Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Ricardo Caram Balthasar

Requisitos Rastreados por Testes: Técnica para Utilização das Especificações de Testes no Rastreamento de Requisitos

São Paulo
2010

Ricardo Caram Balthasar

Requisitos Rastreados por Testes: Técnica para Utilização das Especificações de Testes no Rastreamento de Requisitos.

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.

Data da aprovação: _____ /_____ /_____

_____
Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dr. Carlos Eduardo de Barros Paes (Membro)
PUC – Pontifícia Universidade Católica de São Paulo

Prof. Dr. José Eduardo Zindel Deboni (Membro)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Ricardo Caram Balthasar

Requisitos Rastreados por Testes: Técnica para Utilização das Especificações de Testes no Rastreamento de Requisitos

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.

Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo
Setembro/2010

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

B197r **Balthasar, Ricardo Caram**

Requisitos rastreados por testes: técnica para utilização das especificações de testes no rastreamento de requisitos. / Ricardo Caram Balthasar. São Paulo, 2010.
86p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Reginaldo Arakaki

1. Rastreamento 2. Requisitos de software 3. Teste de software 4. Processo de software 5. Engenharia de software 6. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

11-05 CDU 004.414.3(043)

AGRADECIMENTOS

Ao Prof. Dr. Reginaldo Arakaki, pelo apoio, dedicação, incentivo e pelos conselhos prestados ao longo deste período, durante todo o processo de orientação e desenvolvimento deste trabalho.

Aos membros da banca de avaliação, professores Dr. José Eduardo Zindel Deboni e Dr. Carlos Eduardo de Barros Paes pelos sábios conselhos, orientações e contribuições dadas durante o exame de qualificação.

Ao meu filho, Bruno, que com menos de dois anos me incentivou com sua alegria e seus inúmeros sorrisos, à minha esposa Érika, pelo amor, carinho e paciência demonstrados ao longo deste período, ao meu mais novo filho, Caio, que mesmo antes de nascer tem me incentivado com seus chutinhos de alegria.

Aos meus familiares, especialmente aos meus pais Leila e Marco Antonio, pelo incentivo e por acreditarem em mim.

E a todas as outras pessoas que, de forma direta ou indireta, me ajudaram no desenvolvimento deste trabalho.

RESUMO

A rastreabilidade em desenvolvimento de sistema é um conjunto de práticas utilizadas para rastrear os requisitos e as mudanças durante o processo de desenvolvimento, tendo como um dos principais objetivos manter as informações organizadas e facilitar a localização. O rastreamento ajuda na compreensão do código fonte, na identificação dos requisitos relacionados e conflitantes e estimula a separação de conceitos e o reuso de componentes, aumentando assim a qualidade geral do processo de software. Apesar dos benefícios, a rastreabilidade demanda a criação de artefatos que não fazem parte do ciclo regular de desenvolvimento. A necessidade de artefatos específicos faz com que a rastreabilidade ser torne obsoleta durante a vida útil do software. Neste trabalho, é apresentada uma técnica para utilizar os cenários de testes para dirigir o processo de desenvolvimento de software e obter a rastreabilidade de requisitos utilizando-se de artefatos já presentes no ciclo regular de desenvolvimento. A técnica utiliza os Cenários de Teste para clarificar requisitos e descrever decisões arquiteturais e utiliza uma versão extendida do TDD (*Test Driven Development*) para implementar os cenários de testes com a finalidade de demonstrar que o rastreio pode ser obtido como um subproduto da atividade de teste. Para demonstrar a técnica são utilizados uma aplicação de conversão de moeda e uma aplicação de linha de produto de software para ponto de venda, de onde a matriz de rastreabilidade é extraída dos artefatos de teste.

Palavras-chave: rastreabilidade, identificação de requisitos, desenvolvimento de software, cenários de teste.

ABSTRACT

Test Tracked Requirements: Technique for Test Specification Usage in Requirements Traceability

System development traceability is a set of practices used to track the requirements and changes during the development process, having as one of its main objectives keeping the information organized in order to facilitate the localization. The traceability supports the source code comprehension, the identification of related and conflicting requirements and stimulates separation of concerns and reuse of components, thus increasing the overall quality of the software process. In spite of the benefits, the traceability demands the creation of artifacts that are not part of the regular development cycle. The need for specific artifacts makes the traceability obsolete during the software life cycle. This work presents a technique for using test scenarios to drive the software development process and to obtain the requirement traceability using artifact available in the regular development cycle. The technique makes use of test scenarios so as to clarify requirements and to describe architectural decisions. Additionally, the technique uses an extended version of TDD (Test Driven Development) to implement the test scenarios so as to demonstrate that the traceability can be obtained as a subproduct from testing. The technique is demonstrated on a currency conversion application and on a point of sales software product line application, from which the traceability matrix is extracted from test artifacts.

Keywords: traceability, requirement identification, software development, test scenarios.

Lista de Ilustrações

Figura 1 – Artefatos de rastreio orientado a modelos 17
Figura 2 – Modelo tradicional de rastreio 18
Figura 3 – Rastreio vertical de requisitos 19
Figura 4 – Caso de teste de uso 24
Figura 5 – Teste de aceitação utilizando FIT 25
Figura 6 – Criação sistemática de teste funcional 27
Figura 7 – Arquitetura do framework xUnit 31
Figura 8 – Modelo tradicional de implementação de teste unitário 32
Figura 9 – Modelo V de Desenvolvimento 34
Figura 10 – Estrutura do *Framework* Scrum 36
Figura 11 – Modelo de desenvolvimento tradicional 40
Figura 12 – Modelo XP de desenvolvimento 41
Figura 13 – Etapas do TDD 43
Figura 14 – Modelo de rastreio de história de usuário 45
Figura 15 – Modelo ágil de desenvolvimento 48
Figura 16 – Desenvolvimento Dirigido e Rastreado por Testes (DDRT) 49
Figura 17 – Modelo geral do rastreio dirigido por testes 50
Figura 18 – Rastreio dirigido por testes 51
Figura 19 – Relatório de carteira em múltiplas moedas 52
Figura 20 – Rastreio de funcionalidade por testes de aceitação 53
Figura 21 – Rastreio de decisões arquiteturais 58
Figura 22 – Teste unitário como elemento de rastreio 60
Figura 23 – Modelo proposto de implementação de teste unitário 62
Figura 24 – Modelo de componentes do PDV Global 64
Figura 25 – Exemplo de especificação de requisito 66
Figura 26 – Estrutura de Logs do PDV Global 70
Figura 27 – Teste unitário com associação ao requisito 71
Figura 28 – Implementação de Teste Unitário com características de rastreio 72
Figura 29 – Malha de rastreio por artefatos de teste 75

Lista de Tabelas

Tabela 1 – Matriz de rastreabilidade 20
Tabela 2 – Falhas de integração 30
Tabela 3 – Teste de Aceitação: Relatórios de Carteira de Ações 55
Tabela 4 – Teste Funcional: Relatórios de Carteira de Ações 55
Tabela 5 – Teste de Aceitação: Exibir múltiplos pedidos no monitor de expedição 68
Tabela 6 – Teste Arquitetural: Exibir múltiplos pedidos no monitor de expedição 71

Lista de Abreviaturas e Siglas

CMMI	*Capability Maturity Model Integration*
CTU	Caso de Teste de Uso
DAS	Desenvolvimento Ágil de Software
FIT	*Framework for Integrated Tests*
IDE	Integrated Development Environment
IEEE	*Institute of Electrical and Electronics Engineers*
LPS	Linha de Produto de Software
PDV	Ponto De Venda
DDRT	Desenvolvimento Dirigido e Rastreado por Testes
ROI	*Return Of Investment*
SEI	*Software Engineering Institute*
TDD	*Test Driven Development*
XP	*Extreme Programming*

Sumário

1 INTRODUÇÃO 9
1.1 Motivação 9
1.2 Objetivo 10
1.3 Resultados Esperados 10
1.4 Método de Trabalho 11
1.5 Trabalhos Relacionados 11
1.6 Organização do Trabalho 14
2 FUNDAMENTOS CONCEITUAIS 15
2.1 Rastreabilidade 16
2.2 Atributos de Rastreio 19
2.3 Técnicas de Rastreio 20
2.4 Testes de Software 21
2.5 Teste de Aceitação 22
2.6 Teste Funcional 26
2.7 Teste de Integração 27
2.8 Teste Unitário 30
2.9 Teste de Regressão 32
2.10 Processos e Modelos de Desenvolvimento com Destaque em Testes 33
2.11 Modelo V de Desenvolvimento 33
2.12 Desenvolvimento Ágil de Software 34
2.13 Scrum 35
2.14 Programação Extrema (XP) 38
2.15 Desenvolvimento Dirigido por Testes (TDD) 42
2.16 Rastreabilidade no Desenvolvimento Ágil de Software 44
2.17 Resumo 45
3 DESENVOLVIMENTO DIRIGIDO E RASTREADO POR TESTES - DDRT 47
3.1 Rastreio de Requisitos pela Especificação Dirigida por Testes 52
3.2 Rastreio de Decisões Arquiteturais 56
3.3 Rastreio de Código Fonte através de Testes Unitários 58
3.4 Criação e Armazenamento dos Artefatos de Rastreio 63
3.5 Resumo 63
4 APLICAÇÃO DE EXEMPLOS PRÁTICOS 64
4.1 Linha de Produto de Sistemas de Ponto de Venda 64
4.2 Aplicação da Proposta 67
4.3 Definindo os Testes de Aceite 67
4.4 Validando a Arquitetura do Sistema através dos Testes de Integração 68
4.5 Implementando os Testes Unitários 71
4.6 Visualizando a Rastreabilidade 73
4.7 Analise de Resultados 76
5 CONCLUSÃO 78
5.1 Contribuições 79
5.2 Sugestões para Futuras Pesquisas 79
REFERÊNCIAS 81

1 INTRODUÇÃO

1.1 Motivação

A Arquitetura de Software é uma disciplina que tem evoluído muito nos últimos anos. Esta disciplina utiliza uma série de atributos de qualidade do processo de desenvolvimento de software e do sistema para medir e trabalhar na garantia da qualidade. A validação das funcionalidades, a facilidade de manutenção e a análise de impacto são qualidades de arquitetura que podem ser obtidas quando a rastreabilidade de requisitos é posta em prática (SOUSA, 2008).

A rastreabilidade, utilizada em todas as fases do processo de desenvolvimento, suporta diversas atividades da engenharia de software, conforme apontam Cleland-Huang et al. (2003). Dentre elas estão:

- Análise de impacto: na identificação dos artefatos afetados por um novo requisito e nos requisitos afetados por um defeito encontrado em um determinado artefato do sistema;
- Validação dos requisitos e do projeto: na identificação dos requisitos não implementados e também na identificação dos artefatos não atrelados a qualquer requisito;
- Geração de cenários de testes;
- Identificação das funcionalidades implementadas;
- Justificativa racional da funcionalidade;
- Identificação de requisitos conflitantes e requisitos dependentes.

Um estudo elaborado por Damian e Chisan (2006) apontou que a rastreabilidade influi positivamente no acompanhamento do projeto, na conformidade da especificação, na revisão, na validação de requisitos e no gerenciamento de mudanças.

O rastreio possibilita a identificação do que foi implementado, a razão ou o motivo da implementação, a metodologia de teste aplicada, assim como a relação entre os artefatos produzidos durante o desenvolvimento (JACOBSSON, 2009).

Todas as técnicas de aplicação de rastreabilidade requerem o empenho de uma equipe multifuncional para criar e manter as ligações entre os requisitos de produtos, a sua origem e a forma como afetam trabalhos posteriores (WESTFALL, 2006).

Em ambientes de desenvolvimento ágil e/ou iterativo evolutivo, este problema pode ser ainda pior, pois requisitos mudam constantemente de acordo com as necessidades e as regras de negócio (JACOBSSON, 2009).

Diante da importância da rastreabilidade no processo de desenvolvimento de software, a motivação deste trabalho é de obter o rastreamento de requisitos como um subproduto da atividade de teste. Fundamenta-se portanto, na utilização das especificações de teste, essencial em qualquer processo de desenvolvimento de software, não apenas como artefato de verificação e validação, mas principalmente como elemento de rastreio capaz de dirigir o processo de desenvolvimento de software. Para tanto, este trabalho propõe uma extensão ao desenvolvimento dirigido por testes (*Test Driven Development* - TDD) na criação do Desenvolvimento Dirigido e Rastreado por Testes com Rastreio (DDRT) ou *Test Driven Development & Tracking* (TDD&T).

1.2 Objetivo

O objetivo deste trabalho é criar uma técnica para utilizar as especificações de testes com a finalidade de dirigir o desenvolvimento de software através de cenários de testes e, dessa forma, rastrear os requisitos em duas direções, verticalmente - entre requisitos e código fonte, e horizontalmente - na relação entre requisitos.

1.3 Resultados Esperados

Os resultados esperados são:

- Apresentar uma técnica para obter a rastreabilidade como um subproduto da atividade de teste;
- Evidenciar os testes como elementos arquiteturais de rastreio através de uma aplicação prática em uma Linha de Produto de Software para Ponto de Venda;

- Demonstrar uma técnica para criação de testes unitários para rastrear diferentes tipos de requisitos.

1.4 Método de Trabalho

O primeiro passo na busca de tal objetivo é a identificação dos benefícios da rastreabilidade proposta pelo Software Engineering Institute/ Capability Maturity Model Integration (SEI/CMMI) através da matriz da rastreabilidade.

Analisa-se a literatura para identificar as principais dificuldades em manter a rastreabilidade e as principais alternativas de implementação, principalmente em ambientes ágeis de desenvolvimento.

A partir do modelo de desenvolvimento dirigido por testes (TDD), em que casos de teste são criados antes mesmo de o código ser escrito, propõe-se a utilização dos cenários de testes para dirigir todo o processo de desenvolvimento.

Descreve-se, então, passo a passo, como utilizar os cenários de teste para rastrear requisitos.

Assim, a proposta é validada através de exemplos práticos com a finalidade de identificar:

- O benefício do rastreio ao qual o teste pretende atender;
- Os tipos de implementação de testes que podem ser aplicados;
- Os benefícios trazidos em implementar o teste para a qualidade geral do processo e para o sistema.

1.5 Trabalhos Relacionados

Rastreabilidade, em desenvolvimento de sistema, é um conjunto de práticas utilizadas para identificar os requisitos e as mudanças durante o processo de desenvolvimento (JACOBSSON, 2009), tendo como um dos principais objetivos manter as informações organizadas e de fácil localização.

Muito utilizada em ambientes tradicionais de desenvolvimento, a rastreabilidade é pouco utilizada em ambientes de desenvolvimento ágil por ser considerada um procedimento excessivamente burocrático. Porém, quando o desenvolvimento do sistema é finalizado e transferido para o cliente ou para a área

de manutenção, o rastreio e a documentação possibilitam a identificação do que foi implementado, a razão ou o motivo da implementação, a metodologia de teste aplicada, assim como a relação entre os artefatos produzidos durante o desenvolvimento (JACOBSSON, 2009).

Jacobsson (2009) propôs um conjunto de práticas para introduzir a rastreabilidade em um ambiente *scrum* de desenvolvimento ágil e concluiu que, apesar de haver algumas práticas de custo zero para o desenvolvimento, outras são custosas e têm impacto negativo no Retorno do Investimento (ROI – *Return of Investment*). O autor sugeriu ainda que as práticas não precisam ser aplicadas em sua totalidade, mas sim combinadas de maneira a atender de forma mais adequada o tipo de projeto que se deseja rastrear.

Um ponto importante para a rastreabilidade ágil é que a informação deve ser gravada de uma maneira que não crie muito trabalho extra. Jacobsson (2009) propõe que uma maneira de se evitar a criação extra de documentação é criar rotinas e ferramentas de tal maneira que a informação de rastreabilidade possa ser obtida sem a necessidade de criação de documentos de rastreio.

Uma dos grandes desafios da rastreabilidade é a possibilidade de se obter o rastreio pelo ciclo natural do desenvolvimento, a chamada “rastreabilidade em um clique”. Hayes et al. (2009) sugere ainda que a integração da rastreabilidade e os novos paradigmas de desenvolvimento - como o TDD - podem introduzir um grande salto para a obtenção da “rastreabilidade em um clique”.

A natureza do desenvolvimento por teste - escrever o teste, escrever o código que passe no teste e refatorar o código - faz com que o código fonte, o sistema, os requisitos e os testes estejam sempre alinhados (HAYES et al., 2009).

As mesmas características que fazem do TDD um modelo propício à rastreabilidade trazem também alguns desafios. O processo dinâmico de desenvolvimento baseado no *refactoring* faz com que as ligações entre requisitos e código fonte sejam constantemente alteradas. Além das dificuldades apontadas pelo *refactoring,* Hayes et al. (2009) apontam outros dois desafios a serem tratados pelo TDD:

- Identificação da data, da origem e do requisitante de uma alteração do sistema;

- Adição de artefatos que façam com que o TDD não seja tão “leve” quanto a comunidade de desenvolvimento ágil defende que os processos ágeis devam ser.

Baseado nestes desafios, Hayes et al. (2009) sugerem que estudos sejam feitos para identificar quais informações de rastreio são importantes para serem coletadas e analisadas pelos desenvolvedores (o que rastrear), e qual a metodologia a ser utilizada pelo TDD para a coleta e a manutenção da rastreabilidade no processo de desenvolvimento (como rastrear).

Cleland-Huang et al. (2003) apontam que uma das principais dificuldades na rastreabilidade é a sua manutenção. Afirmam que a maioria dos métodos e ferramentas não disponibiliza meios para que os *links* de rastreio sejam atualizados no tempo correto com o intuito de evitar que a rastreabilidade torne-se obsoleta e propõem uma rastreabilidade orientada a eventos. Neste modelo, uma mudança de requisitos é representada por uma série de eventos de mudança de tal forma que uma mensagem de evento é disparada cada vez que uma mudança é solicitada para que uma ação (manual ou automatizada) seja tomada a fim de restabelecer o rastreio entre os artefatos.

Apesar da importância deste modelo na identificação dos artefatos afetados por uma mudança de requisitos, o modelo proposto por Cleland-Huang et al. (2003) não diminui o custo nem o esforço de atualização dos *links* de rastreio, que algumas vezes precisam ser analisados e modificados de maneira não automatizada.

Cenários escritos em XML representam o modelo proposto por Naslavsky et al. (2005). O levantamento de requisitos, a análise de sistemas, a definição da arquitetura do sistema e dos componentes, assim como os detalhes de implementação são expressos por cenários. A relação entre os artefatos se dá através das identificações dos cenários. O modelo proposto é muito semelhante ao rastreio orientado a modelos, a rastreabilidade é bem definida e forte, mas de alto custo de manutenção e implementação.

Entretanto, por ser escrita em XML e com artefatos identificados por códigos, a solução provê bons mecanismos de busca, facilitando a implementação de ferramentas que ajudam na análise de impacto e na identificação dos requisitos de usuários nos artefatos implementados no sistema.

1.6 Organização do Trabalho

O Capítulo 2 - Fundamentos Conceituais - apresenta os conceitos relacionados à rastreabilidade e os principais trabalhos relacionados à rastreabilidade, testes e desenvolvimento ágil de sistemas.

O Capítulo 3 - Rastreio de Requisitos Dirigidos por Testes - apresenta os testes com as características de rastreio, as implementações de testes e os benefícios de qualidade resultantes dos testes.

O Capítulo 4 - Aplicação de exemplos práticos - utiliza os testes propostos no Capítulo 3 em uma linha de produto de software para sistema de ponto de venda.

No Capítulo 5 - Conclusão - apresenta os aspectos positivos e negativos, as contribuições do trabalho, as propostas para trabalhos futuros e os resultados obtidos.

2 FUNDAMENTOS CONCEITUAIS

Neste capítulo são apresentados os principais conceitos que servem de base para o desenvolvimento deste trabalho: testes de software, modelos de desenvolvimento de software baseados em teste e a rastreabilidade de requisitos.

O ponto principal deste trabalho, a rastreabilidade de requisitos, é abordado no primeiro item deste capítulo. Sendo uma disciplina da gerência de requisitos, a rastreabilidade lida com a possibilidade de se identificar o requisito em cada um dos artefatos criados durante a implementação do sistema. O item aborda também os principais benefícios desta disciplina, assim como os meios tradicionais de implementação.

O teste de software é um conjunto de processos desenvolvido para verificar que o software criado faz o que foi programado para fazer, assim como verificar que o software não faz nada que não seja intencional (PEZZE e YOUNG, 2008). Presente em quase todos os modelos de desenvolvimento de software, o teste de software implementado em diversas etapas do desenvolvimento é utilizado principalmente para auxiliar na garantia da qualidade.

Dentre os modelos e processos de desenvolvimento de software, este trabalho destaca o Modelo V e os modelos de desenvolvimento que foram base para a criação do manifesto ágil: programação extrema (XP) e desenvolvimento dirigido por testes (TDD).

O Modelo V é considerado uma extensão ao modelo de desenvolvimento em cascata, o qual adiciona a cada etapa do processo do modelo em cascata um artefato de validação (RODRIGUES, 2006). Assim como no modelo cascata, o Modelo V possui fases bem definidas, em que o avanço para a próxima fase do desenvolvimento acontece somente quando todas as possibilidades e análises da fase anterior foram esgotas.

A programação extrema ou *Extreme Programming* (XP) é um modelo de desenvolvimento baseado na simplicidade e na comunicação informal, no qual o desenvolvimento é dirigido por testes. A programação, a análise, o desenho e os testes são feitos em pares e não são restritos a uma única área. Além disso, a integração é feita logo após a implementação através dos testes de integração (BECK e ANDRES, 2004).

Uma das técnicas utilizadas pelo XP e que se tornou notória também fora dos ambientes ágeis, o desenvolvimento dirigido por testes (TDD), reforça que só se deve implementar o código necessário para que os casos de testes passem. Com isto, só existe código se houver testes para validar o código que será escrito (BECK, 2002).

2.1 Rastreabilidade

O IEEE define rastreabilidade como sendo o grau em que uma relação pode ser estabelecida entre dois ou mais produtos do processo de desenvolvimento, especialmente os produtos com predecessor-sucessor ou com a relação mestre e subordinado. (“IEEE 610”, 1990).

Ratanotayanon et al. (2009) definem rastreamento como sendo a habilidade de descrever e seguir o ciclo de vida de um requisito em ambas as direções: do requisito ao código fonte e do código fonte ao requisito.

O rastreamento ajuda na compreensão do código fonte, na identificação dos requisitos relacionados (rastreio de requisitos) e conflitantes (análise de impacto) e estimula a separação de conceitos, assim como o reuso de componentes, aumentando, assim, a qualidade geral do processo de software.

Ajudar a manter todas as informações organizadas e fáceis de localizar é um dos objetivos da rastreabilidade. Além de facilitar a busca das informações, os elos entre os diversos artefatos produzidos e as decisões que definiram a elaboração são importantes subprodutos da rastreabilidade. Os principais beneficiados por este tipo de rastreamento são os projetos de longa duração, visto que as decisões tomadas são de difícil memorização; o mesmo problema ocorre em projetos com grandes equipes (JACOBSSON, 2009).

Além de ser de grande importância para projetos de longa duração, a rastreabilidade de requisitos é muitas vezes obrigatória no desenvolvimento de sistemas para agências de governo (ASUNCION et al., 2007).

