Um modelo de avaliação dos requisitos no processo de desenvolvimento de software

Luiz Gustavo Mendes Rodrigues

Trabalho Final de Mestrado Profissional em Computação

Instituto de Computação
Universidade Estadual de Campinas

Um modelo de avaliação de requisitos no processo de desenvolvimento de software

Luiz Gustavo Mendes Rodrigues

Fevereiro de 2006

Banca Examinadora:

- **Prof. Dr. Mario Jino (Orientador)**
  Faculdade de Engenharia Elétrica e de Computação - Unicamp
- **Prof. Dr. Adalberto Nobiato Crespo**
  Centro de Pesquisa Renato Archer - CenPRA
- **Prof. Dr. Rogério Drummond Burnier Pessoa de Mello Filho**
  Instituto de Computação - Unicamp
- **Profa. Dra. Eliane Martins (Suplente)**
  Instituto de Computação – Unicamp

Um modelo de avaliação dos requisitos no processo de desenvolvimento de software

Campinas, 21 de fevereiro de 2006.

Prof. Dr. Mario Jino
(Orientador)

Prof. Dr. Cid Carvalho de Souza
(co-orientador)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na área de Engenharia de Software.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Rodrigues, Luiz Gustavo Mendes

R623m Um modelo de avaliação dos requisitos no processo de desenvolvimento de software / Luiz Gustavo Mendes Rodrigues -- Campinas, [S.P. :s.n.], 2006.

Orientadores : Mário Jino; Cid Carvalho de Sousa

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software – Qualidade. 2. Engenharia de software. 3. Software – Testes. I. Jino, Mário. II. Sousa, Cid Carvalho de. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Dedico este trabalho aos meus pais, que sempre me incentivaram a buscar novos conhecimentos e experiências, às minhas irmãs queridas, e em especial a minha esposa Veridiana, pelo companheirismo, paciência e dedicação.

Agradecimentos

Muito obrigado ao Prof. Mario Jino pela orientação e, pela oportunidade de aprendizado durante todo o tempo de estudo. O aprofundamento na área de testes de sistemas abriu para mim muitas oportunidades durante este período.

Muito obrigado ao Prof. Marcelo Maia pelo auxílio dado em momento que mais precisei.

Muito obrigado aos meus pais e irmãs pelo constante incentivo. Palavras sinceras de incentivo sempre nos dá força para seguir em frente.

Muito obrigado a minha esposa por estar sempre ao meu lado em todos os momentos.Te amo muito! Agradeço também ao meu cão, Hans, pelos momentos de descontração. Afinal, faz parte!

Muito obrigado ao meu amigo e colega de viagens e estudos Marcus Valério. Seu companheirismo foi muito importante ao longo do curso. E também agradeço aos colegas de curso e em especial aos que se tornaram meus amigos: Dinailton, Adão, Wennder, João Marcelo Josko e Marco Ganhoto. Sentirei saudades.

Muito obrigado aos colegas e amigos que de alguma forma me ajudaram na elaboração do conteúdo deste trabalho: Márcia, Diclei, Marcy, Patrícia, Edward, Guilherme e Daniela. Em especial aos amigos Rafael Capanema e Elton Costa.

Muito obrigado ao Sr. Sebastião Bruneli e à Ana Cláudia pela dedicação em me buscar de madrugada na rodoviária. Serei sempre grato!

Resumo

Literatura recente aponta que aproximadamente 40% dos fatores de fracasso de um projeto estão relacionados com defeitos nos requisitos do projeto. Este problema tem elevado o custo da construção de um software bem como afetado a qualidade dos produtos entregues. Qualidade de software está fortemente ligada à qualidade do processo de desenvolvimento de software e teste de software tem grande importância no processo de garantia da qualidade, pois as técnicas de teste são as últimas a serem aplicadas para assegurar que o software satisfaz suas especificações.

Este trabalho propõe uma adaptação no Modelo V de desenvolvimento de software, visando criar mecanismos para melhorar a qualidade dos requisitos de software e para dar suporte às atividades de teste, como uma maneira de melhorar a eficácia do processo de desenvolvimento de software. O Modelo Proposto introduz atividades para validação de requisitos, que aplicam técnicas relacionadas a teste de software desde o início de um projeto de software.

O Modelo Proposto foi avaliado por meio de um estudo experimental aplicado em projetos reais. Os resultados indicam que as técnicas introduzidas no Modelo V são eficazes e que ganhos em qualidade e custo são obtidos pelo uso do Modelo Proposto.

Abstract

Recent literature indicates that approximately 40% of the failure factors of a project are related to defects in project requirements. This problem has increased the cost of software development as well as has affected the quality of delivered products. Software quality is strongly connected with the quality of the software development process and software testing is very important in the quality assurance process, as test techniques are the last ones applied to assure that software meets its requirements.

This work proposes an adaptation in the V Model of software development, aimed to create mechanisms to improve the quality of software requirements and to support testing activities, as a way to improve the efficacy of the software development process. The Proposed Model introduces techniques for requirements validation, which apply software test activities since the early stages of a software project.

The Proposed Model was evaluated by means of an experimental study applied to real projects. The results indicate that the techniques introduced in the V Model are efficacious and that gains in quality and cost are obtained by using the Proposed Model.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Contexto

Uma pesquisa americana publicada em 1994 pelo Instituto Standish Group [STD94], utilizando uma amostra de 365 companhias entrevistadas, representando 8.380 aplicações, chegou às seguintes informações:

- Os EUA gastaram US$ 250 bilhões por ano em desenvolvimento de aplicação de tecnologia da informação em aproximadamente 175 mil projetos.
- 31% de todos os projetos foram cancelados antes do seu término, representando um desperdício da ordem de US$ 81 bilhões.
- 53% de todos os projetos chegaram ao final tendo custado 189% do valor estimado, representando US$ 59 bilhões em custo adicional, atrasaram em até 222% da estimativa original além de serem entregues com apenas 61% das características originalmente especificadas.
- 16% foram entregues no prazo e dentro do orçamento.

A pesquisa ainda explorou as principais razões que levaram a problemas de projeto, como mostra a Tabela 1.1. Frente aos principais fatores de fracasso de um projeto mostrado na Tabela 1.1 podemos entender a necessidade de melhorias nas formas de entendimento e elaboração dos requisitos de um sistema. Os problemas relacionados com os requisitos do sistema são os primeiros da lista e totalizam quase 40% das causas de dificuldade de um projeto.

De acordo com Pressman [PRE02], o custo de correção de defeitos na fase de projeto é cerca de três a seis vezes mais alto do que na fase de definição de requisitos. E este custo aumenta ainda mais quando a correção de defeitos é realizada em fases mais avançadas do processo de desenvolvimento, como a fase de teste.

Portanto, por que deixar a busca e eliminação de defeitos para as últimas fases do processo de desenvolvimento, sabendo-se que quanto mais se distancia das fases iniciais mais cara ficará sua posterior correção? Como identificar os defeitos antes do início da construção do produto? A resposta está no bom entendimento do que o usuário está pedindo. Para este fim existe um documento que traduz o que o usuário pede, o documento de requisitos, que especifica tanto os requisitos funcionais quanto os não-funcionais; quando bem elaborado, o documento de

requisitos reduz em muito a quantidade de defeitos nas fases futuras e melhora a qualidade do software.

Tabela 1.1 – Principais fatores que tornam um projeto crítico [STD94]

Fatores que tornam um Projeto Crítico	% de Respostas
1. Falta de Especificação do Usuário	12,8%
2. Requisitos Incompletos	12,3%
3. Mudança de Requisitos	11,8%
4. Falta de Apoio Executivo	7,5%
5. Tecnologia Imatura	7,0%
6. Falta de Recursos	6,4%
7. Expectativas irreais	5,9%
8. Objetivos obscuros	5,3%
9. Tempo irreal	4,3%
10. Tecnologia nova	3,7%
Outros	23,0%

A norma IEEE610 [IEEE610-90] define qualidade como sendo o grau em que um sistema, componente ou processo satisfaz as necessidades dos usuários. Para a elaboração de bons documentos os processos usados para desenvolver um projeto de software têm enorme importância. Assim, qualidade do software está fortemente ligada à qualidade do processo de desenvolvimento de software. Os testes de software têm grande importância no processo de garantia da qualidade, pois visam assegurar que o software satisfaz suas especificações. Assim, garantia de qualidade de software é uma atividade que é aplicada ao longo do processo e teste de software é um elemento crítico da garantia de qualidade de software [PRE02]. Sendo assim porque não desenvolver atividades relacionadas a teste desde o início do projeto?

1.2 Motivação

Apesar da grande relevância dos requisitos para a construção de um software, grande parte dos processos de construção de software praticados nas empresas não dá a devida atenção a este artefato. Inúmeras fontes de pesquisa mostram que o fracasso de grande parte dos projetos

ocorreu por falhas na elaboração dos requisitos do software [STD94] [SHE92] [BOE81] [GAO92].

Problemas que permanecem sem serem encontrados até a fase de teste são no mínimo 20 vezes mais caros para consertar do que se fossem encontrados na fase de levantamento dos requisitos [FAU97]. Assim, a validação de requisitos torna-se uma atividade essencial no processo de desenvolvimento de software, pois visa garantir que os requisitos do software reflitam as funcionalidades levantadas pelo cliente. Algumas dentre as diversas técnicas de validação existentes podem ser usadas em diversos momentos na construção do software. São o caso da prototipação e do teste de requisitos, que estão incorporadas no modelo do processo proposto neste trabalho. Além da validação dos requisitos veremos, no modelo do processo proposto, o emprego dessas técnicas na atividade de teste de aceitação.

1.3 Objetivos

Este trabalho propõe uma adaptação do modelo V de desenvolvimento de software [MYE79], visando criar mecanismos que melhorem a qualidade dos requisitos de software e auxiliem a atividade de teste melhorando, assim, a eficácia do processo de desenvolvimento de software.

O modelo do processo proposto é fruto da análise de técnicas para validação de requisitos, buscando identificar aquelas cujos princípios melhor se adaptem ao contexto do teste de software. Assim será possível dentro do modelo do processo proposto relacionar atividades da Engenharia de Requisitos com atividades de Teste de Software.

Um estudo experimental foi realizado para avaliar a viabilidade, melhoria e produtividade do modelo do processo proposto. Um time foi montado para aplicar o processo proposto em alguns projetos. Estes projetos foram usados como amostras para o experimento de avaliação da eficácia do modelo do processo proposto.

1.4 Organização do Trabalho

Para um melhor entendimento da abordagem deste trabalho, é necessário conhecer o escopo do trabalho e a sua inserção no contexto dos assuntos abordados. No Capítulo 2

descrevemos a *Engenharia de Requisitos* e, no Capítulo 3, descrevemos o contexto do *Teste de Software*. O Capítulo 4 descreve o *Modelo V do Processo de Desenvolvimento de Software*, que é comparado com o modelo do processo proposto.

O Capítulo 5 contém uma descrição detalhada do modelo do processo proposto, incluindo as técnicas de suporte ao processo, suas particularidades e as etapas de sua execução.

O Capítulo 6 apresenta os resultados do estudo experimental do modelo do processo proposto em comparação ao Modelo V do Processo. O estudo foi realizado em uma empresa do ramo de cartões de crédito e contou com a participação de equipes de teste, de projeto e de desenvolvimento de software, e usuários finais.

O Capítulo 7 finaliza com as considerações finais do trabalho, ressaltando suas contribuições, limitações e perspectivas futuras.

2 Engenharia de Requisitos

A dinâmica do ambiente das empresas exige, nos dias de hoje, a construção de softwares eficientes e eficazes. Porém, até meados da década de 90 a gestão da informação não havia acompanhado a mesma dinâmica. Assim sendo, de lá para cá, muito se fez para que o processo, hoje chamado de engenharia de software, suportasse a rapidez e a exigência de qualidade na construção de um software. Atividades foram criadas para melhorar o processo de construção de um software. Uma delas é a engenharia de requisitos que se baseia na simples transformação do que se pede para o que se faz. O objetivo principal desta engenharia é chamado requisito.

Podemos definir requisito como sendo uma declaração de um serviço ou uma restrição que deve ser implementada em um sistema [KOT98]. Para Kulak um requisito é definido como qualquer coisa que uma aplicação de computador possa fazer para seus usuários [KUL00].

A análise da engenharia de requisitos é desenvolvida sob os pontos de vista do desenvolvedor e do usuário. Do ponto de vista do analista/desenvolvedor, a engenharia de requisitos refere-se ao que o sistema deve fazer e não ao como deve fazê-lo. Está associada, portanto, a uma situação futura, sobre a qual deve ser capaz de fazer previsões e analisar alternativas [LOP99].

Os requisitos de software são a base a partir da qual a qualidade é medida. Desta forma, a falta de conformidade aos requisitos significa falta de qualidade [CAR01]. Cabe à engenharia de requisitos, como sub-área da engenharia de software, aperfeiçoar os processos de gerenciamento do ciclo de vida dos requisitos [ZAN02]. Deve também propor métodos, ferramentas e técnicas que promovam o desenvolvimento do documento de requisitos, para que este produto retrate o conhecimento do problema, em conformidade à satisfação dos stakeholders[1] e aos padrões de qualidade, relacionado ao que se quer produzir com tecnologia da informação para solução dos problemas ou como oportunidade de negócio.

A informação é um produto que envolve opinião e ponto de vista de pessoas, com interesses e prioridades diversos [ZAN02]. Para sua captura são utilizadas técnicas apropriadas ao ambiente e adequadas ao perfil e disponibilidade de tempo do público-alvo. Para representação

[1] Um stakeholder em uma organização é (por definição) algum grupo ou indivíduo que pode afetar ou ser afetado pela realização dos objetivos da empresa [KHA01].

são utilizadas ferramentas do domínio do engenheiro de requisitos que, necessariamente, não são do conhecimento e domínio da fonte de informação.

A engenharia de requisitos procura sistematizar o processo de definição de requisitos. Essa sistematização é necessária porque a complexidade dos sistemas exige que se preste mais atenção ao correto entendimento do problema antes do comprometimento de uma solução [COR02] [LEI94].

Outros fatores relacionados ao ambiente no qual acontece a engenharia de requisitos afetam o processo de definição de requisitos, positiva e negativamente. Pohl [POH94] cita cinco fatores principais relacionados ao ambiente:

- **Métodos** – o processo é influenciado pelos métodos utilizados em seu contexto, uma vez que se concentram em diferentes aspectos (a utilização de análise estruturada deverá produzir uma especificação formal totalmente diferente da obtida pelo uso de análise orientada a objetos);
- **Ferramentas** - a especificação final depende das ferramentas utilizadas durante o processo. Se uma ferramenta de análise baseada em representações formais for utilizada, poderão ser identificadas e removidas inconsistências que, de outro modo, estariam presentes na especificação final;
- **Aspectos sociais** – o ambiente da equipe de engenharia de requisitos afeta sua eficiência no trabalho. Um ambiente no qual as pessoas se entendem e se relacionam melhor produzirá um resultado muito melhor;
- **Competência** – pessoas têm competências e qualificações distintas. O resultado final é freqüentemente melhor se pessoas com as qualificações corretas e a melhor competência conduzirem o processo de requisitos;
- **Restrições econômicas** – limitam os recursos (pessoas, ferramentas, tempo, etc.) que podem ser usados durante o processo. Nem sempre podemos afirmar que com mais recursos um melhor resultado poderia ser obtido. Contudo, se os recursos estiverem abaixo de determinado limite, o resultado certamente será de pior qualidade.

2.1 Os participantes do processo

Parte do entendimento das atividades desenvolvidas no processo de engenharia de requisitos envolve a identificação de seus participantes, caracterizando seus papéis e suas responsabilidades no contexto do processo.

Uma das características do processo de engenharia de requisitos é a diversidade de interesses das pessoas envolvidas. Existem pessoas interessadas no sistema como uma solução para um problema ou um apoio na condução de uma atividade do mundo real, pessoas interessadas na solução de problemas relacionados ao desenvolvimento de tais sistemas, pessoas ligadas a órgãos reguladores interessados em como o uso desses sistemas afeta o ambiente no qual serão instalados, estabelecendo regras e/ou restrições que deverão ser obedecidas pelos sistemas desenvolvidos naquele domínio específico de regulamentação. Essas pessoas são denominadas, genericamente, *stakeholders*. A seguir são listados os principais papéis e funções no processo de requisitos [KOT98]:

- Especialista do Domínio: responsável por prover informações sobre o domínio de aplicação e do problema específico a ser resolvido naquele domínio;
- Usuário final: quem usa o sistema após a entrega;
- Engenheiro de Requisitos: responsável por identificar e especificar os requisitos do sistema;
- Engenheiro de Software: responsável por desenvolver o protótipo do sistema, caso exista;
- Gerente do projeto: responsável pelo planejamento do projeto;

2.2 O que é um requisito

Segundo Abbott [ABB86], é uma função, restrição, ou outra propriedade que precisa ser fornecida, encontrada, ou atendida para satisfazer às necessidades do usuário do futuro sistema.

A norma IEEE 1233 [IEEE1233-98], define requisito como:

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

b) uma condição ou capacidade que precise ser atendida ou estar presente em um sistema ou componente, para satisfazer um contrato, uma norma, uma especificação ou outro documento imposto formalmente;
c) uma representação documentada de uma condição ou capacidade, tal como definidas em a ou b.

Um bom conjunto de requisitos é necessário para um projeto, especialmente um projeto de sistema de computador, para ser bem sucedido. Aí está onde muitos projetos falham; eles não fazem exatamente o que deveriam fazer. Na realidade, muitos projetos de sistemas dispõem somente de um prazo limite para liberação, um orçamento, e uma vaga noção do que deve ser feito [COL04].

Uma boa definição para requisitos [LEI94] é mostrada a seguir.

"***Requisitos:*** *Condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo objetivo*."

