UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE APOIO AOS TESTES BASEADOS EM REQUISITOS

LEANDRO DA CUNHA

BLUMENAU
2010

2010/2-17

LEANDRO DA CUNHA

FERRAMENTA DE APOIO AOS TESTES BASEADOS EM REQUISITOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Everaldo Artur Grahl - Orientador

BLUMENAU
2010

2010/2-17

FERRAMENTA DE APOIO AOS TESTES BASEADOS EM REQUISITOS

Por

LEANDRO DA CUNHA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
Membro:	Prof. Jacques Robert Heckmann, Mestre – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 13 de dezembro de 2010

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que sempre me apoiou nos momentos mais difíceis.

À Elisangela Fischer, por estar ao meu lado em todos os momentos.

Ao meu orientador, Everaldo Artur Grahl, pelo seu apoio e por ter acreditado na conclusão deste trabalho.

Nas grandes batalhas da vida, o primeiro passo para a vitória é o desejo de vencer.

Mahatma Gandhi

RESUMO

Este trabalho tem como objetivo o desenvolvimento de uma ferramenta de apoio aos testes baseados em requisitos. A ferramenta visa auxiliar a documentação de requisitos, tornando-os mensuráveis através de propriedades definidas pelo analista de sistemas. É utilizada também a técnica de grafos de causa e efeito para auxiliar o analista a montar uma visão mais lógica do requisito, evitando assim possíveis ambigüidades. O trabalho aborda também a geração de casos de testes básicos para os requisitos, buscando maior agilidade no processo de construção de testes. A ferramenta disponibiliza uma automação do plano de testes, no qual qualquer alteração em um requisito gera uma alteração no plano de testes, facilitando a rastreabilidade de impactos causados por mudanças. O desenvolvimento é realizado sobre a plataforma *Eclipse Rich Client Platform* (*Eclipse* RCP), utilizando principalmente os frameworks SWT e GEF.

Palavras-chave: Teste de software. Testes baseados em requisitos. Eclipse.

ABSTRACT

The objective this work is to develop a requirement based testing support tool. It aims to help requirement documentation, turning them measurable through properties defined by the system analyst. The cause effect graph technique is used to support the system analyst to build a more logic requirement view, avoiding possible ambiguities. It addresses yet a basic test case generation to the requirements, looking for more agility of test cases construction process. The tool provides test plan automation, in which any requirement update triggers a test plan update, making easier to trace the impact triggered by requirement changes. The development is performed using the Eclipse Rich Client Platform (Eclipse RCP), using mainly SWT and GEF frameworks.

Key-words: Software engineering. Software testing. Requirement based testing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CEG – *Cause Effect Graph*

GEF – *Graphical Editing Framework*

RAP – *Rich Application Platform*

RBT – *Requirement Based Testing*

RCP – *Rich Client Platform*

SWT – *Standard Widget Toolkit*

SUMÁRIO

1 INTRODUÇÃO

Diversas empresas veem o processo de teste de software como uma atividade onerosa dentro do desenvolvimento de novos produtos e na manutenção de produtos existentes. Muitas empresas organizam a área de teste de software como uma atividade que deve ser realizada pelos próprios programadores, para garantir que os requisitos foram atendidos.

O papel do teste é encontrar erros e validar o produto desenvolvido em relação aos requisitos. Freqüentemente os testes são selecionados de forma aleatória ou *ad-hoc* (momentânea) e os casos de teste são desenvolvidos de uma forma não estruturada e não sistemática. Esta é uma realidade encontrada no desenvolvimento de softwares comerciais, onde há recursos limitados e tempo escasso para o teste (RYSER; GLINZ, 2000, p. 1-2).

Parrington e Roper (1989, p. 9) já afirmavam que o tema central da engenharia de software é a produção de sistemas com qualidade. Para que seja possível atingir tal qualidade, é necessário possuir uma metodologia para especificar os requisitos e realizar a análise, assim como utilizar ferramentas de suporte ao desenvolvimento do produto. Porém, o ponto determinante para atingir o objetivo é possuir testes de software realmente efetivos.

Para Gutiérrez et al. (2006, p. 1), realizar testes de forma apropriada tem se tornado mais complexo dia após dia. Esta complexidade reforça a necessidade do uso de técnicas que assegurem a qualidade do produto final. Como o maior objetivo do teste de software é verificar se os requisitos foram adequadamente atendidos pelo sistema (faz o que deveria fazer), é imprescindível que sejam realizadas verificações que busquem apurar esta conformidade.

Existem várias abordagens que podem ser adotadas para testar um sistema. Podem ser utilizados o código fonte, os cenários definidos nos casos de uso, os diagramas de atividades ou de componentes, entre outros. Este trabalho irá abordar a construção de casos de testes baseados em requisitos. Este tipo de teste tem o objetivo de verificar se as necessidades do usuário, registradas na fase de elicitação de requisitos, foram de fato atendidas pelo software.

Os testes baseados em requisitos são realizados basicamente através da criação de condições de testes e *checklists* de funcionalidades. As condições de teste são preparadas inicialmente e de maneira genérica durante a fase de elicitação de requisitos e detalhadas pouco a pouco no decorrer do ciclo de vida de desenvolvimento do software até a preparação dos dados que serão usados nos testes do sistema (RIOS et al., 2007, p. 59).

Diante do exposto, foi necessário estudar e desenvolver uma ferramenta que facilitasse

o processo de testes, buscando maior eficiência no trabalho realizado e redução dos custos provenientes desta atividade. Optou-se aqui por construir uma extensão para a plataforma *Eclipse* RCP.

1.1 OBJETIVOS DO TRABALHO

Este trabalho teve como objetivo principal o desenvolvimento de uma ferramenta de apoio à geração de casos de testes baseados em requisitos de software.

Os objetivos específicos do trabalho são:

a) disponibilizar uma ferramenta que apóie o desenvolvimento de testes utilizando grafos de causa e efeitos, particionamento de classes de equivalência e valores limite;
b) gerar casos de testes a partir de propriedades definidas para os requisitos escritos em linguagem natural, de forma semi-automática;
c) ser desenvolvido como uma extensão do ambiente de desenvolvimento Eclipse.

1.2 ESTRUTURA DO TRABALHO

No capítulo 2 é feita uma revisão bibliográfica abordando os temas de requisitos, testes baseados em requisitos, documentação de teste de software, geração de testes baseados em requisitos e a construção de extensões para a ferramenta Eclipse. Por fim, são apresentadas algumas ferramentas que possuem funcionalidades semelhantes ao trabalho desenvolvido.

O terceiro capítulo apresenta como foi realizado o desenvolvimento do trabalho, passando pelos principais requisitos a serem atendidos, pela especificação e pela implementação, terminando com a apresentação da operacionalidade da ferramenta.

Ao final, o quarto capítulo apresenta as conclusões obtidas a partir deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são descritos pontos importantes sobre os requisitos para a fase de construção dos testes e o processo de desenvolvimento como um todo. A seção 2.2 apresenta o conceito de testes baseados em requisitos. Na seção 2.3 são abordadas as técnicas de geração de casos de testes que serão utilizadas no desenvolvimento do trabalho. A seção 2.4 segue como uma explanação do Eclipse, apresentando principalmente sua arquitetura. Na seção 2.5 são apresentados trabalhos correlatos.

2.1 REQUISITOS

Os requisitos expressam as características e restrições do produto de software do ponto de vista de satisfação das necessidades do usuário, e, em geral, independem da tecnologia empregada na construção da solução, sendo a parte mais crítica e propensa a erros no desenvolvimento de software (MACORATTI.NET, 1999, p. 1).

Requisitos são objetivos ou restrições estabelecidas por clientes e usuários que definem as diversas propriedades do sistema. Os requisitos de software são, obviamente, aqueles dentre os requisitos de sistema que dizem respeito a propriedades do software (MACORATTI.NET, 1999, p. 1).

Segundo Perry (1988, p. 115), os testes em um processo de desenvolvimento de software deveriam ser iniciados junto à fase de levantamento de requisitos. Isso porque é neste momento que as mais importantes decisões do projeto são tomadas. Os requisitos são a base para o projeto do sistema, que então é utilizado no desenvolvimento do produto final. Desta forma, caso os requisitos contenham erros, a aplicação também estará errada.