A rastreabilidade, nos modelos tradicionais de desenvolvimento, é geralmente mantida e solicitada pela gerência de configuração. Neste caso, o elemento de rastreio entre os requisitos e o código fonte são os artefatos gerados na fase de

Design e Projeto - os diagramas estáticos e dinâmicos do sistema. Conforme ilustrado nas Figuras 1 e 2, um requisito é transcrito em modelos estáticos e dinâmicos durante as fases de projeto; feito isto, ele é transcrito na linguagem de programação que produzirá os componentes do sistema final.

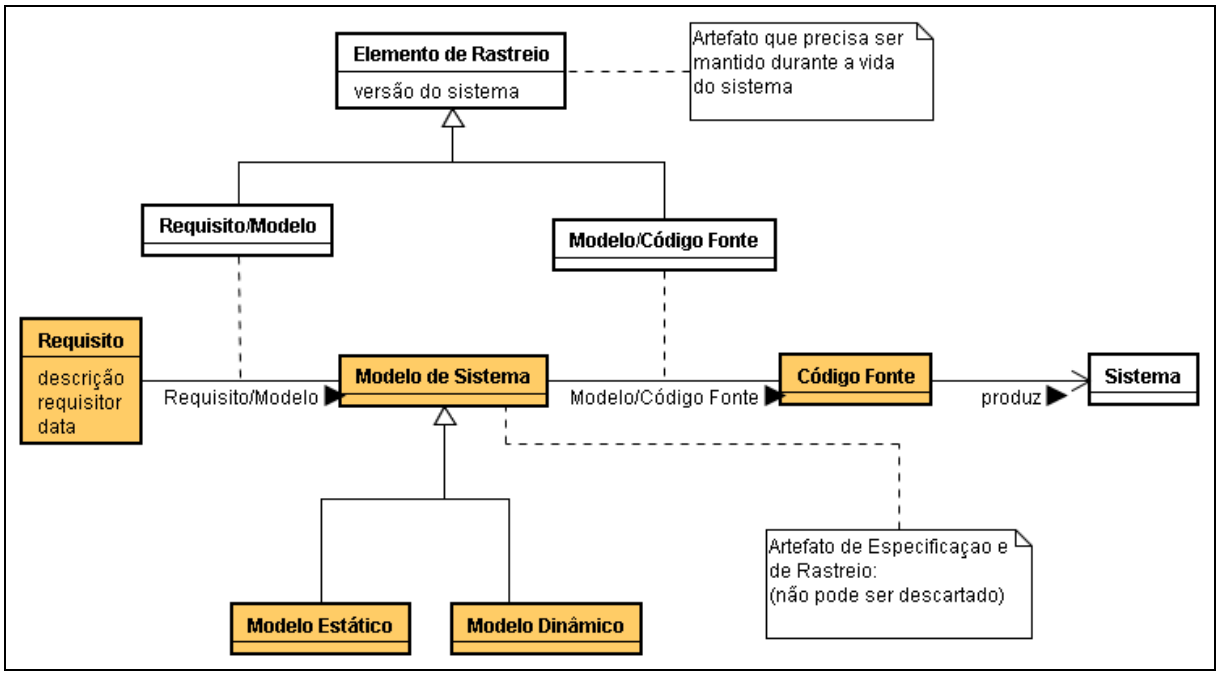


Figura 1 – Artefatos de rastreio orientado a modelos
Fonte: Elaborado pelo autor

Estes modelos são geralmente suportados pela hierarquia estrita do modelo de desenvolvimento tradicional, no qual o gerente de projeto ou o gerente de configuração determinam que os desenvolvedores devem inserir e documentar as informações de rastreio como parte do trabalho de codificação, diretrizes estas seguidas pelo time de desenvolvimento (JACOBSSON, 2009). Em ambientes de desenvolvimento ágil, esta hierarquia não está necessariamente presente.

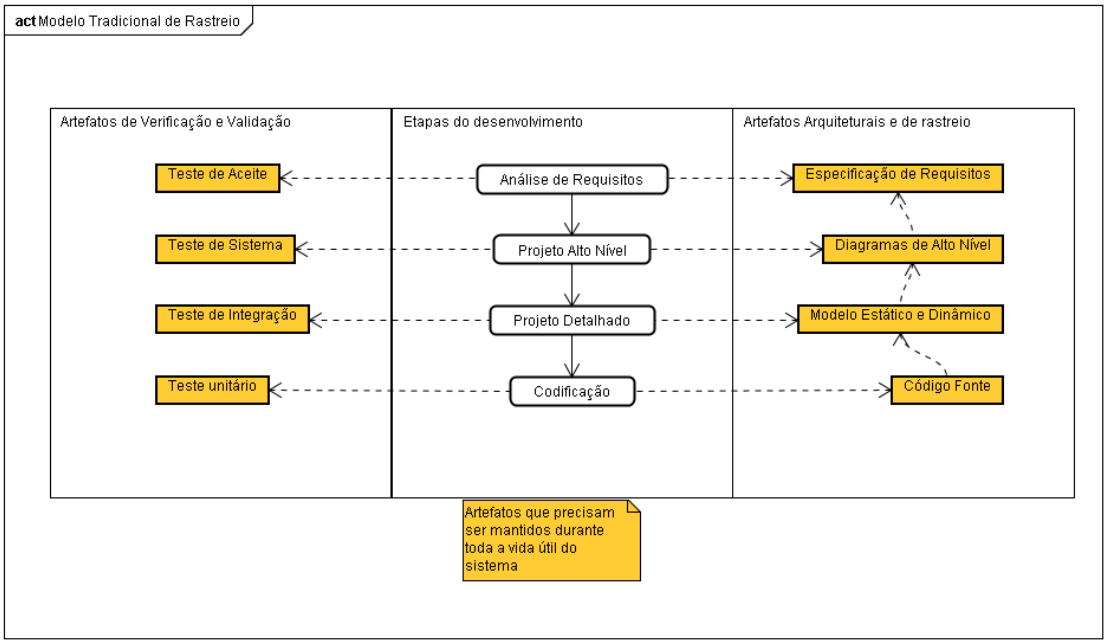


Figura 2 – Modelo tradicional de rastreio
Fonte: Elaborado pelo autor

Rastrear implica não apenas identificar como os requisitos foram implementados, mas também a origem do requisito, as razões ou os problemas para os quais o requisito procura uma solução, a dependência entre requisitos, os artefatos (testes, códigos, entre outros), assim como a versão do sistema em que o requisito foi implementado.

A rastreabilidade vertical, utilizada para identificar todos os artefatos produzidos para implementar um requisito, pode ser implementada através da matriz de rastreabilidade, de ferramentas de análise de requisitos ou através de marcações de rastreio (WESTFALL, 2006). Já a rastreabilidade horizontal, que faz a ligação entre requisitos, geralmente é mantida pelas ferramentas de análise de requisitos.

Conforme ilustra a Figura 3, quando analisada no sentido da origem dos requisitos para o código fonte, a rastreabilidade vertical implica não apenas em identificar como os requisitos foram implementados, mas também assegurar a completeza da especificação para garantir que cada requisito tenha sido devidamente implementado e testado.

Quando analisada no sentido inverso, a rastreabilidade vertical facilita o entendimento dos artefatos criados no processo de desenvolvimento, provendo informações adicionais, como a descrição do requisito e sua origem e ajudando a

garantir que o sistema continue evoluindo no caminho correto em relação ao original e que os artefatos produzidos sejam criados para atender a um determinado requisito, ou seja, que o escopo do projeto não está sendo indevidamente aumentado.

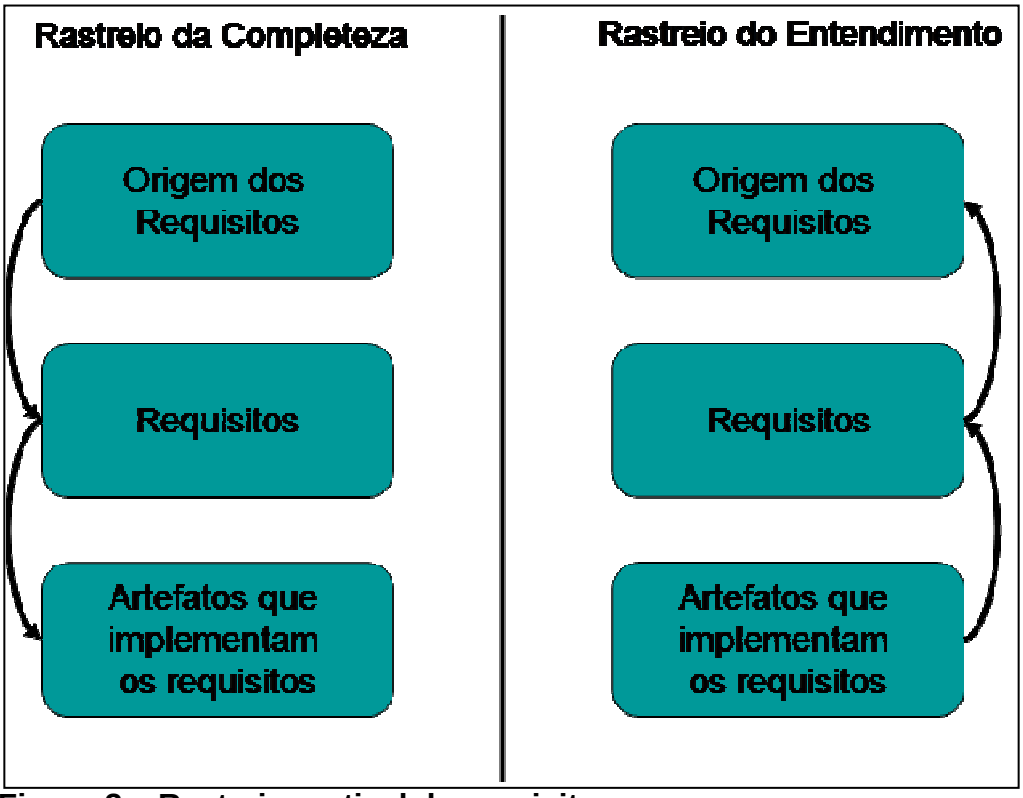


Figura 3 – Rastreio vertical de requisitos
Fonte: Adaptado de Westfall (2006)

2.2 Atributos de Rastreio

- **Requisito com Origem**: Identifica a razão de negócio pela qual os requisitos foram solicitados, devendo conter informações relevantes do cliente para validar se o problema inicial foi convertido em requisitos.
- **Requisito com Artefatos de Sistema:** Também conhecido como rastreamento vertical, identifica todos os artefatos produzidos e afetados na implementação de um determinado requisito.
- **Requisito com Versão do Sistema:** Identifica a partir de qual versão do sistema o requisito e a funcionalidade do sistema foram implementados e disponibilizados para o usuário.

- **Requisito com Requisito:** Identifica os requisitos interrelacionados, visto que mudanças de comportamento em um requisito podem acarretar mudanças de comportamento em outro. Indica também requisitos conflitantes e/ou contraditórios.

2.3 Técnicas de Rastreio

Segundo Westfall (2006), existem três maneiras para se rastrear requisitos: rastreabilidade por matriz, por marcação e por ferramentas de gerenciamento de requisitos.

Na rastreabilidade por matriz, modelo clássico de rastreabilidade, cada requisito e cada artefato produzido são identificados por um código único que pode ser referenciado em uma tabela ou planilha que contém todos os requisitos e os respectivos artefatos, conforme ilustrado na Tabela 1. Apesar de ser uma solução simples, a manutenção desta matriz é uma tarefa não automatizada e que exige um intenso trabalho (WESTFALL, 2006).

Origem	**Requisito**	**Diagrama de Domínio**	**Diagrama de Componentes**	**Código Fonte**	**Teste de Aceitação**	**Teste Funcional**	**Manual do Usuário**
Caso de Uso #210	Conversor de moeda	Modelo Estático DD2.0	Modelo de interação DC2.2	Moeda.cpp Conversor.cpp	UAT 2.1 UAT 2.2 UAT 2.3	CT 2.1.1 CT 2.1.2 CT 2.2.1	Cap. 2

Tabela 1 – Matriz de rastreabilidade
Fonte: Adaptado de Westfall (2006)

De maneira semelhante à rastreabilidade por matriz, a rastreabilidade por marcação necessita que cada requisito e artefato tenham identificação única. Estas identificações são utilizadas para estabelecer uma conexão de um artefato com o artefato imediatamente superior a ele, formando assim uma corrente de rastreabilidade. Sua vantagem em relação à rastreabilidade por matriz deve-se ao fato de a rastreabilidade por marcação não necessitar da criação de um documento de rastreio, pois a identificação de rastreio pode ser aplicada diretamente aos artefatos criados no desenvolvimento do sistema (WESTFALL, 2006).

Por se tratar de um trabalho custoso, a atualização dos diagramas durante a evolução do sistema e das mudanças dos requisitos é pouco realizada, quebrando assim o elemento de rastreio. Pelo fato de se tornarem desatualizados, esses diagramas podem ser considerados um elemento de rastreio estático (EGYED e GRÜNBACHER, 2002).

As ferramentas mais atuais de levantamento e análise de requisitos provêm mecanismos de rastreio. Mesmo utilizando ferramentas, a rastreabilidade ainda é uma tarefa não automatizada, de alto custo, que exige um intenso trabalho e de difícil evolução.

Segundo Asuncion et al. (2007) e Egyed e Grünbacher (2002), os seguintes problemas são apontados quando ferramentas de análise de requisito são utilizadas na rastreabilidade:

- Alto custo das ferramentas comerciais, tanto em relação à aquisição da licença de uso quanto em relação às horas necessárias para o aprendizado;
- Inconsistências na representação de um mesmo artefato;
- Desatualização dos artefatos cadastrados na ferramenta em relação aos artefatos utilizados na construção do sistema;
- O não comprometimento com a rastreabilidade total da cadeia de desenvolvimento, isto é, rastreamento dos requisitos, dos modelos e diagramas e do código fonte;
- Dificuldade de escalar.

2.4 Testes de Software

O teste de software é um processo ou um conjunto de processos desenvolvidos para verificar que o software criado faz o que foi programado para fazer, assim como verificar que o software não faça nada que não seja intencional (PEZZE e YOUNG, 2008).

Presente em quase todos os modelos de desenvolvimento de software, o teste de software implementado em diversas etapas do desenvolvimento é utilizado principalmente para auxiliar na garantia da qualidade.

Myers (2004) afirma que o teste de software consiste de uma série de processos criados para tornar o software previsível e consistente, oferecendo, assim, nenhuma surpresa para os usuários.

Projetar os testes no início do processo de desenvolvimento traz diversas vantagens. Os testes são projetados enquanto os detalhes sobre as necessidades e as especificações técnicas estão vivos na mente dos analistas. Além disso, os casos de testes podem revelar inconsistências ou incompletudes nas especificações (PEZZE e YOUNG, 2008).

Testar é o processo de avaliar se o software está sendo executado corretamente. Portanto, a confiança de que o software está sendo bem executado está diretamente relacionada à quantidade e à qualidade de testes realizados (BECK, 2002).

Nesta seção, resumem-se as atividades do processo de testes relevantes para o desenvolvimento deste trabalho. São elas: teste de aceitação, teste funcional, teste de integração, teste unitário e teste de regressão.

2.5 Teste de Aceitação

O teste de aceitação é elaborado e executado em conjunto com o cliente com o objetivo de verificar se o sistema se comporta de acordo com o especificado e atende as necessidades do usuário.

Validando o sistema sob o ponto de vista do requisitante, o teste de aceitação não é influenciado por interpretações dos requisitos traduzidas nas especificações de sistema, de componentes ou na codificação (GOLDSMITH, 2002a).

O teste de aceitação é uma atividade de validação focada principalmente na aceitabilidade do produto e inclui julgamento sobre a utilidade e a usabilidade em vez de ser um julgamento sobre a conformidade com o requisito (PEZZE e YOUNG, 2008), orientando assim a decisão de quando o produto deveria ser liberado em seu estado atual.

A orientação para liberação do produto não pode ser baseada em dados obtidos através de critérios subjetivos. Portanto, a decisão da liberação do produto sob o ponto de vista dos testes de aceitação é normalmente baseada em testes

estatísticos, incluindo confiança, disponibilidade e tempo médio entre falhas (PEZZE e YOUNG, 2008).

Uma abordagem menos formal para o teste de aceitação é testar o sistema diretamente com usuários distribuindo uma versão preliminar do sistema e coletar pareceres sobre falhas e usabilidade. Os testes *alfa e beta*, nomes comuns a este tipo de teste de aceitação, são executados por uma amostra de usuários que deve refletir o perfil operacional dos usuários da aplicação final (PEZZE e YOUNG, 2008).

Segundo Myers (2004), o objetivo do teste de aceitação é comparar o programa com os requisitos do sistema e com as necessidades dos usuários finais.

Beato (2008) propõe que Casos de Teste de Uso elaborados nas fases iniciais do projeto, durante a análise de requisito, sejam também utilizados nas fases finais do projeto, durante os testes de aceitação. Como ilustrado na Figura 4, os Casos de Teste de Uso (CTU) são utilizados para detalhar os Casos de Uso ou as histórias de usuário através de cenários de testes. O CTU é composto por um cabeçalho com informações sobre o Caso de Uso (nome do sistema, do caso de uso, data de elaboração, objetivo principal e versão do artefato), e por um conjunto de casos de teste que contém informações sobre cada caso de teste (número sequencial, condição, dados de entrada, resultado esperado e observações).

Caso de Teste de Uso

Sitema: **Elaborado por:**

Caso de Uso: **Executado por:**

Data da Elaboração: **Versão:**

Objetivo principal:

Seq	Condição	Dados de Entrada	Resutado Esperado	Observações

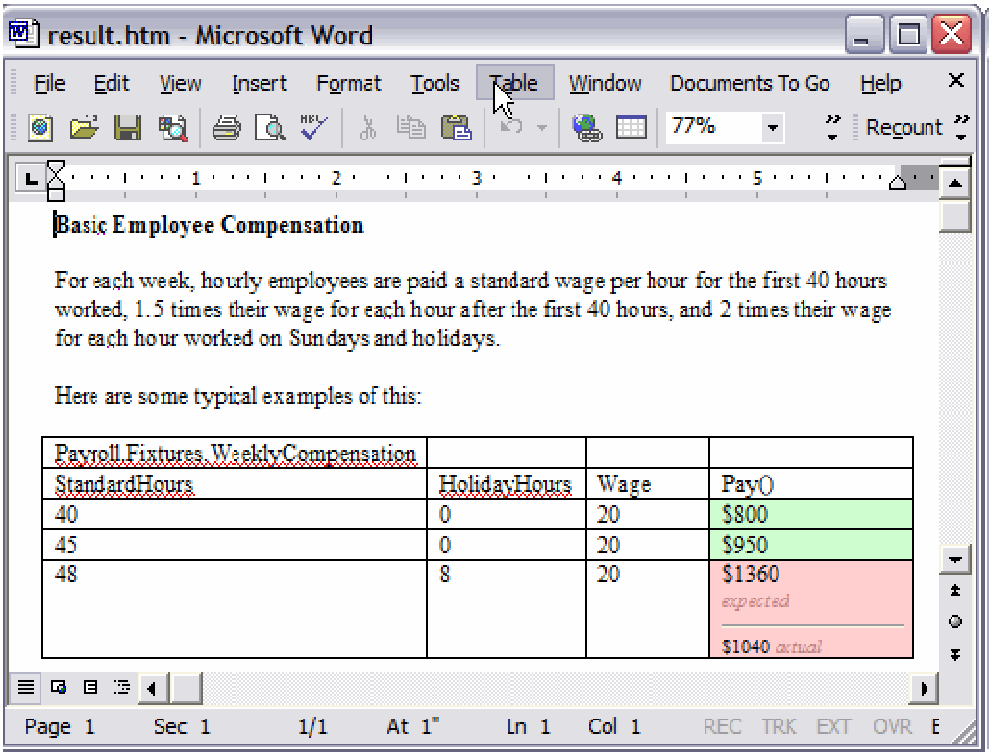
Figura 4 – Caso de teste de uso
Fonte: Adaptado de Beato (2008)

Na mesma linha, Ricca et al. (2007) e Park e Maurer (2008) propõem a utilização (FIT) na criação dos testes de aceitação desenvolvidos pelo usuário, pelo analista de negócio e pelos engenheiros de garantia de qualidade, como um elemento para clarificar requisitos e como um elemento de aceite para orientar a decisão de quando o produto possa ser liberado. Ricca et al. (2007) apontam ainda que os testes de aceite elaborados através do *framework* FIT são de grande importância para o entendimento dos requisitos.

O *Framework for Integrated Tests* (FIT) é uma ferramenta utilizada na elaboração de testes sob a perspectiva de negócio que utiliza tabelas para representar os testes e a geração automática dos resultados de execução destes testes. Um de seus objetivos é permitir que pessoas sem conhecimento de programação possam programar os testes de aceitação dos requisitos (MUGRIDGE e CUNNINGHAM, 2005).

O FIT é formado por duas partes principais: as tabelas, que descrevem os testes e são elaboradas pelos usuários em conjunto com o analista de negócio; e os *"fixtures"*, que são desenvolvidos pelo programador e fazem a ligação das tabelas com o sistema a ser testado.

A Figura 5 ilustra a elaboração de um teste de aceitação utilizando a tabela FIT e o código *"fixture"* correspondente para testar o requisito.




```
public class WeeklyCompensation : ColumnFixture
{
    public int StandardHours;
    public int HolidayHours;
    public Currency Wage;

    public Currency Pay()
    {
        WeeklyTimesheet timesheet = new WeeklyTimesheet(StandardHours, HolidayHours);
        return timesheet.CalculatePay(Wage);
    }
}
```

Figura 5 – Teste de aceitação utilizando FIT
Fonte: Extraído de Cunningham (2002)

2.6 Teste Funcional

O teste funcional é um processo de tentativa de se encontrar discrepâncias entre o programa e a especificação externa. Uma especificação externa é uma descrição precisa do comportamento do programa a partir do ponto de vista do usuário final (MYERS, 2004).

Para executar um teste funcional, a especificação externa é analisada para derivar um conjunto de casos de teste. Este processo de geração dos casos de teste procura responder à questão “quais casos de teste devem ser utilizados para exercitar o programa” (MYERS, 2004) e (PEZZE e YOUNG, 2008).

Enquanto o teste de aceite é utilizado para verificar se o software faz aquilo que ele foi desenhado para fazer, o teste funcional é utilizado para verificar as exceções, ou seja, se o software realmente não faz nada que não tenha sido solicitado a fazer. Enfatizando o que o sistema não deve fazer, os casos de teste com frequência conseguem revelar fraquezas e incompletudes nas especificações externas (PEZZE e YOUNG, 2008).

Como o universo de possibilidades do que o sistema não deve fazer é muito maior do que o universo das funcionalidades das quais derivam os testes de aceitação, a derivação dos casos de teste funcionais a partir de especificações externas é um processo analítico complexo que não deve ser executado sem critério por força bruta (PEZZE e YOUNG, 2008). A Figura 6 ilustra o processo sistemático de criação de testes funcionais para produzir casos de testes mais fáceis de monitorar e menos dependentes da capacitação dos projetistas de teste.

A derivação de casos de teste funcionais a partir de especificações externas, construído por um processo sistemático como o ilustrado na Figura 6, é uma execução mais trabalhosa e que gera menos casos de teste quando comparada com a execução por força bruta, mas produz casos de teste mais fáceis de monitorar e repetir e que são menos dependentes da capacitação dos projetistas de teste.