2.3 Impacto da descoberta de defeitos

A importância da engenharia de requisitos no contexto de desenvolvimento de software advém do fato de que a identificação e documentação corretas dos requisitos é fundamental para o sucesso do software. Pesquisas têm comprovado que muitos projetos de implementação de software têm falhado por problemas de requisitos de software [BOE81] [GAO92], ou seja, os requisitos obtidos muitas vezes são incompletos, mal entendidos ou ambíguos [MAR01].

Além de ser um fator crítico de sucesso para a implementação de software, no que se refere ao atendimento das reais necessidades dos usuários, os requisitos têm um forte impacto no custo total de projetos de software. Defeitos na especificação de requisitos, comuns nas etapas iniciais do desenvolvimento de software, quando detectados tardiamente, apresentam um alto custo para sua correção [MAR01], conforme apresentado na Tabela 2.1 [FAU97].

Podemos perceber pelos dados da Tabela 2.1 que o custo de reparo de um defeito nas fases iniciais tem um valor muito menor do que quando reparado nas etapas finais do ciclo de vida; obviamente isto tem um forte impacto no custo total do projeto de software.

Tabela 2.1 - Custos relativos de reparo do software nas diferentes etapas do ciclo de vida

Etapas	Custo Relativo de Reparo
Análise	1-2
Projeto	5
Implementação	10
Teste Unitário	20
Teste do Sistema	50
Manutenção	100

O resultado da descoberta antecipada de defeitos é ilustrado também na Figura 2.1 pelo curso da curva chamada de "Novo". O valor da detecção de defeitos antecipadamente aumenta no início, mas rapidamente cai. Esta situação contrasta com a situação típica refletida pelo curso da curva chamada de "Velho", onde os defeitos não são identificados até o início dos testes de sistema quando a correção dos mesmos é mais cara. Prevenção antecipada de defeitos nos requisitos envolve identificar e corrigir problemas antes que eles se propaguem para as próximas fases. A Figura 2.1 também ilustra, pelas curvas "Velho" e "Novo", os ganhos associados à prevenção de defeitos. Prevenção de defeitos é mais eficaz durante a fase de análise de requisitos quando o custo de correção é baixo [BLA01].

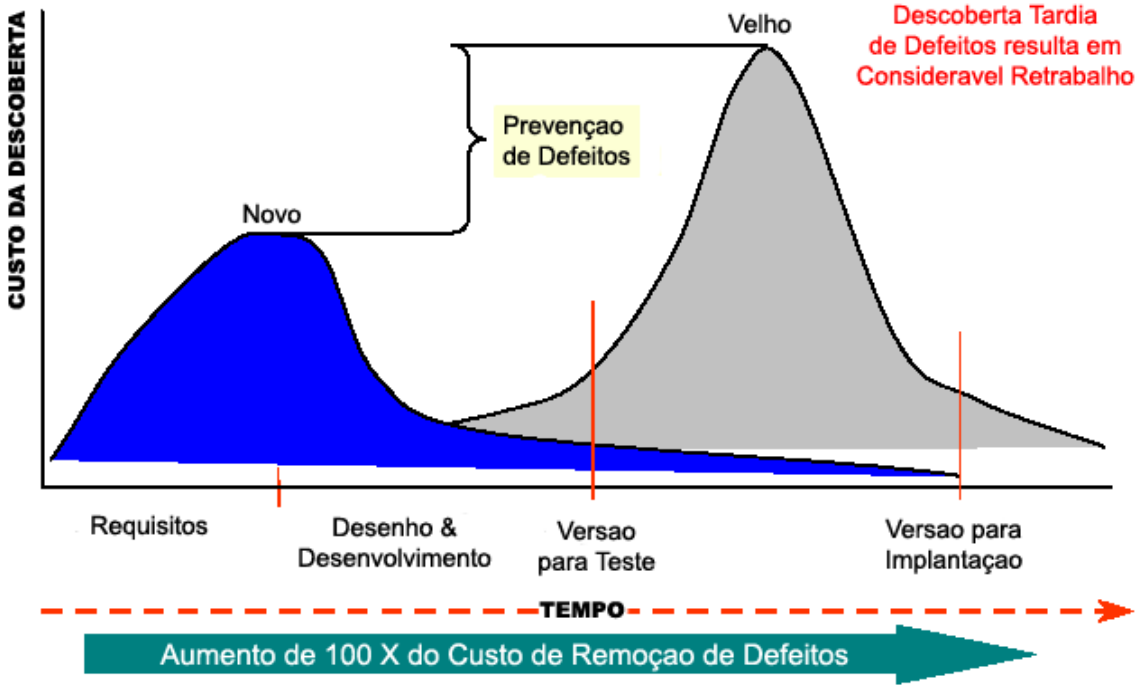

Figura 2.1 - Identificação e Prevenção de Defeitos [BLA01].

Quatro conceitos associados ao teste de software são essenciais para o entendimento do restante deste trabalho; suas definições são dadas na norma IEEE 610 [IEEE610-90]. *Engano* é a ação humana que pode levar a um *defeito*. *Defeito*, por sua vez, é a manifestação física do engano em uma representação do software. *Falha* é a ocorrência observável (perceptível) de um ou mais defeitos quando o software é testado ou utilizado em campo. *Erro* é o estado intermediário ou final, caracterizado por um comportamento incorreto ou um desvio da especificação.

Desta maneira, o processo de ocorrência de falhas durante o teste ou o uso de um programa pode ser resumido da seguinte forma: um *engano* (ação humana) leva a um *defeito* (deficiência algorítmica) que, se ativado, pode provocar um *erro* (estado incorreto do programa); um *erro* que se propaga até a saída provoca a ocorrência de uma *falha*. Um caso de teste é o *revelador de defeito* se uma falha ocorre ao ser executado. Se nenhuma falha ocorre, o caso de teste é *não-revelador de defeito* [CHA02].

2.4 Testabilidade de um requisito

Um defeito em requisito ocorre de várias formas. Um requisito de software incompleto é interpretado de diversas formas, enquanto um requisito testável tem que ser completo, consistente e claro (não-ambíguo). Um requisito testável pode incluir algum conhecimento implícito do domínio, mas este conhecimento deve ser claro ou documentado com a garantia que o requisito está consistentemente ligado com o contexto do sistema a ser testado. Algum erro de interpretação em potencial do requisito é um defeito [BLA01].

Podemos citar como algumas das formas de defeito em requisito as contradições ou problemas de interação. Estes defeitos surgem devido a inconsistências em ou contradições entre requisitos. Estes problemas podem ser difíceis de identificar quando requisitos são documentados de maneira formal ou semi-formal, assim como em textos descritivos. Freqüentemente informações contraditórias são descritas em várias páginas de documentação e são encaminhadas para equipes de duas ou mais pessoas. Embora abordagens rigorosas e inspeções manuais possam auxiliar reduzindo as contradições e imperfeições, estes procedimentos estão ainda associados aos limites da capacidade humana e são muito dependentes do envolvimento pessoal.

2.5 Validação de Requisitos

A engenharia de requisitos enquanto um processo de "construção" dos requisitos divide-se em quatro etapas: elicitação, análise, especificação e validação dos requisitos. A elicitação dos requisitos expande a declaração informal dos requisitos, oferecendo subsídios para análise dos requisitos que resultará nos requisitos acordados entre usuários e desenvolvedores. Os requisitos acordados devem ser especificados na etapa de especificação dos requisitos, gerando esboços da documentação dos requisitos [MAR01].

Após a documentação dos requisitos ter sido produzida, inicia-se o processo de validação, buscando determinar se os requisitos estão corretos, ou seja, descritos de forma apropriada, procurando eliminar problemas de incompleteza, ambigüidade e inconsistência [COR02].

A preocupação maior desta fase é com a qualidade do documento de requisitos produzido [COR02]. Uma lista de tarefas é sugerida nesta fase e várias questões devem ser respondidas para que se dê continuidade às fases posteriores.

Validação de requisitos visa garantir que os requisitos reflitam a funcionalidade levantada pelos clientes. A atividade de validação de requisitos requer um analista, usuário ou cliente decidindo a validade de cada requisito. Modelos provêem um meio para que os clientes entendam precisamente os requisitos e auxiliem na identificação das omissões [BLA01].

A principal técnica utilizada na validação de requisitos é basicamente aquela utilizada em outras atividades do processo de software: revisões. Além desta técnica, são também utilizadas validação por Protótipos, Validação de Modelos e Teste.

2.5.1 Revisões

Lopes [LOP02] define esta técnica de validação de requisitos da seguinte forma:

> "Revisões consistem de reuniões estruturadas nas quais um grupo de pessoas, prévia e cuidadosamente escolhidas, após lerem e analisarem o documento de requisitos, reúnem-se para discutir os problemas encontrados e chegar a uma solução de consenso em relação às ações a serem adotadas para corrigi-los".

Esta técnica exige formalização das reuniões documentando tudo o que foi discutido para melhor andamento das discussões. Exige-se também um rigoroso critério de seleção dos

participantes deste processo. Pessoas com diferentes bases de conhecimento trazem mais experiência à reunião de revisão e tornam mais provável a identificação de problemas nos requisitos. Além disso, ao discutir os problemas identificados com profissionais de outras especializações, os *stakeholders* tendem a entender melhor as razões que levaram a mudanças nos requisitos por eles propostos [LOP02].

Inspeção de Software

Inspeção de software é um método de análise estática para verificar as propriedades de qualidade de produtos de software. Segundo Christiensen [CRIS01], inspeção de software tem sido o tipo de revisão de software mais estudado e utilizado.

Os objetivos principais da inspeção de software são identificar erros específicos num documento ou sistema, identificar erros sistemáticos no processo de desenvolvimento e identificar desvios em relação às especificações e padrões. A inspeção de software cobre não somente inspeção de requisitos, mas qualquer artefato produzido ao longo do processo de desenvolvimento de software.

Algumas técnicas foram criadas para guiar a atividade de inspeção. As principais técnicas são a *ad hoc*, *checklists* e a PBR (Leitura Baseada em Perspectiva). Todas elas são técnicas de leitura com características particulares. Na técnica PBR cada inspetor assume a perspectiva de um usuário específico do documento. A base da técnica ad hoc está na experiência do time que realiza a inspeção e na pouca formalidade do processo de inspeção. Na técnica de checklist o inspetor seque uma lista de itens com características a serem revisadas, mas ainda aplicando leitura ad hoc para identificar os defeitos.

2.5.2 Desenvolvimento de Protótipos

Com o auxílio de um *stakeholder,* um desenvolvedor tem a possibilidade de construir protótipos para validar um conjunto de requisitos. Este tipo de validação não se restringe a discussões de interfaces com o usuário. Um protótipo simula uma parte do sistema e pode ser desenvolvido de diversas maneiras.

Se possível, use uma ferramenta de simulação que garanta que o cliente não espere utilizar o protótipo no produto final – jogue fora o protótipo. Se o desenvolvedor achar que há alguma

possibilidade de reutilização do protótipo, ele deve ser avaliado da mesma maneira como avaliamos a reutilização de código, planos, modelos, testes, e assim por diante.

Um protótipo para validação dos requisitos deverá incluir um número razoável de facilidades para possibilitar o uso prático do sistema. Caso contrário, os *stakeholders* não poderão utilizar o sistema de uma forma natural, invalidando o esforço de validação [LOP02].

Os protótipos são construídos na maioria das vezes para simular importantes interfaces e as principais funcionalidades do sistema pretendido, não necessariamente em ambientes similares aos de produção. Eles, em sua grande maioria, não incluem tratamento de erros, respostas corretas a entradas inválidas ou encerramento claro de um processo [BRO87].

Lopes [LOP02] sugere que caso seja escolhida a prototipação como técnica de elicitação dos requisitos, validar requisitos por meio do protótipo pode ser uma boa idéia.

Os requisitos de um sistema nem sempre são totalmente descritos, nem totalmente entendidos. Prototipação auxilia o cliente na formalização ou solidificação dos requisitos do sistema.

Seguem abaixo as vantagens da construção de um protótipo:

- Extração dos requisitos;
- Validação dos requisitos;
- Ponto de auxílio na comunicação entre desenvolvedores e *stakeholders;*
- Treinamento de usuários;
- Demonstra o sistema operando (acompanhamento do trabalho frente ao usuário);
- Teste do sistema (teste paralelo).

Conforme Spinellis [SPI03] retrata em pesquisa realizada examinando 39 projetos envolvendo construção de protótipos, os gerentes ressaltaram as seguintes vantagens:

- Um produto final usável;
- Requisitos mais corretos;
- Melhor qualidade no desenho do sistema;
- Fácil manutenção;
- Menor custo na construção do sistema.

2.5.3 Validação de Modelos

A modelagem provê uma forma de formalização do requisito. A disciplina e estrutura do processo de modelagem ajudam a eliminar imperfeições, e o resultado do modelo constrói uma base para ferramentas auxiliarem na detecção antecipada de imperfeições e contradições no processo de desenvolvimento. Análise da testabilidade do requisito é o processo de refinamento e elucidação dos requisitos por meio de modelos usando uma combinação de processo e ferramentas de análise automatizada para desenvolver requisitos livres de defeitos [BLA01].

A documentação de requisitos pode conter modelos desenvolvidos segundo diversas notações. Segundo Kotonya e Sommerville [KOT98] três são os objetivos desta atividade:

- Demonstrar que cada um dos modelos é consistente de forma a conter toda a informação necessária e não haver conflitos entre suas diferentes partes;
- Demonstrar, na existência de diversos modelos, que eles são consistentes externa e internamente. Isto inclui verificar nomenclatura por meio dos modelos, consistência das referências múltiplas, etc;
- Demonstrar que os modelos refletem precisamente os requisitos reais dos stakeholders. Esta é a parte mais difícil da validação de modelos, uma vez que consiste na produção de argumentos capazes de convencer que o modelo realmente representa os requisitos.

2.5.4 Teste de Validade de Requisitos

Uma das características de qualidade de um requisito bem elaborado é que ele possa ser testado. Uma boa maneira de identificar problemas nos requisitos, tais como falta de completitude ou ambigüidade, é tentar propor casos de teste para um requisito. É preciso ter em mente que o objetivo de se propor casos de teste nesta fase é validar o requisito e não o sistema [LOP02].

Lopes [LOP02] ainda sugere que “ao testarmos requisitos, tal como na atividade de testes, devemos registrar cuidadosamente o resultado dos testes. Assim, convém fornecer um modelo a quem for testar os requisitos para que o resultado do teste seja registrado a contento”.

Também se propõe o teste de requisitos criando-se critérios de aceitação para validar os requisitos criados pelo *stakeholder* em conjunto com as equipes de desenvolvimento e de teste.

Porém, estes critérios devem estar ligados aos casos de teste criados para cada requisito e que, posteriormente, servirão para os testes do sistema.

2.6 Esforço na construção dos testes

As tarefas relativas ao projeto dos testes são tipicamente manuais e propensas a erros e podem gastar aproximadamente 60% do esforço gasto nos testes. Empresas têm relatado o gasto de 50% do esforço durante a fase de teste em criação e depuração de roteiro de testes. Automatizando o processo de construção e manipulação dos testes ou de criação de roteiros pode resultar em considerável diminuição dos custos e maior eficácia nos testes [BLA01].

Safford [SAF00] relatou resultados condicionando a redução de custo, esforço, e tempo do ciclo de desenvolvimento à eliminação dos defeitos dos requisitos e automatização dos testes. Os relatos de Safford resumem os seguintes benefícios:

- Requisitos de melhor qualidade para o desenho e implementação do sistema ajudam a eliminar retrabalho nestas fases tanto quanto durante os testes;
- Testes gerados de um modelo de verificação podem eliminar até 90% de esforço em criação manual de testes e depuração;
- O número de casos de teste pode ser reduzido e a fase de execução pode ser melhorada, eliminando redundância de testes;
- Um conhecido nível de cobertura dos requisitos pode ser planejado e medido durante a execução dos testes.

3 Teste de Software

Teste de software é um elemento crítico da garantia de qualidade de software e representa a revisão final da especificação, projeto e geração de código [PRE02]. O desenvolvimento de sistemas de software envolve uma série de atividades de produção em que as oportunidades para injetar a falibilidade humana são enormes. Assim, por causa da inabilidade humana de realizar e de se comunicar com perfeição, o desenvolvimento é acompanhado por uma atividade de garantia de qualidade.

Segundo Pressman [PRE02], não é raro o teste demandar 40% do esforço total de um projeto. Isto deve-se à busca por qualidade em determinados softwares de algumas organizações.

Para Myers [MYE79] teste é o processo de execução de um programa com a intenção de achar defeitos. E a intenção estaria associada à adição de algum valor ao programa, ou seja, melhorar a qualidade ou a segurança do programa. Caso de teste é o instrumento pelo qual é possível identificar um defeito. Um bom caso de teste é aquele que tem alta probabilidade de encontrar um defeito ainda não descoberto.

Sabemos que é impossível testar aplicando todos os possíveis casos de teste de um programa devido a restrições de tempo e custo. Assim, é necessário determinar quais os casos de testes que serão utilizados a fim de que a maioria dos defeitos existentes possa ser detectada e que o número de casos de teste utilizados não seja tão grande a ponto de ser impraticável.

Várias técnicas foram criadas para sistematizar a condução dos testes de software; associadas a essas técnicas, ferramentas de teste de software foram desenvolvidas para apoiar a aplicação dessas técnicas. Isto porque, na falta de ferramentas de teste de software, o processo teria que ser realizado manualmente, tornando impraticável a aplicação da maioria das técnicas de teste.

O projeto efetivo de casos de tese é importante, mas a estratégia que usamos para executá-los também é. Se a atividade de teste for conduzida ao acaso, defeitos no projeto podem passar despercebidos durante o teste. Portanto, uma estratégia sistemática deve ser estabelecida para o teste de software, ou seja, um conjunto de atividades que podem ser planejadas antecipadamente e conduzidas sistematicamente.

Para Pressman [PRE02], o teste de software é composto por uma seqüência de quatro etapas. Inicialmente, o teste focaliza cada componente individualmente, garantindo que ele

funciona adequadamente como uma unidade. Em seguida, os componentes devem ser montados ou integrados para formar o pacote de software completo. Depois do software integrado, um conjunto de teste de alto nível é conduzido. Critérios de validação precisam ser aplicados para tentar garantir que o software satisfaz todos os requisitos funcionais.