A forma como os requisitos são descritos pode contribuir ou não para a qualidade do produto final. Trata-se de uma das fases mais críticas do desenvolvimento de um novo produto.

É amplamente divulgado que requisitos incompletos, escritos de maneira pobre e mal comunicados são responsáveis por algo em torno de 50 a 70% das falhas de projetos de software. Pesquisas na indústria mostram que a causa raiz de 56% de todos os erros identificados em projetos são introduzidos na fase de levantamento de requisitos (MKS,

2009).

Perry (1988, p. 115) afirma que uma das entregas do levantamento de requisitos deve ser uma descrição detalhada da solução recomendada, destacando o método indicado para satisfazer as necessidades, e ainda lembra que esta é a entrada para a fase de projeto do software.

Os requisitos funcionais são a descrição das diversas funções que clientes e usuários querem ou precisam que o software faça. Eles definem a funcionalidade desejada do software. O termo função é usado no sentido genérico de operação que pode ser realizada pelo sistema, seja através de comandos dos usuários ou pela ocorrência de eventos internos ou externos ao sistema (MACORATTI.NET, 1999, p. 1).

2.2 TESTES BASEADOS EM REQUISITOS

Tradicionalmente, o teste de software é definido como o processo de execução de um programa com a intenção de encontrar erros. Está muito claro que esta definição é inadequada para sistemas complexos. Pode-se definir o teste de software como uma parte do processo de desenvolvimento de software que deveria estar presente em todos os seus estágios, buscando verificar se o produto construído está de acordo com as necessidades dos usuários (RAMACHANDRAN, 1996, p. 1-2).

Os testes baseados em requisitos visam verificar se o sistema executa corretamente as funcionalidades e se é capaz de sustentar esta correção após a sua utilização por um período de tempo contínuo. Os testes de requisitos devem ser considerados concluídos após os programas tornarem-se operacionais, embora os requisitos possam ser testados individualmente durante as fases anteriores do ciclo de vida (RIOS et al., 2007, p. 58).

Os testes baseados em requisitos buscam demonstrar que o sistema atende às necessidades e, portanto, tem um valor. Estes testes são considerados os mais importantes, pois apenas eles garantem que o software está fazendo algo realmente útil (GOLDSMITH, 2009, p. 1).

Segundo Mogyorodi (2008, p. 1), o RBT (*Requirement Based Testing* – Teste baseado em requisitos) é um processo rigoroso utilizado para melhorar a qualidade dos requisitos, para derivar o menor número possível de casos de testes e cobrir 100% dos requisitos. O RBT é composto por duas técnicas, sendo a primeira uma revisão de ambigüidades (*ambiguity*

review) na definição dos requisitos e a segunda a construção de um grafo de causa e efeito (MOGYORODI, 2008, p. 1).

O processo RBT estabiliza a definição da interface da aplicação mais cedo, porque os requisitos para a interface com o usuário se tornam bem definidos e são escritos de forma não ambígua e testável. Isto permite o uso de ferramentas de captura/reprodução mais cedo no ciclo de desenvolvimento de software.

A revisão de ambigüidades é usada na fase de requisitos para identificar pontos passíveis de erros devido à falta de concisão. A intenção dela é identificar pontos obscuros ou imprecisos nos requisitos.

O grafo de causa e efeito é uma técnica de *design* de casos de testes que deve ser utilizada após os requisitos terem passado pela revisão de ambigüidade e após o conteúdo dos requisitos terem sido aprovados. A técnica de grafo de causa e efeito é vista mais adiante, no tópico 2.4.

2.3 TÉCNICAS PARA GERAÇÃO DE CASOS DE TESTES

Existem muitas técnicas para geração de casos de testes a partir de requisitos funcionais, quando estes requisitos são expressos em notações formais e precisas, tais como a notação Z ou especificações algébricas. Apesar disso, a maior parte da indústria de software trabalha com requisitos escritos em linguagem natural (GUTIÉRREZ et al., 2006, p. 1).

Por outro lado, existem técnicas bastante utilizadas para transformar requisitos em casos de testes que validam se as condições documentadas na especificação foram realmente atendidas pelo sistema.

Srivastava, Patel e Chatrola (2009, p. 1) afirmam que testes são geralmente descritos como um grupo de procedimentos que servem para avaliar alguns aspectos de um pedaço de um software. Testes também podem ser descritos como um processo para revelar defeitos no software ou para estabelecer que o software possua um grau específico de qualidade. Eles cobrem atividades tanto de validação quanto de verificação. O domínio dos testes inclui: revisões técnicas, testes de unidade, de integração, de sistema, de aceitação, de usabilidade, entre outros.

Existem basicamente dois tipos de testes, sendo o de caixa branca, onde o foco dos casos de testes desenvolvidos é cobrir as condições apresentadas no código fonte do sistema, e

o de caixa preta, no qual o foco é testar os requisitos funcionais da aplicação, sem se preocupar com a sua lógica interna (SRIVASTAVA; PATEL; CHATROLA, 2009, p. 1).

Os testes da caixa preta são chamados de testes funcionais por estarem baseados nos requisitos funcionais do software. Espera-se poder verificar aquilo que se pretende que o programa faça. Exemplos de técnicas de teste de caixa preta são os grafos de causa-efeito, testes baseados em tabela de decisão e teste do valor limite (LEITE, 2000, p.1).

O teste de caixa preta permite verificar a validade do software em relação aos requisitos. Por exemplo, suponha que o cliente solicitou que o software calculasse o valor médio ponderado das três notas dos alunos. O desenvolvedor construiu uma função que calcula a média aritmética. Os testes de caixa preta devem permitir verificar que a função desenvolvida não é a função correta, utilizando como entrada para o teste valores que não resultam na mesma nota caso a média calculada for aritmética, e não a ponderada.

De acordo com os requisitos, a média deve ser calculada pela fórmula média = (a*4 + b*5 + c*6)/15. Desta forma, para o caso de teste cujos valores sejam {a = 2, b=3, c=4}, o valor esperado é 3,1333, e para o caso {a = 5, b=6, c=8}, o valor esperado é 6,5333.

Aplicando alguma técnica de teste da caixa preta chega-se ao valor média = 3 para o primeiro caso de teste e média = 6,333 para o segundo, uma vez que o desenvolvedor implementou uma função que calcula a média aritmética e não a média ponderada.

Algumas técnicas de testes de caixa preta são:

a) grafos de causa e efeito, utilizados para aplicações que precisam de testes intensivos para controles, onde são desenvolvidos testes utilizando dados que representam a entrada de dados no software (as causas) e as ações ou saídas correspondentes (os efeitos);
b) particionamento de equivalência, onde os dados de entrada e saída do teste são divididos em classes e apenas alguns valores pertencentes a cada classe são utilizados nos testes;
c) valores limite, onde os valores utilizados como entrada no software são selecionados de acordo com os valores mínimo e máximo possíveis para a aplicação.

As técnicas de particionamento de classes de equivalência e valores limites são utilizadas no momento da geração da tabela de decisão para limitar o número de entradas geradas, bem como potencializar a chance de expor os erros ao executar os testes.

Os testes gerados a partir da técnica não deverão ser considerados como imutáveis ou fechados. Mais testes poderão ser adicionados pelo analista de testes, bem como alterações

nos testes gerados deverão ser permitidas. Mudanças na especificação dos requisitos poderão requerer que os testes sejam gerados novamente, buscando cobrir itens incluídos e tratar situações ainda não previstas no momento da primeira geração.

A seguir é apresentada uma descrição mais detalhada de cada uma destas técnicas.

2.3.1 Grafo de causa e efeito

O grafo de causa e efeito (*Cause-Effect Graphing* - CEG) é um método utilizado para derivar casos de testes a partir de uma especificação descrita em linguagem natural para validar sua respectiva implementação (NURSIMULU; PROBERT, 1995, p. 1).