A identificação das Funcionalidades Testáveis Independentemente facilita a criação de casos de teste e simplifica o problema, permitindo que cada funcionalidade seja analisada separadamente. A próxima etapa é selecionar os valores representativos, identificando diferentes categorias de dados como valores limite, de erros, cadeias vazias, entre outros (PEZZE e YOUNG, 2008). Os dados

devem ser selecionados para mostrar que o sistema não faz nada que não foi solicitado para fazer. Os casos de testes são então criados através das especificações dos casos de testes que serão implementados, criando o código auxiliar necessário para execução.

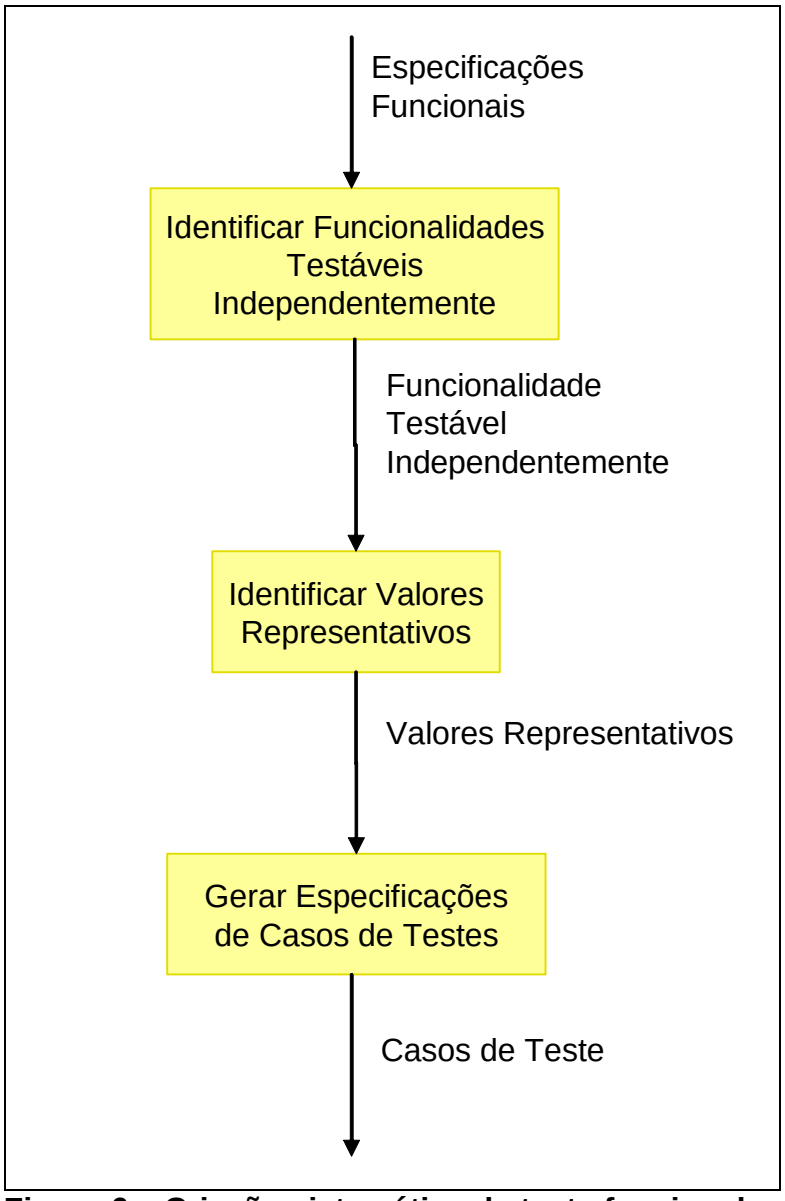


Figura 6 – Criação sistemática de teste funcional
Fonte: Adaptado de Pezze e Young (2008)

2.7 Teste de Integração

O teste de integração valida o sistema como um todo: entradas e saídas, funcionalidades requeridas, características não funcionais como desempenho, documentação, comportamento em ambiente de produção, entre outros. (RODRIGUES, 2006) e (GOLDSMITH, 2002a).

O teste de integração é focado na construção do sistema, sendo elaborado pela equipe técnica de desenvolvimento (desenvolvedores e testadores) (RODRIGUES, 2006).

Este teste verifica se os subsistemas e componentes estão se comportando de acordo com sua especificação, valida a interface com os outros componentes e as entradas e saídas, conforme aponta Goldsmith (2002a). Além disso, valida se o componente se comunica com outras partes do sistema de maneira correta, conforme o especificado no desenho do sistema (RODRIGUES, 2006).

O teste de integração verifica a compatibilidade dos módulos e é construído a partir do teste unitário e das inspeções. Testes de integração são, de certa forma, uma verificação dos testes unitários, isto é, falhas identificadas durante o teste de integração podem ser um sinal de teste unitário insatisfatório (PEZZE e YOUNG, 2008).

As falhas de integração são geralmente causadas por especificações incompletas, falhas na implementação da interface dos componentes, erros não tratados ou pré e pós condições não respeitadas.

Na Tabela 2 é apresentado um resumo com alguns exemplos de falha de integração e as consequências destas falhas na indústria (PEZZE e YOUNG, 2008). A tabela é composta das seguintes colunas:

- Falha de integração: indica uma possível classe de falha de especificação ou implementação;
- Exemplo: contém um exemplo da falha e a conseqüência desta falha.

Falha de Integração	**Exemplo**
Interpretação inconsistente dos parâmetros ou valores A interpretação de cada módulo pode ser razoável, mas eles são incompatíveis	Unidades não combinam: uma mistura de sistema métrico com o sistema inglês (metros e jardas), acredita-se, levou à perda do satélite *Mars Climate*, em setembro de 1999.
Violações dos valores de domínio ou dos limites de capacidade ou tamanho	Estouro de buffer, um limite de capacidade implícito (não-verificado) imposto por um módulo, é violado por

Premissas implícitas sobre intervalos de valores ou tamanhos	outro e torna-se uma vulnerabilidade de segurança notória. Por exemplo, algumas versões do servidor Web Apache 2, entre 2.0.35 e 2.0.50, podiam ter o buffer estourado no momento da expansão das variáveis de ambiente durante a carga do arquivo de configuração.
Efeitos colaterais nos parâmetros ou recursos	Um módulo frequentemente usa recursos que não são mencionados explicitamente em sua interface. Problemas de integração ocorrem quando estes efeitos implícitos de um módulo interferem com os de outro. Por exemplo, usar um arquivo temporário “tmp” pode ser invisível até a integração com outro módulo que também tenta usar um arquivo temporário “tmp” no mesmo diretório de arquivos descartáveis.
Perda ou má compreensão de funcionalidades Funcionalidades mal especificadas podem levar a premissas incorretas sobre os resultados esperados	A contagem de acessos em sites da Web pode ser feita de muitas maneiras diferentes: por endereço IP único, por acesso, incluindo ou excluindo *spiders*, e assim por diante. Os problemas ocorrem se a interpretação assumida no módulo de contagem for diferente da usada pelos módulos clientes.
Problemas não funcionais	Propriedades não funcionais, tais como desempenho, normalmente são explicitamente especificadas somente quando são uma característica esperada. Mesmo quando o desempenho não é

	explicitamente especificado, espera-se que o software forneça resultados em um tempo razoável. A interface entre os módulos pode reduzir o desempenho para menos que um limite mínimo aceitável
Incompatibilidades dinâmicas Muitas linguagens e *frameworks* permitem a ligação dinâmica. Os problemas podem ser causados por falhas nas "conexões" quando os módulos são integrados.	Chamadas polimórficas podem ser levadas a dinamicamente ativar métodos incompatíveis.

Tabela 2 – Falhas de integração
Fonte: Pezze e Young (2008, p. 433)

Uma estratégia para o teste de integração é efetuar os testes de maneira incremental com a montagem dos módulos em subsistemas cada vez maiores. Quando se utiliza da montagem incremental, é necessário utilizar o código de apoio - "*drivers*" e "*stubs*" que simularão os componentes "clientes" e os componentes "provedores", respectivamente – para que os subsistemas possam ser devidamente testados (MYERS, 2004) (PEZZE e YOUNG, 2008).

2.8 Teste Unitário

Os testes unitários são implementados pelos próprios desenvolvedores na mesma linguagem de programação utilizada para desenvolver os módulos do sistema.

Os testes unitários geralmente são implementados pelo *framework* xUnit. Disponível em diversas linguagens de programação, as implementações dos xUnit compartilham uma mesma arquitetura de componentes conforme mostra a Figura 7.

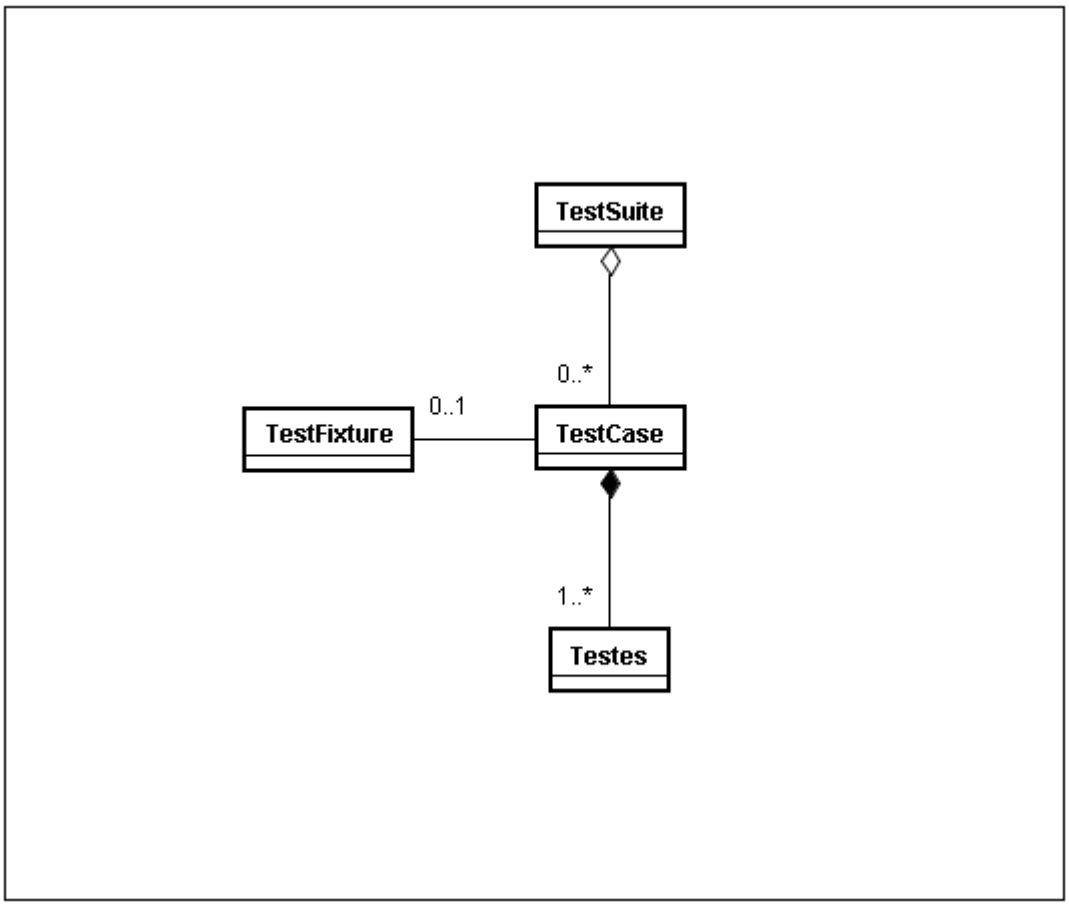


Figura 7 – Arquitetura do framework xUnit
Fonte: Elaborado pelo autor

No xUnit, o *TestSuite* é utilizado para agrupar um conjunto de *TestCases* geralmente associado a um componente de sistema. Desta maneira, como ilustra a Figura 8, o *TestSuite* é criado para testar o Componente de Sistema. Este *TestSuite* é composto por um conjunto de *TestCases* que testam as diversas Classes e Módulos que fazem parte do Componente de Sistema. Por sua vez, cada *TestCase* contém um conjunto de testes que validam cada método ou função da Classe ou do Módulo sendo validado pelo *TestCase* utilizando de uma massa de dados controlada e definida pelo *TestFixture.*

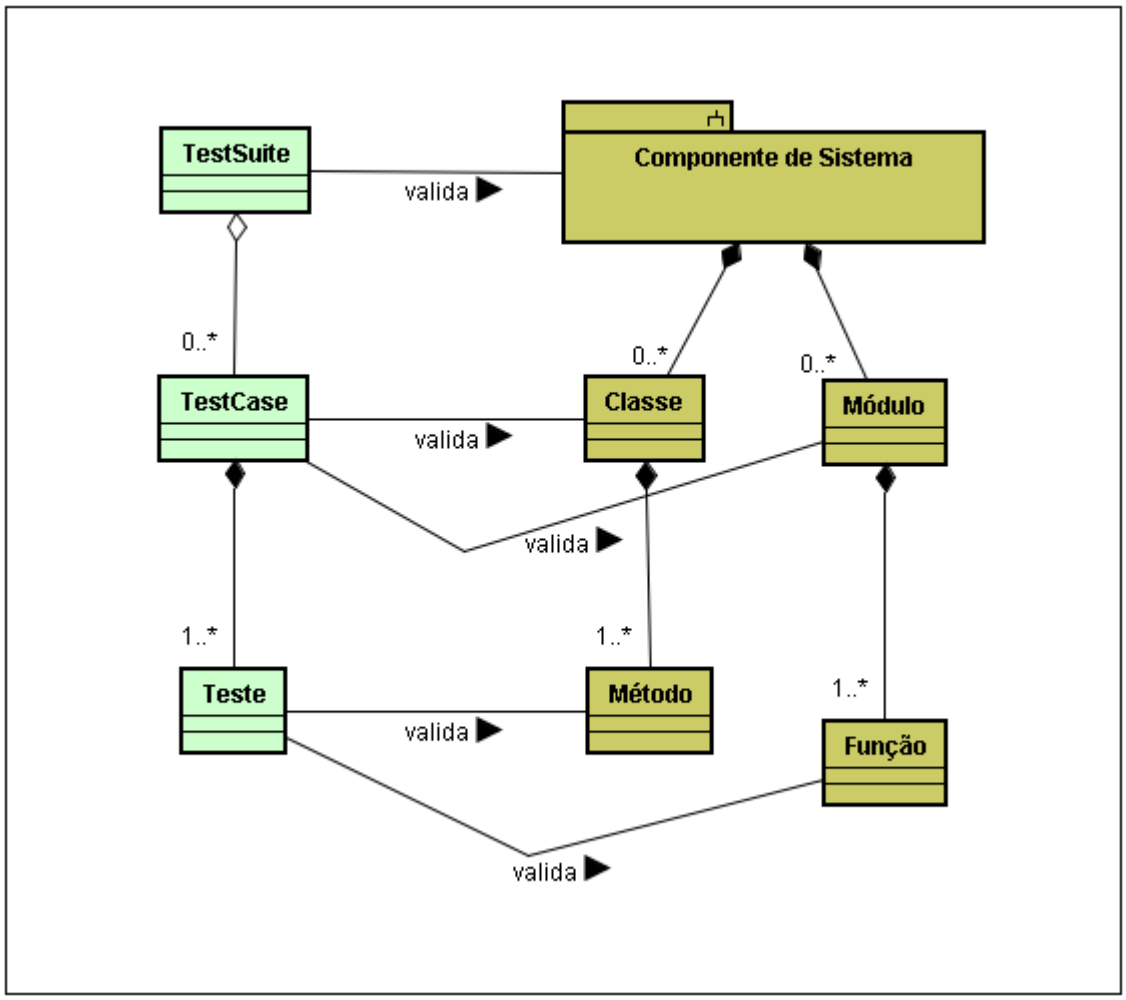


Figura 8 – Modelo tradicional de implementação de teste unitário
Fonte: Elaborado pelo autor

2.9 Teste de Regressão

O Teste de Regressão é realizado depois de fazer uma melhoria funcional ou reparação do sistema. Sua finalidade é determinar se a alteração tenha regredido outros aspectos do programa. Myers (2004) afirma que o Teste de Regressão é importante porque as mudanças e correções de erros tendem a ser mais propensos a erros do que o código do programa original.

Executado pelo grupo de desenvolvimento de testes, o Teste de Regressão verifica novamente os casos de testes aprovados em versões prévias do software, protegendo assim o sistema contra alterações indesejadas (PEZZE e YOUNG, 2008).

Executar todos os casos de teste para grandes produtos pode requerer muitas horas ou dias de execução, e pode depender de recursos escassos, como hardware de teste dispendioso (PEZZE e YOUNG, 2008). Devido aos ciclos de liberação rápida de software, no entanto, os profissionais muitas vezes têm pouco tempo para realizar testes de regressão (CHITTIMALLI e HARROLD, 2008).

Para melhorar a eficiência dos testes de regressão para que ele possa ser usado na prática para testar o software alterado foram desenvolvidas algumas técnicas para reduzir o seu custo.

As técnicas de seleção de casos de teste de regressão são baseadas ou no código ou na especificação. As técnicas baseadas em códigos selecionam um caso de teste para a execução se ele exercita uma porção do código que tenha sido modificada. Critérios baseados na especificação selecionam casos de teste para execução se for relevante para a porção da especificação que foi alterada (PEZZE e YOUNG, 2008).

Algumas destas técnicas, como a proposta por Chittimalli e Harrold (2008), utilizam uma matriz de rastreio de requisitos por cenário de testes para identificar os cenários de testes que precisam ser executados durante os testes de regressão.

2.10 Processos e Modelos de Desenvolvimento com Destaque em Testes

2.11 Modelo V de Desenvolvimento

O modelo V representa o ciclo de vida de desenvolvimento de software, segundo o modelo *Waterfall* ou cascata de desenvolvimento e seus respectivos artefatos de validação (RODRIGUES, 2006).

Apesar da similaridade com o modelo em cascata, o modelo V foi criado para reagir ao modelo em cascata, no qual os testes são encarados como uma etapa ao final do processo de desenvolvimento. A primeira mudança em relação ao modelo em cascata foi propor diferentes níveis de testes, um para cada fase de desenvolvimento (GOLDSMITH, 2002a).

De maneira diferente do modelo de desenvolvimento em cascata, no qual os testes são deixados para as fases final do desenvolvimento, o modelo V descreve os testes como uma atividade de grande importância no desenvolvimento de software,

sendo um dos mais conhecidos modelos de desenvolvimento voltado a testes (GOLDSMITH, 2002b).

O modelo V, ilustrado na Figura 9, destaca a existência dos testes para cada fase do desenvolvimento. Mesmo sendo executados apenas no final do processo de desenvolvimento, os testes são especificados dentro de cada fase, facilitando assim a validação e a compreensão dos artefatos produzidos.

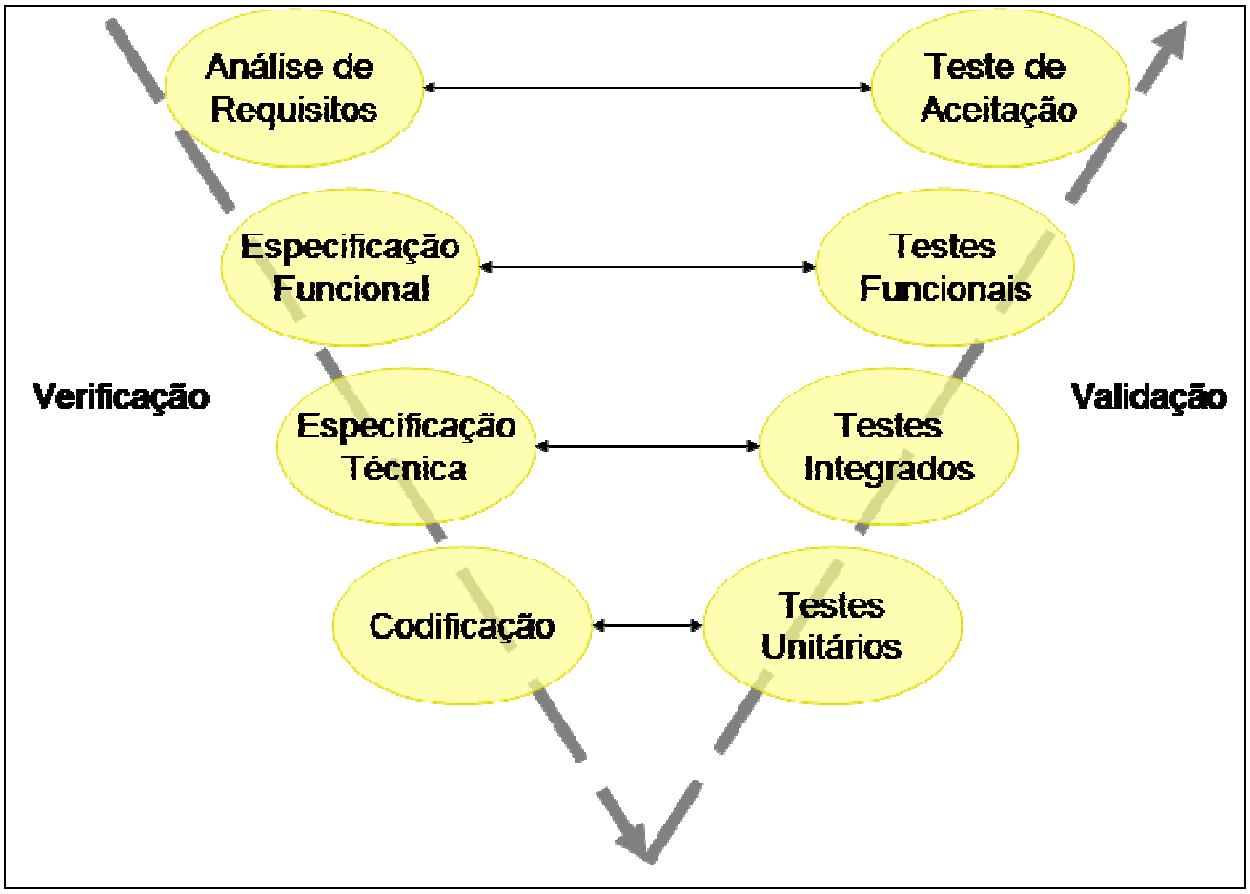


Figura 9 – Modelo V de Desenvolvimento
Fonte: Elaborado pelo autor

2.12 Desenvolvimento Ágil de Software

O termo Desenvolvimento Ágil de Software (DAS) é utilizado para definir um conjunto de metodologias de desenvolvimento de software baseado no desenvolvimento iterativo de software e nas orientações publicadas pelo Manifesto Ágil[1] (HANSSEN, 2007).

[1] Do inglês *Agile Manifesto (www.agilemanifesto.org).*

- Indivíduos e interações entre os mesmos em lugar de processos e ferramentas: enfatiza a auto-organização do time e a comunicação próxima e informal com o cliente;
- Software em funcionamento em lugar de documentação abrangente: o foco do desenvolvimento é o sistema funcionando; a prova de que o desenvolvimento está sendo efetuado é a possibilidade de se ter o sistema, mesmo que com parte das funcionalidades, disponível para utilização a qualquer momento. Apenas a documentação extremamente necessária é produzida;
- Colaboração com o cliente em lugar de negociação de contratos: o cliente faz parte do processo e é tão responsável pelo sucesso do desenvolvimento quanto os desenvolvedores. O cliente participa desde o início do processo com a definição dos requisitos até o final, verificando os resultados parciais e finais;
- Responder a mudanças em lugar de seguir um plano: o desenvolvimento é dinâmico assim como as necessidades do cliente. O planejamento é iterativo e mínimo, com foco na flexibilidade e na resposta a mudanças.