Os métodos de teste caixa-preta concentram-se nos requisitos funcionais do software. São usados para demonstrar que as funções do software estão operacionais, que a entrada é adequadamente aceita e a saída é corretamente produzida, e que a integridade da informação externa (por exemplo, uma base de dados) é mantida. Teste caixa-branca é baseado num exame rigoroso de detalhes procedimentais e de implementação. Caminhos lógicos internos ao software são testados, definindo casos de testes que exercitam conjuntos específicos de condições e/ou laços [PRE02].

3.1 Fases de teste de software

3.1.1 Teste de unidade

O teste de unidade focaliza o esforço de verificação na menor unidade de projeto do software – o componente ou módulo de software. Usando a descrição de projeto em nível de componente como guia, caminhos de controle importantes são testados para descobrir defeitos dentro das fronteiras do módulo. O teste de unidade é orientado para caixa-branca e o passo pode ser conduzido em paralelo para diversos componentes [PRE02].

Como um componente não é um programa isolado, um módulo pseudocontrolador (driver) e módulos pseudocontrolados (stub) devem ser desenvolvidos para cada teste de unidade. Na maioria das aplicações um pseudocontrolador nada mais é do que um "programa principal", que aceita dados do caso de teste, passa tais dados ao componente (a ser testado) e imprime os resultados relevantes. Os pseudocontrolados servem para substituir módulos que são subordinados (chamados pelo) ao componente a ser testado. Um pseudocontrolado usa a interface dos módulos subordinados, pode efetuar um mínimo de manipulação de dados, imprime a verificação da entrada e devolve o controle ao módulo que está sendo processado [PRE02].

3.1.2 Teste de Integração

O teste de integração é uma técnica sistemática para construir a estrutura do programa enquanto, ao mesmo tempo, conduz testes para descobrir defeitos associados às interfaces. O objetivo do teste de integração é usar componentes testados em nível de unidade para construir a estrutura de programa determinada pelo projeto. Existem dois tipos de teste de integração, a não-incremental e incremental. Pressman [PRE02] sugere a abordagem incremental para os testes de integração, que propõem a construção e teste do programa em pequenos incrementos, em que os defeitos são fáceis de isolar e corrigir. Dentre as estratégias incrementais existentes para o teste de integração as principais são a integração descendente, a integração ascendente e a integração sanduíche.

3.1.3 Teste de Sistema

Teste de sistema abrange um conjunto de diferentes tipos de teste cuja finalidade principal é exercitar por completo o sistema baseado em computador. Apesar de cada tipo de teste ter uma finalidade distinta, todos visam verificar se os elementos do sistema foram adequadamente integrados e executam as funções a eles alocadas [PRE02].

São quatro os tipos de teste de sistema mais comuns. Teste de recuperação, em que o software é forçado a falhar e verifica-se se a recuperação foi adequadamente realizada. Teste de segurança, em que se tenta verificar se os mecanismos de proteção incorporados a um sistema vão de fato protegê-lo de invasão imprópria. Teste de fadiga, em que o sistema é executado de modo que demande recursos em quantidade, freqüência ou volume anormais. E por fim, teste de desempenho, que testa o desempenho do software quanto ao tempo de resposta e uso de recursos durante sua execução, no contexto de um sistema integrado.

3.1.4 Teste de Validação

Após a finalização do teste de sistema, o sistema de software deve ser submetido a um conjunto de testes caixa-preta para demonstrar a sua conformidade aos requisitos definidos nas Especificações dos Requisitos do Software – é o Teste de Validação.

Baseado na descrição de um plano de teste e na definição dos procedimentos de teste, o teste de validação visa assegurar que os requisitos funcionais estão satisfeitos, todas as características comportamentais foram implementadas e todos os requisitos de desempenho foram alcançados [PRE02]. Revisão da configuração e testes alfa e beta são processos adotados nesta fase.

3.2 Técnicas de Teste de Software

Para reduzir os custos associados ao teste, é fundamental a aplicação de técnicas que indiquem como testar o software e de critérios que determinem quando parar os testes, de forma que esta atividade possa ser conduzida de modo planejado e sistemático [PRE02].

As técnicas de teste de software fornecem ao desenvolvedor uma abordagem sistemática de como fazer o teste, ou seja, um conjunto de métodos para ajudar a descobrir a maior quantidade de defeitos possível.

Segundo Presssman [PRE02], quando o software para computador é considerado, existem duas maneiras dele ser testado: teste caixa-preta ou teste funcional, teste conduzido na interface do software, e teste caixa-branca ou teste estrutural, baseado num exame criterioso dos detalhes procedimentais.

As duas técnicas de teste têm focos distintos, isto é, são abordagens que se complementam e que visam revelar diferentes classes de defeitos.

3.2.1 Técnica Funcional

Esta técnica enfoca os requisitos funcionais do software, ou seja, visa derivar conjuntos de condições de entrada que vão exercitar todos os requisitos funcionais de um programa [PRE02]. Assim sendo, uma especificação correta do software é essencial para que o teste funcional seja de boa qualidade.

Também conhecida como teste caixa-preta, Myers [MYE79] descreve-a como estratégia de teste na qual o testador vê o programa como uma caixa preta. Isto é, o testador está completamente despreocupado com a estrutura e com o comportamento interno do programa. Ao contrário, o testador está interessado somente em achar em quais circunstâncias o programa não

se comporta conforme suas especificações. Os dados de testes são derivados exclusivamente das especificações.

O teste funcional tenta encontrar defeitos das seguintes categorias: funções incorretas ou omitidas, defeitos de interface, defeitos de estrutura de dados ou de acesso à base de dados externa, defeitos de comportamento ou desempenho e defeitos de iniciação e término [PRE02].

a) Particionamento de Equivalência

Como já citado anteriormente, é praticamente impossível a execução de todos os possíveis testes em um sistema. Portanto, estamos limitados a testar um pequeno conjunto de casos de testes. Uma maneira de criar estes conjuntos é realizar uma boa seleção de casos de teste segundo duas propriedades: casos de teste que reduzem, de um valor que é maior do que um, o número adicional de casos de teste que precisam ser projetados para atingir um teste razoável e casos de teste que nos dizem algo sobre a presença ou ausência de classes de erros, ao invés de um erro associado somente com o teste específico em mãos [MYE79].

Para Pressman [PRE02], as classes de equivalência podem ser definidas de acordo com as seguintes diretrizes:

- Se uma condição de entrada especifica um *intervalo*, uma classe de equivalência válida e duas inválidas são definidas.
- Se uma condição de entrada exige um *valor* específico, uma classe de equivalência válida e duas inválidas são definidas.
- Se uma condição de entrada especifica um membro de um *conjunto*, uma classe de equivalência válida e uma inválida são definidas.
- Se uma condição de entrada especifica é *booleana*, uma classe de equivalência válida e uma inválida são definidas.

b) Análise de Valores Limites

É uma técnica de projeto de casos de teste que completa o particionamento de equivalência. Em vez de selecionar qualquer elemento de uma classe de equivalência, a análise de valor limite leva à seleção de casos de teste nas “bordas” da classe. Em vez de focalizar somente as condições de entrada, a técnica deriva casos de teste também para o domínio de saída [MYE79].

De acordo com PRESSMAN [PRE02], as diretrizes para a análise de valores limites são semelhantes em muitos aspectos às que são fornecidas para o particionamento de equivalência:

- 1 – Se uma condição de entrada especifica um intervalo limitado pelos valores *a* e *b*, casos de testes devem ser projetados com os valores *a* e *b* e imediatamente acima e imediatamente abaixo de *a* e *b.*
- 2 – Se uma condição de entrada especifica vários valores, casos de teste devem ser desenvolvidos para exercitar os números mínimo e máximo. Valores imediatamente acima e imediatamente abaixo do mínimo e do máximo também são testados.
- 2 – Aplicar as diretrizes 1 e 2 às condições de saída. Por exemplo, considere que uma tabela de temperatura *versus* pressão é esperada como saída de um programa de análise de engenharia. Casos de teste devem ser projetados para criar um relatório de saída que produza o número máximo (e mínimo) admissível de entradas na tabela.
- 4 – Se as estruturas de dados internas do programa têm limites prescritos (por exemplo, um vetor tem um limite definido de 100 entradas), certifique-se de projetar um caso de teste para exercitar a estrutura de dados no seu limite.

c) Grafo de Causa-Efeito

Estabelece requisitos de teste baseado nas possíveis combinações das condições de entrada que os critérios anteriores não exploram. Primeiramente, são levantadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) do programa. A seguir é construído um grafo relacionando as causas e efeitos levantados. Esse grafo é convertido em uma tabela de decisão a partir do qual são derivados os casos de teste [DOM02].

3.2.2 Técnica Estrutural

O teste estrutural ou de caixa branca é uma técnica de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de teste. Esta estratégia de teste permite examinar a estrutura interna do programa.

Para Pressman [PRE02], o teste caixa branca permite derivar os casos de teste que:

- garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez;
- exercitem todas as decisões lógicas para valores falsos ou verdadeiros;

- executem todos os laços em suas fronteiras e dentro de seus limites operacionais;
- exercitem as estruturas de dados internas para garantir a sua validade.

Como o teste estrutural é baseado na análise do código fonte, as técnicas evidentemente dependem da linguagem de programação utilizada.

a) Teste de Caminhos Básicos

O método do caminho básico consiste em o projetista de casos de teste calcular uma medida da complexidade lógica de um projeto procedimental e usar essa medida como guia para definir um conjunto básico de caminhos de execução [PRE02].

Complexidade ciclomática é a métrica de software que fornece uma medida quantitativa de complexidade lógica de um programa. Quando usada no contexto do método de teste de caminhos básicos, o valor calculado para a complexidade ciclomática define o número de caminhos independentes de um conjunto base de um programa e nos fornece um limite superior para a quantidade de testes que deve ser conduzida para garantir que cada comando tenha sido executado pelo menos uma vez [PRE02].

Grafo de Fluxo de Controle ou *Grafo de Programa*, serve com uma ferramenta útil para entender esta técnica e ilustrar a sua abordagem. Os critérios mais conhecidos baseados no fluxo de controle são:

- **Todos-nós**: Todos os nós (ou comandos) devem ser executados pelo menos uma vez; dessa forma, a execução do programa deve incluir cada vértice do Grafo de Fluxo de Controle pelo menos uma vez;
- **Todos-arcos**: Todos os arcos (ou decisões) devem ser executados pelo menos uma vez, ou seja, exige que a execução do programa inclua menos uma vez cada aresta do Grafo de Fluxo de Controle;
- **Todos-caminhos**: Todos os caminhos possíveis em um Grafo de Fluxo de Controle devem ser executados pelo menos uma vez, requerendo assim, que todos os caminhos possíveis do programa seja executados (impraticável, pois o número de caminhos pode ser muito grande).

b) Teste de Fluxo de dados

Os critérios baseados em fluxo de dados baseiam-se na análise de fluxo de dados do programa. A análise do fluxo de dados é utilizada tradicionalmente na implementação de compiladores, para otimização de código. Este tipo de análise considera as relações entre definições e usos de variáveis, preenchendo a lacuna (critérios muito fracos e critério extremamente exigente) existente entre os critérios baseados no fluxo de controle [RAP85].

Rapps e Weyuker [RAP85] definem uso de uma variável como sendo:

- **c-uso** (uso computacional): quando a variável aparece em uma expressão ou em um comando de saída.
- **p-uso** (uso predicativo): quando a variável ocorre em um predicado e, portanto, afeta o fluxo de controle do programa (este uso está associado ao arco).

3.2.3 Técnica baseada em Defeitos

O teste baseado em defeitos utiliza informações sobre os tipos de defeitos mais freqüentes no processo de desenvolvimento de software para derivar os requisitos de teste. A ênfase da técnica está nos defeitos que o programador ou projetista pode introduzir durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência. Semeadura de Defeitos e Análise de Mutantes são critérios típicos que se concentram em defeitos [MAL03]:

- **Semeadura de defeitos**: neste critério, uma quantidade conhecida de defeitos é semeada artificialmente no programa. Após o teste, do total de defeitos encontrados verificam-se quais são originais e quais são semeados. Usando estimativas de probabilidade, o número de defeitos originais ainda existentes no programa pode ser calculado.
- **Análise de mutantes**: é um critério que utiliza um conjunto de programas ligeiramente modificados (mutantes) obtidos a partir de um determinado programa P para avaliar o quanto um conjunto de casos de teste T é adequado para o teste de P. O objetivo é encontrar um conjunto de casos de teste capaz de revelar, pela execução de P, as diferenças de comportamento existentes entre P e seus mutantes.

3.3 Norma IEEE 829-1998

Existem diversos tipos de documentos usados em teste de software, criados em muitos casos pelas necessidades encontradas por cada organização. A norma IEEE 829 [IEEE829-98] agrupa diversas soluções e apresenta algumas boas práticas na área. Criada em 1983 e revisada em 1998, a norma têm como proposta descrever um conjunto básico de documentos de teste de software.

Os documentos definidos pela norma são oito e estão distribuídos em três diferentes categorias: planejamento, especificação e relatos de teste. Estes documentos são definidos a seguir [CRE04]:

- **Plano de Teste** – Apresenta o planejamento para execução do teste, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste. Identifica os itens e as funcionalidades a serem testadas, as tarefas a serem realizadas e os riscos associados com a atividade de teste.

A tarefa de especificação de testes é coberta por 3 documentos:

- **Especificação de Projeto de Teste** – Refina a abordagem apresentada no Plano de Teste e identifica as funcionalidades e características a serem testadas pelo projeto e por seus testes associados. Este documento também identifica os casos e os procedimentos de teste, se existirem, e apresenta os critérios de aprovação.
- **Especificação de Caso de Teste** – Define os casos de teste, incluindo dados de entrada, resultados esperados, ações e condições gerais para a execução do teste.
- **Especificação de Procedimento de Teste** – Especifica os passos para executar um conjunto de casos de teste.

Os relatórios de teste são cobertos por 4 documentos:

- **Diário de Teste** - Apresenta registros cronológicos dos detalhes relevantes relacionados com a execução dos testes.
- **Relatório de Incidente de Teste** - Documenta qualquer evento que ocorra durante a atividade de teste e que requeira análise posterior.
- **Relatório-Resumo de Teste** – Apresenta de forma resumida os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste e provê avaliações baseadas nesses resultados

- **Relatório de Encaminhamento de Item de Teste** – Identifica os itens encaminhados para teste no caso de equipes distintas serem responsáveis pelas tarefas de desenvolvimento e de teste.

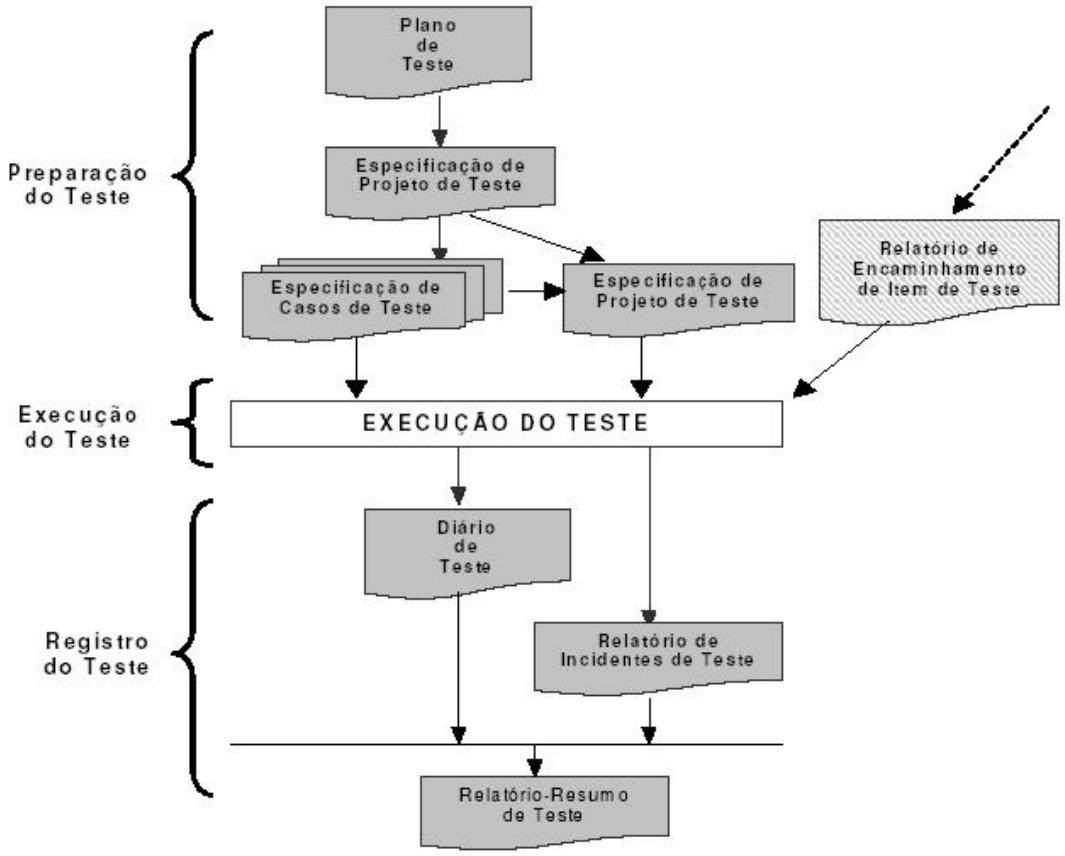

Figura 3.1 - Relacionamentos entre os documentos de teste [CRE04].

A norma separa as atividades de teste em três etapas: preparação do teste, execução do teste e registro do teste. A Figura 3.1 mostra os documentos que são produtos de cada uma das etapas e os relacionamentos entre eles.