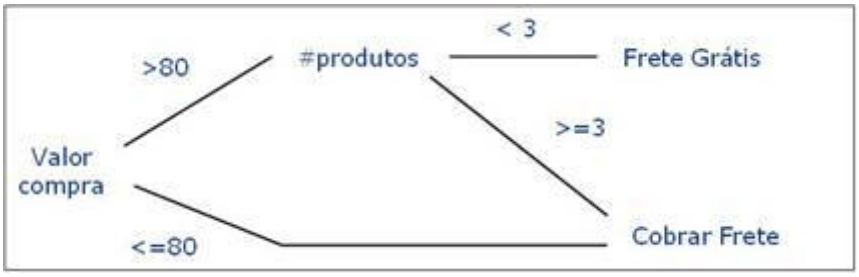
Na análise do CEG, primeiramente são identificadas as causas, os efeitos e as restrições a partir de uma especificação realizada em linguagem natural. Na seqüência é desenvolvido um grafo representado por nós (as causas e os efeitos) e arestas (ligando as causas aos seus respectivos efeitos) marcadas com sinais que representam as restrições. Restrições são formadas por operadores lógicos como AND, OR e NOT. Em seguida, este grafo é convertido em uma tabela de decisão, que por sua vez pode ser convertida para casos de uso e casos de testes (SRIVASTAVA; PATEL; CHATROLA, 2009, p. 1).

Uma causa é qualquer condição nos requisitos que pode afetar as saídas do programa. Um efeito é a resposta do programa para alguma combinação de condições de entrada.

De acordo com Leite (2000, p1), a técnica oferece uma representação concisa das condições lógicas e das ações correspondentes. A técnica segue quatro passos:

a) causas (condições de entrada) e efeitos (ações) são relacionados para um módulo e um identificador é atribuído a cada um;
b) um grafo de causa-efeito (descrito a seguir) é desenvolvido;
c) o grafo é convertido numa tabela de decisão;
d) as regras da tabela são convertidas em casos de teste.

A figura 1 exemplifica um grafo de causa e efeito para uma regra sobre a cobrança ou não de frete de acordo com o valor da compra.



Fonte: Dias Neto (2010, p. 1).

Figura 1 - Exemplo de grafo de causa e efeito (CEG)

Para o grafo apresentado na figura 1, podemos gerar a tabela de decisão conforme tabela 1.

Valor compra		**#produtos**		**Frete grátis**	**Cobrar frete**
> 80	**<=80**	**<3**	**>=3**		
V	F	V	F	V	F
V	F	F	V	F	V
F	V	V	F	F	V
F	V	F	V	F	V

Tabela 1 – Exemplo de tabela de decisão

2.3.2 Particionamento de classes de equivalência

Exceto para as mais triviais aplicações de software, normalmente é considerado impossível testar todas as combinações de entradas logicamente viáveis para um sistema de software. Por conseguinte, a seleção de um bom subconjunto que tenha a maior probabilidade de encontrar a maioria dos erros é uma tarefa importante e útil para os testadores executarem (ECLIPSE PROCESS FRAMEWORK COMPOSER, 2008, p. 1).

Os testes baseados na análise de classes de equivalência são uma forma de análise de teste de caixa preta que tenta reduzir a quantidade total de testes potenciais a um conjunto mínimo que irá descobrir a maioria dos erros possíveis. Trata-se de um método que decompõe o conjunto de entradas e saídas em uma quantidade finita de classes de equivalência que permitem a escolha de um valor de teste representativo para cada classe. O teste que resulta do valor representativo para uma classe é chamado de "equivalente" para os outros valores na mesma classe. Se nenhum erro for encontrado no teste de valor representativo, conclui-se que todos os outros valores equivalentes também não identificarão nenhum erro.

O poder das classes de equivalência reside na sua capacidade de orientar o testador a usar de uma estratégia de amostragem para reduzir a explosão combinatória dos testes

potencialmente necessários. A técnica fornece uma base lógica através da qual um subconjunto do total concebível de testes pode ser selecionado. Aqui estão algumas categorias de áreas problemáticas para uma grande gama de testes que podem se beneficiar da consideração das classes de equivalência:

a) combinação de variáveis independentes;
b) variáveis dependentes baseadas em relacionamentos hierárquicos;
c) variáveis dependentes baseadas em relacionamentos temporais;
d) relacionamentos agregados baseados em exemplares de mercado;
e) relacionamentos complexos que podem ser modelados.

A teoria da decomposição de equivalência como proposto por Glenford Myers (MYERS, 1979) tenta reduzir a quantidade total de casos de teste necessários pela decomposição das condições de entrada em uma quantidade finita de classes de equivalência. Dois tipos de classe de equivalência estão classificados: o conjunto de entradas válidas para o programa é considerado como *classe de equivalência válida*, e todas as outras entradas são incluídas na *classe de equivalência inválida*.

Aqui está um conjunto de diretrizes para identificar classes de equivalência (ECLIPSE PROCESS FRAMEWORK COMPOSER, 2008, p. 1):

a) se uma condição de entrada especifica um conjunto de valores (tal como, o programa "aceita valores entre 10 e 100"), então uma classe de equivalência válida (entre 10 e 100) e duas classes de equivalência inválidas (inferior a 10 e superior a 100) são identificadas;
b) se uma condição de entrada especifica um conjunto de valores (tais como, "o pano pode ser de várias cores: VERMELHO, BRANCO, PRETO, VERDE e MARROM"), então uma classe de equivalência válida (os valores válidos) e uma classe de equivalência inválida (todos os outros valores inválidos) são identificadas. Cada valor de classe de equivalência válida deve ser tratado distintamente;
c) se a condição de entrada for especificada como uma situação "dever ser" (tal como, "o texto de entrada deve estar em caixa alta"), então uma classe de equivalência válida (caracteres maiúsculos) e uma classe de equivalência inválida (todas as outras entradas, exceto caracteres maiúsculos) são identificadas;
d) tudo que acabou "com longa duração" antes da tarefa ser feita é uma classe de equivalência. Tudo que foi feito em um curto intervalo de tempo antes do programa terminar é outra classe. Tudo que foi feito exatamente antes do programa

começar outra operação é outra classe;

e) se for especificado que um programa deve trabalhar com tamanho de memória entre 64M e 256M. Então esta dimensão de tamanho é uma classe de equivalência. Qualquer outro tamanho de memória, que seja maior que 256M ou menor que 64M, não pode ser aceito.

A decomposição do evento de saída depende das entradas do programa. Mesmo que diferentes classes de equivalência de entrada possam ter o mesmo tipo do evento de saída, as classes de equivalência de entrada devem ser tratadas distintamente.

2.3.3 Análise de valor limite

Em cada uma das classes de equivalência, considera-se que as condições limítrofes tenham uma maior taxa de sucesso na identificação de resultados com falhas do que as condições não limítrofes. As condições limítrofes são os valores, imediatamente acima ou abaixo dos limites de cada classe de equivalência (ECLIPSE PROCESS FRAMEWORK COMPOSER, 2008, p. 1).

Os testes que resultam das condições limítrofes fazem uso dos valores, mínimo (min), logo abaixo do mínimo (min+), logo acima do máximo (max-), e máximo (max) do intervalo que precisa ser testado. Ao testar valores limítrofes, os testadores escolhem alguns casos de teste para cada classe de equivalência. Para a amostra de testes relativamente pequena, a probabilidade de descoberta de falha é elevada. É dado ao testador algum alívio no ônus de testar uma enorme quantidade de casos em uma classe de equivalência de valores que não são susceptíveis de produzir grandes diferenças nos resultados dos testes.