As orientações e os princípios publicados pelo Manifesto Ágil surgiram a partir de diversos métodos e práticas de desenvolvimento ágil, dentre eles o *Scrum*, a programação extrema e o desenvolvimento dirigido por testes.

2.13 Scrum

Scrum é um *framework* de programação interativa que utiliza tempos fixos de ciclos de desenvolvimento chamados *Sprints*. É formado por um time multidisciplinar com membros de diversas áreas, como, por exemplo, desenvolvedores, testadores, gerente de projeto, e assim por diante. (BUDWIG et al., 2009). É uma proposta de processo para gerenciamento ágil de projetos de software (SCHWABER, 2004).

Fundamentado na teoria de controle de processos empíricos, o *Scrum* emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar os riscos, sendo sustentado por três pilares: transparência, inspeção e adaptação (SCWABER e SUTHERLAND, 2010).

Sendo um dos pilares do *Scrum*, a inspeção é utilizada para rastrear o processo de desenvolvimento para que possam ser detectadas as variações inaceitáveis no processo. Existem três pontos para inspeção no *Scrum*: a reunião diária, que é utilizada para inspecionar o processo em direção à meta do *Sprint*; as reuniões de revisão do *Sprint* e de planejamento do *Sprint*, que são utilizadas para inspecionar o progresso em direção à meta do *Release*; e a retrospectiva do *Sprint*, que é utilizada para revisar o *Sprint* anterior.

O *framework Scrum* é baseado em um processo iterativo incremental ilustrado na Figura 10. O círculo inferior representa um *Sprint*, uma iteração de atividades de desenvolvimento de duração fixa que ocorre uma após a outra e na qual nenhuma mudança de escopo é permitida. A saída de cada iteração é um incremento do produto. O círculo superior representa a inspeção diária que ocorre durante a iteração, em que membros da equipe se reúnem para inspecionar cada uma das atividades dos outros e fazer as adaptações necessárias. Esse ciclo se repete até que o fim do projeto seja decretado (SCHWABER, 2004).

Bergström (2008) afirma que por conta dessas iterações curtas e reuniões diárias, a equipe saberá exatamente quais as tarefas que ainda precisam ser completadas, se os membros da equipe fizeram as tarefas que haviam sido escolhidas e se o time está progredindo para finalizar o *Sprint* de acordo com o que foi planejado na reunião de planejamento.

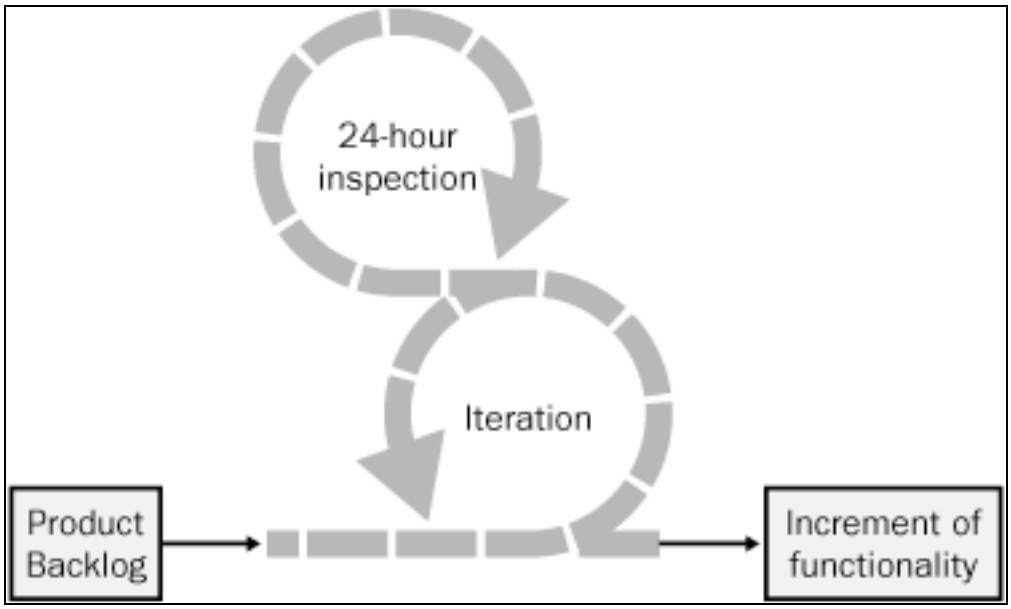


Figura 10 – Estrutura do *Framework* Scrum
Fonte: Extraído de Schwaber (2004)

O *Scrum* possui apenas quatro artefatos e três papéis: o Dono do Produto, a Equipe e o Scrum Máster. Todo o gerenciamento do projeto é dividido entre estes três papéis (SCHWABER, 2004).

O Dono do Produto é responsável por representar os interesses de todos os interessados no projeto e seu sistema resultante. É o único responsável pelo gerenciamento do *Backlog* do Produto e por garantir o trabalho realizado pela equipe (SCHWABER e SUTHERLAND, 2010).

O *Backlog* do Produto é uma lista priorizada de todos os requisitos, incluindo novas funcionalidades, funções, tecnologias, aprimoramentos e correções de defeito, conforme aponta Bergström (2008). Os itens do *Backlog* do Produto são geralmente representados por História de Usuário ou Casos de Uso. Testes de aceitação podem frequentemente substituir as descrições textuais mais detalhadas com uma descrição testável do que o item do *Backlog* do Produto deve fazer uma vez que esteja completo (SCHWABER e SUTHERLAND, 2010).

As Equipes Scrum são multidisciplinares, auto-organizadas e autogerenciadas, sendo elas as responsáveis por desenvolver os itens do *Backlog* do Produto priorizados para cada *Sprint*.

A Equipe, junto com o Dono do Produto, selecionam os itens do *BackLog* do Produto que serão implementados no próximo *Sprint*. A Equipe então identifica uma lista de tarefas necessárias para transformar os itens do *Backlog* do Produto em um incremento pronto. Esta lista de tarefas identificadas para o *Sprint* é chamada de *Backlog* do *Sprint*.

No final de cada *Sprint*, que dura aproximadamente de duas a quatro semanas, o Dono do Produto pode ver o resultado e obter uma demonstração de como funciona o sistema até aquele momento. Uma vez que os itens do *Backlog* do Produto e do *Sprint* são limitados, é fácil manter um rastreio dos novos recursos e mudanças que foram feitas nos diferentes *Sprints* (BERGSTRÖM, 2008).

Além do *Backlog* do Produto e do *Backlog* do *Sprint*, o *Scrum* produz outros dois artefatos: o *Burndown* do *Release* e o *Burndown* do *Sprint*.

O *Burndown* do *Release* é representado através de um gráfico e registra a soma das estimativas dos esforços restantes do *Backlog* do Produto ao longo do tempo. O esforço estimado deve estar em uma unidade de medida de trabalho

definido pela Equipe *Scrum*. A unidade de tempo utilizada geralmente são os *Sprints*.

O *Burndown* do *Sprint* representa a quantidade restante de trabalho do *Backlog* do *Sprint* ao longo do tempo do *Sprint*. Também representado em forma de gráfico, o *Burndown* do *Sprint* é utilizado para que a Equipe possa gerenciar seu progresso para completar o trabalho de um *Sprint*.

O *Scrum* Máster é responsável por garantir que todos da Equipe estejam aderindo aos valores, às regras e às praticas do *Scrum*. Ajudando a Equipe a entender como usar o autogerenciamento e a interdisciplinaridade, o *Scrum* Máster é um membro da Equipe que ajuda a realizar as mudanças necessárias para que a Equipe consiga otimizar o ambiente de desenvolvimento dentro da organização.

O *Scrum* utiliza a transparência e a inspeção contínua do processo para rastrear o desenvolvimento dos itens do *Backlog* do Produto dentro dos *Sprints* e, assim, minimizar os riscos (SCHWABER e SUTHERLAND, 2010).

A natureza emergente do *BackLog* do Produto, com foco apenas nos itens de alta prioridade, maximiza a probabilidade de que o investimento no detalhamento de requisitos é de valor.

E, por fim, a maneira como o *Scrum* aborda a implementação e a entrega dos requisitos por *Sprint* atende a uma das características do rastreamento de requisitos: assegurar a completeza da especificação para garantir que cada requisito tenha sido devidamente implementado e testado.

2.14 Programação Extrema (XP)

A Programação Extrema, ou *Extreme Programming*, propõe um conjunto de princípios para o desenvolvimento ágil de software. Dentre esses princípios temos: programação em pares, desenvolvimento dirigido por testes, projeto (*design)* simples, validação dos requisitos por testes de aceitação e planejamento contínuo a cada duas semanas (BECK e ANDRES, 2004). Utilizado geralmente por equipes pequenas de desenvolvimento, é um método de desenvolvimento que foca na interação entre os membros da equipe e o cliente.

Assim como nos outros modelos de desenvolvimento ágil, os princípios do XP surgiram a partir do fato de que a mudança é constante e, consequentemente,

promoveu práticas de desenvolvimento de software que apoiam mudanças frequentes de requisitos, planos e resultados. Como exemplo, o XP emprega o uso de técnicas de coleta de requisitos estruturados denominados Histórias de Usuário. Histórias de Usuário são as necessidades do cliente descritas através de funcionalidades e comportamentos do sistema (LEE et al., 2003). Algumas equipes podem agrupar histórias para identificar grupos de maior funcionalidade (SCHWABER, 2004).

Os requisitos, funcionalidades e histórias geralmente são escritos em cartões de índice. Quando a equipe tem a lista de necessidades definida, os cartões são fixados a uma parede. Os cartões podem ser organizados de alguma forma, por exemplo, por prioridade, risco, custo ou dependências. O principal objetivo dos cartões é o de servir como veículos para conduzir a discussão com o cliente sobre a funcionalidade do sistema e o comportamento esperado. Muitas vezes referida como "promessas de conversa", as histórias, periodicamente revisitadas pela equipe, trazem o foco de volta para as necessidades do cliente (LEE et al., 2003).

As histórias devem ser escritas e planejadas em unidades de funcionalidade. O cliente precisa entender o valor daquela história para que o planejamento das histórias que precisam ser implementadas seja orientado pelo custo-benefício de cada história. O XP não utiliza o termo requisito, pois um requisito é algo obrigatório; um sistema com apenas 20% de requisitos selecionados de maneira correta pode atender a todos os benefícios esperados pela construção do sistema (BECK e ANDRES, 2004).

Durante as conversas com o cliente, as Histórias de Usuário são traduzidas em Testes de Aceitação, nos quais o cliente especifica cenários capazes de testar quando uma história de usuário foi corretamente implementada (WELLS, 1999).

Nos modelos tradicionais de desenvolvimento, como ilustrado na Figura 11, o processo de desenvolvimento inicia-se na fase de requisitos e é finalizada na fase de integração e testes, em sequência e de uma só vez. Os artefatos produzidos na fase de análise de requisitos são passados para a fase de projeto, que produz os artefatos de projeto a serem implementados na fase de codificação para que, então, sejam integrados e testados na fase de testes de integração.

A Figura 12 ilustra o modelo sob demanda utilizado pelo XP. As histórias de usuário são detalhadas apenas momentos antes de serem implementadas. Os testes são criados a partir das especificações, o projeto dos componentes é feito para atender às necessidades do teste e a codificação é feita para atender aos testes e ao projeto das interfaces. O projeto das interfaces é então refinado durante a fase de codificação, a partir da qual se pode identificar qual a próxima história a ser detalhada.

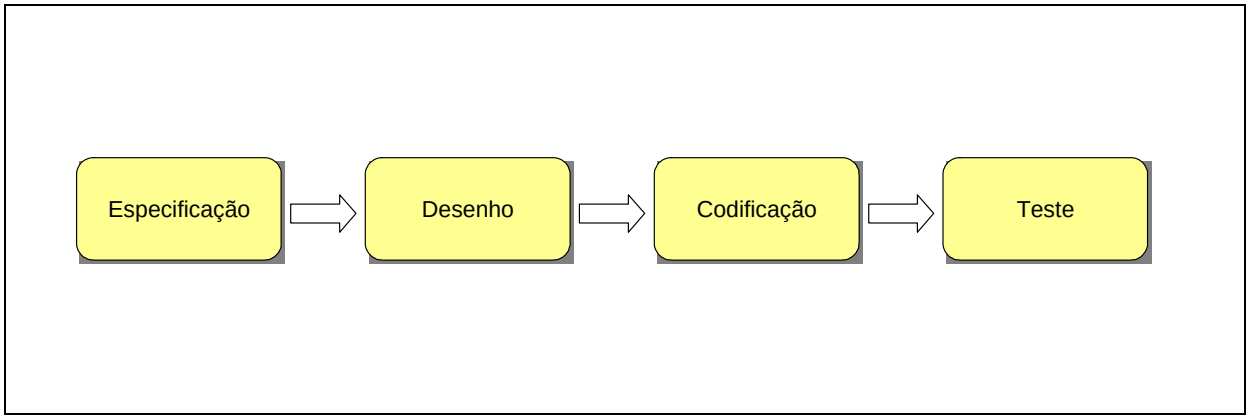


Figura 11 – Modelo de desenvolvimento tradicional
Fonte: Extraído de Beck e Andres (2004)

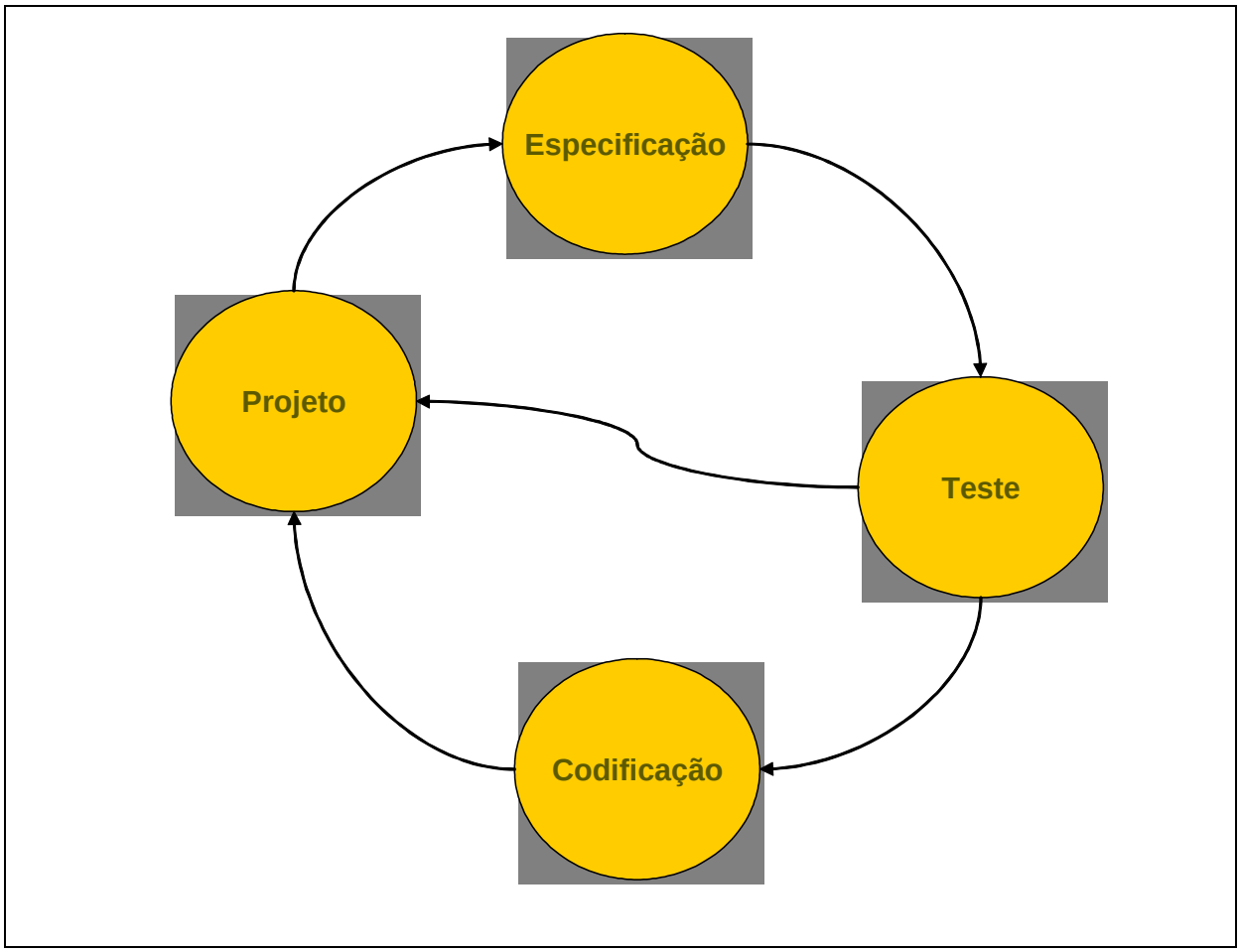


Figura 12 – Modelo XP de desenvolvimento
Fonte: Extraído de Beck e Andres (2004)

O princípio do desenvolvimento dirigido por testes traz a proposta de se escrever os casos de testes antes mesmo de se escrever o código fonte. Com essa prática, a equipe consegue produzir o código com 100% de cobertura de teste. A equipe se torna mais confiante, pois sabe que qualquer alteração será validada pelos testes durante a integração do código. Beck e Andres (2004) afirmam que o desenvolvimento dirigido por teste pode resolver os seguintes problemas do processo de codificação:

- Aumento de escopo: toda inserção de código é um potencial ponto de falha e, muitas vezes, o desenvolvedor é tentado a adicionar um pedaço de código prevendo que este pedaço poderá ser utilizado no futuro. Ao focar a codificação para resolver o problema de fazer o teste passar, evita-se que o desenvolvedor adicione código desnecessário para resolver o teste que está falhando;

- Acoplamento e coesão: se escrever testes se torna uma tarefa difícil, é um sinal de que há um problema de projeto. Então, é hora de refatorar para tornar o sistema capaz de diminuir o acoplamento e aumentar a coesão;
- Confiança: ao escrever um código utilizando os testes automatizados como justificativa, o desenvolvedor ganha a confiança da equipe e a equipe ganha confiança em relação ao software;
- Ritmo: o processo de escrever os testes, escrever o código, refatorar, testar, codificar, refatorar, e assim sucessivamente, dita o ritmo de desenvolvimento, fazendo com que o desenvolvedor trabalhe sempre em tarefas pequenas e contínuas.

A rastreabilidade, a capacidade de vincular o que mudou no sistema com o motivo da mudança, também é construída em XP, embora a informação não seja rotineiramente registrada. A única mudança necessário para implementar a rastreabilidade é fazer um registro físico de informações. Se uma linha de código está mudando, é porque um teste unitário está sendo escrito, o qual é uma parte do teste de nível de sistema. O teste de sistema foi criado a partir de uma história de usuário selecionada para ser implementada em uma determinada iteração e estava pronta para ser implantada no dia em que passou pelo teste de aceitação (BECK e ANDRES, 2004).

2.15 Desenvolvimento Dirigido por Testes (TDD)

O Desenvolvimento Dirigido por Testes, *Test Driven Development* (TDD) é uma prática de desenvolvimento ágil baseada em testes, em que estes são construídos antes dos artefatos a serem testados (HAYES et al., 2009). A construção do teste antes da construção do artefato a ser testado faz com que o desenvolvedor esteja constantemente tomando decisões de projeto e análise, sendo um processo em que o código fonte, os casos de teste e as decisões de arquitetura e construção evoluem simultaneamente.

Como todo método ágil, o TDD é baseado em pequenas e rápidas interações, em que, ao final de cada iteração, um conjunto de funcionalidades é entregue com seus respectivos testes de aceite e unitários.

As etapas do TDD são ilustradas na Figura 13. O primeiro passo é adicionar rapidamente um teste, pequeno o suficiente para falhar. Em seguida, o teste é executado para garantir que o teste criado realmente esteja falhando. O código é então criado para fazer com que o teste passe. A quarta etapa é executar os testes novamente para verificar se nenhum teste está falhando; a falha em algum teste implica em reescrever o código até que nenhum teste falhe. Por último, quando todos os testes passarem, deve-se fazer a refatoração do código para retirar as duplicações e deixar o código mais simples o possível (AMBLER, 2010).

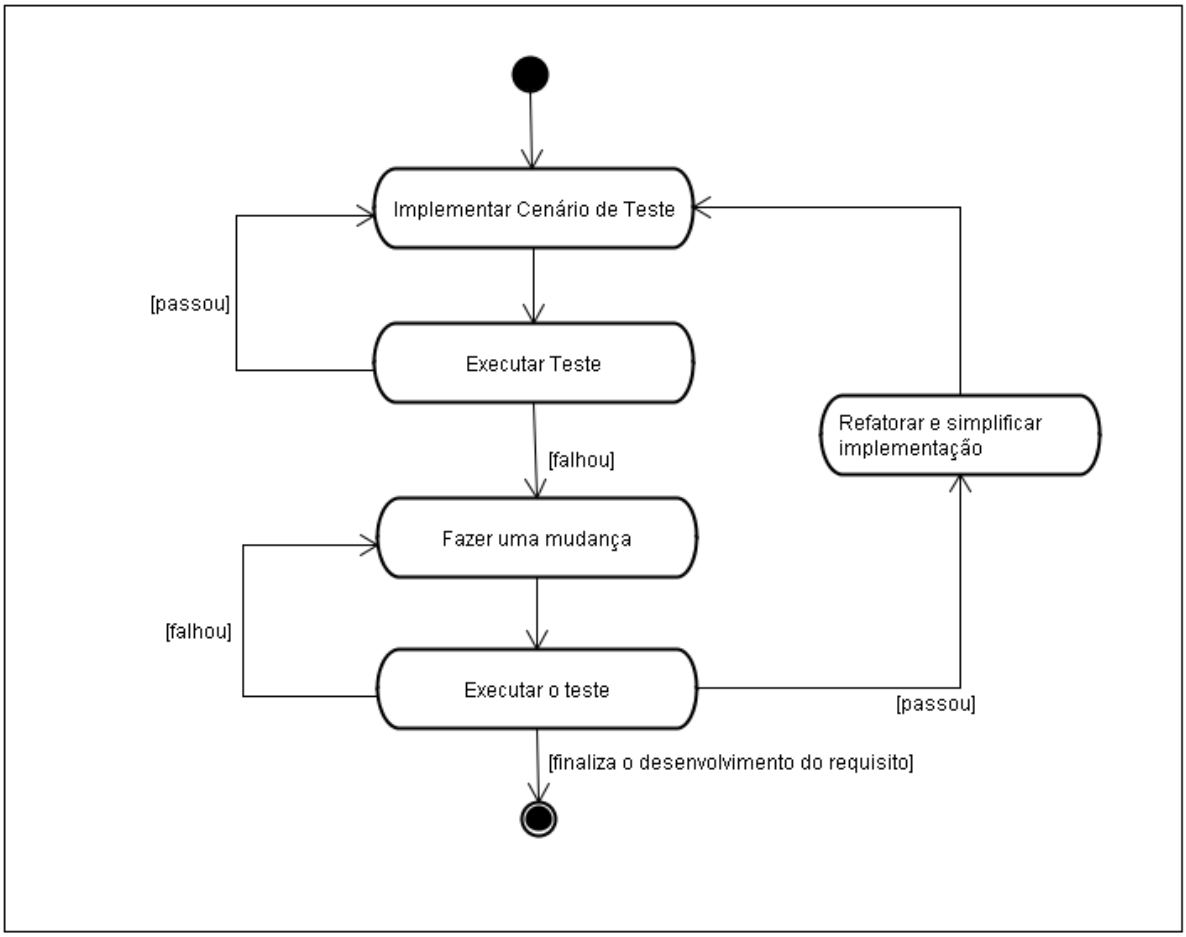


Figura 13 – Etapas do TDD
Fonte: Adaptado de Ambler (2010)

O TDD é uma prática disciplinada de desenvolvimento de software que enfoca o projeto de software, em que, através de pequenas e frequentes iterações, os testes unitários automatizados são escritos, seguidos então pelo código de produção (KENT, 2001 apud JANZEN, 2005).