4 O Modelo V do Processo de Desenvolvimento de Software

A engenharia de requisitos deve ser entendida dentro de um contexto mais amplo, ou seja, o contexto do desenvolvimento de software. O desenvolvimento de software utiliza modelos para descrever os artefatos de projeto produzidos ao longo do desenvolvimento de software. Estes modelos são importantes e são muitas vezes usados como estandartes de várias campanhas de aperfeiçoamento da engenharia de software.

Dentre os diversos modelos existentes podemos citar os mais conhecidos como: o Modelo Cascata, o Modelo Espiral, o Modelo Baseado em Componentes e o Modelo Concorrente. Todos eles buscam descrever a forma como as etapas de desenvolvimento ocorrem e como interagem.

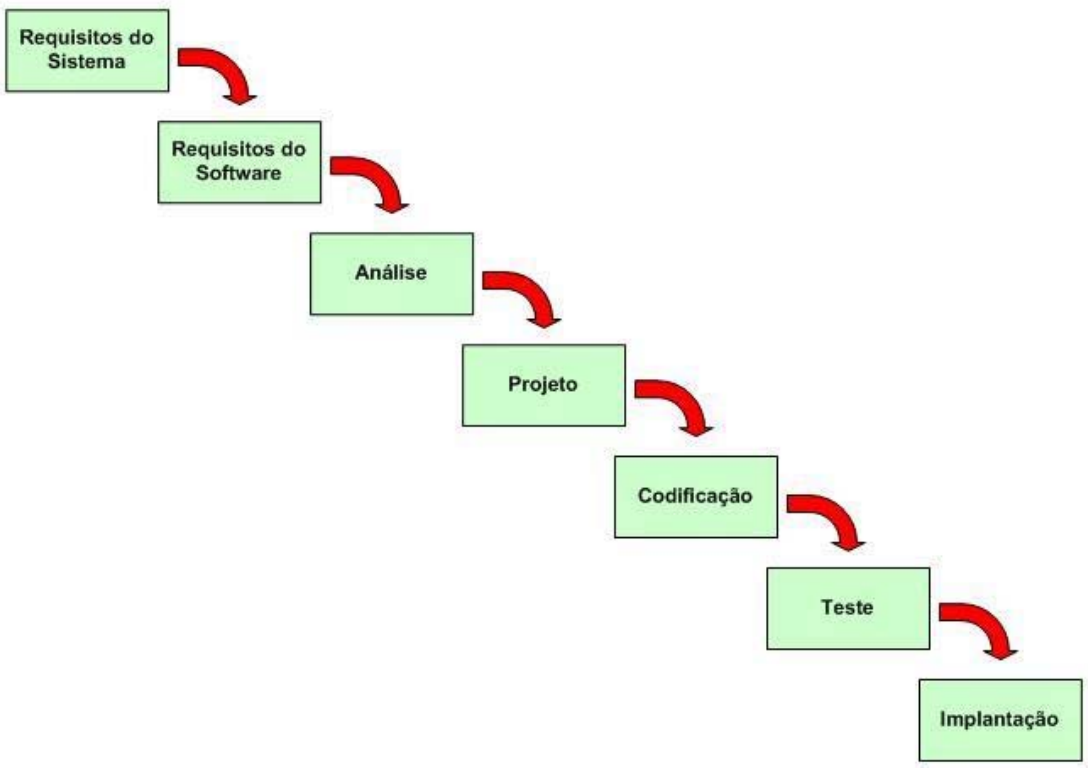

Figura 4.1 – Representação do modelo em Cascata.

Embora existam muitos modelos de representação das etapas de desenvolvimento de software (ciclo de vida de software), poucos representam tão bem estas etapas como o modelo em Cascata. O modelo em Cascata apresenta a seqüência das etapas do ciclo de vida de software de forma mais didática do que os demais. No entanto, ele não representa adequadamente o processo

de interação dessas etapas, conforme preconizado por metodologias de desenvolvimento de software mais recentes [MAR01]. Apesar disto, este modelo é adotado por várias organizações em que a complexidade dos projetos não exige um modelo de desenvolvimento de software mais elaborado.

Existem formas diversas de representação do modelo em cascata; uma dessas formas, o Modelo V, é utilizada neste trabalho.

A engenharia de requisitos ocorre de forma intensiva nas primeiras etapas do ciclo de vida, abrangendo até o Projeto (Figura 4.1)**,** podendo se estender para as demais etapas, dependendo do paradigma de engenharia de software adotado [MAR01].

4.1 O Modelo V

O Modelo V foi criado com o objetivo de melhorar a eficiência e efetividade do desenvolvimento de software. O modelo é uma representação gráfica do ciclo de vida de desenvolvimento de um sistema.

O teste é o foco primário na construção de um sistema e sua realização, segundo o modelo, deriva da maneira como um software é desenhado e construído. O Modelo V descreve os tipos de teste para cada estágio de desenvolvimento de um software. O modelo possui, claramente, uma atividade de teste associada a cada atividade de desenvolvimento, conforme a Figura 4.2 [COL04].

O Modelo V destaca a existência de vários níveis de teste e mostra sua relação com as diferentes fases de desenvolvimento. *Teste de componente* é baseado em código e executado por desenvolvedores para demonstrar que seus pequenos pedaços de funções de códigos executáveis são confiáveis. *Teste de Interface* demonstra que duas ou mais unidades ou suas interações trabalham apropriadamente juntas, e tende a focar em interfaces especificadas no desenho do sistema. Quando todas as unidades e suas várias integrações tiverem sido testadas, o *Teste de Sistema* demonstra que o sistema funciona como se fosse o ambiente de produção para prover as funcionalidades especificadas no desenho do sistema. Finalmente, quando a equipe de desenvolvimento completa estes testes, os requerentes do projeto ou os usuários executam os *Testes de Aceitação* para confirmar que o sistema faz, de fato, o que foi pedido [GOL02].

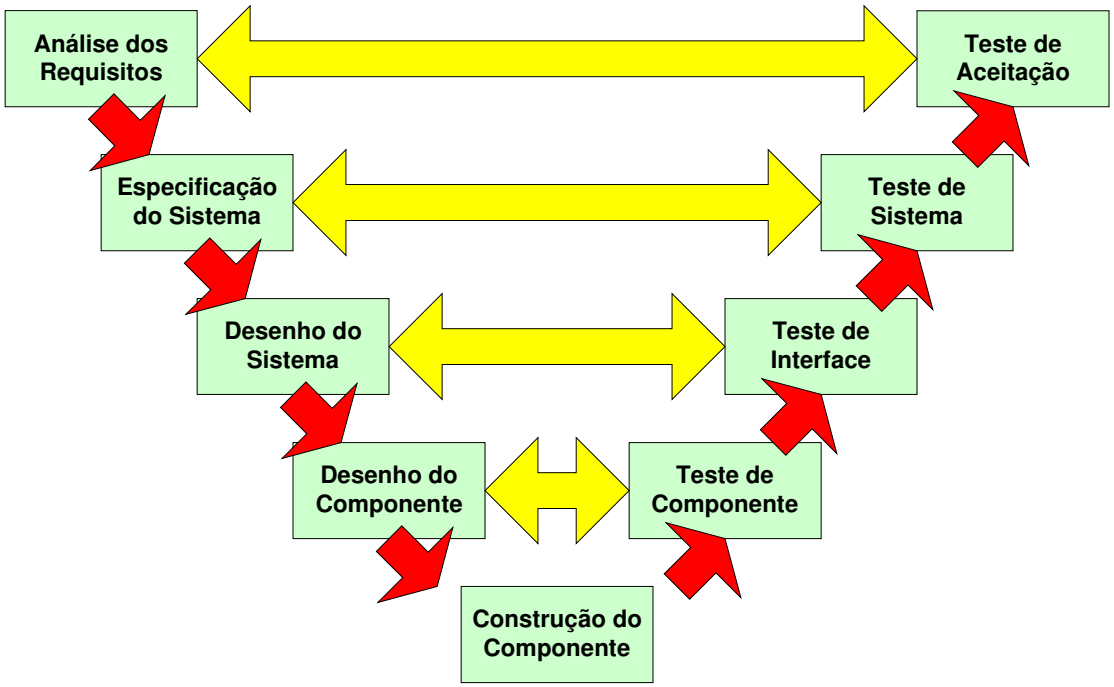

Figura 4.2 – Representação do Modelo V.

4.2 Fases da Construção de um Software segundo o Modelo V

O processo de desenvolvimento de um sistema é tradicionalmente como um modelo em cascata em que cada passo segue o próximo. O modelo tradicional não implica que alguns dos passos em um processo devam ser finalizados antes que o próximo passo comece, ou que o passo anterior não terá que ser revisado mais tarde no desenvolvimento. O modelo V útil para a visualização de como cada passo se relaciona com outros. A seguir, descrevemos cada etapa do modelo.

4.2.1 Caso de Negócio

O passo inicial no desenvolvimento é uma investigação do negócio baseado em um Caso de Negócio produzido por um cliente para um sistema. O caso de negócio esboça um novo sistema, ou altera um sistema existente, que trará benefícios ao negócio, e esboça os custos esperados quando estiver desenvolvendo e rodando o sistema [COL04].

4.2.2 Análise dos Requisitos

O próximo passo é definir um conjunto de requisitos, que é a lista do cliente definindo "o que" o sistema deve realizar para satisfazer suas necessidades. Os requisitos podem ser funcionais ou não-funcionais [COL04].

4.2.3 Especificação do Sistema

Requisitos são então passados para os desenvolvedores, que produzem uma especificação do sistema. A especificação do sistema muda o foco de "o que" o sistema deverá realizar para "como" ele irá realizar por meio de uma definição em termos computacionais, levando em conta requisitos funcionais e não-funcionais [COL04].

4.2.4 Desenho do Sistema

Outros desenvolvedores produzem um desenho do sistema a partir da especificação do sistema. O desenho do sistema contém as funcionalidades requeridas e estas são projetadas em vários componentes. As relações entre esses componentes são também definidas aqui. O documento deve resultar em um detalhado desenho de sistema que deverá realizar o que é requerido pela especificação de sistema [COL04].

4.2.5 Desenho do Componente

Cada componente tem um desenho correspondente que descreve em detalhes exatamente como ele irá executar seu pedaço do processo [COL04].

4.2.6 Construção do Componente

Finalmente, cada componente é construído e está pronto para os processos de teste [COL04].

4.2.7 Teste de Componente

No Modelo V (vide Figura 4.2), o primeiro nível de teste é o teste de componente, também chamado de teste de unidade. Ele envolve assegurar que cada funcionalidade especificada no desenho do componente tenha sido implementada corretamente no componente.

O teste de componente é sempre realizado pelo desenvolvedor, visto que é a única pessoa que entende como o componente trabalha. O problema com um componente é que ele representa somente uma pequena parte da funcionalidade de um sistema e se relaciona com outras partes do sistema, que podem não ter sido construídas ainda. Para superar isto, o desenvolvedor constrói ou usa software especial para simular que o componente está trabalhando como se todo o sistema já estivesse construído [COL04].

4.2.8 Teste de Interface

Como os componentes são construídos e testados separadamente, ao serem acoplados deve-se verificar se interagem corretamente. Dois componentes que tenham passado nos testes de componentes podem não operar corretamente quando conectados. Estes testes podem ser feitos por especialistas ou pelos desenvolvedores.

Teste de interface não é focado em "o que" os componentes fazem mas se eles se comunicam conforme especificado no desenho do sistema. O desenho do sistema define as relações entre componentes e isto envolve algumas questões:

- O que um componente pode esperar de outro componente em termos de serviços?
- Como estes serviços irão responder?
- Como eles serão entregues?
- Como manipular condições não-padrão, ou seja, erros?

Testes são construídos para responder cada uma dessas questões. Eles são organizados para verificar todas as interfaces, até que todos os componentes tenham sido construídos e conectados uns com os outros, produzindo todo o sistema [COL04]. Myers [MYE79] divide o teste de interface em duas fases: teste de integração e teste funcional. Para ele, uma prévia validação deve ser feita utilizando técnicas pertinentes tais como particionamento de equivalência, análise de valores limites e grafo de causa-efeito. O objetivo é ajudar antecipadamente no processo de teste de sistema.

4.2.9 Teste de Sistema

Uma vez que todo o sistema foi construído ele precisa ser testado conforme a "Especificação do Sistema" para verificar se estão sendo entregues as funcionalidades requeridas. Este tipo de teste ainda é focado na construção do sistema, ou seja, no desenvolvimento, embora saibamos que testadores de sistemas são empregados para fazê-lo. Seguem abaixo duas abrangências do teste de sistema:

- Teste de Sistema não é, em sua essência, o exame das partes individuais do projeto, mas sim a avaliação do sistema como um todo.
- Teste de sistema pode envolver vários tipos especializados de teste para verificar se todos os requisitos funcionais e não funcionais estão de acordo com as especificações.

Assim sendo, além dos requisitos funcionais podemos incluir os seguintes tipos de teste para os requisitos não-funcionais:

- Desempenho - Os critérios de desempenho estão satisfeitos?
- Volume - Grandes volumes de informações podem ser manipulados?
- Fadiga - Picos de volume de informações podem ser manipulados?
- Documentação - A documentação é utilizável?
- Robustez - O sistema permanece estável em circunstâncias adversas?

Há muitos outros tipos de testes para os requisitos não-funcionais: usabilidade, segurança, configuração, compatibilidade, estabilidade, confiança, restauração e manutenibilidade. A necessidade de aplicar-se um ou outro são ditadas pelo quanto se espera do sistema [MYE79].

4.2.10 Teste de Aceitação

Teste de aceitação examina o sistema contra os requisitos. É similar ao teste de sistema em que todo o sistema é verificado, mas a diferença importante está na mudança do foco:

- Teste de sistema verifica se o sistema entregue é o que foi especificado;
- Teste de aceitação verifica se o sistema faz o que foi requisitado.

O cliente, e não o desenvolvedor, deve efetuar o teste de aceitação. O cliente sabe o que é solicitado do sistema para alcançar valor para o negócio; somente uma pessoa qualificada pode fazer este julgamento. As condições de teste criadas para os testes de aceitação podem ser usadas

pela equipe de desenvolvimento na fase anterior, ou seja, no teste de sistema. Deste modo, alguns defeitos já podem ser levantados antecipadamente.

4.2.11 Teste de Versão

Mesmo que um sistema esteja implementado de acordo com seus requisitos, há ainda um aspecto a ser examinado que irá beneficiar o negócio. A ligação do *Caso de Negócio* para o teste de versão é mais "livre" que os outros testes, mas é ainda importante.

Teste de versão diz respeito à verificação se o novo sistema ou o sistema alterado funciona no ambiente de negócio da empresa, principalmente o ambiente técnico. Ele aborda questões como:

- Afetará algum sistema já em funcionamento?
- É compatível com outros sistemas?
- Tem desempenho aceitável sob carga pesada de informações?

Estes testes são sempre executados nas empresas pelo time de operadores de sistema. Pode parecer óbvio que o time de operadores deveria ser envolvido desde o início do projeto para dar sua opinião do impacto que um novo sistema pode ter. Eles deveriam então ter certeza que o "Caso de Negócio" é relativamente bom no mínimo no que diz respeito a custos financeiros e custos operacionais. Porém, na prática, vários times de operadores se envolvem em um projeto somente semanas antes da implantação o que pode resultar em grandes problemas [COL04].

5 O Processo de Engenharia de Requisitos Proposto

O objetivo é propor um processo de validação de requisitos dentro do Modelo V, o qual reduza os defeitos nos requisitos de software. Para isto são descritas as técnicas utilizadas e as particularidades do processo proposto.

5.1 Introdução

Algumas características devem estar presentes em uma especificação de requisitos de software, conforme especificado pela norma IEEE 830 [IEEE830-98]: correção, não ambigüidade, completude, consistência, classificação quanto à importância e/ou estabilidade, além de ser verificável, modificável e rastreável.

Kotonya & Summerville [KOT98] destacam que linguagem natural é a única notação existente passível de compreensão por todos os potenciais leitores dos documentos de requisitos. Reconhecem, contudo, que requisitos expressos dessa forma são potencialmente difíceis de entender, podendo ser ambíguos, surpreendentemente obscuros e mal interpretados. Davis [DAV93] destaca que uma das razões para a alta ambigüidade contida em especificações expressas em linguagem natural é o fato de que, ao nos expressarmos verbalmente, a entonação, os movimentos das mãos e a linguagem corporal ajudam a clarear a idéia que se pretende transmitir. Na linguagem escrita, contudo, essa parte da expressão verbal não existe, contribuindo para sua ambigüidade.

Uma das formas de minimizar o não entendimento completo da documentação do requisito é envolver as pessoas certas na construção e elicitação dos requisitos. Assim há a oportunidade de esclarecer a idéia que se pretende transmitir no documento.

No tocante aos fatores relacionados à competência das pessoas envolvidas no processo de levantamento de requisitos, todos os integrantes do time possuem diferentes perfis que agregam conhecimento técnico, esforço, opiniões, que juntos convergem na criação de um documento final.

Levando em consideração que a fase de levantamento de requisitos é a base de todas as atividades da engenharia de software subseqüentes, um documento com muitas falhas e equívocos leva a um esforço indevido e muitas vezes à construção de softwares que não atendem

as necessidades do usuário ou a requisitos não testáveis. Decisões tomadas na fase de desenho do sistema irão afetar a testabilidade de um requisito. A importância da fase de requisitos no ciclo de vida da construção de um software freqüentemente não é levado em consideração.

Na tentativa de diminuir os problemas causados pela pouca relevância dada à fase inicial do projeto, algumas iniciativas foram implantadas no processo proposto para melhorar a qualidade do software. Uma dessas iniciativas envolve a atuação da equipe de teste de software.

Um dos papéis de um testador no ciclo de construção de um software é colaborar com os usuários a fim de projetar e executar os testes funcionais. Embora seja uma tarefa essencial, este papel é realizado somente no final da fase de desenvolvimento do software (ou seja, o testador espera que uma interface com o usuário esteja disponível para iniciar o teste funcional).