Algumas recomendações ao escolher valores limítrofes:

a) para uma variável de ponto flutuante, se a sua condição válida for entre `-1,0` e `1,0`, testar `-1,0`, `1,0`, `-1,001` e `1,001`;

b) para um inteiro, se a gama de entrada válida for entre `10` e `100`, testar `9`, `10`, `100 e 101`;

c) se um programa espera uma letra maiúscula, testar os limites A e Z. Teste `@` e `[` também, porque no código ASCII, `@` está logo abaixo de A e `[` está logo depois de Z;

d) se a entrada ou saída de um programa é um conjunto ordenado, preste atenção no

primeiro e no último elemento do conjunto;

e) se a soma das entradas tiver que ser um número específico (`n`), testar o programa onde a soma seja `n-1`, `n e n+1`;

f) se o programa aceita uma lista, testar os valores da lista. Todos os outros valores são inválidos;

g) ao ler ou escrever em um arquivo, verificar o primeiro e o último caractere do arquivo;

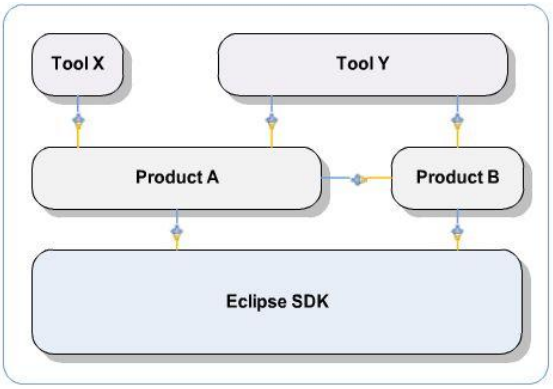
h) o menor valor nominal do dinheiro é um centavo ou o equivalente. Se o programa aceita um determinado intervalo, entre a e b, testar a `-0,01` e b `+0,01`;

i) para uma variável com vários conjuntos de valores, cada conjunto de valores é uma classe de equivalência. Se os subintervalos não estiverem sobrepostos, testar os valores dos limites, logo acima do limite superior e logo abaixo do limite inferior.

2.4 DESENVOLVIMENTO DE EXTENSÕES PARA A FERRAMENTA ECLIPSE

O Eclipse não é um único e enorme programa escrito em Java, mas sim um pequeno programa que provê a funcionalidade típica de um *"*carregador de componentes*"* chamado *Plug-in loader*. O Eclipse é cercado por centenas de milhares de *plug-ins* (ou componentes), que não são nada mais do que programas Java estendendo alguma das suas funcionalidades. Cada um destes componentes pode tanto consumir serviços providos por outro *plug-in* quanto disponibilizar uma funcionalidade para ser consumida por outros componentes (ECLIPSE PLUGIN DEVELOPMENT, 2008).

A figura 2 exemplifica as possibilidades providas pela plataforma, onde os componentes *Product A* e *Product B* utilizam *extensions* para construir alguma funcionalidade adicional para a plataforma. Os componentes *Tool X* e *Tool Y* utilizam *extension points* do *Product A* para adicionar mais alguma funcionalidade específica.



Fonte: Eclipse Plugin Development (2008, p. 1).
Figura 2 – Possibilidades de extensão do Eclipse

O Eclipse utiliza os conceitos de *extension point* e *extensions*, disponibilizando classes bases que podem ser estendidas com implementações de funcionalidades mais específicas. Por exemplo, é possível construir um editor de XML estendendo um editor padrão de texto, onde é necessário desenvolver apenas os tratamentos específicos para XML (ECLIPSE PLUGIN DEVELOPMENT, 2008).

Originalmente, a plataforma *Eclipse* foi projetada para funcionar como uma plataforma de ferramentas abertas. A partir do *Eclipse* 3.0, sua arquitetura foi reformulada para que seus componentes pudessem ser usados para construir praticamente qualquer aplicação cliente. O conjunto mínimo de *plug-ins* necessário para construir uma aplicação *rich client* é coletivamente conhecido como RCP. Essas aplicações ricas ainda são baseadas em um modelo de *plug-in* dinâmico e a interface gráfica com o usuário (*Graphical User Interface* – GUI) é construída usando os mesmos *kits* de ferramentas e pontos de extensão. No entanto, a diferença chave é que o ambiente de trabalho está sob controle de baixa granularidade do desenvolvedor do *plug-in* com aplicações RCP. Observe que o *Eclipse IDE* é, por si só, uma aplicação RCP (MINOCHA, 2006, p. 1).

2.4.1 Arquitetura RCP

O RCP emprega uma estrutura leve de componentes de software baseada em *plug-ins*. Esta arquitetura fornece extensibilidade e integração sem emendas. Tudo no RCP (e no Eclipse, de forma geral), com exceção do seu núcleo de execução, é um *plug-in*. O RCP é arquitetado de modo que seus componentes possam ser unidos para construir qualquer aplicação cliente, usando um modelo de encaixe dinâmico, conjuntos de ferramentas, e pontos de extensão. O gerenciador de leiautes e o gerenciamento de espaço de trabalho estão sob o

controle do desenvolvedor de *plug-ins*.

Os seguintes componentes constituem o RCP:

a) *Core platform*: responsável por descobrir e executar dinamicamente os plug-ins desenvolvidos;
b) *Standard Widget Toolkit* (SWT): biblioteca gráfica desenvolvida para renderizar os componentes de tela;
c) JFace: camada que trabalha sobre o SWT, abstraindo algumas funções básicas para o desenvolvedor;
d) Eclipse *Workbench*: representa o ambiente básico de desenvolvimento do Eclipse. O *workbench* do Eclipse é cercado por componentes que representam a sua identidade visual, utilizando os componentes *Perspectives*, *Views*, *Editors*, *Projects* e *Workspaces*.

2.4.2 Componentes de visualização e edição – *Views* e *Editors*

O conceito fundamental de interface organizada em blocos do Eclipse consiste em *views* e *editors*. A extensibilidade do Eclipse permite que o desenvolvedor crie ou estenda as *views* e os *editors* que já acompanham a plataforma. O desenvolvimento de *views* é facilitado através de *frameworks* e *extension points*, onde o desenvolvedor se concentra na construção do conteúdo da *view*, deixando o comportamento padrão a cargo da plataforma (SHAVOR, 2003, p. 315-317).

A figura 3 apresenta a estrutura de classes de uma view. Ao desenvolver uma nova visualização, é necessário estender as classes *ViewPart* e *ContentProvider*. A primeira é responsável pelos aspectos visuais da interface, enquanto a segunda fornece os dados que devem ser exibidos. Ao estender a classe *LabelProvider*, é possível alterar como o conteúdo é apresentado na visualização.