Maximilien e Williams (2003) afirmam que a utilização do TDD reduziu a taxa de defeito em cerca de 50 por cento em comparação com um sistema semelhante que foi construído utilizando uma abordagem tradicional de teste de unidade. Afirmam ainda que o projeto foi executado dentro do prazo esperado com um mínimo impacto no desenvolvimento.

A rastreabilidade também pode ser construída em TDD. Uma linha de código é alterada em consequência da criação do teste unitário. Como o desenvolvedor nunca trabalha mais do que poucos minutos sem ter a certeza de que os testes continuam sendo executados com sucesso, a matriz de rastreabilidade pode ser automaticamente restabelecida ao final de cada execução dos testes automatizados.

2.16 Rastreabilidade no Desenvolvimento Ágil de Software

A rastreabilidade em ambientes de desenvolvimento ágil deve ser iniciada o mais cedo possível; ela não pode ser intrusiva para o time de desenvolvimento e deve ser obtida através de artefatos que já são naturalmente produzidos pela equipe (LEE et al., 2003). Da mesma maneira, Lee et al. (2003) propõem a utilização de ferramentas e rotinas para a obtenção da rastreabilidade sem a necessidade de se utilizar documentos extras para a rastreabilidade.

Lee et al. (2003) sugerem a criação de uma ferramenta capaz de levar em conta os princípios e práticas adotados pelos times de desenvolvimento ágil, em que todos os artefatos produzidos durante o desenvolvimento (Documentos de Requisito, Histórias de Usuários, Testes de Aceite e código fonte) são armazenados em um repositório comum, integrado a IDE de desenvolvimento. Qualquer atualização em um dos artefatos é anotada como uma possível quebra de rastreio, que é corrigida no momento da implementação do teste de aceite ou da codificação.

Já Jacobsson (2009) propõe pequenas mudanças nos artefatos produzidos durante o desenvolvimento, como a identificação do *stakeholder* que solicitou um requisito, detalhes sobre o problema que originou o requisito, adição do código do requisito aos cartões de histórias de cada funcionalidade, referência cruzada de requisitos, entre outros.

Apesar de ser uma proposta que apenas acrescenta novas informações nos artefatos comumente utilizados pelo *Scrum*, alguns destes novos atributos podem

trazer alguns custos ao projeto, custos estes que não compensam seus benefícios. Um estudo detalhado dos custos e benefícios destes atributos pode ser encontrado no trabalho de Ratanotayanon et al. (2009).

Ratanotayanon et al. (2009) afirmam que uma rastreabilidade, para suportar a compreensão do código fonte, pode ser uma grande oportunidade para os ambientes de desenvolvimento ágil, orientados a testes e centrados no código fonte. Conforme mostra a Figura 14, a simples ativação de uma história de usuário ou tarefa na criação dos casos de teste ou do código fonte pode prover a rastreabilidade do requisito e o aumento da compreensão do código fonte.

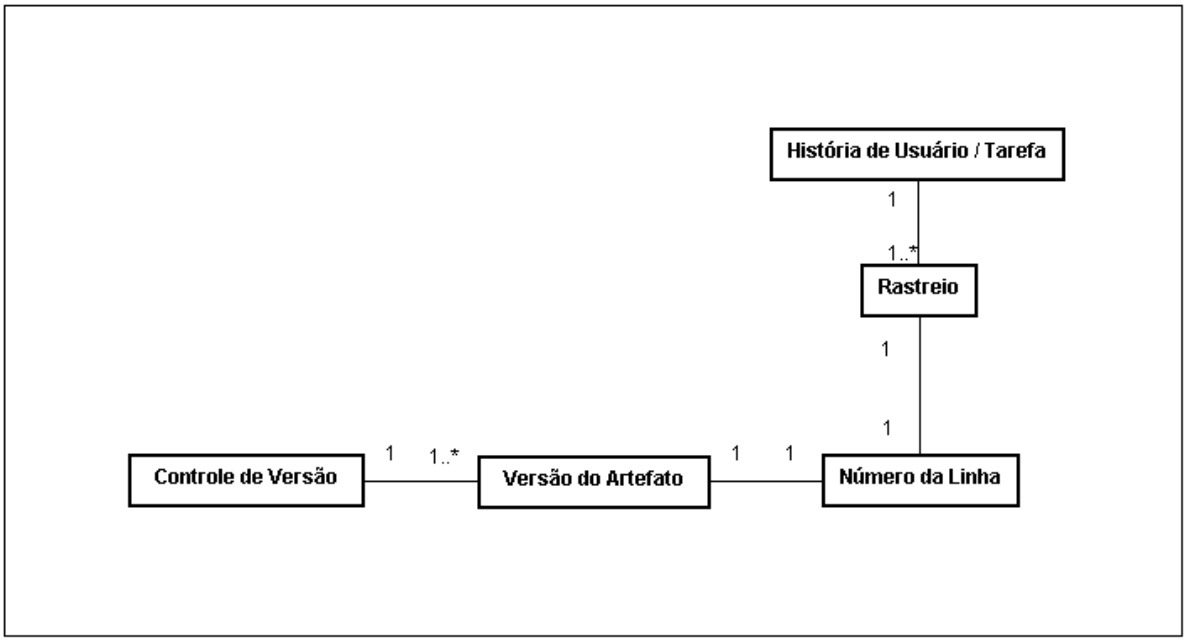


Figura 14 – Modelo de rastreio de história de usuário
Fonte: Adaptado de Ratanotayanon et al. (2009)

2.17 Resumo

A rastreabilidade de requisitos é um importante elemento arquitetural que ajuda na garantia da qualidade de software.

Apesar de ser bastante estudada em ambientes tradicionais de desenvolvimento de software, a rastreabilidade nem sempre é utilizada por seu alto custo de manutenção e pela falta de suporte ao rastreio de todos os artefatos gerados durante o processo de desenvolvimento.

O teste de software utilizado em todas as etapas do desenvolvimento de software permite identificar defeitos nas estruturas internas do software, para garantir que as funcionalidades desenvolvidas estejam de acordo com os requisitos do sistema e para verificar se o sistema está sendo construído de acordo com os requisitos não funcionais.

Foram também detalhados neste capítulo alguns modelos de desenvolvimento que utilizam os testes durante o processo de desenvolvimento, com destaque para o TDD e o modelo V de desenvolvimento. Enquanto o TDD é o modelo mais utilizado pelos desenvolvedores de sistemas, o modelo V destaca a importância de se utilizar os testes em todas as etapas do processo de desenvolvimento.

Conclui-se que o teste é o artefato que está presente em todas as etapas de desenvolvimento e, se utilizado para dirigir o processo de desenvolvimento, torna-se um importante elemento de rastreio.

3 DESENVOLVIMENTO DIRIGIDO E RASTREADO POR TESTES - DDRT

Neste capítulo, apresenta-se a proposta do presente trabalho, que é um método para utilizar os artefatos de testes como elementos de rastreio de requisitos durante o processo de desenvolvimento de software. Propõe-se estender o modelo de Desenvolvimento Dirigido por Testes (TDD), em que os casos de testes dirigem o desenvolvimento do sistema, para criar o Desenvolvimento Dirigido e Rastreado por Testes (DDRT), técnica proposta por este trabalho.

O teste é um artefato arquitetural presente nos modelos tradicionais de desenvolvimento de software e também muito utilizado em ambientes de desenvolvimento ágil.

Nos modelos de desenvolvimento tradicionais, os testes aparecem em todas as etapas do desenvolvimento, ora como artefatos de validação, ora como artefatos de verificação.

Já em ambientes de desenvolvimento ágil, os testes são utilizados como artefatos arquiteturais para a elaboração dos requisitos e também na construção do código fonte. Neste modelo, os testes são construídos em uma colaboração entre desenvolvedores e *stakeholders* e, geralmente, são automatizados por ferramentas, permitindo que sejam executados constantemente.

Por esta razão, os testes de aceitação são um tipo de artefato que se mantém atualizado, refletindo o estado atual do sistema e, como consequência, o código fonte. Trata-se, portanto, de um artefato que não se perde e que não se torna desatualizado, conforme mostra a Figura 15.

No modelo de desenvolvimento ágil, como o TDD - *Test Driven Development* - , o grande objetivo dos testes é garantir de forma automatizada que:

1. Todos os requisitos foram implementados;
2. A implementação de novos requisitos não gerou impacto em requisitos já implementados.

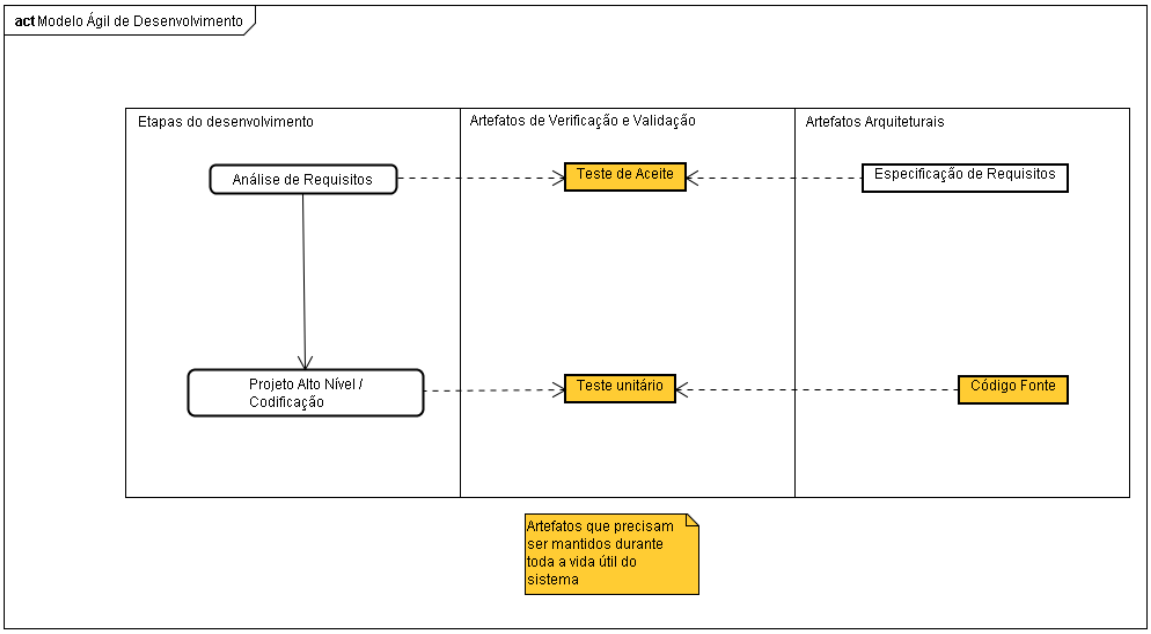


Figura 15 – Modelo ágil de desenvolvimento
Fonte: Elaborado pelo autor

No DDRT, os testes são utilizados para refletir os requisitos (funcionais e não funcionais) no código fonte. Conforme ilustra a Figura 16, os testes de aceite, funcionais e não funcionais (arquiteturais) são elaborados para refletir os requisitos do usuário, os requisitos de sistema e os requisitos arquiteturais no código fonte.

Cada requisito é validado com o usuário através do teste de aceite. Os testes de aceite são então complementados com os testes funcionais que são utilizados para validar que o sistema não faz aquilo que não foi solicitado para fazer. Os testes arquiteturais são utilizados para validar os requisitos não funcionais, qualidades e restrições de implementação.

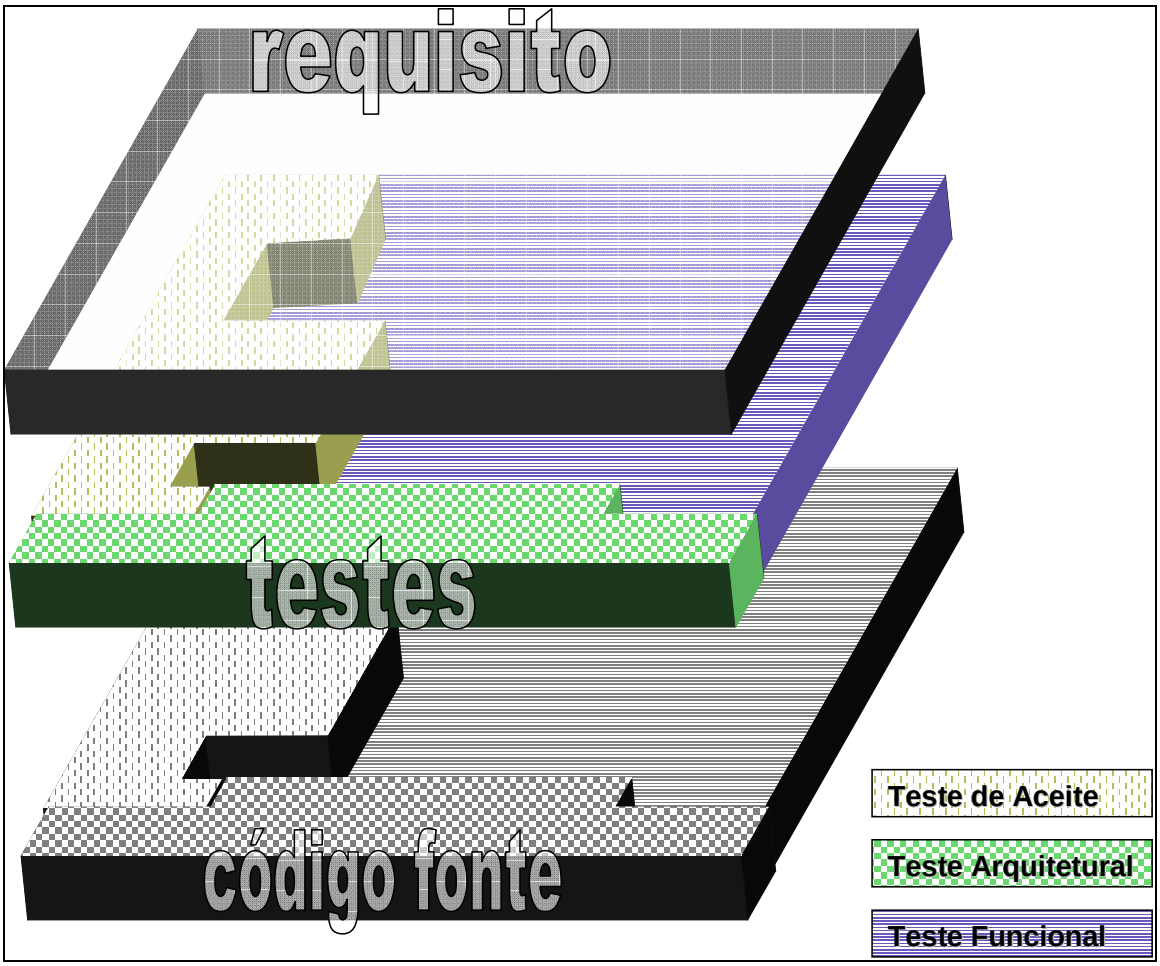


Figura 16 – Desenvolvimento Dirigido e Rastreado por Testes (DDRT)
Fonte: Elaborado pelo autor

A proposta desta técnica é dirigir todo o processo de desenvolvimento através de cenários de testes, os quais serão visualizados no código fonte através dos testes unitários.

A proposta geral do rastreio dirigido por testes é ilustrada na Figura 17. Todo o processo é dirigido pelos cenários de teste. A funcionalidade descrita através do teste de aceitação e pelo teste funcional é complementada com os testes de arquitetural. O *TestSuite* é o elo de ligação para rastrear os componentes de software necessários para a funcionalidade, validando se os requisitos do usuário, os requisitos de sistema e as decisões de arquitetura do projeto foram implementadas para a funcionalidade.

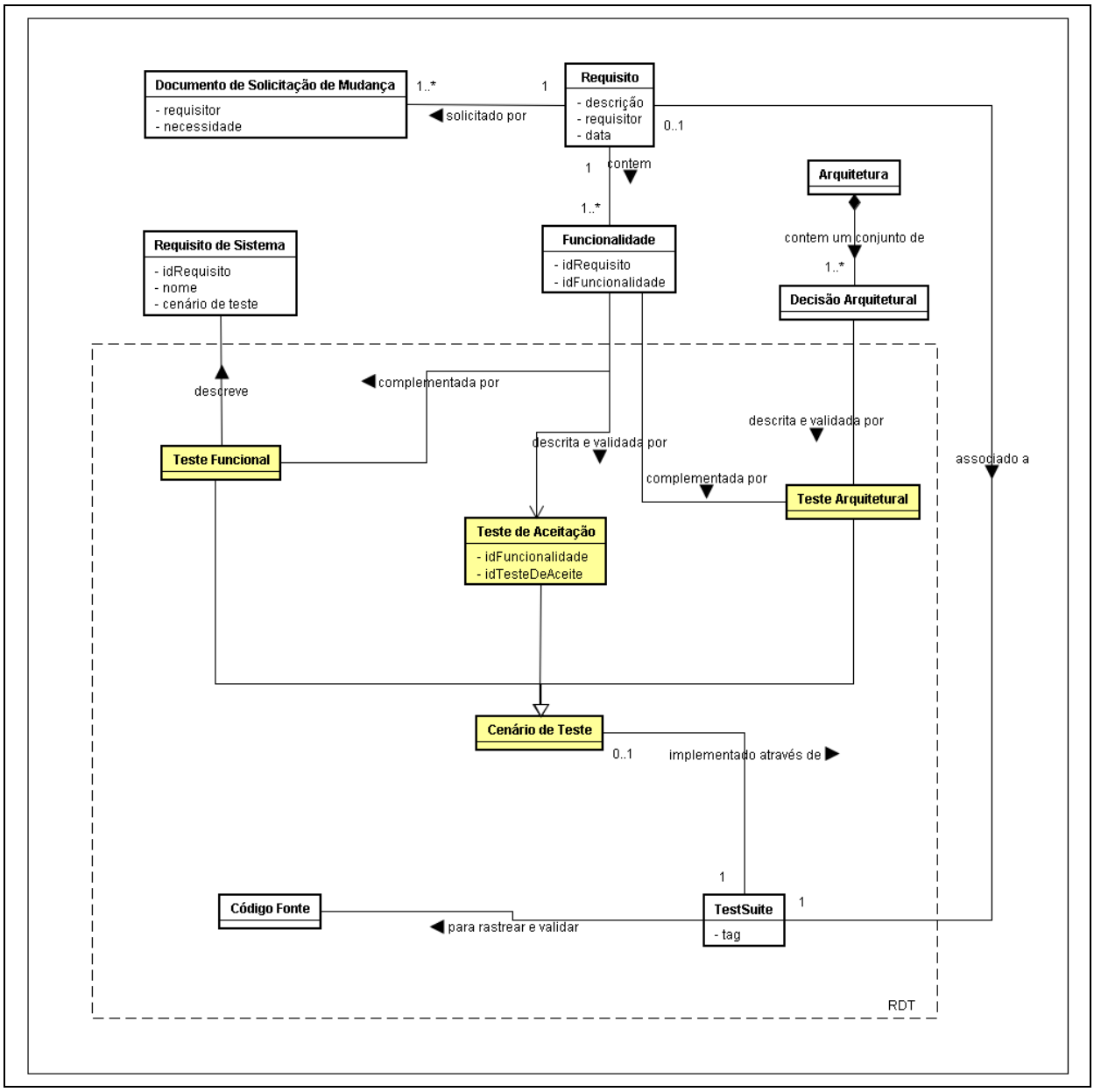


Figura 17 – Modelo geral do rastreio dirigido por testes
Fonte: Elaborado pelo autor

Conforme ilustra a Figura 18, os cenários de teste de aceitação são criados para descrever e validar os requisitos, os testes funcionais complementam os testes de aceitação, clarificando como o sistema deve se comportar em cenários de exceções e limítrofes, a interação interna entre os componentes é especificada através dos cenários de testes de integração ou arquiteturais, que também são utilizados para testar características não funcionais. Todos os cenários de testes criados em cada uma das fases são então implementados utilizando-se da técnica TDD.

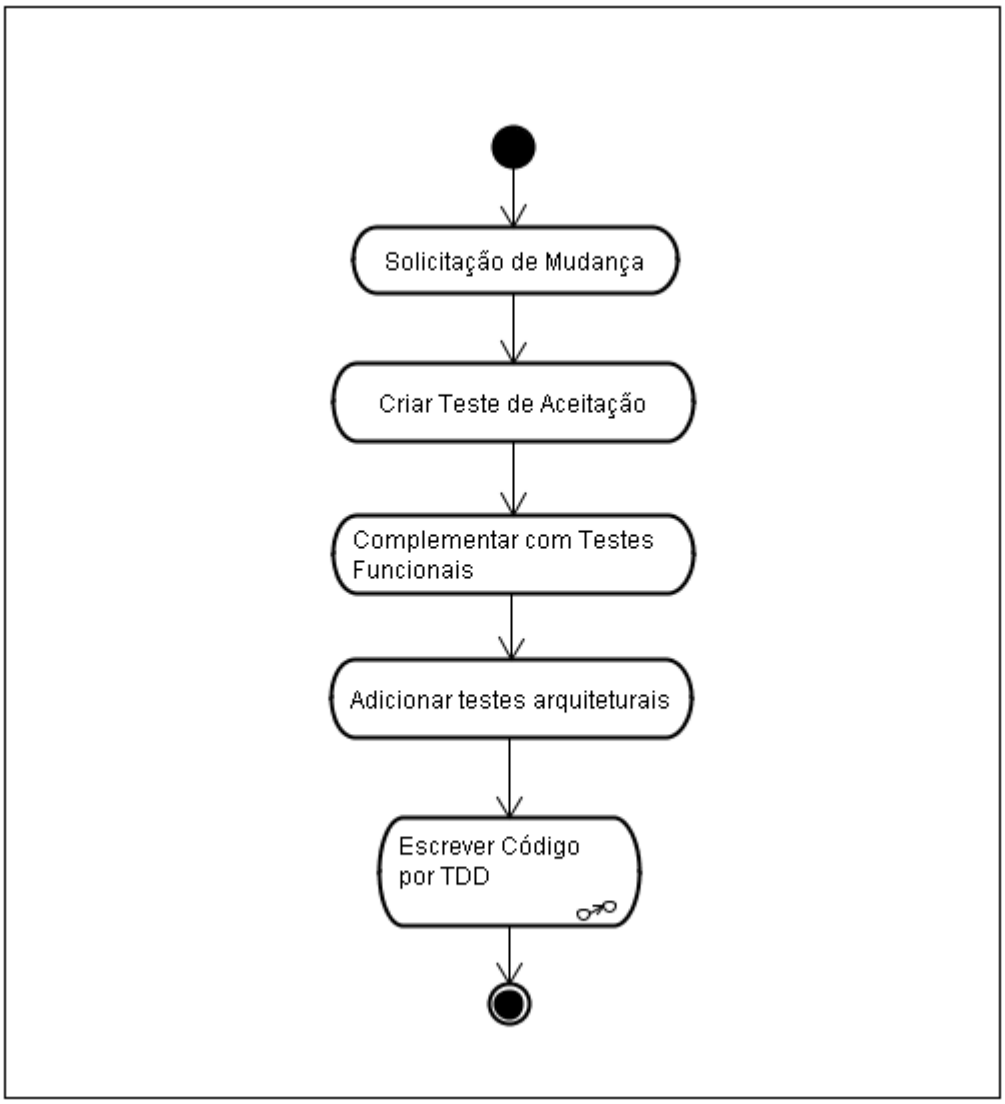


Figura 18 – Rastreio dirigido por testes
Fonte: Elaborado pelo autor

Os cenários de testes podem ser especificados utilizando-se os Casos de Teste de Uso ou através das tabelas FIT apresentadas no Capítulo 2. O importante é escrever todas as especificações na forma de teste para que o desenvolvedor possa implementá-las através dos testes unitários.