Chris DeNardis [DEN00] relata sua experiência:

> "... a equipe de teste começou a interagir com a equipe de desenvolvimento com o objetivo de melhorar a comunicação e a troca de conhecimentos técnicos entre os dois times. Os conhecimentos de teste são passados à equipe de desenvolvimento para que ela consiga realizar melhor seus testes (teste unitário e teste de sistema) e a equipe de teste saberá com mais antecedência quais as alterações o sistema sofrerá, ou terá mais contato com os requisitos do software".

Este tipo de interação tem sido adotado por muitas equipes de desenvolvimento e teste, porém percebemos que uma importante interação ainda não tem sido adotada nos projetos de construção de software: o envolvimento da equipe de teste no levantamento de requisitos.

Percebemos que nos processos mais comuns de construção de software, como o Modelo V, a equipe de teste é envolvida tardiamente. Em alguns casos, como o do artigo anteriormente citado, a equipe de teste é envolvida no início dos testes da equipe de desenvolvimento para melhora da qualidade dos testes após a codificação.

Ellen Gottesdiener [GOT02] diz que o envolvimento do time de teste e de especialistas em qualidade na elaboração dos requisitos produz requisitos de melhor qualidade. E este ponto é reforçado por Cynthia Cohen [COH04] que afirma que o engajamento consistente e antecipado da equipe de teste na fase de elicitação dos requisitos pode prover oportunidades para o entendimento comum das necessidades por parte das equipes.

Ramachandran [RAM96] também defende que uma forma de melhoria da qualidade dos requisitos seria identificar claramente, desde o início do processo de desenvolvimento, as interações da equipe de teste com a equipe de desenvolvimento; as atividades de teste começam com a validação e verificação dos requisitos e a criação de condições de teste. Para Ramachandran cada fase de desenvolvimento deve possuir em paralelo um processo de teste associado.

Figura 5.1 - Adaptação do Modelo V do processo de desenvolvimento de software.

5.2 O Modelo V do Processo de Desenvolvimento de Software Proposto

O processo procura melhorar a qualidade do processo de desenvolvimento dentro do contexto estudado. Para isto ele enfoca três pilares importantes: um modelo de representação das etapas de desenvolvimento de software, os documentos gerados pelas atividades de teste e as técnicas de auxílio à validação dos requisitos de software.

O Modelo V dá ao processo de desenvolvimento de software uma atenção maior ao teste de software. Apesar de o Modelo V enfatizar a atividade de teste em diversos níveis, a grande maioria das equipes de desenvolvimento e teste não utiliza todo o potencial do modelo conforme afirmam Goldsmith e Graham [GOL02-1].

O modelo proposto não é totalmente novo em relação ao Modelo V mas ele enfatiza a necessidade das atividades de validação desde o início do projeto. A Figura 5.1 mostra através da elipse azul a atividade de Revisão da Qualidade presente nas fases de Análise de Requisitos e Teste de Aceitação dentro do Modelo Proposto. A atividade de Revisão da Qualidade inclui atividades como teste de requisitos na fase de Análise de Requisitos e automação de testes na fase de Teste de Aceitação.

5.2.1 Validação dos requisitos do sistema

Esta fase é iniciada pela equipe de desenvolvimento juntamente com os usuários. Nela, os trabalhos de elicitação, análise e especificação dos requisitos são direcionadas conforme as técnicas escolhidas para cada uma das fases.

Ao final da etapa de especificação dos requisitos, o documento dos requisitos é gerado. A partir daí, a equipe de teste é envolvida para validação dos requisitos já documentados, conforme mostra a Figura 5.2.

Com o documento dos requisitos em mãos, a equipe de teste inicia as atividades de criação dos casos de teste (processo 1 na Figura 5.2) . Em paralelo à criação das condições de teste a prototipação é também iniciada (processo 3 na Figura 5.2). Ao término do processo de criação dos casos de teste o relatório de quantidade de casos de teste por requisito é gerado e analisado (processo 2 na Figura 5.2). A análise deste relatório nos dirá também se existe algum requisito que deva também utilizar a prototipação para sua validação (arco conectando o processo 2 ao processo 3 na Figura 5.2). As linhas pontilhadas significam que o processo de prototipação será utilizado no processo proposto quando as funcionalidades exigirem tal tipo de validação. Caso contrário, utilizamos somente a técnica de teste de requisitos para validação (processo 1 na Figura 5.2).

Terminados os três processos, é gerada uma lista com os problemas encontrados e ações acordadas. A lista de problemas deve relatar todos os problemas identificados no documento de especificação (processo 4 na Figura 5.2), classificando-os de alguma forma, como por exemplo:

ambigüidade, inconsistência, etc. As ações acordadas são ações a serem executadas a partir dos problemas relacionados na lista de problemas [MAR01].

A seguir detalharemos cada processo dentro da etapa de validação dos requisitos: Criação dos casos de teste e Teste dos requisitos, Análise da Quantidade de Requisitos por Casos de Teste, Prototipação e Alteração dos Requisitos.

5.2.1.1 Criação dos Casos de teste e Teste dos Requisitos

É preciso ter em mente que o objetivo do uso de casos de teste (processo 1 na Figura 5.2) nesta fase é validar o requisito e não o sistema. Todos os participantes desta atividade, ou seja, usuários, equipes de desenvolvimento e de teste, devem ter ciência disto.

Outro ponto muito importante nesta atividade é a participação ativa das pessoas chaves. Elas devem ser usuários experientes que consigam abstrair as novas funcionalidades visualizando o procedimento correto para testar o requisito.

Esta técnica foi escolhida por possuir algumas vantagens. São elas: iniciação do processo de criação de casos de teste na fase inicial do projeto, reuso dos casos de teste em fases posteriores e interação da equipe de teste com a equipe de desenvolvimento e do projeto desde o início.

Neste processo é muito importante a documentação dos testes para que os resultados possam ser analisados posteriormente e que os casos de testes dos requisitos possam ser aproveitados nas próximas fases como fontes de conhecimento futuro. Utilizamos dois documentos para armazenar os casos de teste apoiados na norma IEEE 829 [IEEE829-98]: Especificação de casos de teste e Especificação de roteiro de teste. As informações armazenadas nestes documentos são mostradas na Figura 5.3 e Figura 5.4.

Esses documentos nos possibilitam armazenar os casos de teste e testá-los, de forma a manter um histórico de todo o processo de teste. Eles servirão posteriormente como repositório de dados do projeto e devem também ser usados, quando possível, nos testes de aceitação para garantir que o que foi acordado no início é realmente o que está sendo entregue [RAM96].

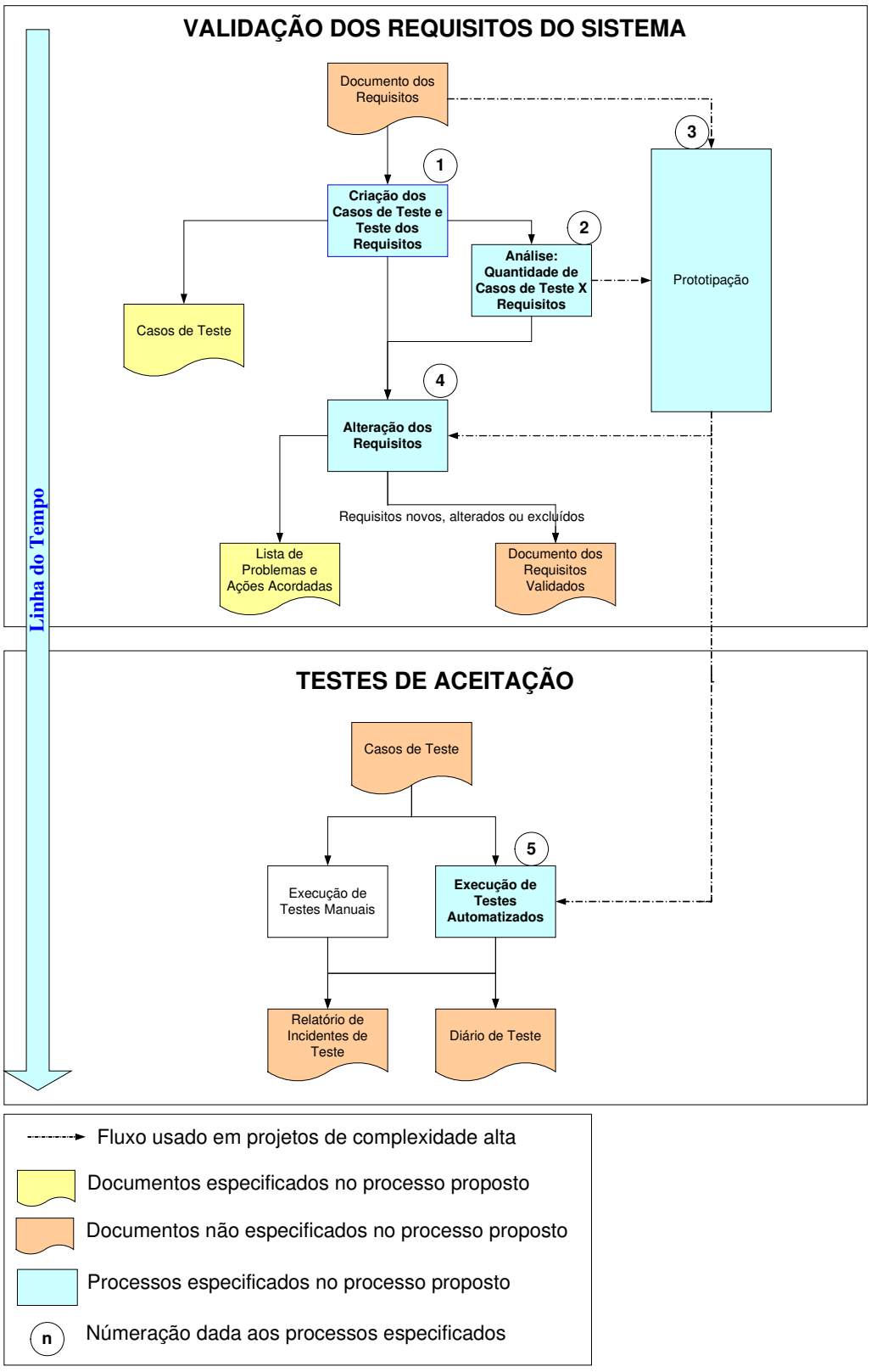

Figura 5.2 - Processo proposto.

Especificação de Casos de Teste	
Número do Requisito	Número seqüencial.
Descrição do Requisito	Breve descrição de forma a identificar o mesmo.
Número da Condição de teste	Número seqüencial seguindo o seguinte formato: requerimento.condição. Ex.: 1.1 = Condição de Teste 1 do Requisito 1.
Condição de Teste	Texto explicativo da condição de teste
Número do Roteiro de Teste	Número seqüencial que referencia o roteiro de teste.

Figura 5.3 - Especificação de Casos de Teste usado na criação e execução dos casos de teste.

Especificação de Roteiro de Teste	
Número do Roteiro	Número seqüencial
Descrição do Roteiro	Breve descrição do roteiro de teste
Escrito por	Nome de quem criou o roteiro de teste
Data	Data da criação do roteiro
Número da Condição de teste	Número da condição de teste que o roteiro irá exercitar. Um roteiro de teste pode exercitar mais de uma condição de teste.
Condição de Teste	Texto explicativo da condição de teste. Mesmo texto contido no Relatório de Casos de teste.
Pré Condições	Condições necessárias para a execução do teste.
Passos	Número seqüencial. Um roteiro de teste pode exigir mais de um passo para sua execução.
Ação	Descrição das ações que devem ocorrer para execução do passo.
Resultado Esperado	O que se espera da ação executada.
Resultado Atual - (P)assou ou (F)alhou	Status do resultado esperado.
Resolução/Comentário	Caso tenha ocorrido alguma falha, explicar o ocorrido. Ou algum comentário importante.

Figura 5.4 - Especificação de Roteiro de Teste usado na criação e execução dos casos de teste.

5.2.1.2 Análise: Quantidade de Requisitos X Casos de Teste

Dificuldades no processo de geração de casos de teste significam que ou o requisito não está claro ou não é testável, o que implica em revisão da definição do requisito.

A análise das ligações de requisitos a casos de teste identifica requisitos não testáveis e/ou requisitos que, por terem muitas ligações, dificultam o processo de verificação da correção. A Figura 5.5 mostra que a simples análise visual dessas ligações aponta requisitos que se enquadram no caso de requisitos não testáveis [SAY03].

Figura 5.5 - Ligações de requisitos a casos de teste [SAY03].

A contagem dos casos de teste por requisitos é uma métrica simples de obter [SAY03]. Os valores obtidos deverão gerar um gráfico, como o mostrado na Figura 5.6. A análise do gráfico resultante é um bom parâmetro para apontar requisitos cuja verificação é complexa, pois exigem a execução de um grande número de casos de teste para concluir sobre sua correção [SAY03]. Estes dados também definem quais requisitos devem ser adicionalmente validados pelo processo de prototipação (processo 3 na Figura 5.2). Ou seja, quanto mais complexo é o requisito, maior a necessidade de validá-lo com qualidade.

A título de exemplo, observamos na Figura 5.6 que para o requisito de número 4 foi criado apenas um caso de teste e que os requisitos de número 3,8 e 11 possuem 6 casos de teste cada um. Já para um único requisito, o requisito de número 9, não foi criado nenhum caso de teste.

A quantidade de casos de teste por requisitos é extraída da Especificação de Casos de Teste descrito na Figura 5.3.

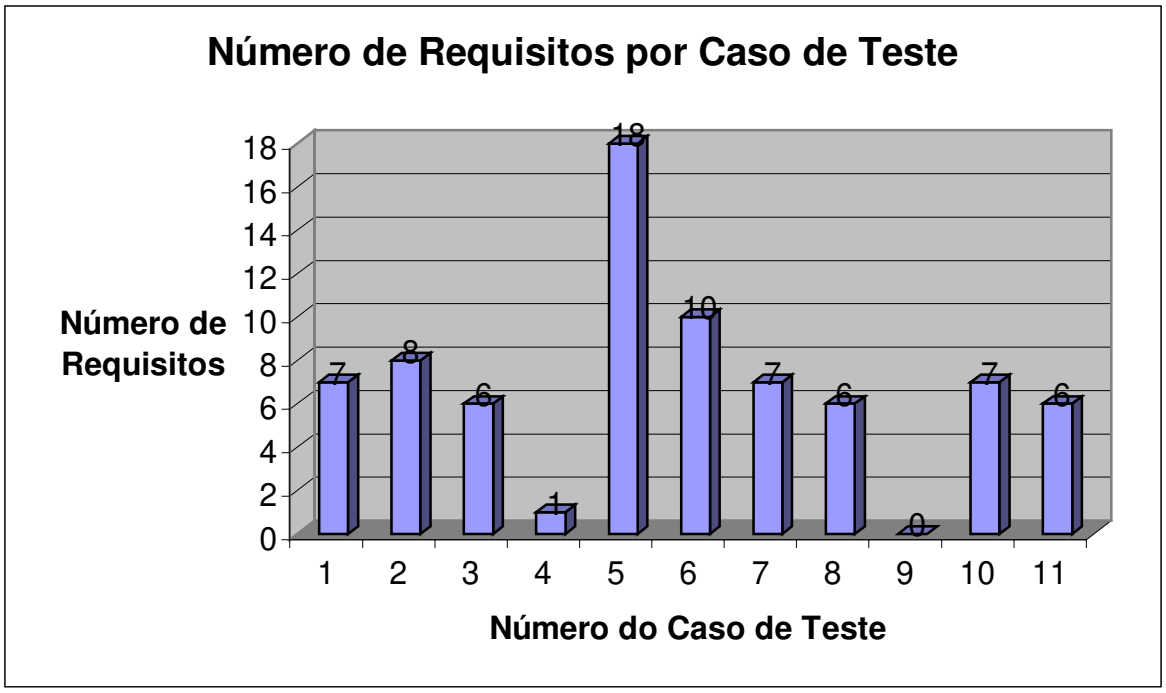

Figura 5.6 - Exemplo de possível distribuição de requisitos por caso de teste.

5.2.1.3 Prototipação

Grande parte dos protótipos usados para validação dos requisitos foi utilizada também no processo de elicitação. Segundo Lopes [LOP02], construir um protótipo apenas para validação dos requisitos não é economicamente atraente. Porém, caso o processo de análise da quantidade de casos de teste por requisitos (processo 2 na Figura 5.2) encontre algum requisito com quantidade alta de casos de teste, recomenda-se o uso da prototipação para sua validação pois é um bom sinal de que o requisito possui uma verificação complexa ou que há necessidade de ampla cobertura de testes.

Outros motivos reforçam o uso da prototipação como ferramenta de validação de requisitos: aquisição de experiência para uso em etapas posteriores do projeto, utilização do protótipo na fase de teste para aumento da cobertura dos testes e interação da equipe de teste com a equipe de desenvolvimento desde o início do projeto para permitir a troca de conhecimentos.

5.2.1.4 Alteração dos Requisitos

Os processos descritos nas Seções 5.2.2.1, 5.2.2.2 e 5.2.2.3 geram um relatório: Lista de problemas e Ações acordadas. Este relatório contém as alterações que devem ser feitas no documento dos Requisitos e ajudarão no processo de rastreamento dos requisitos.

5.2.2 Teste de Aceitação

Conforme já foi descrito em seções anteriores, o processo proposto se apóia no Modelo V. Neste modelo, as últimas fases do processo de construção do software são validações e verificações entre o que foi acordado em fases anteriores com o que está sendo entregue. A fase de Teste de Aceitação verifica se o que foi descrito na fase de Análise de Requisitos está sendo entregue. No processo proposto, para os projetos em que a prototipação foi utilizada no processo de validação dos requisitos, utilizamos o protótipo nos testes de aceitação.