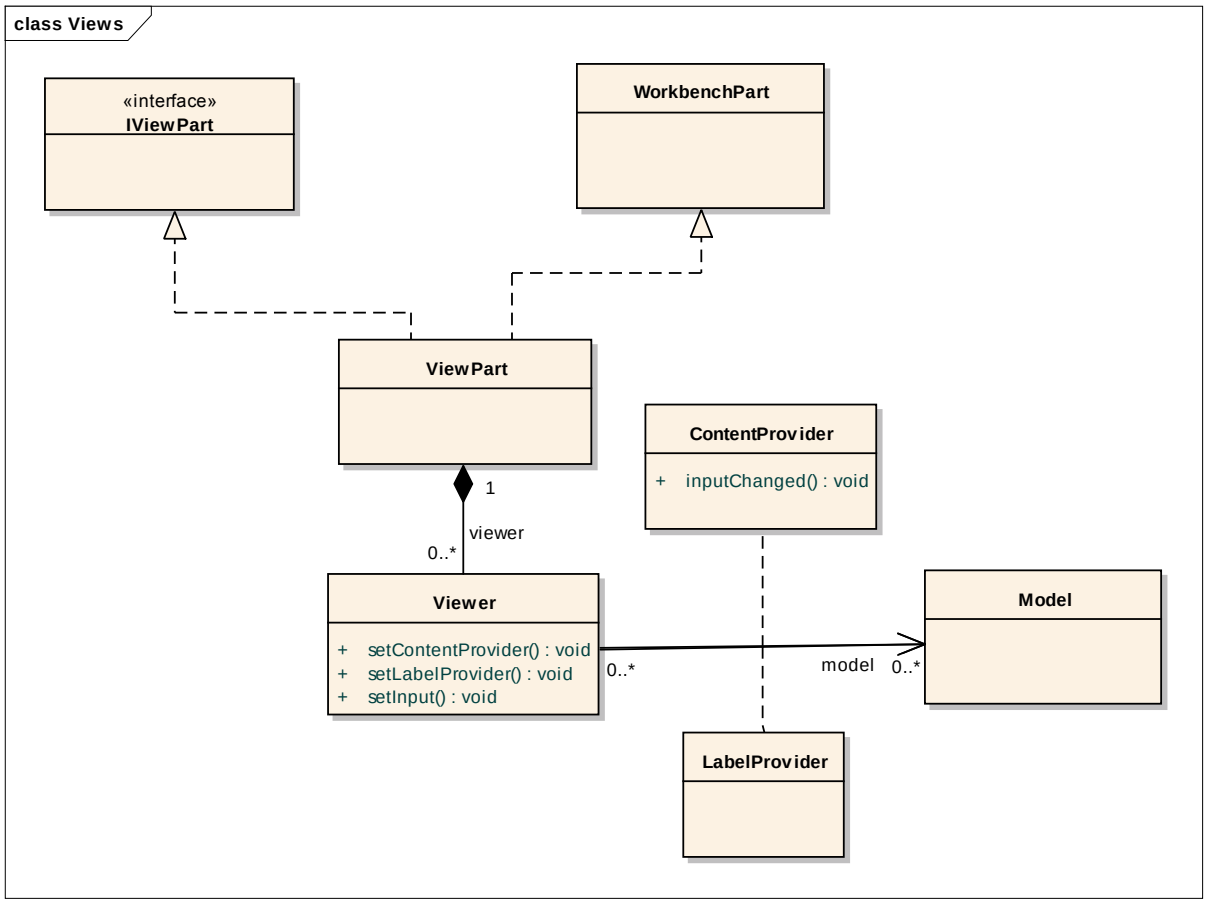


Figura 3 – Arquitetura de funcionamento das *Views*

2.5 TRABALHOS CORRELATOS

Foram pesquisadas algumas ferramentas que se propõe a realizar a geração de casos de teste de forma automatizada. Por outro lado, existe uma ampla gama de ferramentas que propõem auxiliar na construção e organização dos casos de testes. Dentre elas podem ser citadas a SilkCentral Test Manager, a TaRGeT e a TestLink.

2.5.1 SilkCentral Test Manager

É um ambiente integrado para construção e execução de testes baseados em requisitos (BORLAND SOFTWARE CORPORATION, 2010). Os testes são escritos a partir de requisitos, porém estes são escritos e mantidos por uma segunda ferramenta, chamada CaliberRM.

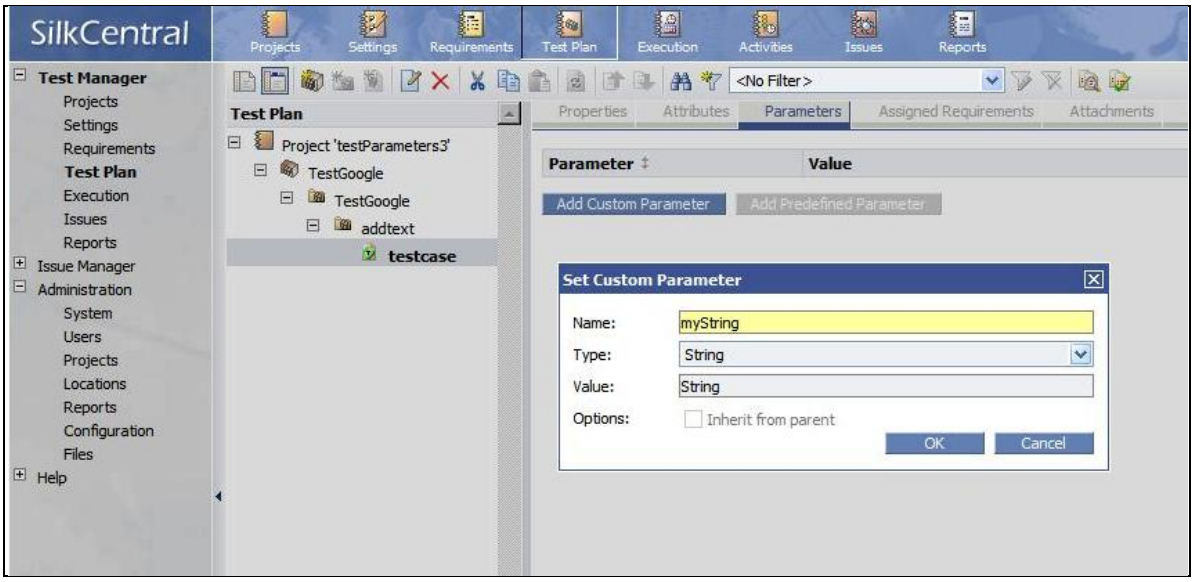


Figura 4 – Ferramenta SilkCentral Test Manager

A figura 4 ilustra a interface da ferramenta. Através da interface do SilkCentral é possível: acompanhar o andamento das execução dos casos de testes, manuais ou automatizados; disparar execuções de scripts de testes; obter relatórios gerencias; verificar em detalhes as falhas relatadas.

2.5.2 TaRGeT

A TaRGeT (*Test and Requirements Generation Tool*) tem como objetivo a criação de cenários de teste que estão diretamente ligados ao documento de casos de uso do sistema. Esta ferramenta é derivada da técnica de automação de testes baseada em modelos (*Based-Model Testing*), na qual os casos de testes são gerados a partir da definição de um modelo formal do sistema a ser testado (BORBA; SILVA, 2010).

Com o intuito de diminuir o esforço na criação de cenários de teste com base nos requisitos, foi definido para a TaRGeT um modelo para descrever os casos de uso do sistema. No modelo existem campos referentes às ações do usuário, resposta do sistema e pré-condições para cada passo do caso de uso. É possível também a criação de fluxos alternativos a partir do reuso de passos já mapeados.

Com o uso da TaRGeT é possível criar casos de teste que são combinações dos passos descritos no modelo formal de caso de uso aceito pela ferramenta. Esses testes podem ser gerenciados pela interface do sistema e suítes podem ser criadas com os testes selecionados com base nos critérios de cobertura, requisitos exercitados entre outros filtros.

A ferramenta possui dois componentes principais:

a) `TaRGeT Test Case Generation`: realiza a geração de suítes de teste que podem ser exportadas em diferentes formatos, além de possibilitar a extensão dessa funcionalidade para implementar novos formatos de saída.

b) `TaRGeT On The Fly Generation`: plug-in responsável por gerenciar os casos de testes gerados pelo TaRGeT Test Case Generation. Esse gerenciamento consiste em criar suítes aplicando filtros de seleção e exportando para vários formatos como Excel e XML.

A figura 5 ilustra a interface da ferramenta estudada.