Nas próximas seções deste capítulo são apresentadas as etapas para se obter o rastreio dirigido por testes. São apresentados os testes de aceite, funcional e arquitetural e os meios de refleti-los no código fonte utilizando-se como exemplo uma aplicação de conversor de moeda[2].

[2] Aplicação utilizada por Beck (2002) para apresentar o TDD.

No conversor de moeda proposto por Beck (2002), o requisito para gerar o relatório da carteira de ações foi alterado para que ele suporte a exibição de ações das companhias negociadas em múltiplas moedas. Conforme ilustrado na Figura 19, o relatório deve exibir cada item com a sua respectiva moeda de negociação. O total geral deve ser exibido de acordo com a moeda local configurada por uma taxa de conversão de câmbio.

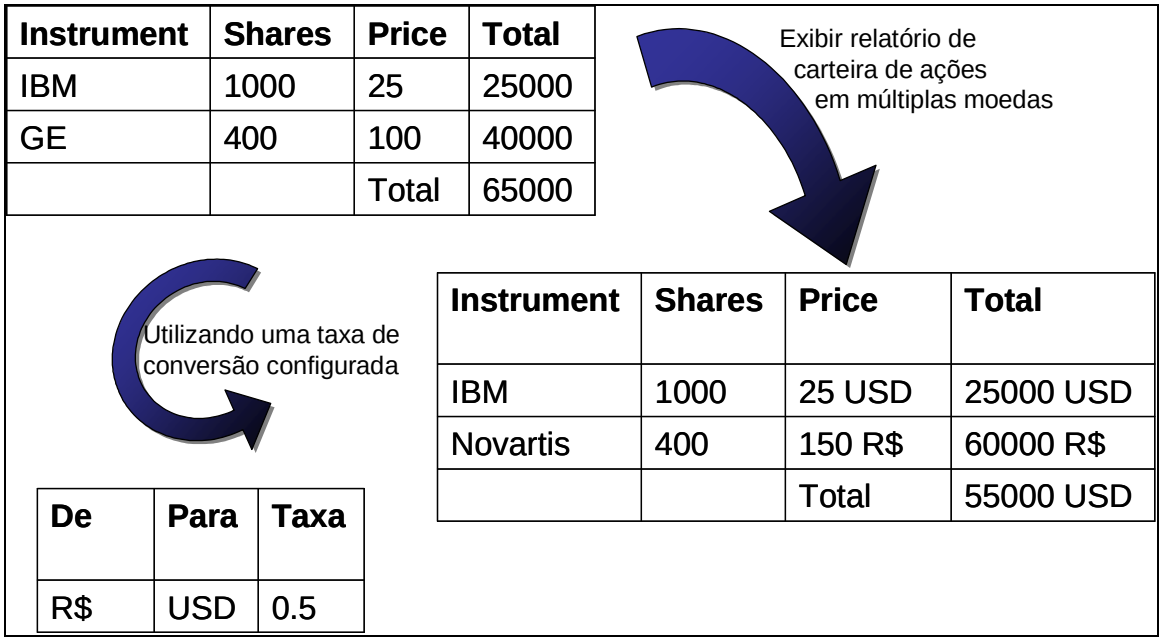


Figura 19 – Relatório de carteira em múltiplas moedas
Fonte: Adaptado de Beck (2002)

3.1 Rastreio de Requisitos pela Especificação Dirigida por Testes

Independentemente do modelo de desenvolvimento adotado – cascata ou XP - a validação do sistema é feita pelo usuário, o qual executa um conjunto de procedimentos para verificar e validar se o sistema apresenta aquilo que foi requisitado.

Um dos principais meios de validar se o sistema atende ao que foi requisitado é através da execução de um conjunto de testes chamados de teste de aceitação. Se implementados e executados apenas na fase final do desenvolvimento, os testes de aceitação atuam somente como um artefato de controle de qualidade, indicando se o sistema pode ser entregue ao usuário e sob quais condições.

Para que os requisitos do usuário possam ser rastreados através dos testes, no DDRT, os testes de aceitação são criados durante a elaboração dos requisitos da

mesma maneira que os testes unitários são criados durante a codificação do sistema no TDD.

Os testes de aceitação são testes funcionais realizados para validar que o sistema se comporta como o esperado, em que estímulos ao sistema produzem resultados que são validados por comportamentos ou dados de saída esperados.

Para facilitar o entendimento e a verificação dos requisitos e também facilitar o rastreio no processo do desenvolvimento, os requisitos devem ser divididos em subtarefas. A divisão em subtarefas facilita a validação da implementação através dos testes unitários, além de aumentar a velocidade da implementação por estimular e permitir a codificação em paralelo por mais de um desenvolvedor.

A Figura 20 ilustra a divisão dos requisitos em subtarefas e os testes relacionados. Cada requisito está associado a um documento de solicitação de mudança, que contém informações sobre o requerente, assim como as necessidades de negócio que motivaram esta solicitação. Os requisitos são então divididos em funcionalidades, as quais são detalhadas pelos testes de aceitação.

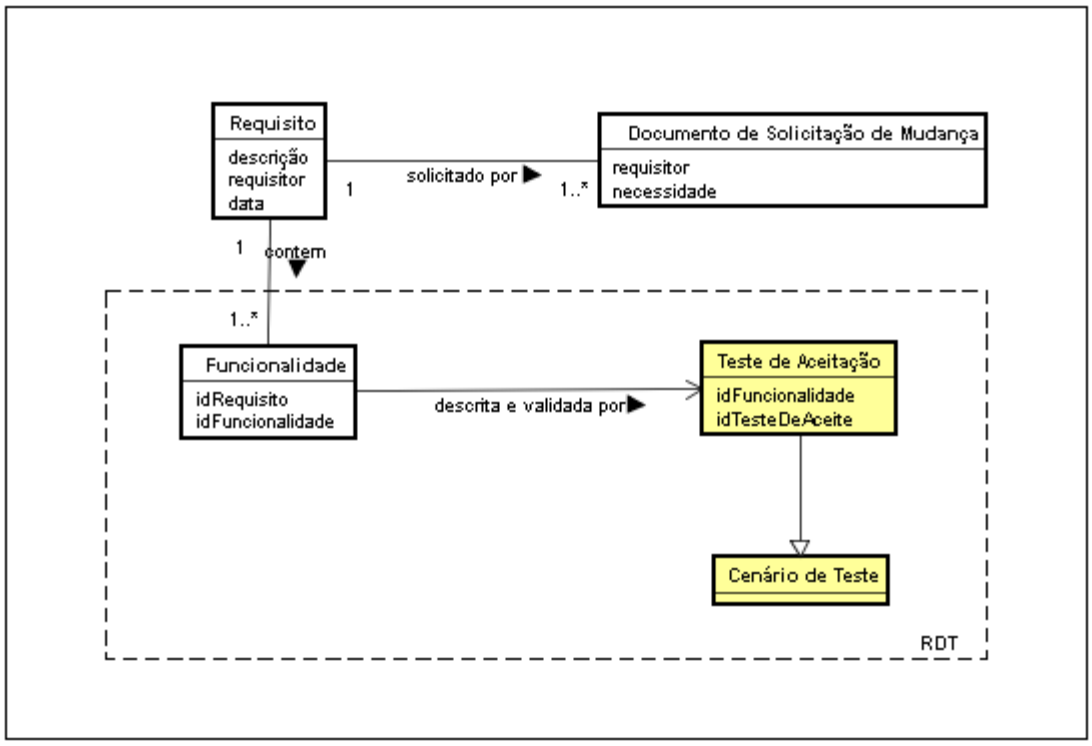


Figura 20 – Rastreio de funcionalidade por testes de aceitação
Fonte: Elaborado pelo autor

Sendo o elemento de rastreio mais próximo do requisito do usuário, o teste de aceitação é utilizado para descrever o requisito em formato de testes e assim validar a funcionalidade a ser implementada. É uma tarefa feita em colaboração com o usuário para validar especificar e aumentar a compreensão dos requisitos.

Quando utilizados como elementos de verificação, no final do processo de desenvolvimento, os testes de aceitação identificam os requisitos que não foram implementados na versão do sistema sendo liberada. A identificação dos requisitos ainda não implementados é um dos atributos da rastreabilidade de requisitos.

O teste de aceitação, da maneira como é proposto por este trabalho, está presente em dois diferentes momentos no processo de desenvolvimento: na fase inicial, durante a clarificação dos requisitos, e na fase final, durante os testes de aceitação.

Um importante benefício desta abordagem é que este mesmo teste, que foi criado pelo usuário em conjunto com os analistas de sistema na fase inicial para descrever o requisito, será utilizado na fase final do processo para verificar se o sistema atende ao que foi solicitado. Portanto, o mesmo artefato que solicita a mudança é utilizado para verificar o produto final.

Para fins ilustrativos, a Tabela 3 apresenta os testes de aceitação para o Requisito “Relatório de Carteira de Ações”.

Teste de Aceitação:	**Relatório de Carteira de Ações**				
Funcionalidade:	**Relatório com ações em múltiplas moedas**				
Requisito:	**1.0**				
Teste	**Identificar valores com a moeda de negociação**				
1. Adicione as ações					
	Adicionar()	Ação	Share	Price	Currency
	true	IBM	100	25	USD
	true	GE	400	100	R$
2. As ações devem ser exibidas com as respectivas moeda de negociação					
	ListaDeAçõesDaCarteira				
	Ação	Share	Price	Currency	
	IBM	100	25	USD	
	GE	400	100	R$	
Teste	**O total por ação deve conter a moeda**				
1. Adicione as ações					
	Adicionar()	Ação	Share	Price	Currency
	true	IBM	100	25	USD
	true	GE	400	100	R$
2. O total por ação deve conter a moeda da negociação					
	ListaTotalPorAção				

	Ação	Total		Currency	
	IBM	2500		USD	
	GE	60000		R$	
Teste	**Exibir total geral de acordo com a moeda local**				
1. Adicione as ações					
	Adicionar()	Ação	Share	Price	Currency
	true	IBM	100	25	USD
	true	GE	400	100	R$
2. Definir a moeda local					
	DefinirMoedaLocal()		moeda		
	true		USD		
3. Definir taxa de conversão					
	DefinirTaxa()	De	Para	Taxa	
		R$	USD	0.5	
3. Obter Total da carteira em moeda local					
			Valor	Moeda	
	obter	totalDaCarteira	55000	USD	

Tabela 3 – Teste de Aceitação: Relatórios de Carteira de Ações
Fonte: Elaborado pelo autor

Os testes de aceitação são completados pelos testes funcionais, como ilustrado na Tabela 4. Ainda que elaborados com base nos requisitos, os testes funcionais são utilizados para validar casos de exceção. São estes testes que validarão se o sistema não faz nada além do que foi requisitado.

Por serem voltados mais para como o sistema soluciona o requisito, os testes funcionais ajudam na identificação dos requisitos conflitantes e relacionados.

Teste Funcional:	**Relatório de Carteira de Ações**				
Funcionalidade:	**Relatório com ações em múltiplas moedas**				
Requisito:	**1.0**				
Teste	**Adicionar ação em moeda não suportada pelo Sistema**				
1. Adicionar ação					
	Adicionar()	Ação	Share	Price	Currency
	False	IBM	100	25	Y

Tabela 4 – Teste Funcional: Relatórios de Carteira de Ações
Fonte: Elaborado pelo autor

Os testes criados são identificados pela versão do sistema e são submetidos à ferramenta de controle de versão para que possam ser rastreados durante a evolução do sistema.

De maneira semelhante às atividades descritas pelo TDD, a mudança de um requisito gera um efeito em cascata. Ao mudar um requisito, o usuário e o analista de teste criam um novo cenário de teste ou alteram um cenário de teste existente. A criação de um novo cenário de teste faz com que novos testes funcionais sejam criados. Os cenários de testes de aceitação e funcional precisam então ser

implementados no código fonte, o que gera uma solicitação de mudança no sistema e, com isso, o rastreio é restabelecido.

3.2 Rastreio de Decisões Arquiteturais

A terceira etapa do processo é a criação dos cenários de testes que serão utilizados para rastrear os componentes e as decisões arquiteturais. Conhecidos também como testes estruturais, eles validam como a estrutura interna dos componentes foi implementada.

Tendo como entrada os cenários de testes de aceitação e funcional, o arquiteto do sistema, em conjunto com os desenvolvedores e o analista de negócios, criam cenários para testar a integração dos componentes internos. Nesta fase do processo, o foco é verificar como os componentes interagirão validando as entradas e saídas de cada componente.

Os testes arquiteturais identificam os subsistemas e componentes afetados pelo requisito e validam todo o ciclo de informação - uma sequência de entradas e saídas entre os componentes de software.

No processo de análise, o arquiteto identifica os cenários de testes necessários para validar a implementação dos interesses arquiteturais associados aos requisitos sendo implementados. No DDRT, os padrões de projeto selecionados para implementação de cada interesse arquitetural são indicados na própria história de usuário, por meio de cenários de testes arquiteturais. O método de escolha das táticas está fora do escopo deste trabalho, no qual se considera apenas a necessidade de se criar cenários de testes para validação das características arquiteturais.

Na elaboração do projeto da arquitetura, o arquiteto de software deve projetar também a estrutura dos testes para validar as decisões de arquitetura dentro de cada requisito sendo implementado para que o software possa atingir os objetivos de qualidade especificados.

Ao se implementar um componente de infraestrutura, o desenvolvedor, além de criar os testes unitários para validar o componente, deve também criar os *fake*

objects ou *mock objects* que serão utilizados para validar a característica não funcional ou arquitetural de uma determinada funcionalidade sendo implementada.

A arquitetura de um software decorre de múltiplos interesses, boa parte deles associados às funções de infraestrutura (persistência, geração de *logs*, tratamento de exceções, dentre outros). A forma como os interesses arquiteturais são especificados varia em função dos atributos de qualidade a que o software precisa atender.

Nesta fase do processo também são criados cenários de teste para avaliar características não funcionais, como facilidade de manutenção e facilidade de identificação de problemas em ambientes de produção. Estes cenários de testes ajudam a rastrear as implementações de código não relacionadas a requisitos funcionais.

Os testes arquiteturais são implementados por utilitários que, através de *fake objects* ou *mock objects*, validam os requisitos não funcionais e as decisões arquiteturais.

Da mesma forma que os requisitos funcionais, as características arquiteturais são também rastreadas através de testes unitários. Para tanto, conforme ilustra a Figura 21, cada característica arquitetural é representada por um *TestCase* abstrato, que é especializado e implementado durante a validação da característica arquitetural dentro de um requisito funcional.

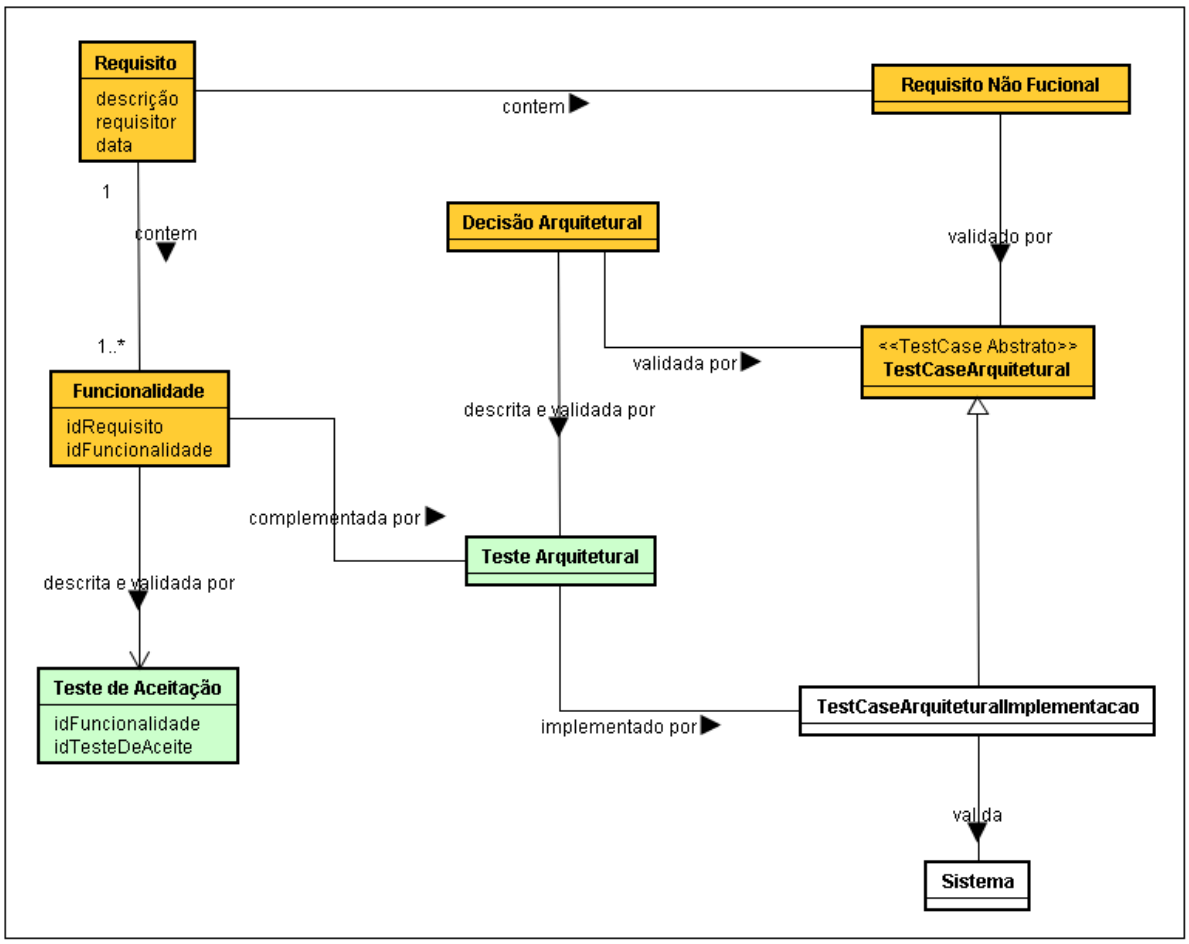


Figura 21 – Rastreio de decisões arquiteturais
Fonte: Elaborado pelo autor

Assim como no teste de aceitação, o teste arquitetural, da maneira como é proposto por este trabalho, está presente em dois diferentes momentos no processo de desenvolvimento: na fase inicial, durante a análise da arquitetura, e na fase de implementação, durante a implementação dos testes unitários.

3.3 Rastreio de Código Fonte através de Testes Unitários

Os testes unitários são implementados pelos próprios desenvolvedores na mesma linguagem de programação utilizada para desenvolver os módulos do sistema. Eles são utilizados para fazer o *link* de rastreio entre o teste de aceite e o código fonte criado para atender aos testes de aceite. Também conhecidos como testes estruturais, os testes unitários são o elemento de rastreio do desenvolvedor por proporcionar:

- Maior compreensão do código fonte;
- Interligação da implementação ao requisito de usuário;
- Identificação dos requisitos relacionados;
- Validação dos requisitos implementados.

Para se ter uma maior compreensão do código fonte e entender a razão pela qual aquela parte do código foi escrita, uma ligação entre o código fonte e o requisito de usuário precisa ser estabelecida.

Os testes unitários completam a rastreabilidade do código proporcionada pela documentação das funções, classes e variáveis, adicionando compreensão dinâmica do código. O aumento da compreensão do código fonte se dá principalmente pela forma como os testes unitários são criados. O nome do teste, o nome da classe que implementa o teste, assim como a documentação associada ao teste unitário são de grande importância para que o teste unitário seja utilizado como elemento de rastreio.

Por ser um elemento de rastreio, o teste unitário precisa conter conexão com o artefato de origem (artefato pai), conexão com o artefato de destino (artefato filho) e versão.

O teste unitário valida se as funções, classes e módulos foram implementados de forma correta para atender um requisito de negócio identificado aqui pelos testes de aceite. Portanto, como mostra a Figura 22, o artefato de rastreio teste unitário tem como artefato de origem ou artefato pai o cenário de teste, e como artefato de destino ou artefato filho as funções, as classes e os módulos.

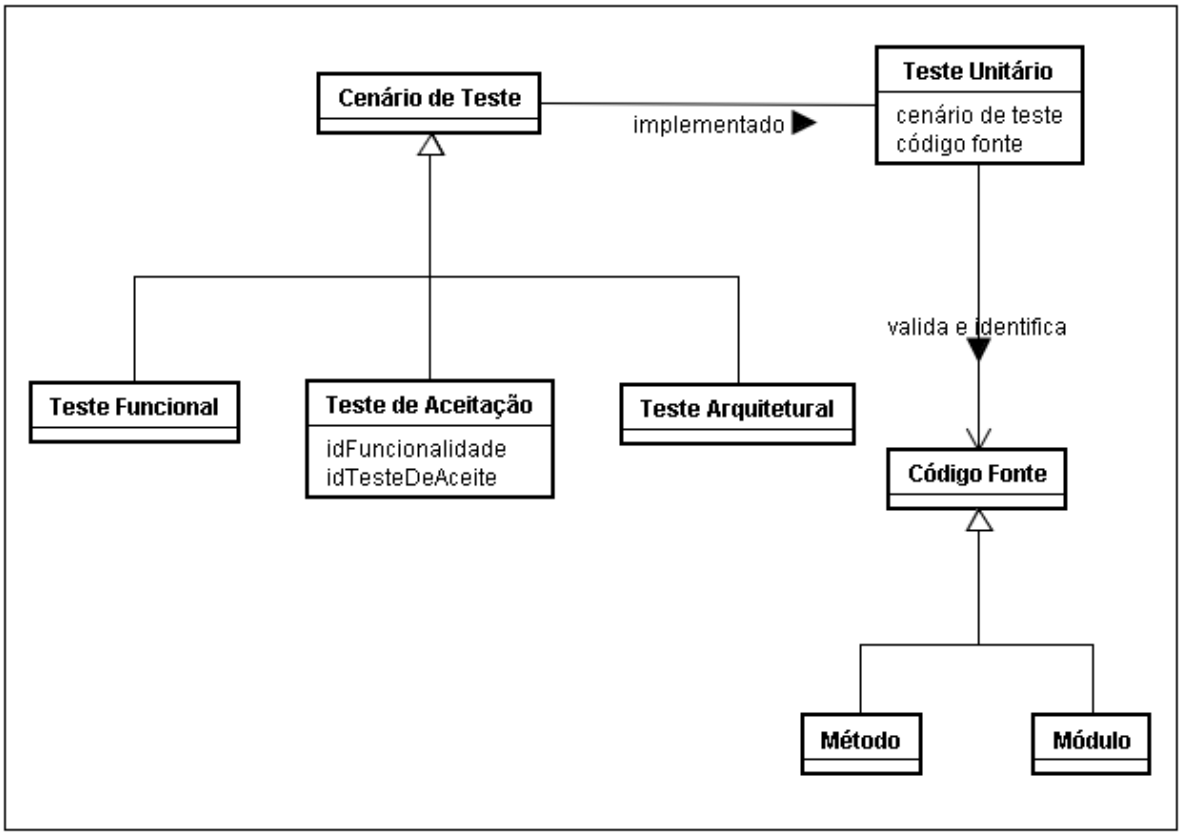


Figura 22 – Teste unitário como elemento de rastreio
Fonte: Elaborado pelo autor

O motivo é a propriedade de rastreio que ajuda na compreensão do artefato sendo rastreado. Para o teste unitário, é a informação que identifica a razão pela qual aquele código fonte foi criado ou a razão pela qual ele está sendo testado.