Como os protótipos são construídos, em grande parte, para validar os requisitos mais complexos, o refinamento da funcionalidade que está sendo validada é alto. Isto permite aos usuários visualizar o resultado esperado com um grau de assertividade igualmente alto. Sendo assim, o protótipo se torna uma ferramenta confiável aos olhos do usuário. Isto porque a construção do mesmo é encerrada, para a funcionalidade que se pretende validar, somente quando o resultado esperado satisfaz as condições desejadas pelo usuário. Em vista disto, o protótipo é uma ferramenta que exercita várias condições do sistema chegando a um resultado esperado.

Assim, a utilização do protótipo nos testes de aceitação possibilita a execução, em menor tempo e com boa precisão, de condições de teste que na maioria das vezes são consideradas muito complexas quando executadas manualmente (cálculos financeiros são bons exemplos). Desta forma, o uso do protótipo como ferramenta de execução dos casos de teste dá à equipe de teste a possibilidade de execução de maior número de casos de teste; o tempo que seria gasto para executar um teste complexo poderia ser usado para o teste de outras funcionalidades que seriam testadas inadequadamente. Os documentos usados na execução dos testes de aceitação são os descritos nas Figuras 5.3 e 5.4.

6 O Estudo Experimental

Este capítulo descreve o contexto e o resultado do estudo experimental [TRA02] realizado para avaliar o processo proposto. Aplicando o método GQM (Goal/Question/Metric) [BAS02] foram construídas as hipóteses a serem verificadas experimentalmente, baseadas nas características apresentadas pelo processo proposto.

O GQM é um método bastante difundido para guiar a escolha das medidas mais adequadas para um processo de mensuração. Sua principal característica é a utilização de uma abordagem top-down para a definição das medidas. O GQM prega que o processo de mensuração não deve ser guiado pelas medidas em si, mas pelos objetivos que se pretende atingir com sua coleta. Neste sentido, as medições só devem ser realizadas se estiverem fundamentadas por uma meta claramente definida. Assim, o ponto de partida para a escolha das medidas é a definição das metas que irão guiar o restante do processo. O passo seguinte à definição das metas é a geração de perguntas que traduzam as metas em aspectos quantitativos e que possam ser alvos de medição. E, em seguida, especificam-se as medidas (métricas) que precisam ser coletadas de forma a obter as especificações necessárias para responder às perguntas geradas.

Os projetos que representam a amostragem do estudo adotam o Modelo V como modelo de desenvolvimento de software, cuja principal característica são fases com delimitações claras (início da fase após o término da fase anterior). A cada término de fase, um documento é criado e aprovado por todos os participantes do projeto.

Os participantes dos projetos se dividem em quatro grupos: usuário final, gerência de projeto, equipe de desenvolvimento de software e equipe de teste. A Tabela 6.1 mostra a participação nas fases do projeto.

O estudo compara dois processos dentro de um dado contexto: o processo antigo apresentado no Capítulo 4 e o processo proposto apresentado no Capítulo 5. O processo proposto distingue-se do processo antigo em dois aspectos: a validação dos requisitos pela equipe de teste pela execução de testes e pela prototipação, e o uso de protótipos na execução de casos de teste durante a fase de teste de aceitação. O contexto do estudo é uma empresa do ramo de cartões de crédito.

Tabela 6.1 - Participantes das fases do Projeto onde P=participante e N=não-participante.

Participantes				
Fases do Projeto	Usuário Final	Gerência de Projeto	Equipe de Teste	Equipe de desenvolvimento de software
Especificação dos Requisitos	P	P	P	N
Especificação do Sistema	P	P	N	P
Desenho do Sistema	P	P	N	P
Desenho do Componente	N	P	N	P
Construção do Componente	N	P	N	P
Teste de Componente	N	P	N	P
Teste de Interface	N	P	N	P
Teste de Sistema	P	P	P	P
Teste de Aceitação	P	P	P	P

6.1 Definição dos Objetivos

6.1.1 Objetivo do Estudo

Analisar o processo proposto

Com o propósito de avaliá-lo

Com respeito à eficácia na revisão/inspeção da Especificação de Requisitos e criação dos casos de teste

Do ponto de vista da equipe de projeto e de teste de software

No contexto do ambiente de desenvolvimento de software no negócio de cartões de crédito.

Requisitos são artefatos produzidos na fase de análise dos requisitos. A quantidade de requisitos é uma forma de medição que reflete o tamanho de um projeto [LIN04]. A necessidade de medição nos leva ao uso da quantidade de requisitos para o melhor emparelhamento dos projetos no estudo experimental proposto. A alteração ou criação de requisitos é conseqüência da execução da atividade de validação sendo uma forma de medição da eficácia deste processo.

O processo proposto introduz atividades de criação de casos de teste nas fases iniciais do processo. Estando inseridos no ambiente de desenvolvimento de software no negócio de cartões de crédito, os projetos possuem grande parte de seus requisitos ligados a cálculos financeiros. E os cálculos ao qual os projetos possuem ou interagem, requerem diversas combinações para serem elegíveis a esta ou aquela condição. Assim, a quantidade de casos de teste é considerada no estudo um importante medidor da eficácia do processo proposto.

Em virtude do objetivo do estudo, abranger as etapas de especificação de requisitos e testes de aceitação do processo de desenvolvimento de software, consideraremos a avaliação do modelo proposto em cada uma das etapas. Assim duas metas distintas, com suas respectivas questões e métricas associadas, são criadas.

6.1.1.1 Objetivo 1

Analisar o processo proposto

Com o propósito de avaliá-lo

Com respeito à eficácia na revisão/inspeção da Especificação de Requisitos

Do ponto de vista da equipe do projeto de software

No contexto do ambiente de desenvolvimento de software no negócio de cartões de crédito.

Questões e Métricas Associadas:

Q 1.1: O número de requisitos alterados após a validação dos mesmos no processo proposto é maior que o resultado apresentado no processo antigo?

Métrica: O número de requisitos alterados após a validação dos mesmos nos dois processos.

Q 1.2: O número de requisitos novos após a validação dos mesmos no processo proposto é maior que o resultado apresentado no processo antigo?

Métrica: O número de requisitos novos após a validação dos mesmos nos dois processos.

Q 1.3: O número de defeitos encontrados nos Testes de Aceitação no processo proposto é menor que o resultado apresentado no processo antigo?

Métrica: O número de defeitos encontrados nos Testes de Aceitação nos dois processos.

6.1.1.2 Objetivo 2

Analisar o processo proposto

Com o propósito de avaliá-lo

Com respeito à eficácia na criação de casos de teste

Do ponto de vista da equipe de teste

No contexto do ambiente de desenvolvimento de software no negócio de cartões de crédito.

Questões e Métricas Associadas:

Q 2.1: O número de casos de teste criado no novo processo proposto é maior que o resultado apresentado no processo antigo?

Métrica: A quantidade de casos de teste para os dois processos.

6.2 Planejamento

6.2.1 Definição das Hipóteses

Para teste das hipóteses aplicaremos o Teste T para dados emparelhados, analisando as diferenças entre o processo proposto e o processo antigo. Neste contexto, utilizaremos as seguintes definições:

- **Hipótese 1**

  – Hipótese nula (μ_d) = 0

  – Hipótese alternativa (μ_d) > 0

Onde , sendo:

– número de requisitos novos utilizando o processo proposto

– número de requisitos novos utilizando o processo antigo

- **Hipótese 2**

  – Hipótese nula (μ_d) = 0

  – Hipótese alternativa (μ_d) > 0

  Onde , sendo:

  – número de requisitos alterados utilizando o processo proposto

  – número de requisitos alterados utilizando o processo antigo

- **Hipótese 3**

  – Hipótese nula (μ_d) = 0

  – Hipótese alternativa (μ_d) < 0

  Onde , sendo:

  – número de erros encontrados utilizando o processo proposto

  – número de erros encontrados utilizando o processo antigo

- **Hipótese 4**

  – Hipótese nula (μ_d) = 0

  – Hipótese alternativa (μ_d) > 0

  Onde , sendo:

  – número de casos de teste criados utilizando o processo proposto

  – número de casos de teste criados utilizando o processo antigo

6.2.2 Descrição da Instrumentação

Instrumentação para o Objetivo 1

A instrumentação para captura dos dados para o estudo das questões Q 1.1 e Q1.2 é mostrado na Figura 6.3. O Documento dos Requisitos (DR) é finalizado após o processo de Especificação dos Requisitos. Em seguida a validação dos requisitos é feita (processo proposto ou processo antigo) e, ao final, o Documento dos Requisitos Validados (DRV) é gerado. O modelo utilizado para o Documento dos Requisitos e para o Documento dos Requisitos Validados está representado na Figura 6.1.

DR é comparado com DRV para extrair o número de alterações. Pelo fato de ambos os documentos serem representados na forma de texto, qualquer inclusão, alteração ou exclusão de sentenças ou palavras que altere o sentido da informação é considerada mudança no requisito. Para exemplificar, suponha o seguinte requisito de uma aplicação hipotética: "O sistema deve gerar um gráfico de barras, totalizando os projetos concluídos *por mês*". Um exemplo de requisito alterado pode ser "O sistema deve gerar um gráfico de barras, totalizando os projetos concluídos *por trimestre*". Deste mesmo requisito, caso seja necessário após a geração de relatório, enviar as informações para outro sistema, precisaremos criar um novo requisito pois se trata de uma nova funcionalidade ("O sistema deve enviar as informações do relatório X para o sistema Y"). Portanto, para cada diferença encontrada nos requisitos temos dois tipos de modificações: inclusão de requisitos novos ou alteração dos requisitos. A quantidade de modificações é o objeto de análise.

No processo de instrumentação da Questão Q 1.3, o instrumento de análise é a quantidade de defeitos contida no Relatório de Incidente de Teste (RIT) (Figura 6.2). O RIT é concluído ao final da fase de Teste de Aceitação. A Figura 6.4 ilustra o processo.

Requisito # (Número do Requisito)	
Tipo	Tipo do Requisito: Funcional ou Não-Funcional. Caso seja não-funcional, classificá-lo.
Nome do Requisito	Nome que consiga identificar o requisito.
Definição do Requisito	Descrição do processo que se pretende criar, alterar ou excluir.

Justificativa	Motivo pelo qual o requisito foi criado e benefícios do mesmo.
Fonte do Requisito	Nome do requerente do requisito e respectiva área/departamento.
Satisfação do Usuário	Grau de satisfação do usuário se o requisito for implementado com sucesso. Escala: de 1 (desinteressado) a 5 (extremamente satisfeito).
Insatisfação do Usuário	Grau de insatisfação do usuário caso o requisito não seja entregue. Escala: de 1 (pouco importa) a 5 (extremamente insatisfeito).
Restrições	Limites/condições contidas no requisito.
Dependência	Lista de requisitos que tenham alguma dependência com este.
Conflitos	Requisitos que não poderão ser implementados caso este seja.
Material de Apoio	Nome de documentos que ilustram e ajudam na explicação deste requisito.

Figura 6.1 - Documento do Requisito utilizado na documentação dos requisitos do projeto.

Incidente # (Número do Incidente)	
Tipo	Tipo do defeito reportado (Resultado Esperado diferente do Atual, Mensagem de Erro inesperada, Ambiente Não Disponível, Lentidão no Processamento).
Resumo do Erro	Breve descrição do defeito.
Detalhamento do Erro	Descrição detalhada do defeito.
Documentos de Referência do Erro	Apontamento para documento de auxílio da explanação do erro.
Número do Caso de Teste	Caso de teste que identificou o defeito.
Status	Status do erro (Aberto, Fechado, Aguardando Reteste, Fechado com Condições).
Prioridade	Prioridade de resolução do erro (Alta, Média e Baixa). Uma prioridade deve ser alta caso o erro impossibilite o andamento dos testes.
Data da Submissão	Data da submissão do defeito.
Submetido por	Nome de quem submeteu o defeito.
Data estimada da Resolução	Data prevista para resolução do defeito. Esta data é estipulada baseada na prioridade do erro.
Área requerente	Time/Área que reportou o erro.

Área direcionada	Time/Área responsável pela resolução do erro.
Detalhes da Solução do Erro	Explicações para solução do erro.
Documentos de referência da Solução do Erro	Apontamento para documento de auxílio da explanação da solução do erro.
Detalhes do Reteste	Descrição detalhada do Reteste.
Documentos de referência do Reteste	Apontamento para documento de auxílio da explanação do reteste.

Figura 6.2 - Relatório do Incidente de Teste utilizado na documentação dos erros.

Figura 6.3 - Instrumentação na fase de Especificação de Requisitos (Q 1.1 e Q 1.2).

Figura 6.4 - Instrumentação na fase de Teste de Aceitação (Q 1.3).

Instrumentação do Objetivo 2

A quantidade dos casos de teste é extraída do documento de Casos de Teste ("seção 5.2.1.1" do Capítulo 5 O Processo de Engenharia de Requisitos Proposto), conforme mostra a Figura 6.5. Este documento é analisado antes do início dos testes de aceitação; a partir dele é criado o relatório para posterior análise.

6.2.3 Seleção do Contexto

O contexto pode ser caracterizado conforme quatro dimensões: o processo (on-line ou off-line); os participantes; a realidade (o problema real ou modelado); e a generalidade (específico ou geral).

Este estudo pressupõe o processo off-line, pois os dados são coletados após a finalização das etapas envolvidas. Os participantes deste estudo são profissionais da área de gestão de projetos e engenharia de software no negócio de cartões de crédito. Foram selecionados projetos de construção de software no negócio de cartões de crédito, portanto aplicados a problemas reais.

O novo processo apresentado é comparado ao processo já existente em projetos de construção de software no negócio de cartões de crédito; assim, o contexto possui um caráter específico.

Figura 6.5 - Instrumentação na fase de Teste de Aceitação (Q 2.1).

6.2.4 Seleção dos indivíduos

Como participantes para o estudo propôs-se utilizar os projetos de construção de software de uma dada empresa no negócio de cartões de crédito. Todos os projetos utilizam o Modelo V como modelo de desenvolvimento de software. Em parte dos projetos utilizou-se o processo proposto (apresentado no Capítulo 5) de desenvolvimento de software e na outra parte utilizou-se o processo antigo (apresentado no Capítulo 4).

A Tabela 6.2 apresenta uma breve descrição dos projetos selecionados. Os projetos foram escolhidos conforme a quantidade de requisitos dos projetos contidos no Documento dos Requisitos (Figura 6.1) do projeto antes do processo de validação dos requisitos. Foram escolhidos projetos com um número de requisitos igual ou maior que quatro e igual ou menor que onze. Estes projetos são considerados similares, pois se assemelham em tamanho (custo de até R$ 50.000,00), criticidade (prejuízos moderados e perdas recuperáveis em termos de conseqüências

de uma possível falha do sistema), tamanho da equipe (a mesma equipe para todos os projetos escolhidos) e padrões adotados. Quatro projetos utilizaram o processo proposto; dois destes projetos usaram o protótipo nos testes de aceitação. Os outros quatro projetos adotaram o processo antigo.

Como pretendemos verificar se o processo proposto é melhor que o processo antigo, utilizamos o Teste T para dados emparelhados. Nestes testes, as amostras devem ser comparadas em pares. O critério utilizado para criação dos pares foi a quantidade de requisitos iniciais do projeto. Assim um projeto que tenha onze requisitos iniciais compõe um par com outro projeto que possua dez requisitos e assim por diante (Tabela 6.3). Lembrando que requisitos iniciais se referem aos requisitos pertencentes ao Documento dos Requisitos antes do processo de validação dos requisitos. Cada par é constituído por um projeto que tenha utilizado o processo proposto e por outro projeto que tenha utilizado o processo antigo.

Tabela 6.2 - Descrição dos projetos.

#	Projeto	Decrição	Quantidade de Requisitos - Antes da Validação dos Requisitos	Processo
1	CEFA	Cálculo de Juros com taxa máxima para clientes em atraso.	8	Proposto com Protótipo no Teste de Aceitação
2	Parcelamento Globestar	Inclusão do parcelamento da fatura do cliente no sistema Globestar	11	Proposto com Protótipo no Teste de Aceitação
3	CIB Versão 2	Segunda versão das funcionalidades para migração dos cartões para plataforma bancária.	6	Proposto sem Protótipo no Teste de Aceitação
4	CIB Versão 3	Terceira versão das funcionalidades para migração dos cartões para plataforma bancária.	7	Proposto sem Protótipo no Teste de Aceitação
5	CIB Versão 1	Primeira versão das funcionalidades para migração dos cartões para plataforma bancária.	6	Antigo
6	Registro Automático de Parcelamento	Identificação automática de parcelamento de fatura.	9	Antigo
7	Cadastro Correios	Cadastro de endereço com validação em base fornecida pelos Correios.	5	Antigo
8	24 Parcelas	Aumento da quantidade máxima de parcelas no processo de parcelamento da fatura.	10	Antigo

6.2.5 Variáveis

Variáveis independentes: o número de novos requisitos, o número de requisitos alterados, o número de defeitos relatados durante os testes de aceitação e o número de casos de teste.

Variáveis dependentes: diferença entre o número de novos requisitos gerados pelo processo proposto e antigo, diferença entre o número de requisitos alterados gerados pelo processo proposto e pelo processo antigo, diferença entre o número de erros relatados durante os testes de aceitação no processo proposto e no processo antigo, e diferença entre o número de casos de teste criados pelo processo proposto e pelo processo antigo.

Tabela 6.3 - Pares criados para aplicação do Teste T para dados emparelhados.

EMPARELHAMENTO				
PAR #	**Elemento 1**		**Elemento 2**	
	Número de Requisitos do Projeto que utilizou o processo proposto	**Nome do Projeto**	**Número de Requisitos do Projeto que utilizou o processo antigo**	**Nome do Projeto**
1	8	CEFA	9	Registro Automático de Parcelamento
2	11	Parcelamento Globestar	10	24 Parcelas
3	6	CIB Versão 2	5	Cadastro Correios
4	7	CIB Versão 3	6	CIB Versão 1

6.2.6 Validade

Validade interna: são utilizados projetos de construção de software no negócio de cartões de crédito; portanto, pressupõe-se que são representativos para o contexto proposto.