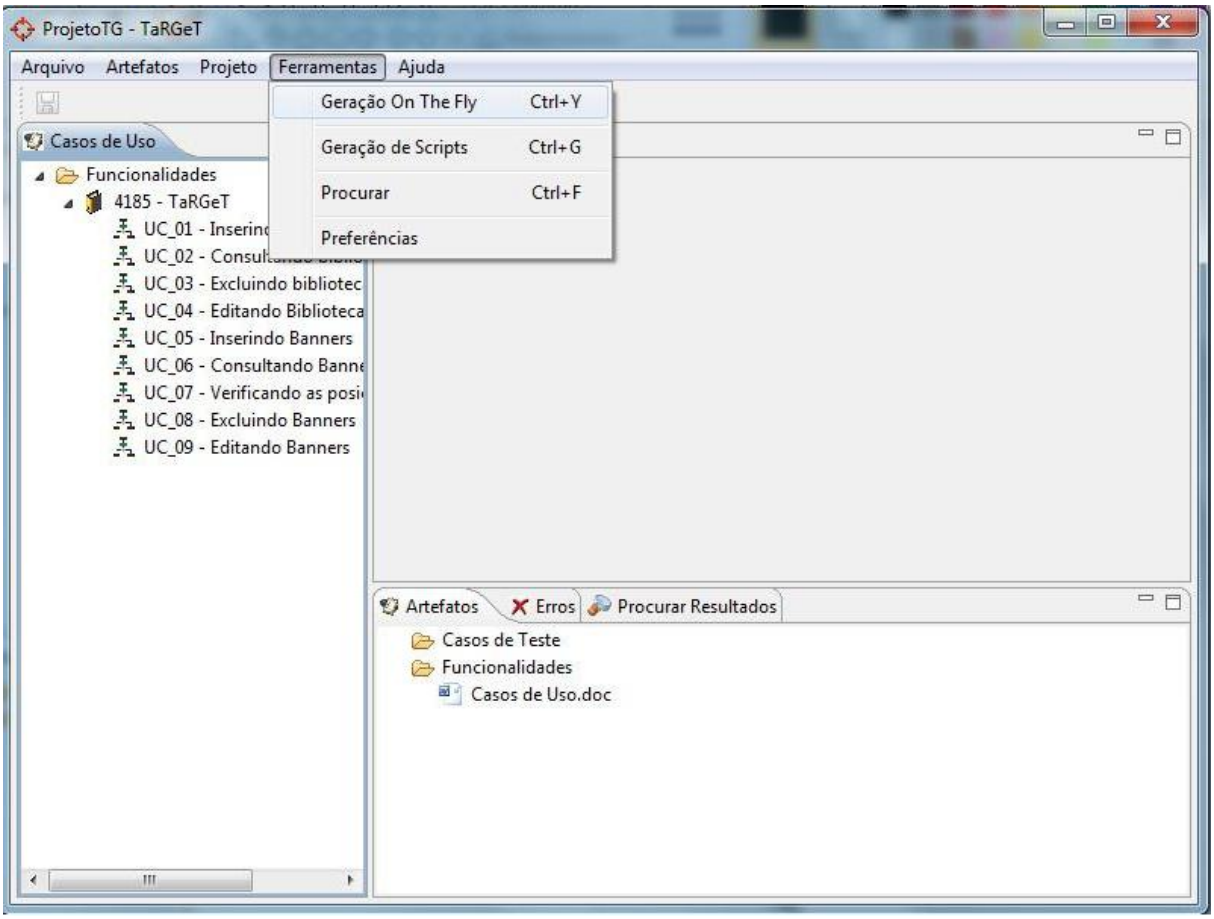


Figura 5 – Registro de casos de uso da ferramenta TaRGeT

2.5.3 TestLink

Ferramenta que se propõe a organizar e gerenciar testes em suítes[1], bem como auxiliar

[1] Uma suíte de testes representa um conjunto de testes que possuem o mesmo objetivo.

na execução dos testes e auxiliar no planejamento de execução de testes (TESTLINK COMMUNITY, 2010). O TestLink não auxilia na geração de casos de testes, apenas auxilia em sua organização e execução.

A figura 6 ilustra o formulário para cadastro de um caso de teste na ferramenta.

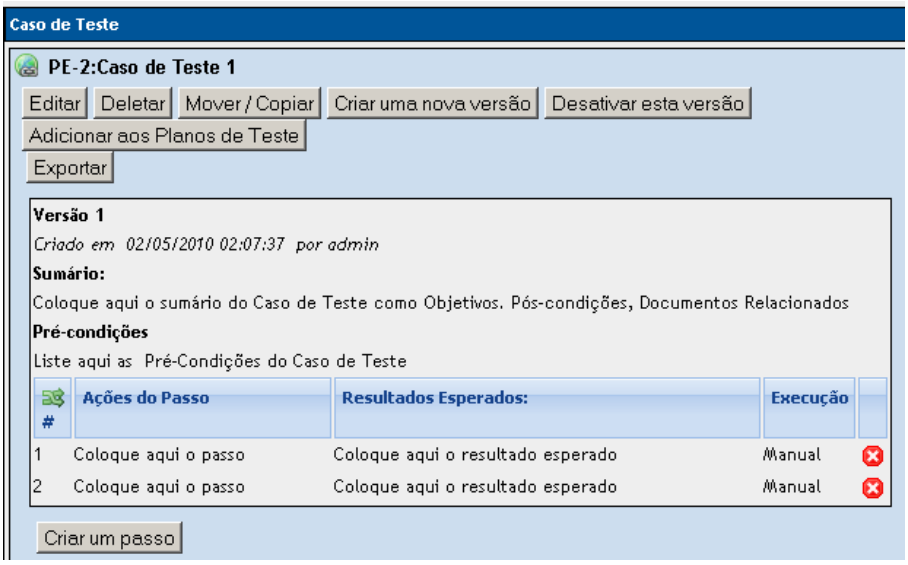


Figura 6 – Documentação de caso de testes no TestLink

O TestLink permite a criação de projetos de testes, que encapsulam suítes de testes, que por sua vez agrupam os casos de testes. Os casos de testes podem ser mapeados para os requisitos, porém a ferramenta não obriga que esta estruturação seja realizada. Os requisitos também podem ser cadastrados pela ferramenta.

A ferramenta também possui assistente de execução de testes, que permite a utilização por mais de um usuário ao mesmo tempo. A execução dos testes pode ser direcionada para um testador específico, associando ele a um caso de teste.

São disponibilizados também relatórios de execução e gerenciamento dos testes.

3 DESENVOLVIMENTO DA FERRAMENTA

O capítulo 3.1 lista os principais requisitos do problema a ser trabalhado. O capítulo 3.2 apresenta especificação da ferramenta desenvolvida. No capítulo 3.3 são apresentados detalhes do desenvolvimento, tais como técnicas e ferramentas utilizadas (capítulo 3.3.1) e a operacionalidade da implementação (capítulo 3.3.2). Por fim, o capítulo 3.4 apresenta os resultados obtidos através do desenvolvimento deste trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Considerando o estudo sobre RBT e os trabalhos correlatos definiu-se um conjunto de requisitos para a ferramenta:

a) permitir a descrição de requisitos funcionais e não funcionais (Requisito Funcional - RF) ;
b) permitir a criação de marcações no texto dos requisitos de modo que propriedades sejam criadas e quantificadas. Cada marcação poderá ter quantas propriedades forem necessárias (RF);
c) permitir a criação de casos de testes de forma que seja mantida a rastreabilidade para os requisitos (RF);
d) automatizar parte da geração de casos de testes, com base em propriedades definidas dentro dos requisitos e no grafo de causa e efeito (RF);
e) apresentar uma matriz de rastreabilidade que indique os requisitos testados e não testados (RF);
f) permitir a execução, passo a passo, dos casos de testes selecionados e armazenar os resultados para posterior avaliação (RF);
g) agrupar os casos de testes em suítes de testes definidas pelo usuário (RF);
h) permitir a criação de um plano de testes baseado nas escolhas do analista de testes e nos requisitos alterados durante o processo de manutenção (RF);
i) permitir a geração de um relatório de falhas encontradas durante a execução dos testes (RF);
j) permitir a geração de um relatório que liste os testes executados e o resultado de

cada um deles (RF);

k) deverá ser desenvolvido em Java (RNF);

l) deverá ser um *plug-in* para o ambiente de desenvolvimento Eclipse (RNF);

m) deverá utilizar o framework XStream 1.3.1 para persistência dos elementos em arquivos XML.

3.2 ESPECIFICAÇÃO

A seção 3.2.1 apresenta os passos necessários para a geração de casos de testes a partir dos requisitos. A seção 3.2.2 apresenta diagramas de atividades construídos para documentar como será o processo de utilização da ferramenta. A seção 3.2.3 apresenta os casos de uso atribuídos aos atores analistas de sistemas e analista de testes. A seção 3.2.4 apresenta diagramas de classes com os pontos principais do desenvolvimento. A seção 3.2.5 apresenta os diagramas de seqüência para as funções de registro de requisitos e geração de casos de testes.

3.2.1 Definição do processo para geração de casos de testes a partir de requisito

A seguir é descrito um esboço dos passos necessários para realizar a geração dos casos de testes a partir dos requisitos descritos pelo analista de sistemas. É demonstrado um processo prático desde a documentação inicial do requisito até o final onde os casos de testes são gerados pela ferramenta.

3.2.1.1 Descrição do requisito

Ao iniciar o processo de desenvolvimento de uma nova funcionalidade para um software já existente ou a produção de um novo software, o primeiro passo é definir os seus requisitos, de modo que fique claro o que deve ser entregue ao final do processo de desenvolvimento. Neste exemplo, será descrito um requisito para um software de testes baseados em requisitos.