Em um teste unitário, o atributo de rastreio motivo é compreendido por:

- Nome e documentação do caso de teste;
- Nome e documentação da função de teste;
- Codificação do teste;
- Validação do código fonte sendo testado.

Geralmente, o *TestSuite* agrupa um conjunto de testes relacionados a um componente do sistema. Neste trabalho, o *TestSuite* atuará como um elemento de ligação entre os testes unitários e os cenários de teste. Cada cenário de teste definido nos Testes de Aceite terá um *TestSuite* que conterá um conjunto de *TestCases* capaz de validar de maneira mais atômica todas as classes, as funções e

os módulos necessários para a implementação do requisito. Assim como descrito pelo xUnit, o *TestCase* estará associado a um *TextFixture*.

Neste trabalho, os *TestSuites* e os *TestCases* não estão relacionados com os componentes de software, e sim com os cenários de teste. A Figura 23 ilustra o modelo de implementação de teste unitário proposto neste trabalho.

O desenvolvedor sabe, a qualquer momento do processo de desenvolvimento, quais os requisitos de usuário que uma determinada função, classe ou módulo está atendendo e como ela está sendo validada (testes de aceite relacionados). Da mesma maneira, os Usuários, Gerentes de Projetos, Testadores e Analistas de Negócio conseguem saber o impacto que uma mudança na funcionalidade ou no requisito pode causar ao sistema, verificando os módulos, classes e funções responsáveis por implementar uma determinada funcionalidade.

A Figura 23 também ilustra que a rastreabilidade horizontal pode ser obtida através dos artefatos de testes inclusive. Duas ou mais funcionalidades ou requisitos estão interrelacionados se compartilham algum módulo em comum do sistema. A interdependência é diretamente proporcional à atomicidade do relacionamento. Dois requisitos que compartilham uma mesma função estão mais relacionados que dois requisitos que compartilham um mesmo módulo ou classe, que, por sua vez, estão mais relacionados que dois requisitos que compartilham um mesmo componente, e que estão mais relacionados que dois requisitos que compartilham um mesmo subsistema.

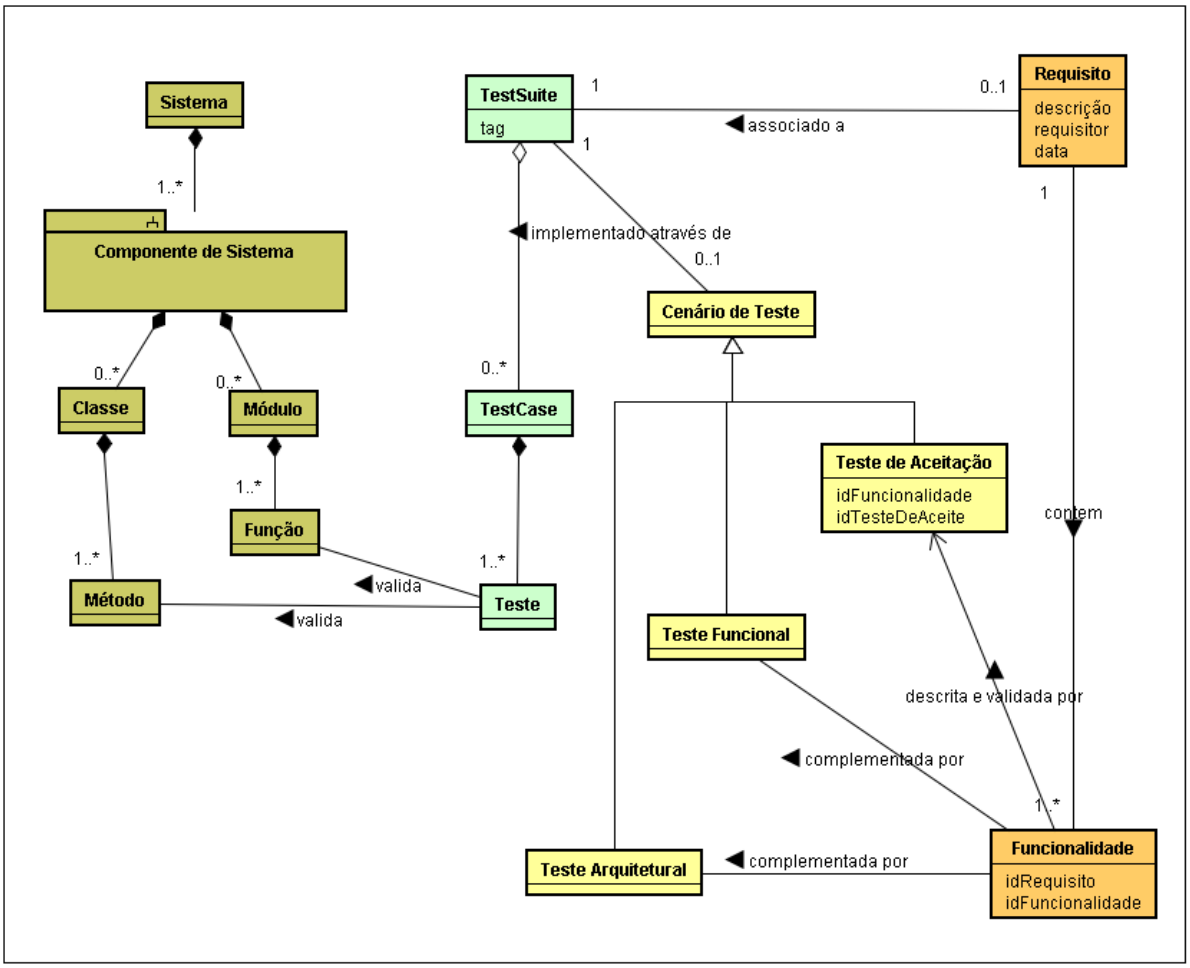


Figura 23 – Modelo proposto de implementação de teste unitário
Fonte: Elaborado pelo autor

De maneira geral, o teste unitário implementado da maneira proposta proporciona:

- Rastreio de baixa granularidade: identifica os componentes de código afetados por um requisito (característica de rastreio);
    - Funções, métodos, classes e configurações.
- Mais baixo nível na validação do acoplamento de requisitos: requisitos que compartilham o mesmo trecho de código estão relacionados entre si (característica de rastreio);
- Testes de características não funcionais e também funcionais (característica de validação);
- Pode ser automatizado por ferramentas (característica de métodos ágeis).

3.4 Criação e Armazenamento dos Artefatos de Rastreio

Os cenários de testes são o elemento de rastreio no DDRT e devem refletir o estado atual do sistema. Assim como o código fonte, os cenários de testes são armazenados nas ferramentas de controle de versão. Desta maneira, é possível acompanhar a evolução dos requisitos e dos cenários em relação ao tempo.

Se armazenados em forma de texto, os cenários de teste podem facilmente ser submetidos a ferramentas de buscas e nas comparações disponibilizadas pelas ferramentas de controle de versão.

É possível também extrair uma lista das funcionalidades existentes em uma determinada versão do sistema, bastando apenas resgatar todos os cenários de teste armazenados para aquela versão determinada.

3.5 Resumo

Os testes acompanham todo o ciclo de desenvolvimento de software. Na fase de análise de requisitos, eles são utilizados para a elaboração das especificações dos testes funcionais.

A partir dos testes funcionais ou de aceite, os testes de sistema são criados, identificando assim os subsistemas e as características não funcionais que serão afetadas pelo requisito sendo validado pelo teste de aceite.

Os testes de integração e unitários são criados a partir dos testes de aceite e de sistema, sendo estes utilizados como elementos de desenho do sistema.

Os testes unitários, desenvolvidos na mesma linguagem de programação do sistema, fecham a cadeia de rastreio, fazendo a ligação bidirecional do requisito ao código fonte.

4 APLICAÇÃO DE EXEMPLOS PRÁTICOS

Este capítulo apresenta a rastreabilidade orientada a testes para uma linha de produto de Sistemas de Ponto de Venda.

Sistemas de Ponto de Venda, ou PDV, são sistemas de automação de lojas que possuem, entre outras, as seguintes funcionalidades: registro das vendas, impressão de documentos fiscais, geração de relatórios gerenciais e controle de recebimento. Quando instalados em restaurantes e lanchonetes, conforme mostra o modelo de componentes do PDV Global da Figura 24, os PDVs são completados com módulos de controle de produção de itens para auxiliar a cozinha e as áreas de entrega.

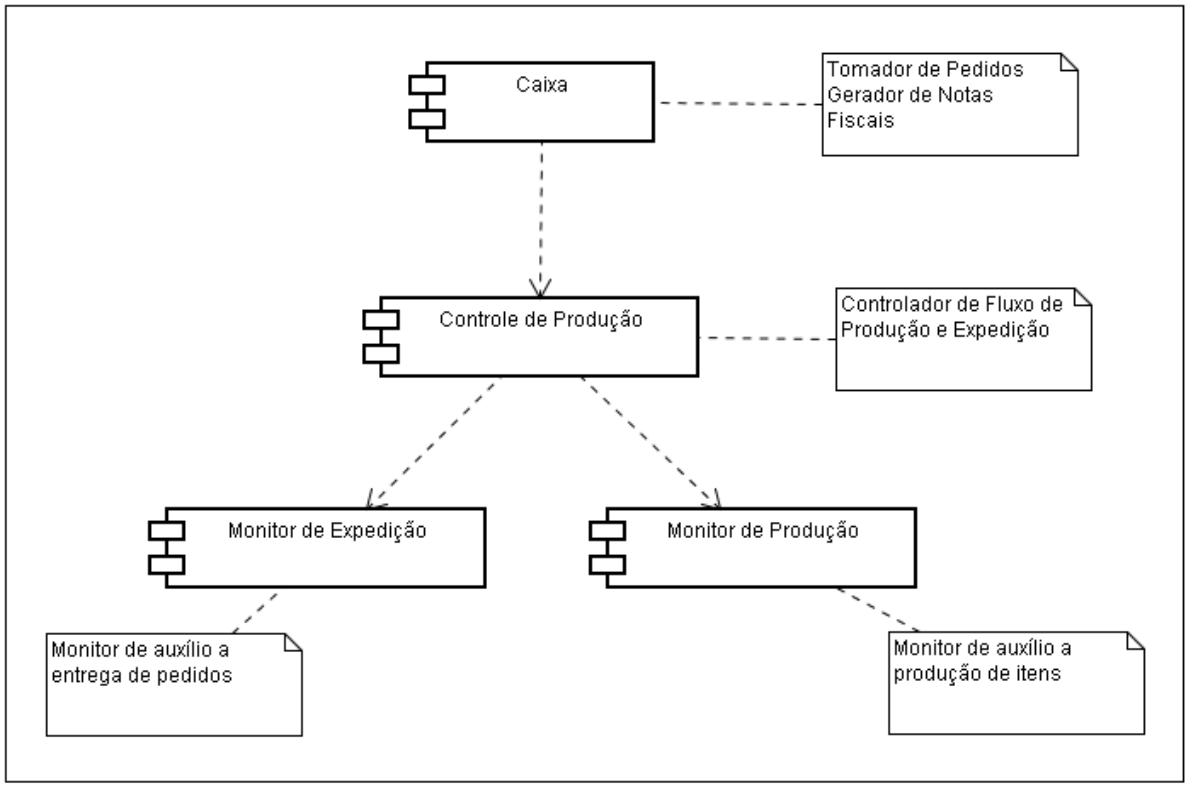


Figura 24 – Modelo de componentes do PDV Global
Fonte: Elaborado pelo autor

4.1 Linha de Produto de Sistemas de Ponto de Venda

O PDV Global é um sistema de ponto de venda instalado em mais de trinta países e construído em uma plataforma de Linha de Produto de Software, LPS.

Os requisitos, elaborados por cada país, podem conter funcionalidades locais, globais obrigatórias ou globais opcionais. De maneira geral, as seguintes regras se aplicam:

- Requisitos globais obrigatórios: são requisitos que devem ser implementados no “*core*” do sistema e que estarão disponíveis como funcionalidades obrigatórias para todos os produtos membros;
- Requisitos globais opcionais: são requisitos que devem ser implementados no “*core*” do sistema, porém, a decisão sobre se a funcionalidade implementada por este requisito será incorporada ao produto dependerá das necessidades de cada produto membro;
- Requisitos locais: são requisitos que serão utilizados apenas em determinada região ou país, não sendo implementadas no “*core*” do sistema. No entanto, a plataforma precisa prover meios para que a funcionalidade seja implementada localmente.

Os requisitos, conforme Figura 25, são especificados através do documento de Especificação de Requisitos, o qual contém um conjunto de funcionalidades agrupadas por um tema central.

Monitor de expedição

Este documento apresenta as funcionalidades necessárias para os Monitores de expedição com o intuito de aumentar a velocidade e a acuidade da entrega dos itens solicitados pelo cliente.

Requerimentos funcionais

Exibição de múltiplos pedidos

Req. #	Description
1.0	Afim de atender múltiplos clientes de maneira simultânea, o monitor deve exibir até 8 pedidos simultâneos organizados em forma de tabela com 4 colunas e duas linhas.
1.1	Em respeito a ordem de chegada dos clientes, os pedidos devem ser exibidos de cima para baixo, da esquerda pra direita de ordenados pelo momento do pedido, pedidos mais velhos deverão ser exibidos na primeira posição. O funcionário irá preparar os pedidos de acordo com a ordem exibida.

Especificação do número do ponto de atendimento

Req. #	Description
2.0	Cada pedido exibido no monitor precisa estar identificado com o número do caixa que tomou o pedido. Este número irá permitir que o funcionário saiba onde deve levar os produtos que estão sendo exibidos no monitor
2.1	O número do ponto de atendimento deverá se exibido no canto esquerdo superior do pedido, precedido da letra "C" A posição facilita a identificação do caixa que tomou o pedido e a letra "C" dá significado ao número.

Especificação do número do pedido

Req. #	Description
3.0	Cada pedido exibido no monitor precisa estar identificado através de um número de pedido, os mesmo impresso no comprovante fiscal do cliente. Este número irá identificar o cliente para qual o pedido pertence.
3.1	O número do pedido deverá ser exibido sempre com 3 dígitos, logo após o numero do caixa e precedido por um "-". A exibição padronizada aumenta a legibilidade do número diminuindo assim o erro na identificação do pedido. A posição facilita a identificação do numero do pedido relacionado ao caixa que tomou o pedido, o "-" separa o numero do caixa do numero do pedido.

Figura 25 – Exemplo de especificação de requisito
Fonte: Elaborado pelo autor

4.2 Aplicação da Proposta

Para demonstrar o rastreio por testes aplicados a esta linha de produto, as etapas definidas no Capítulo 3 são executadas através de exemplos.

A rastreabilidade começa na criação dos testes de aceite utilizados para validar os requisitos obtidos através das histórias de usuário. As características não funcionais são verificadas e validadas pelos testes de integração, e por último, os testes unitários validam o código fonte e completam o rastreio, conectando os requisitos ao código fonte.

4.3 Definindo os Testes de Aceite

Os testes de aceite produzem a primeira conexão de rastreio entre requisitos e código fonte e são também um elemento de verificação que garante a implementação e a avaliação dos requisitos.

A Tabela 5 apresenta os testes de aceite para o Requisito “Monitor de Expedição-Exibição de múltiplos pedidos”.

Teste de Aceite:	**Monitor de Expedição**		
Funcionalidade:	**Exibição de múltiplos pedidos**		
Requisito:	**1.0**		
Teste	**MaximoNumeroDePedidos**		
1. Inicie a aplicação			
	iniciar	Caixa 1	
	iniciar	Caixa 2	
	iniciar	Monitor de Expedição	
2. Faça 2 pedidos no caixa 1			
	Entre	pedido	C1-001
	Pague	pedido	C1-001
	Entre	pedido	C1-002
	Pague	pedido	C1-002
3. O Monitor deve exibir os dois pedidos			
	ListaDePedidosExibidos		
	Caixa	número do pedido	
	C1	001	
	C1	002	
4. Faça 3 pedidos no caixa 2			
	Entre	pedido	C2-001
	Pague	pedido	C2-001
	Entre	pedido	C2-002
	Pague	pedido	C2-002
	Entre	pedido	C2-003

	Pague	pedido	C2-003
5. Faça mais 5 pedidos no caixa 1			
	Entre	pedido	C1-003
	Pague	pedido	C1-003
	Entre	pedido	C1-004
	Pague	pedido	C1-004
	Entre	pedido	C1-005
	Pague	pedido	C1-005
	Entre	pedido	C1-006
	Pague	pedido	C1-006
	Entre	pedido	C1-007
	Pague	pedido	C1-007
6. O Monitor deve exibir apenas os oito primeiros pedidos			
	ListaDePedidosExibidos		
	Caixa	número do pedido	
	C1	001	
	C1	002	
	C2	001	
	C2	002	
	C2	003	
	C1	003	
	C1	004	
	C1	005	

Tabela 5 – Teste de Aceitação: Exibir múltiplos pedidos no monitor de expedição
Fonte: Elaborado pelo autor

A conexão de rastreio entre o teste de aceite e o código fonte é feita através dos testes unitários. Os testes unitários avaliam os aspectos arquiteturais e, tal como os testes de aceite, a implementação dos requisitos no código fonte.

4.4 Validando a Arquitetura do Sistema através dos Testes de Integração

Os testes de integração e arquitetural validam as características não funcionais do sistema. Além de validar como os componentes e subsistemas interagem na implementação das funcionalidades, os testes de integração são utilizados para rastrear todos os componentes envolvidos na implementação de uma funcionalidade.

Já os testes arquiteturais são utilizados para validar aspectos arquiteturais da funcionalidade. Características não funcionais como geração de logs, capacidade de monitoração e divisão em camadas são avaliadas pelos cenários de testes arquiteturais e seus componentes de teste arquitetural.

Por ser um sistema instalado em muitos pontos de vendas distribuídos em mais de 30 países, o PDV Global conta com três níveis de suporte ao sistema para identificar e corrigir os problemas que acontecem em campo.

O nível 1 de suporte é formado por uma equipe de colaboradores responsável pela configuração e instalação do PDV para um determinado país. O nível 2 é formado pela equipe responsável por programar e selecionar as funcionalidades do PDV Global que estarão disponíveis para uma determinada região ou para um conjunto de países. Já o nível 3 de suporte é formado pela equipe que desenvolve o núcleo do sistema, ou seja, as funcionalidades globais obrigatórias e opcionais que estarão disponíveis para todas as regiões.

O sistema de log do PDV Global precisa ser desenvolvido de tal maneira a auxiliar os diversos níveis de suporte a identificar a causa de um problema ocorrido em campo. Problemas de configuração devem ser direcionados à equipe responsável pela configuração do PDV, ou seja, a equipe de suporte nível 1. Já problemas relacionados à localização, programação e seleção das funcionalidades da região são de responsabilidade da equipe de suporte nível 2. E, finalmente, problemas relacionados a funcionalidades globais devem ser direcionados à equipe de suporte nível 3, ou seja, a equipe responsável pelo desenvolvimento do núcleo do sistema.

Dentro da arquitetura do PDV Global, os logs são elementos arquiteturais que ajudam a descobrir problemas em campo e devem estar presentes em todas as implementações dos requisitos funcionais.

A Figura 26 ilustra a estrutura estática de gravação de logs do PDV Global. O escritor de log, representado pela classe LogGravador, é o principal ponto de entrada para criar entradas de log e escrevê-los em seus respectivos destinos representados pela classe LogOuvinte. Ele cria uma instância de uma entrada de log contendo as informações a serem registradas, e interage com os outros objetos que filtram a entrada de log, identifcam as categorias, formatam e despacham para os destinos adequados.

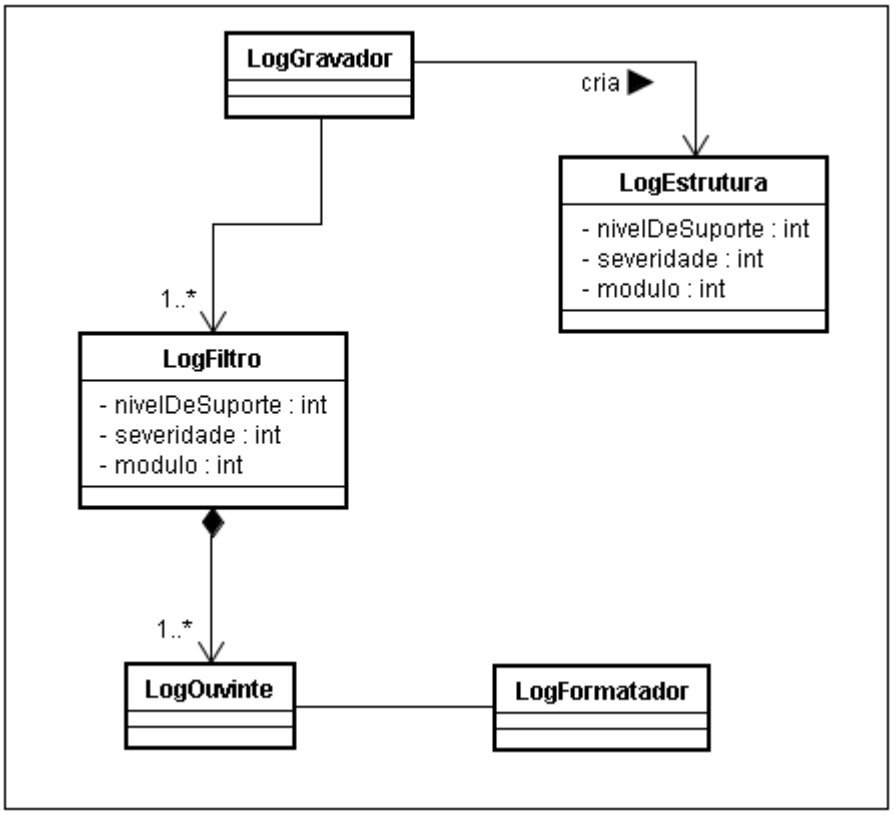


Figura 26 – Estrutura de Logs do PDV Global
Fonte: Elaborado pelo autor

Conforme apresentado na Tabela 6, os Cenários de Teste que validam os requisitos funcionais são então complementados com o cenários de teste que validam as decisões arquiteturais para a funcionalidade “Exibição de múltiplos pedidos”.