Validade de conclusão: para o teste das hipóteses usamos o Teste T para dados emparelhados. Testes de dados emparelhados são tipicamente usados para estudos nos quais há a intenção de se verificar se um novo tratamento, técnica, ou método trabalha melhor que um método existente, sem ter que se preocupar com outros fatores ou outras questões que possam influenciar os resultados. Este tipo de teste é usual para populações onde o tamanho da amostra é pequeno (menor ou igual a 30). No caso de amostras de tamanho pequeno, a quantidade de informações para fundamentar suas conclusões é pequena não sendo confiável utilizar os valores presentes na distribuição normal padrão. Assim, na metodologia do teste há uma tabela de

Distribuição T (Tabela 6.8) com valores diferentes dos valores da distribuição normal padrão. Os valores da tabela dependem do tamanho da amostra.

Para melhor interpretação dos dados gerados pelos indivíduos em estudo a criação de pares nos permite eliminar a discrepância na análise de um projeto que tenha cinco requisitos com outro projeto que tenha onze requisitos. Assim usamos pares com quantidades de requisitos próximas para evitar distorções (Tabela 6.3).

Uma alternativa para comparação dos dados seria submeter o indivíduo da amostra (os projetos selecionados) aos dois processos em análise. Porém, algumas limitações nos impedem a montagem de tal cenário. A primeira delas é o custo para a empresa na qual se realiza o estudo, pois a execução do mesmo projeto duas vezes torna o custo inviável para a empresa. A segunda limitação é em relação à equipe envolvida. Uma mesma equipe executando o mesmo projeto duas vezes pode mascarar o resultado. O fator repetição influenciaria o processo, pois a probabilidade da segunda execução do projeto ser mais bem sucedida que a primeira é muito grande.

Validade de construção: as métricas definidas são utilizadas para estabelecer comparações entre o processo proposto e o processo antigo. Os métodos de coleta de dados foram bem explicitados para garantir que os dados corretos fossem coletados. Coletas feitas de forma incorreta, como a comparação de relatórios incompletos ou inacabados, podem influenciar o resultado final do estudo. Procedimentos foram criados e formalizados para facilitar o entendimento. Uma equipe foi designada e treinada para coleta de dados após reunião com todas as equipes envolvidas. Assim evitamos o aparecimento de versões diversas de uma mesma coleta por falta da definição de papéis e responsabilidades de cada um no processo. A reunião também teve um caráter de validação do processo de coleta de dados, alinhando-os com os objetivos do estudo. Desta forma conseguimos maior veracidade dos fatos e procedimentos criados aumentando assim a correspondência entre a estratégia de medida e os conceitos pretendidos.

Validade externa: a técnica foi empregada dentro do ambiente de desenvolvimento de software de uma empresa no negócio de cartões de crédito. Portanto, não é possível generalizar os resultados obtidos para a indústria; para obter resultados mais gerais seria necessária a aplicação do modelo proposto em outros projetos.

6.3 Análise e Interpretação dos Resultados

O estudo experimental foi realizado durante o ano de 2004 e o primeiro semestre de 2005 na empresa do ramo de cartões de crédito. Dez pessoas participaram do experimento.

Os participantes que realizaram o estudo experimental utilizando o processo proposto foram treinados na utilização desta técnica antes da realização do estudo. O treinamento foi realizado em uma sessão de quatro horas no início de 2004. Os participantes foram treinados em dois grupos: um dos grupos composto somente pelos usuários finais e o outro grupo formado pela equipe de teste, equipe de desenvolvimento e gerência de projeto.

6.3.1 Análise Quantitativa

Aplicando o percentual de requisitos alterados em relação aos requisitos iniciais de cada projeto (Tabela 6.4) percebemos um aumento significativo no percentual de alterações ocorridas nos requisitos. Dentre os projetos que utilizaram o processo proposto o valor percentual é maior nos projetos que usaram a prototipação (CEFA e Parcelamento Globestar) como técnica de validação de requisitos. Observamos também uma tendência de aumento da alteração dos requisitos em projetos onde a quantidade de requisitos iniciais é maior, tanto para os projetos que utilizaram o processo proposto como para os que utilizaram o processo antigo.

Tabela 6.4 - Percentual de Requisitos Alterados por Projeto.

Número do Par	Projetos	Processo	Número de Requisitos Iniciais	Número de Requisitos Alterados	Percentual de Requisitos Alterados
1	**CEFA**	Proposto	8	7	87,5%
2	**Parcelamento Globestar**	Proposto	11	9	81,8%
3	**CIB Versão 2**	Proposto	6	4	66,7%
4	**CIB Versão 3**	Proposto	7	5	71,4%
1	**Registro Automático de Parcelamento**	Antigo	9	4	44,4%
2	**24 Parcelas**	Antigo	10	6	60,0%
3	**Cadastro Correios**	Antigo	5	2	40,0%
4	**CIB Versão 1**	Antigo	6	2	33,3%

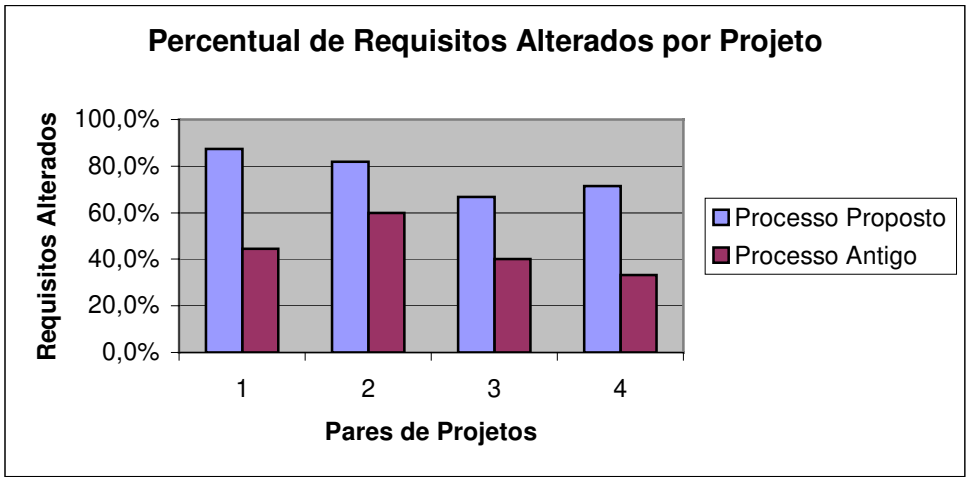

Figura 6.6 - Percentual de Requisitos Alterados por Projeto.

A variação percentual de requisitos novos em relação aos requisitos iniciais de cada projeto também foi satisfatória em comparação ao processo antigo (Tabela 6.5). O menor percentual encontrado no processo proposto foi superior ao maior percentual encontrado no processo antigo.

Tabela 6.5 - Percentual de Requisitos Novos por projeto.

Número do Par	Projetos	Processo	Número de Requisitos Iniciais	Número de Requisitos Criados	Percentual de Requisitos Novos
1	**CEFA**	Proposto	8	4	50,0%
2	**Parcelamento Globestar**	Proposto	11	5	45,5%
3	**CIB Versão 2**	Proposto	6	3	50,0%
4	**CIB Versão 3**	Proposto	7	3	42,9%
1	**Registro Automático de Parcelamento**	Antigo	9	2	22,2%
2	**24 Parcelas**	Antigo	10	2	20,0%
3	**Cadastro Correios**	Antigo	5	2	40,0%
4	**CIB Versão 1**	Antigo	6	2	33,3%

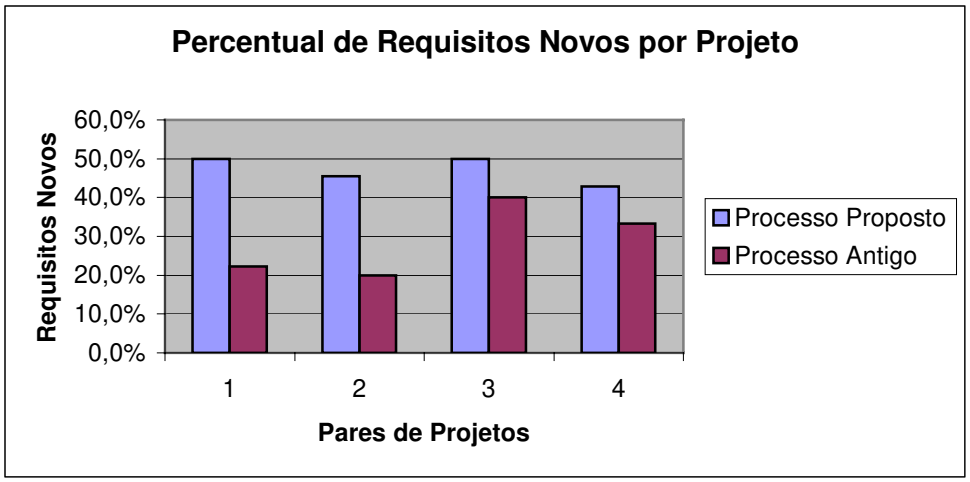

Figura 6.7 - Percentual de Requisitos Novos por projeto.

Aplicando a proporção de casos de teste por requisitos (Tabela 6.6) visualizamos três cenários distintos: processo proposto com protótipo no teste de aceitação, processo proposto sem protótipo no teste de aceitação e processo antigo. Na comparação entre o processo proposto e o processo antigo os dados do processo proposto sempre são superiores aos dados do processo antigo. Porém, quando comparamos os dados do processo proposto com protótipo com o processo proposto sem protótipo, percebemos que os valores do processo proposto com protótipo são bem superiores.

Tabela 6.6 - Proporção de Casos de Teste por Requisitos.

Número do Par	Projetos	Processo	Número de Requisitos	Quantidade de Casos de Teste	Número de Casos de Teste por Requisitos
1	**CEFA**	Proposto com Protótipo no Teste de Aceitação	12	1365	113,8
2	**Parcelamento Globestar**	Proposto com Protótipo no Teste de Aceitação	16	953	59,6
3	**CIB Versão 2**	Proposto sem Protótipo no Teste de Aceitação	9	274	30,4
4	**CIB Versão 3**	Proposto sem Protótipo no Teste de Aceitação	10	287	28,7
1	**Registro Automático de Parcelamento**	Antigo	11	316	28,7
2	**24 Parcelas**	Antigo	12	246	20,5
3	**Cadastro Correios**	Antigo	7	130	18,6
4	**CIB Versão 1**	Antigo	8	215	26,9

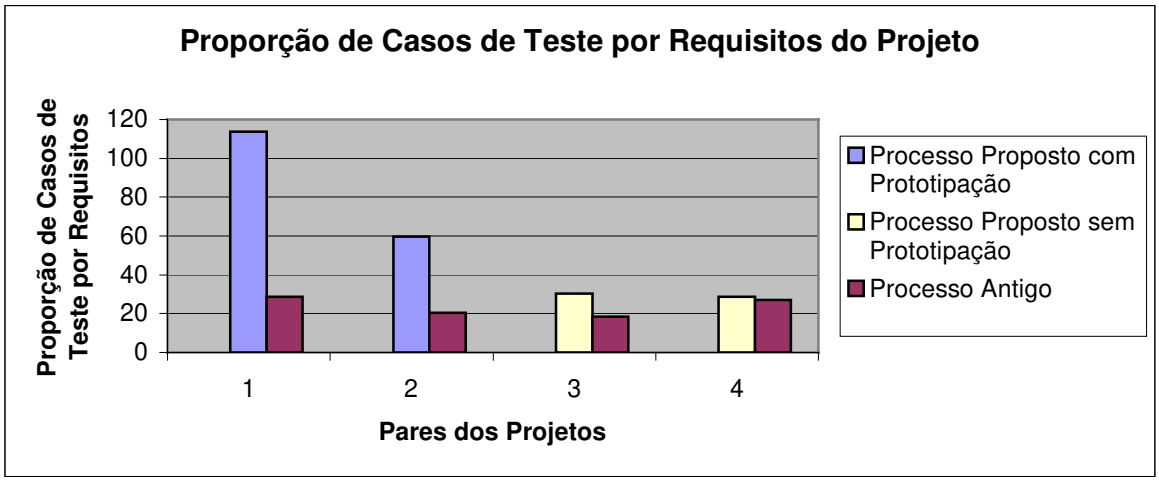

Figura 6.8 - Proporção de Casos de Teste por Requisitos em modo gráfico.

A Tabela 6.7 mostra a quantidade de defeitos encontrados nos projetos em estudo durante os Testes Aceitação. Vemos que houve um decréscimo em todos os projetos que usaram o processo proposto em comparação com os projetos que utilizaram o processo antigo.

Tabela 6.7 - Quantidade de defeitos encontrados nos Testes de Aceitação.

Número	Projetos	Processo	Quantidade de Defeitos durante os Testes de Aceitação
1	**CEFA**	Proposto	8
2	**Parcelamento Globestar**	Proposto	13
3	**CIB Versão 2**	Proposto	15
4	**CIB Versão 3**	Proposto	20
1	**Registro Automático de Parcelamento**	Antigo	32
2	**24 Parcelas**	Antigo	25
3	**Cadastro Correios**	Antigo	21
4	**CIB Versão 1**	Antigo	25

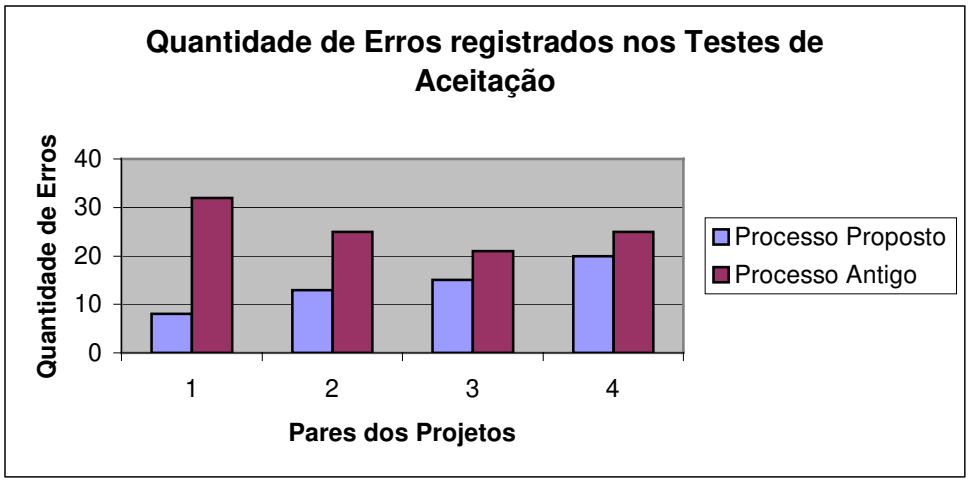

Figura 6.9 - Quantidade de erros encontrados nos Testes de Aceitação.

6.3.2 Aplicação do Teste Estatístico

Calculado a estatística *t* ($t = [\, d \,/\, (\, S \,/\, \sqrt{n}\,)]$, onde *d* é a média da diferença entre os pares, S é o desvio padrão e *n* é a quantidade de pares), com os dados experimentais, este deve ser comparado com o valor de *t* crítico, tabelado em função do número de graus de liberdade (Tabela 6.8) e o nível de significância escolhido para o teste (*n*). Caso o valor de *t* seja maior que o valor de *t* crítico (unilateral - 95%), a hipótese nula é rejeitada.

A Figura 6.10 ilustra o teste estatístico. O gráfico mostra a região de aceitação da hipótese H_0 (nula) para o teste unicaudal, chamada de P. A região chamada de α se refere à região de rejeição da hipótese H_0. E *p* é o valor do teste que, neste caso por se encontrar dentro de α, rejeitou H_0. Caso *p* não esteja dentro de α, H_0 será aceito [BAR04].

Tabela 6.8 – Distribuição T.

Graus de Liberdade	90th Percentile	95th Percentile	97.5th Percentile	98th Percentile	99th Percentile
1	3.078	6.314	12.706	31.821	63.657
2	1.886	2.920	4.303	6.965	9.925
3	1.638	2.353	3.182	4.541	5.841
4	1.533	2.132	2.776	3.747	4.604
5	1.476	2.015	2.571	3.365	4.032
6	1.440	1.943	2.447	3.143	3.707
7	1.415	1.895	2.365	2.998	3.499
8	1.397	1.860	2.306	2.896	3.355

9	1.383	1.833	2.262	2.821	3.250
10	1.372	1.812	2.228	2.764	3.169
11	1.363	1.796	2.201	2.718	3.106
12	1.356	1.782	2.179	2.681	3.055
13	1.350	1.771	2.160	2.650	3.012
14	1.345	1.761	2.145	2.624	2.977
15	1.341	1.753	2.131	2.602	2.947
16	1.337	1.746	2.120	2.583	2.921
17	1.333	1.740	2.110	2.567	2.898
18	1.330	1.734	2.101	2.552	2.878
19	1.328	1.729	2.093	2.539	2.861
20	1.325	1.725	2.086	2.528	2.845
21	1.323	1.721	2.080	2.518	2.831
22	1.321	1.717	2.074	2.508	2.819
23	1.319	1.714	2.069	2.500	2.807
24	1.318	1.711	2.064	2.492	2.797
25	1.316	1.708	2.060	2.485	2.787
26	1.315	1.706	2.056	2.479	2.779
27	1.314	1.703	2.052	2.473	2.771
28	1.313	1.701	2.048	2.467	2.763
29	1.311	1.699	2.045	2.462	2.756
30	1.310	1.697	2.042	2.457	2.750
40	1.303	1.684	2.021	2.423	2.704
60	1.296	1.671	2.000	2.390	2.660

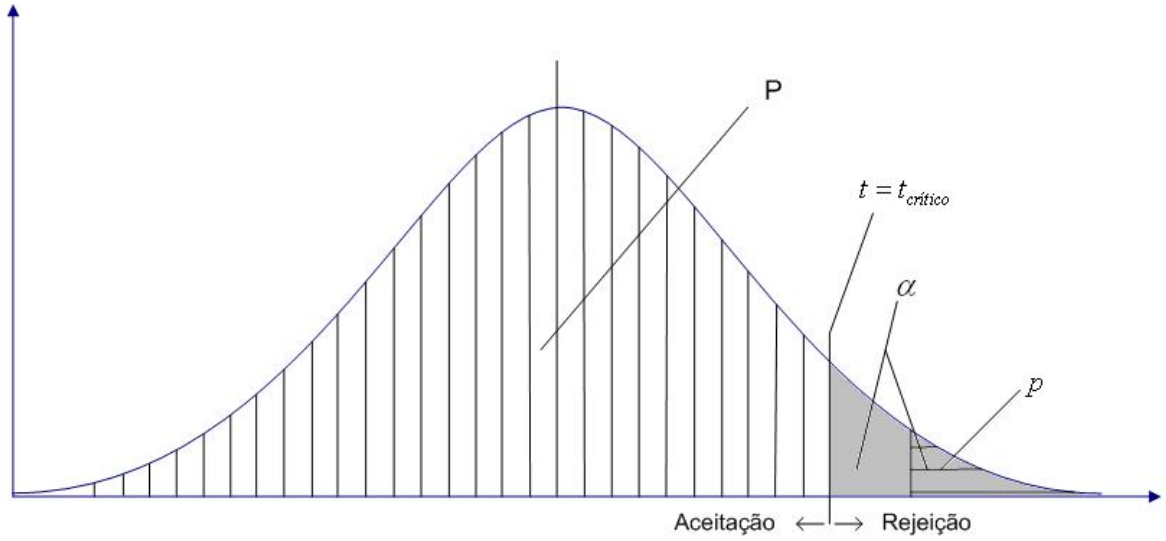

Figura 6.10 - Região de aceitação (P), rejeição (α) e o valor do teste (*p*).