RF0001 - O sistema deve permitir que seja realizado o registro de laudos de testes. O sistema deve solicitar que o usuário informe um número identificador para o laudo, uma data de início e uma data de fim para os testes. O sistema deverá calcular o valor pago pelo laudo utilizando a seguinte regra: - Caso a quantidade de semanas utilizadas for igual a um, então tarifa = 100 por semana; - Caso a quantidade de semanas utilizadas for maior do que um e menor do que cinco, então a tarifa = 75 por semana; - Caso a quantidade de semanas utilizadas for maior do que cinco semanas, então tarifa = 50 por semana.

Quadro 1 – Requisito exemplo

Um requisito, para ser considerado bem descrito, não deve conter ambigüidades e deve estar devidamente quantificado. Para garantir estas propriedades, o RBT propõe uma etapa chamada revisão de ambigüidades. Esta etapa não faz parte do escopo deste trabalho. Será assumido que o requisito está bem descrito.

3.2.1.2 Elencar palavras chaves (propriedades)

A etapa seguinte consiste em elencar quais são as palavras chaves do requisito descrito. Esta etapa deve ser realizada pelo analista de sistemas, pois é ele quem detém as informações necessárias para tal. Estas são utilizadas como propriedades, que podem expressar entidades que o software deverá controlar, bem como condições lógicas que deverão ser atendidas.

Ao analisar o requisito, foram grifadas as palavras consideradas como chaves para o exemplo.

R0001 - O sistema deve permitir que seja realizado o registro de laudos de testes. O sistema deve solicitar que o usuário informe um **número identificador** para o laudo, uma **data de início** e uma **data de fim** para os testes. O sistema deverá calcular o **valor pago** pelo laudo utilizando a seguinte regra: - Caso a quantidade de **semanas** utilizadas for igual a um, então **tarifa = 100** por semana; - Caso a quantidade de **semanas** utilizadas for maior do que um e menor do que cinco, então a **tarifa = 75** por semana; - Caso a quantidade de **semanas** utilizadas for maior do que cinco semanas, então **taria = 50** por semana.

Quadro 2 – Palavras chaves grifadas

As propriedades elencadas não representam necessariamente todas as existentes no requisito, e desta forma fica a critério do analista o nível de cobertura de testes desejada. As palavras grifadas representam as causas e os efeitos do requisito.

3.2.1.3 Quantificando as propriedades

Em seguida, as propriedades elencadas para o requisito devem ser quantificadas e qualificadas. Esta etapa é importante para a geração dos casos de testes.

O quadro 3 apresenta uma parte das palavras chaves já com suas propriedades definidas.

Número identificador Pode ficar nulo: Não Intervalo Menor número possível: 1 Maior número possível: 999999 Contagem:1 **Data de início** Pode ficar nulo: Não Intervalo Menor data possível: 31/12/1950 Maior data possível: 31/12/2050 Tipo de intervalo: DIAS Contagem: 1

Quadro 3 – Exemplo de propriedades quantificadas e qualificadas

No modelo apresentado, cada palavra chave contém propriedades, que ajudam a definir de forma mais completa e clara o escopo do problema a ser resolvido. Estas palavras chaves representam dados de entrada ou saída para o software que deve ser testado. A definição clara e concisa destas informações já no requisito permite que os testes sejam escritos mais cedo e reduz a necessidade de revisão mais tarde.

3.2.1.4 Montando o grafo de causa e efeito

Após visualizar as palavras chaves/propriedades do requisito, é possível prosseguir com a montagem do grafo de causa e efeito. Este grafo apresenta uma visão lógica do relacionamento existente entre as palavras chaves e suas propriedades, além de auxiliar na visualização de possíveis itens faltantes no requisito.

A figura 7 apresenta um grafo de causa e efeito montado para o problema citado no requisito estudado.

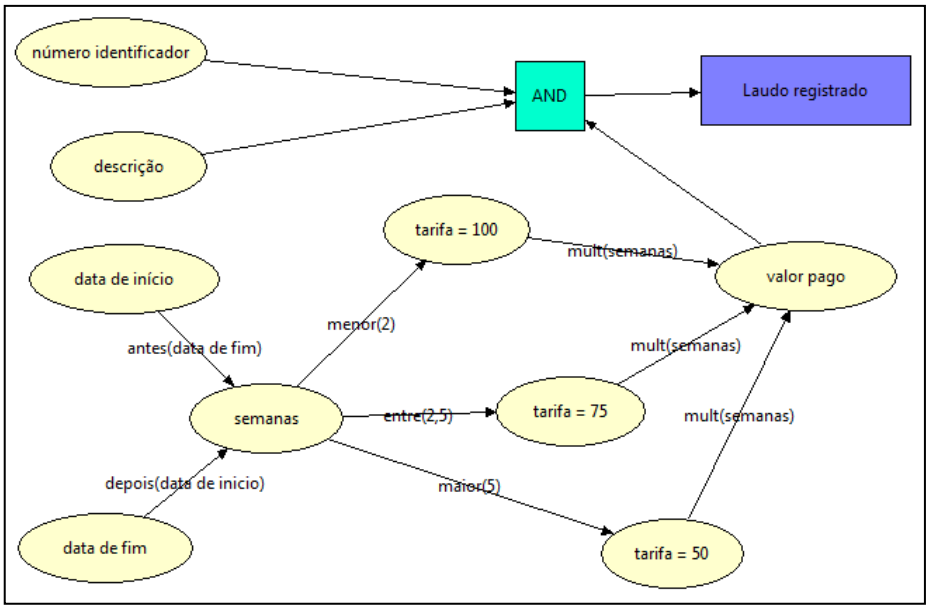


Figura 7 – Grafo de causa e efeito

3.2.1.5 Geração dos casos de testes

O grafo de causa e efeito e as propriedades definidas para o requisito são utilizados como insumos para a geração dos casos de testes. Os casos de testes são combinações de entradas que podem ser usadas para validar se o software possui o conjunto mínimo de funcionalidades esperadas e se estas funcionalidades foram implementadas corretamente do ponto de vista do usuário.

A tabela 2 apresenta alguns exemplos de casos de testes gerados com tais informações, com o foco apenas para o sub-grafo que define qual a taxa deve ser cobrada na emissão do laudo.

Neste momento as técnicas de particionamento de classes de equivalência e valores limite são aplicadas nas propriedades e no modelo criado. Estas técnicas funcionam como uma espécie de filtro para a geração de casos de testes, fazendo com que uma quantidade mínima de testes seja gerada para cobrir a maior quantidade possível de situações de teste.

Tabela 2 – Casos de testes gerados pela ferramenta

Id do caso de teste	Data de Início	Data de fim	Valor da taxa
#01	20/11/2010	27/11/2010	R$ 100,00
#02	20/11/2010	28/11/2010	R$ 150,00
#03	20/11/2010	30/12/2010	R$ 450,00

A tabela gerada é formada por uma identificação, as entradas e em seguida as saídas esperadas do sistema para cada caso de teste. Quanto maior o número de informações

adicionadas às propriedades das palavras chaves, maior o detalhamento dos testes ao final do processo.

3.2.2 Diagrama de atividades

A seguir, serão apresentados os diagramas de atividades para os processos de geração de casos de testes, montagem do plano e execução dos testes.

3.2.2.1 Geração de casos de testes

A figura 8 ilustra a seqüência de atividades executadas pelo usuário da ferramenta ao criar um novo requisito.