Teste de Aceite:	**Monitor de Expedição**		
Funcionalidade:	**Exibição de múltiplos pedidos**		
Requisito:	**1.0**		
Teste	**Log nível 1 ao atingir máximo número de pedidos**		
1. Inicie a aplicação			
	Iniciar	Caixa 1	
	Iniciar	Monitor de Expedição	
2. Faça 10 pedidos no caixa 1			
	Entre	pedido	C1-001
	Pague	pedido	C1-001
	Entre	pedido	C1-002
	Pague	pedido	C1-002
	Entre	pedido	C1-003
	Pague	pedido	C1-003
	Entre	pedido	C1-004

	Pague	pedido	C1-004
	Entre	pedido	C1-005
	Pague	pedido	C1-005
	Entre	pedido	C1-006
	Pague	pedido	C1-006
	Entre	pedido	C1-007
	Pague	pedido	C1-007
	Entre	pedido	C1-008
	Pague	pedido	C1-008
	Entre	pedido	C1-009
	Pague	pedido	C1-009
	Entre	pedido	C1-010
	Pague	pedido	C1-010
6. O módulo Monitor de Expedição deve adicionar duas linhas de log de suporte nível 1 alertando que dois pedidos não estão sendo exibidos.			
	ListaDeLogsNivel1		
	NívelDeSuporte	Módulo	Código
	1	MonitorExpedição	ME-200
	1	MonitorExpedição	ME-200

Tabela 6 – Teste Arquitetural: Exibir múltiplos pedidos no monitor de expedição
Fonte: Elaborado pelo autor

4.5 Implementando os Testes Unitários

Os testes unitários são o elemento de rastreio mais próximo do código fonte, devendo conter informações que façam a ligação com o teste de aceite e com o requisito pelo qual o teste unitário está sendo criado. O *TestCase*, conforme segue abaixo, é uma classe implementada na mesma linguagem de programação do GlobalPDV e é desenvolvido seguindo os mesmos padrões de codificação do sistema.

No cabeçalho do arquivo, como mostra a Figura 27, são inseridas algumas informações de rastreio, como o nome do cenário de teste, os códigos do teste de aceite e do requisito, bem como o número da revisão do teste do repositório de código fonte, o nome do desenvolvedor responsável pelo teste e a data de criação.

```
/**
 * ExibirNumeroDoCaixa.java
 * Este TesteCase valida a estória de usuário Número do ponto de
atendimento
 *
 * @TesteDeAceite ME-02.2010
 * @Requerimento  ME-2010.01
 *
 *
 * $Revision: 1.8 $
 * $Date: 2007/11/26 16:24:43 $ (of revision)
 * $Author: rcaram $ (of revision)
*/
```

Figura 27 – Teste unitário com associação ao requisito
Fonte: Elaborado pelo autor

A preparação dos dados do teste unitário - *TestFixture* - é efetuada na função de inicialização do teste. Isto é feito para que, nas funções de teste, fiquem apenas as chamadas funções que estão sendo testadas e rastreadas além das funções de validação de resultados, conforme Figura 28.

```
	/**
	 * ---- TestFixture ----
	 * Prepara o ambiente para teste
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
		caixaNumero2 = new Caixa(2);
		caixaNumero12 = new Caixa(12);
		pedidoCaixa2 = caixaNumero2.novoPedido();
		pedidoCaixa12 = caixaNumero12.novoPedido();
		xmlPedidoCaixa2 =caixaNumero2.criarXMLParaMonitor(pedidoCaixa2);

		mockSocket = new MockSocket();
		caixaNumero2.setInterfaceDeComunicacao(mockSocket);
		caixaNumero12.setInterfaceDeComunicacao(mockSocket);
	}

	/**
	 *
	 * Testa se o XML que representa o pedido contém o número do caixa
que tomou o pedido.
	 * @throws java.lang.Exception
	 */
	@Test
	public void testeRepresentacaoXMLDoPedidoDoCaixa(){
		String xml = null;

		//Método sendo testado
		xml = caixaNumero2.criarXMLParaMonitor(pedidoCaixa2);

		//Validando
		Assert.assertThat(xml,
JUnitMatchers.containsString("<CaixaDeOrigem numero=\"2\"\\>"));

		//Método sendo testado
		xml = caixaNumero12.criarXMLParaMonitor(pedidoCaixa12);

		//Validando
		Assert.assertThat(xml,
JUnitMatchers.containsString("<CaixaDeOrigem numero=\"12\"\\>"));

	}
```

Figura 28 – Implementação de Teste Unitário com características de rastreio
Fonte: Elaborado pelo autor

4.6 Visualizando a Rastreabilidade

A Figura 29 apresenta a malha de rastreio criada a partir da instanciação dos objetos de rastreio da Figura 17 para rastrear requisitos através das especificações de testes.

Os requisitos, divididos em histórias de usuário, são validados pelos testes de aceite, formando assim o primeiro elemento de rastreio.

Os testes de integração, criados a partir do teste de aceite, formam o segundo elemento de rastreio e são utilizados para identificar os componentes de software ou os requisitos de sistema, o quais são utilizados na implementação do requisito de usuário sendo testado.

Como último elemento de rastreio, o teste unitário promove a conexão final, identificando as funções e métodos que implementam o requisito de usuário e constituindo o nível de maior granularidade de rastreio.

Os requisitos de usuário solicitados pelo documento de Especificação de Requisitos contêm detalhes das necessidades do usuário, sem a visão do sistema, divididos em subtópicos chamados de Funcionalidades.

Cada funcionalidade é então transformada em Requisitos de Sistema e validada através dos Testes de Aceite. Neste momento, é apresentado ao usuário como o sistema atenderá à necessidade de negócio solicitada pela Especificação de Requisitos. O Teste de Aceite, além de clarificar as necessidades do usuário com exemplos de como o sistema se comportará para atender a tal requisito, serve como um “contrato” firmado entre as partes e também como *link* de rastreio e base para a criação dos testes que validarão os métodos e funções que conferirão comportamento ao sistema.

Os testes unitários criados para validar os métodos e funções do sistema fazem a ligação final do requisito ao código fonte, proporcionando assim o rastreio vertical bidirecional. Conforme mostra a Figura 30, o desenvolver consegue a qualquer momento visualizar quais são as funcionalidades e o requisito de usuário que estão sendo atendidos pela função “criarXmlDoPedidoDoCaixa”. Da mesma maneira, os Usuários, Gerentes de Projetos, Testadores e Analistas de Negócio conseguem saber o impacto que uma mudança da funcionalidade “Exibir número do

caixa” pode causar ao sistema ao saber que os módulos Caixa e Monitor serão afetados, mas que apenas os métodos “criarXmlDoPedidoDoCaixa”, “EnviarPedidoParaMonitor” serão afetados com isso.

A rastreabilidade horizontal também é obtida utilizando-se da conexção criada pelos artefatos de testes. É possível verificar na Figura 29 que as funcionalidades “Exibir número do caixa” e “Exibir número do pedido” estão relacionadas, pois se utilizam da mesma função do sistema “criarXmlDoPedidoDoCaixa”.

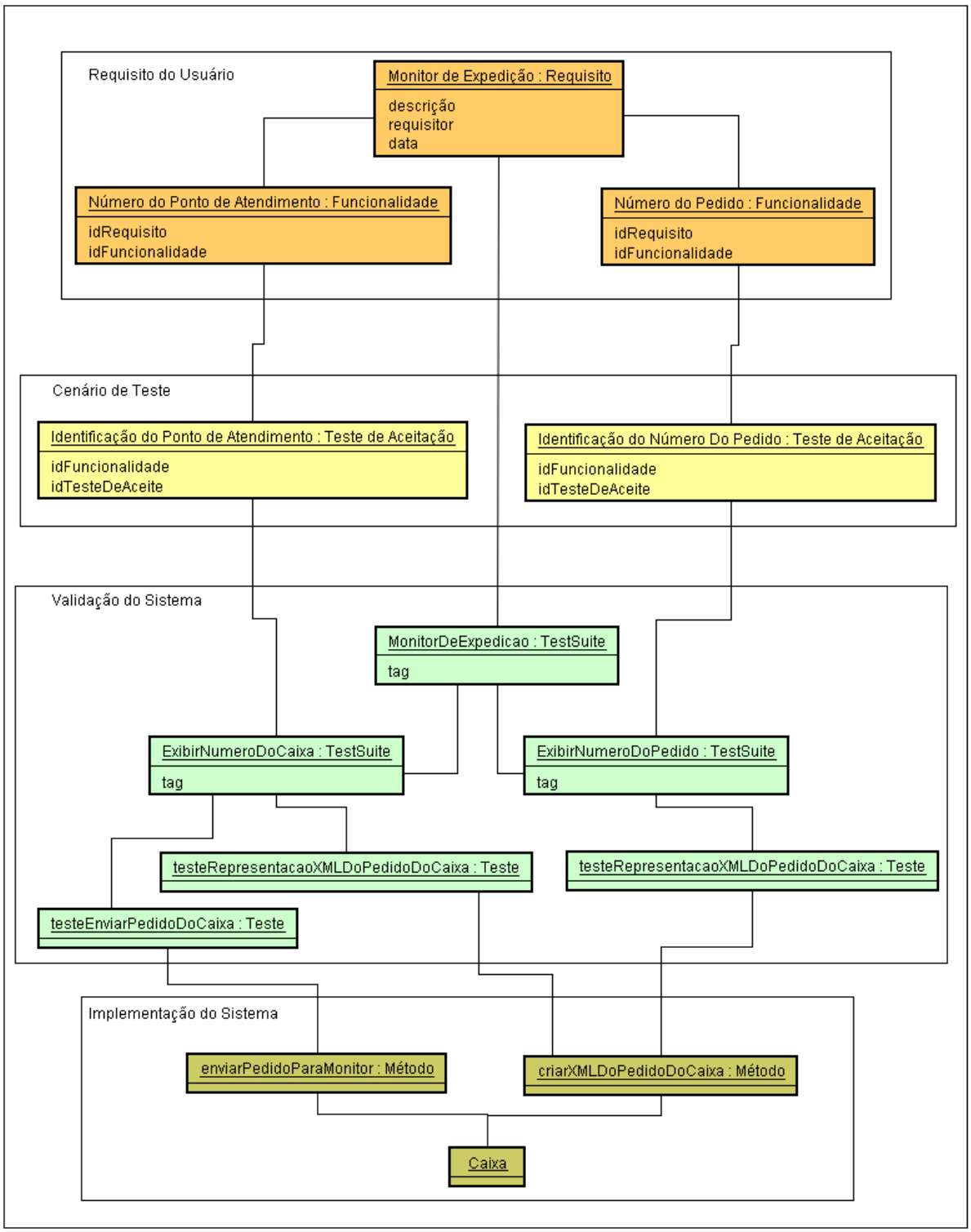


Figura 29 – Malha de rastreio por artefatos de teste
Fonte: Elaborado pelo autor

Como pôde-se observar, a rastreabilidade é automática: a alteração em um requisito através de uma história de usuário faz com que o teste de aceite seja atualizado, fazendo com que o teste unitário seja atualizado, o que, por fim, reflete-

se no código fonte. Todo rastreio é feito na linguagem de cada membro da equipe: o usuário altera a história de usuário correspondente; os testadores, membros da equipe de garantia da qualidade, alteram os testes de aceite; e os desenvolvedores, por sua vez, alteram a arquitetura, os testes unitários e o código fonte.

Já em um modelo tradicional, a rastreabilidade é controlada pela matriz de rastreabilidade, em uma planilha à parte do processo, sem a utilização da linguagem dos usuários e dos membros da equipe de desenvolvimento.

4.7 Analise de Resultados

Os resultados obtidos na aplicação desta proposta são apresentados utilizando-se dos mesmos critérios utilizados na avaliação do *framework* de rastreabilidade apresentado por Sousa (2008).

- Representação: o critério de representação caracteriza como a abordagem representa as informações de rastreabilidade. No DDRT, a rastreabilidade é representada pelos cenários de teste e implementada pelos testes unitários;
- Mapeamento e Granularidade: o critério de mapeamento e granularidade indica a direção e o nível de granularidade do rastreio, ou seja, se a abordagem é capaz de gerar a rastreabilidade do requisito para o código fonte e do código fonte para o requisito, e se a abordagem consegue rastrear além do artefato (baixa granularidade), os detalhes internos que formam o artefato (alta granularidade). No DDRT, os Cenários de Testes são utilizados para descrever os requisitos funcionais e não funcionais, e a implementação dos cenários através de testes unitários estabelece uma ligação bidirecional entre os requisitos e o código fonte. Por utilizar o TDD na construção do sistema, o DDRT atinge um alto número de cobertura de teste e, portanto, um alto nível de detalhes internos do código fonte;
- Escalabilidade: o critério de escalabilidade analisa se é possível aplicar a abordagem de um sistema de grande porte. O DDRT foi aplicado apenas em uma pequena parte do PDV Global. Mais estudos precisam

ser efetuados para verificar o impacto do DDRT em sistemas de grande porte.

Análise de Impacto de Mudança: o critério de análise de impacto de mudança avalia a forma como a abordagem inclui suporte para detectar o impacto das mudanças sobre os artefatos relacionados. O DDRT prove meios de se identificar os cenários testes que serão afetados por uma mudança e, consequentemente, os testes unitários e códigos fontes. Porém, um estudo com ferramentas de análise estática de código precisa se feito para identificar o real suporte do DDRT na análise de impacto.

5 CONCLUSÃO

O DDRT propõe a utilização de cenários de teste para rastrear os requisitos durante o processo de desenvolvimento de software. Para rastrear os requisitos, são utilizados cenários padronizados de teste, escritos por cenários de casos de uso e tabelas de automação FIT. Esses cenários validam os requisitos do sistema elaborados através de histórias de usuários, assim como decisões arquiteturais e requisitos não funcionais.

O DDRT é uma proposta de extensão do TDD, que aplica os conceitos de testes funcional e estrutural para auxiliar o desenvolvedor na construção de sistemas dirigida por testes. Devido ao fato de os testes serem elementos sempre presentes no processo de produção de software e acompanharem naturalmente a evolução dos requisitos e das solicitações de mudança, com essa proposta, acredita-se ser possível validar e verificar os requisitos e as decisões arquiteturais, identificar os requisitos conflitantes, aumentar a qualidade do sistema, assim como identificar os requisitos implementados no código fonte.

Por meio de dois casos de aplicação apresentados neste trabalho, observa-se que é possível utilizar os cenários de testes e testes unitários - embora sejam necessárias adaptações ao TDD e na implementação dos testes unitários - para rastrear requisitos sem que seja necessário a criação de uma matriz de rastreabilidade,

Uma limitação da técnica é em relação ao rastreio de decisões arquiteturais, que possibilita o rastreio de requisitos não-funcionais mas não das decisões que definem a estrutura dos componentes e a relação entre eles.

Outra limitação da técnica esta no rastreio vertical no sentido código fonte-requisito, visto que, não é possível apenas com a malha de ligação criada através dos testes identificar com precisão qual o requisito responsável por ume pedaço de código que esta sendo compartilhado por um conjunto de métodos e funções exercitadas pelos testes unitários.

5.1 Contribuições

A técnica DDRT demonstra que a utilização dos cenários de testes para dirigir o desenvolvimento de sistemas habilita os testes como elemento para rastrear requisitos.

Os cenários de testes provêem mais informações para o desenvolvedor no momento da elaboração dos testes unitários, ajudando assim na elaboração do projeto de software durante a fase de codificação.

A criação de cenários de testes para validar as decisões arquiteturais e a implementação destes cenários através de teste unitário clarificam decisões de codificação que auxiliam no entendimento do código fonte.

A técnica apresentada foi capaz de rastrear códigos fonte associados à infraestrutura (segurança, apresentação, tratamento de erros, facilidade de identificação de problemas em campo, suporte, dentre outras questões), assim como rotinas associadas aos requisitos funcionais e processos de negócio.

5.2 Sugestões para Futuras Pesquisas

Este trabalho propõe a técnica de Desenvolvimento Dirigido e Rastreado por Testes (DDRT) e, embora tenha sido demonstrado como o DDRT pode ser utilizado para se extrair o rastreio de requisitos, prover mais informações para o desenvolvedor e identificar os requisitos conflitantes, espera-se que trabalhos futuros sejam feitos para comparar a técnica DDRT com outras propostas de rastreio de requisitos.

Outro ponto que precisa ser estudado é o impacto que a técnica DDRT pode ter sobre a agilidade do processo de desenvolvimento, principalmente em ambientes de desenvolvimentos ágeis.

Nesta linha, este trabalho será utilizado como base de pesquisa do laboratório LabMed do IPT na área de medições de processo de desenvolvimento ágil de software.

Outra proposta para trabalhos futuros é o desenvolvimento de ferramentas que facilitem a criação dos cenários de testes e a criação automatizada de testes unitários para refletir os cenários de testes.

Finalmente, outra proposta para trabalho futuro é a criação de uma ferramenta de análise estática de teste unitário e código fonte para a extração automática da tabela de rastreabilidade.

REFERÊNCIAS

AMBLER, S. W. **Introduction to Test Driven Design (TDD)**, 2010. Disponível em: <http://www.agiledata.org/essays/tdd.html>. Acesso em: 22 ago. 2010.

ASTELS, D. **Test-Driven Development: A Practical Guide**. 2.ed. Prentice Hall, 2003.

ASUNCION, H. U.; FRANÇOIS, F.; TAYLOR, R. N. An end-to-end industrial software traceability tool. In: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07. **Anais...** . Dubrovnik, Croatia, 2007. p.115.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice (2nd Edition)**. 2ed. Addison-Wesley Professional, 2003.

BEATO, S. T. S. **Casos de Teste de Uso: uma contribuição para o desenvolvimento de software dirigido por testes**. Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 2008.

BECK, K. **Test Driven Development: By Example**. Addison-Wesley Professional, 2002.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. 2 ed. Addison-Wesley Professional, 2004.

BERGSTRÖM, A. **Software Configuration Management in Scrum Projects**. Lund University, jun. 2008.

BUDWIG, M.; JEONG, S.; KELKAR, K. When user experience met agile: a case study. In: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems. **Anais...** . Boston, MA, USA: ACM, 2009. p.3075-3084.

CHITTIMALLI, P. K.; HARROLD, M. J. Regression test selection on system requirements. In: Proceedings of the 1st India software engineering conference. **Anais...** . Hyderabad, India: ACM, 2008. p.87-96.

CLELAND-HUANG, J.; CHANG, C.; CHRISTENSEN, M. Event-based traceability for managing evolutionary change. **IEEE Transactions on Software Engineering**, v. 29, n. 9, p. 796-810, 2003.

CUNNINGHAM, W. **Introduction To FIT**. 7. sep. 2002. Wiki. Disponível em: <http://fit.c2.com/wiki.cgi?IntroductionToFit>. Acesso em: 21 jul. 2010.

DAMIAN, D.; CHISAN, J. An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. **IEEE Trans. Softw. Eng.**, v. 32, n. 7, p. 433-453, 2006.

EGYED, A.; GRÜNBACHER, P. Automating Requirements Traceability: Beyond the Record & Replay Paradigm. In: Proceedings of the 17th IEEE international conference on Automated software engineering. **Anais...** . IEEE Computer Society, 2002. p.163.

GOLDSMITH, R. F. **The Forgotten Phase**. jul. 2002a. Disponível em: <http://www.gopromanagement.com/articles.html>. Acesso em: 8 mar. 2010.

GOLDSMITH, R. F. **This or That, V or X?** sep. 2002b. Disponível em: <http://www.gopromanagement.com/articles.html>. Acesso em: 8 mar. 2010.

HANSSEN, G. **Agile Software Product Line Engineering**. 2007.

HAYES, J. H.; DEKHTYAR, A.; JANZEN, D. S. Towards traceable test-driven development. In: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. **Anais...** . IEEE Computer Society, 2009. p.26-30.

IEEE Standard Glossary of Software Engineering Terminology. .**IEEE Std 610.12-1990**, 1990. Disponível em: <10.1109/IEEESTD.1990.101064>. Acesso em: 20 jul. 2010.

JACOBSSON, M. **Implementing Traceability In Agile Software Development**. 2 feb. 2009. Lund University. Disponível em: <www.cs.lth.se/EDA920/2009/2009-02_Rapport.pdf>. Acesso em: 5 mar. 2010.

JANZEN, D. S. Software architecture improvement through test-driven development. In: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. **Anais...** . San Diego, CA, USA: ACM, 2005. p.240-241.

LEE, C.; GUADAGNO, L.; JIA, X. An Agile Approach to Capturing Requirements and Traceability. **Proceedings of the 2nd international workshop on traceability in emerging forms of software engineering**, v. 2003, p.17-23, 2003.

LINES, M.; BARNES, J.; HOLMES, J.; AMBLER, S. W. Agile Rational Unified Process: RUP experiences from the trenches. 15 feb. 2008. CT316. Disponível em: <http://www.ibm.com/developerworks/rational/library/edge/08/feb08/lines_barnes_holmes_ambler/index.html#N10151>. Acesso em: 30 abr. 2010.

MAXIMILIEN, E. M.; WILLIAMS, L. Assessing test-driven development at IBM. In: Proceedings of the 25th International Conference on Software Engineering. **Anais...** . Portland, Oregon: IEEE Computer Society, 2003. p.564-569.

MUGRIDGE, R.; CUNNINGHAM, W. **Fit for Developing Software: Framework for Integrated Tests**. Prentice Hall, 2005.

MYERS, G. J. **The Art of Software Testing, Second Edition**. 2ed. Wiley, 2004.
NASLAVSKY, L.; ALSPAUGH, T. A.; RICHARDSON, D. J.; ZIV, H. Using scenarios to support traceability. In: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering - TEFSE '05. **Anais...** . Long Beach, California, 2005. p.25.

PARK, S. S.; MAURER, F. The benefits and challenges of executable acceptance testing. In: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral. **Anais...** . Leipzig, Germany: ACM, 2008. p.19-22.

PEZZE, M.; YOUNG, M. **Teste e Análise de Software**. Bookman, 2008.

RATANOTAYANON, S.; SIM, S. E.; GALLARDO-VALENCIA, R. Supporting Program Comprehension in Agile with Links to User Stories. In: AGILE Conference. **Anais...** . Los Alamitos, CA, USA: IEEE Computer Society, 2009. p.26-32.

RICCA, F.; TORCHIANO, M.; CECCATO, M.; TONELLA, P. Talking tests: an empirical assessment of the role of fit acceptance tests in clarifying requirements. In: Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting. **Anais...** . Dubrovnik, Croatia: ACM, 2007. p.51-58.

RODRIGUES, L. G. M. **Um modelo de avaliação dos requisitos no processo de desenvolvimento de software**. 21 feb. 2006. Instituto de Computação. Disponível em: <http://libdigi.unicamp.br/document/?code=vtls000388272>. Acesso em: 8 mar. 2010.

ROZANSKI, N.; WOODS, E. **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley Professional, 2005.

SCHWABER, K. **Agile Project Management with Scrum**. 1ed. Microsoft Press, 2004.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide**. fev. 2010. Disponível em: <http://www.scrum.org/scrumguides/>. Acesso em: 3 ago. 2010.

SOMMERVILLE, I. **Software Engineering: (Update)**. 8ed. Addison Wesley, 2006.

SOUSA, A. **Traceability Support in Software Product Lines**. Universidade Nova de Lisboa. Faculdade de Ciências e Tecnologia, 2008.

WELLS, D. **Acceptance Tests**. 1999. Disponível em: <http://www.extremeprogramming.org/rules/functionaltests.html>. Acesso em: 19 ago. 2010.

WESTFALL, L. **Bidirectional Requirements Traceability**. feb. 2006. Disponível em: <http://www.westfallteam.com/software_configuration_managmenet_papers_&_resources.htm>. Acesso em: 12 abr. 2010.