6.3.3 Verificação das Hipóteses

Na análise das hipóteses (Tabela 6.9), o teste T foi conclusivo no sentido de afirmar que o processo proposto melhora a eficácia na revisão/inspeção da especificação dos requisitos em relação às métricas dos números de requisitos alterados (H1), número de requisitos novos (H2) e número de defeitos encontrados (H3). A eficácia na criação dos casos de teste (H4) não foi aceita pois H_0 foi aceito no teste estatístico.

Tabela 6.9 - Resultados do Teste T.

Teste T				
Variável	**H1: Requisitos Alterados (RA)**	**H2: Requisitos Novos (RN)**	**H3: Número de Erros Encontrados (EE)**	**H4: Número de Casos de Teste (CT)**
Média da Diferença (d)	2,75	1,75	-11,75	572
	4	4	4	4
Graus de Liberdade (n - 1)	3	3	3	3
Desvio Padrão (S)	0,50	0,96	8,73	600,07
Erro da média ($= S / \sqrt{n}$)	0,25	0,48	4,37	300,04
t	11,00	3,66	-2,69	1,91
$t_{crítico}$ (95%)	2,353	2,353	-2,353	2,353
$t > t_{crítico}$	Sim	Sim	Sim	Não
Resultado	$\mu_d > 0$	$\mu_d > 0$	$\mu_d < 0$	$\mu_d = 0$

6.3.4 Conclusão

Embora os testes estatísticos relacionados à eficácia na criação dos casos de teste não tenham sido conclusivos (Hipótese 4), o sucesso com os testes relacionados às técnicas de validação dos requisitos nos indica que elas são viáveis e podem auxiliar no processo como um todo. Ao contrário dos testes estatísticos, as análises quantitativas do processo proposto nos mostram melhoras em todas as métricas do estudo, reafirmando assim, o ganho em eficácia no processo proposto.

6.4 Lições Aprendidas

Em uma repetição do experimento devemos considerar alguns pontos, vistos como limitações de sua primeira execução.

Aprimoramento do treinamento: sugere-se para uma repetição do experimento, que as ferramentas a serem utilizadas nos experimentos sejam apresentadas aos participantes durante o treinamento. Ao invés de apenas expositivo, o treinamento deve permitir que os participantes apliquem as técnicas propostas. Apesar das técnicas aplicadas pelo processo proposto não terem sido usadas por todo o time, como por exemplo a equipe de projeto, o conhecimento mais detalhado da técnica ajudará a convencer quem gerencia o cronograma do projeto da necessidade de aumento do tempo de execução de determinadas atividades.

Instrumentação: as informações geradas pelo processo proposto foram todas armazenadas em planilhas eletrônicas e em diretórios em rede. Alguns participantes expressaram dificuldades no controle das planilhas eletrônicas. Assim sugeriram a construção de um sistema automatizado que dê suporte ao processo proposto.

7 Conclusão

7.1 Síntese do trabalho

O grande desafio das empresas que desenvolvem sistemas de software é construir softwares de qualidade e dentro do prazo inicialmente estipulado. Estas empresas encontram dificuldades na adequação e na aplicação de técnicas que diminuam o custo final e que não causem atrasos, mantendo a qualidade na entrega.

Para que as empresas consigam realizar e fazer cumprir suas metas de forma eficaz é necessário dar ênfase à atividade de teste ao longo do ciclo de desenvolvimento. Também é necessário criar mecanismos que facilitem a integração de todo o processo de construção de software com atividades ligadas à qualidade do produto de software. Assim, a adoção de técnicas de teste em fases iniciais, conforme foi demonstrado no trabalho apresentado, diminui a propagação de erros ao longo do projeto e, conseqüentemente, melhora a qualidade da entrega dos sistemas.

Com o objetivo de melhorar a eficácia do processo de desenvolvimento de software, este trabalho apresentou um modelo de processo adaptado do Modelo V. Este modelo foi criado a partir de aspectos teóricos, oriundos de uma revisão bibliográfica sobre aspectos de engenharia de requisitos e teste de software, e a partir de aspectos práticos, oriundos de um estudo de caso que sustenta o modelo de processo proposto.

O trabalho utilizou técnicas existentes na bibliografia que possibilitaram a reutilização dos artefatos na fase de teste de aceitação. Isto foi possível utilizando as condições de teste criadas para os testes de validação de requisitos na fase de teste de aceitação pelo usuário. Assim foi possível verificar com mais cuidado se o que foi pedido no início, foi realmente entregue. Os protótipos construídos para validação dos requisitos também foram reutilizados na fase de teste de aceitação, o que propiciou o aumento da cobertura dos testes.

O estudo experimental realizado mostrou que o modelo de processo proposto foi melhor do que o Modelo V de processo. Isto foi constatado tanto na revisão dos requisitos como na criação dos casos de teste. A melhoria trouxe os seguintes benefícios: maior quantidade de condições de teste executadas nos testes de aceitação, maior quantidade de requisitos novos e alterados no processo de validação dos requisitos e menor quantidade de defeitos nos testes de aceitação. A

diminuição da quantidade de defeitos nos testes de aceitação aponta um importante ganho de qualidade e de custo no processo, ou seja, um menor número de defeitos encontrados sugere menor quantidade de defeitos propagados até as fases finais do processo de desenvolvimento e menor tempo na execução dos testes de aceitação.

7.2 Trabalhos Futuros

Entre os trabalhos futuros, podemos citar:

- Criar uma base histórica dos projetos da organização para servir de fonte de informação para a criação de métricas. Esta base serviria para definição de limites superior e inferior aceitáveis para os resultados de cada métrica criada, permitindo melhor interpretação dos dados. As métricas serviriam para tomada de decisões corretivas no processo de construção do software.
- O empacotamento padronizado dos dados experimentais pode servir como base para uma organização concreta da informação empírica, possibilitando armazenar artefatos diferentes como os resultados e experiências finais dos projetos realizados.
- Acrescentar ao estudo de caso métricas para medir a qualidade final do produto e o ganho relativo ao esforço de desenvolvimento. A medida utilizada para medição da qualidade final do produto seria a quantidade de defeitos encontrados em produção para os produtos desenvolvidos segundo o processo proposto e segundo o processo antigo. Já a medida utilizada na medição do ganho sobre o esforço seria a quantidade total de pessoas-mês despendida no processo proposto e no processo antigo.

8 Referências Bibliográficas

[ABB86] Abbott, R.J.; **An Integrated Approach to Software Development**; Nova York:John Wiley, 1986.

[BLA01] Blackburn, Mark R.; Busser, Robert; Nauman, Aaron; **Removing Requirement Defects and Automating Test**, Software Productivity Consortium NFP, 2001.

[BAR04] Barbetta, P.A.; Reis, M.M.; Bornia, A.C.; **Estatística para Cursos de Engenharia e Informática;** São Paulo: Atlas, 2004.

[BAS02] Basili, V.R.; Caldiera, G.; Rombach, H. D.; **The Goal Question Metric Approach**; Encyclopedia of Software Engineering. Vol. 1, John Wiley & Sons, Inc., 2nd ed., 2002, pp. 578-583.

[BOE81] Boehm, B.; **Software Engineering Economics**; Prentice-Hall, 1981.

[BRO87] Brooks Jr., Frederick P; **No Silver Bullet: Essence and Accidents of Software Engineering**; IEEE Computer. Vol. 20, n. 4, p. 10-19, 1987.

[CAR01] Carvalho, Ana E. S. de; Tavares, Helena C.; Castro, Jaelson B.; **Uma estratégia para Implantação de uma Gerência de Requisitos Visando a Melhoria dos Processo de Software**; WER01 - Workshop em Engenharia de Requisitos, Buenos Aires, Argentina, 2001. Anais p. 32-54.

[CHA02] Chaim, M. L.; Maldonado, J. C.; Jino, M.; **Processo de Depuração depois do Teste: Definição e Análise**; Boletim de Pesquisa e Desenvolvimento, Empresa Brasileira de Pesquisa Agropecuária, Outubro de 2002. Disponível em http://www.cnptia.embrapa.br/modules/tinycontent3/content/2002/bolpesq5.pdf. Acessado em: 10 jan. 2006.

[COH04] Cohen, Cynthia F.; Birkin S. J.; Garfield M. J.; Webb H. W.; **Managing conflict in software testing;** Communications of the ACM. Vol. 47, n. 1, 2004.

[COL04] Coley Consulting; **Business System Advice**; Atualizado em Maio de 2004. Disponível em: http://www.coleyconsulting.co.uk/article.htm. Acessado em: 10 jan. 2006.

[COR02] Cordeiro, Marco A.; **Uma Ferramenta Automatizada de Suporte ao Processo de Gerenciamento de Requisitos**; Dissertação de Mestrado em Ciências, Centro de Ciências Exatas e de Tecnologia – CCET, Pontifícia Universidade Católica do Paraná – PUCPR, 2002.

[CRE04] Crespo, A. N.; Silva, O. J.; Borges, C. A.; Salviano, C. F.; Argollo, M. T. , Jino, M..; **Uma Metodologia de Teste de Software no Contexto de Melhoria de Processo**; III Simpósio Brasileiro de Qualidade de Software – SBQS, Brasília-DF, 2004.

[CRIS01] Christiensen, M., Thayer, R.; **The Project Manager´s Guide to Software Engineering´s Best Practices**; IEEE Computer Society Press., 2001.

[DAV93] Davis, A. M.; **Software Requirements – Objects, Functions and States**; PTR-Prentice Hall, 2nd ed., 1993.

[DEN00] Denardis, Chris; **Perspectives from a Test Manager**; Software Testing & Quality Engineering Magazine; Setembro/Outubro de 2000; Disponível em http://www.stickyminds.com/. Acessado em: 10 jan. 2006.

[DOM02] Domingues, A. L. S.; **Avaliação de Critérios e Ferramentas de Teste para Programas OO**; Dissertação de Mestrado, ICMC-USP, São Carlos – SP, 2002.

[FAU97] Faulk, S. R.; **Software Requirements: A Tutorial**; in Software Requirements Engineering, 2nd. ed., IEEE CS Press, 1997, pp. 128-149.

[GAO92] US General Accounting Office, **Mission Critical Systems: Defense Attempting to Address Major Software Challenges**; GAO/IMTEC-93-13, December 1992.

[GOL02] Goldsmith, Robin F.; **This or That, V or X?**; Agosto de 2002. Disponível em: http://www.sdbestpractices.com/documents/s=8815/sdm0208e/. Acessado em: 10 jan. 2006.

[GOL02-1] Goldsmith, Robin F.; Graham, D.; **The Forgotten Phase**; Julho de 2002; Disponível em: http://www.sdmagazine.com/documents/s=7224/sdm0207e/0207e/. Acessado em: 10 jan. 2006.

[GON03] Gonçalves, K. V.; **Teste de Software em Aplicações de Banco de Dados Relacional**; Trabalho Final de Mestrado Profissional, Unicamp, Campinas – SP, 2003.

[GOT02] Gottesdiener, E.; **Top Ten Ways Projects Teams Misure Use Cases and How to Correct Them**; The Rational Edge; Julho de 2002. Disponível em: http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/jul02/TopTenWaysJul02.pdf . Acessado em: 10 jan. 2006.

[IEEE829-98] **IEEE Standard for Software Test Documentation**, Nova Iorque: IEEE, ANSI/IEEE Std 829-1998, 1998.

[IEEE830-98] **IEEE Recommended Practice for Software Requirements Specification**; Nova Iorque: IEEE, ANSI/IEEE Std 830-1998, 1998.

[IEEE1233-98] **IEEE Guide for Developing System Requirements Specifications**; Nova Iorque: IEEE, ANSI/IEEE Std 1233-1998, 1998.

[IEEE610-90] **IEEE Standard Glossary of Software Engineering Terminology**; IEEE Std 610.12-1990, 1990.

[KHA01] Khalifa, G.; Irani, Z.; Baldwin, L. P.; Jones, S.; **Evaluating Information Technology With You In Mind**; Department of Information Systems and Computing, Brunel

University, Uxbridge, UK, 2001. Disponível em: http://www.ejise.com/volume-4/volume4-issue1/issue1-art5.htm. Acessado em: 10 jan. 2006.

[KOT98] Kotonya, G. and Sommerville, I.; **Requirements Engineering: Process and Techniques**; John Wiley and Sons, 1998.

[KUL00] Kulak, D. and Guiney, E.; **Use Cases – Requirements in Context**; Addison-Wesley, 2000.

[LEI94] Leite, Júlio C.S.P.; **Engenharia de Requisitos**"; 1 ed.; PUC-RIO; Rio de Janeiro; 1994; (Notas de Aula).

[LIN04] Lindström, B.; **A Software Measurement Case Study Using GQM;** Dissertação de Mestrado, Department of Communications Systems, Lund Institute of Technology, Lund University, Suécia, 2004.

[LOP99] Lopes, Paulo Sérgio N. D.; **Engenharia de Requisitos**; Comentário e avaliação da Introdução e do Capítulo 1 do livro "Requirements Engineering", Seminário de Análise de Requisitos, Universidade de São Paulo, Março, 1999.

[LOP02] Lopes, Paulo Sérgio N. D.; **Uma Taxonomia da Pesquisa na Área de Engenharia de Requisitos**; Dissertação Mestrado em Ciência da Computação, Instituto de Matemática e Estatística , Universidade de São Paulo – USP, 2002.

[MAL03] Maldonado, J. C; Barbosa E. F.; **Teste de Software: Teoria e Prática**; XVII Simpósio Brasileiro de Engenharia de Software - SBES´2003, Manaus, 2003.

[MAR01] Martins, Luiz E.G.; **Uma Metodologia de Elicitação de Requisitos de Software Baseada na Teoria da Atividade**; Tese de Doutorado em Engenharia Elétrica, Faculdade de Engenharia Elétrica e de Computação - FEEC, Universidade Estadual de Campinas, 2001.

[MYE79] Myers, G. J.; **The art of Software Testing**; 1ª ed., John Wiley & Sons, Inc., 1979.

[POH94] Pohl, K.; **The Three Dimensions of Requirements Engineering: A Framework and its Applications**; Information Systems; Vol. 19, n. 3, p. 243-258, 1994.

[PRE02] Pressman, R.S.; **Engenharia de Software**; 5ª ed., MacGraw-Hill, 2002.

[RAM96] Ramachandran, M.; **Requirements-Driven Software Test: A Process-Oriented Approach**; Software Enginnering Note, ACM Sigsoft, Vol. 21, n. 4, p 66-70, 1996.

[RAP85] Rapps, S. and Weyuker, E. J.; **Data Flow Analysis Tecniques for Test Data Selection**; IEEE Transactions on Software Engineering, vol. 11(4). Abril, 1985.

[ROY70] ROYCE, W. W.; **Managing the development of large software systems**"; Proccedings of IEEE WESCON, p. 1-9, 1970.

[SAF00] Safford, Ed; **Test Automation Framework, State-based and Signal Flow Examples**; Twelfth Annual Software Technology Conference; Maio de 2000.

[SAY03] Sayão, Miriam; Staa, Arndt V.; Leite, Júlio C. S. P.; **Qualidade em Requisitos;** Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro – RJ, Outubro de 2003.

[SHE92] Sheldon, F.T.; Kavi, K.M.; Tausworthe, R.C.; Yu, J.T.; Brettschneider, R.; Everett, W.W.; **Reliability Measurement from Theory to Practice**; IEEE Software, Vol. 9, ed. 4, Julho de 1992.

[SPI03] Spinellis, D.; **Advantages of Prototype Development**; Software Development; Dezembro de 2003; Disponível em http://www.dmst.aueb.gr/dds/etech/swdev/index.htm. Acessado em: 10 jan. 2006.

[STD94] Standish; **The Chaos Report**; The Standish Group International, Inc., USA, 1994.

[TRA02] Travassos, G. H.; Gurov, D.; Amaral, E. A. G.; **Introdução à Engenharia de Software Experimental**. Relatório Técnico RT-ES-590/02, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

[ZAN02] Zanlorenci, E. P; Burnett, R. C.; **Certificação de qualidade em engenharia de requisitos;** SBQS 2002 - Simpósio Brasileiro de Qualidade de Software, 2002; Anais p. 71-83.