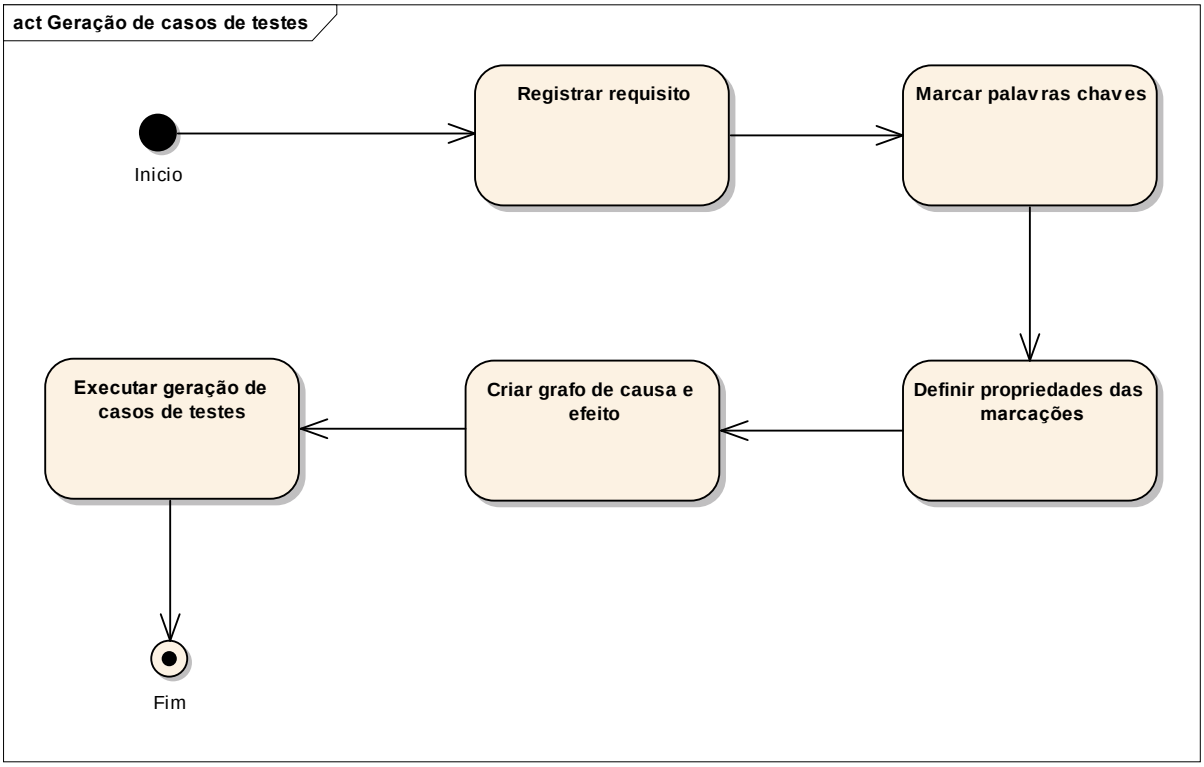


Figura 8 – Diagrama de atividades para geração de casos de testes

O processo é iniciado com o registro do novo requisito na ferramenta. Após descrever o requisito de forma completa, é necessário que o analista de sistemas realize uma revisão de ambigüidade, porém esta etapa não é suportada pela ferramenta.

Com o requisito já descrito, o próximo passo consiste em marcar as palavras chaves do requisito. As palavras chaves são utilizadas para quantificar o requisito de tal forma que seja

possível gerar casos de testes baseando-se em cada uma delas. Considera-se que, realizando o processo de testes focados nestas palavras, ao final será possível afirmar que o requisito foi testado em sua totalidade.

Na etapa seguinte, as palavras chaves são de fato quantificadas. Cada palavra chave recebe propriedades que serão utilizadas para guiar a ferramenta quanto aos testes que devem ser gerados para cada palavra chave.

Com as palavras chaves definidas e com suas propriedades devidamente registradas, é iniciado o processo de geração do grafo de causa e efeito. O grafo de causa e efeito apresenta de uma forma mais estruturada a relação entre as palavras chaves, bem como suas restrições, e auxilia na identificação de outras palavras chaves ou propriedades que não foram identificadas nos passos anteriores. O grafo pode ser gerado também para mais de um requisito. Esta situação é bastante comum porque em diversos momentos a definição de um requisito ajuda na complementação de outro.

Com o grafo de causa e efeito definido é possível realizar a geração dos casos de testes utilizando todas as informações adicionadas ao requisito. O processo de geração utiliza as propriedades geradas para as palavras chaves como fonte de informações para o caso de teste e o grafo de causa e efeito como fonte de mapeamento da relação entre cada uma das palavras chaves.

3.2.2.2 Montagem do plano de testes

A figura 9 ilustra a adição de requisitos e casos de teste ao plano de testes de uma nova versão do produto em construção ou que está sofrendo manutenção.

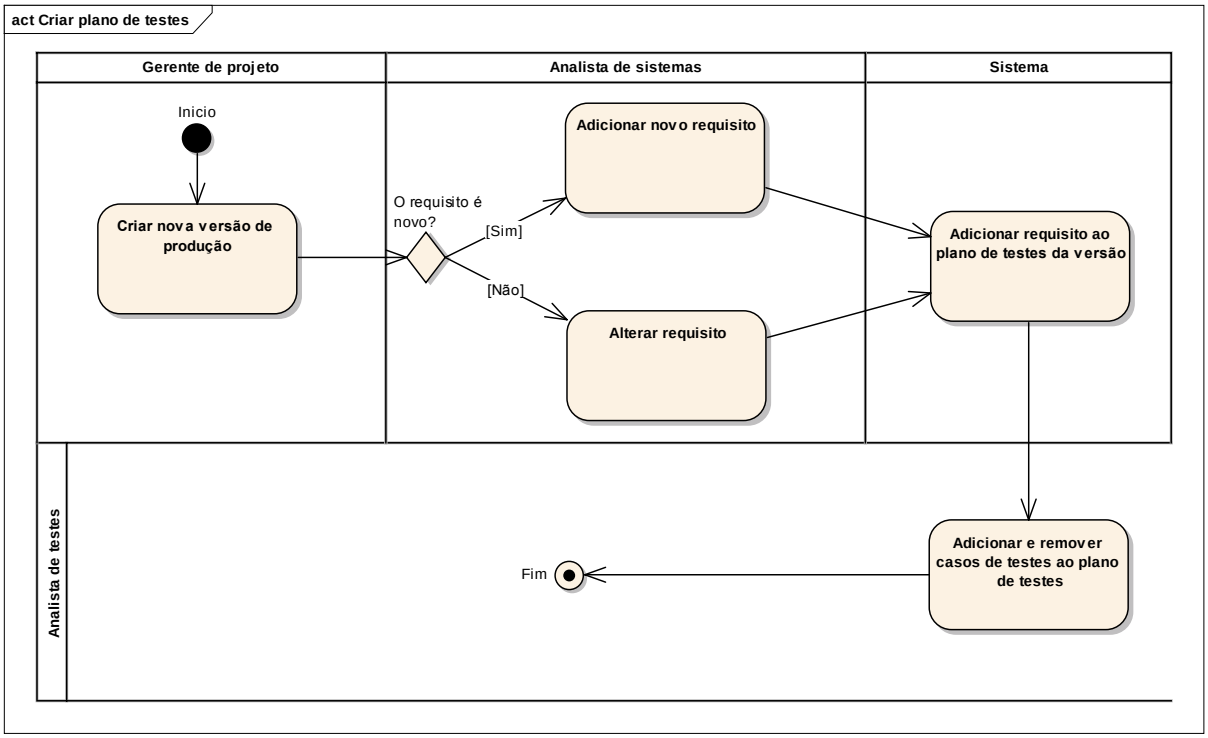


Figura 9 – Inclusão de requisitos e casos de teste no plano de testes

Quem dá inicio ao processo é o gerente de projetos. Ele é responsável por criar uma nova versão de produção, a partir da qual, qualquer alteração nos requisitos existentes ou qualquer adição de requisito impacta diretamente no plano de testes.

O analista de sistemas, ao alterar/adicionar um requisito, não possui a visão necessária para avaliar se a sua ação gera algum tipo de impacto na etapa de testes, e mesmo o analista de testes pode ter dificuldades em alguns momentos para identificar quais os testes que devem ser selecionados para avaliar se as alterações não impactaram no funcionamento anterior no software. Neste momento o sistema assume a responsabilidade de adicionar os testes relacionados aos requisitos afetados ao plano de testes, de forma que todas as alterações serão cobertas na fase de execução de testes. O grafo de causa e efeito é muito importante neste momento, pois ele ajuda a elencar os testes daqueles requisitos que não foram diretamente alterados, mas fazem parte de um grafo de causa e efeito que possui outros requisitos que foram.

Adicionalmente, o analista de testes pode ainda incluir mais testes no plano ou remover aqueles que foram considerados desnecessários ou fora do escopo.

3.2.2.3 Execução do plano de testes

A figura 10 ilustra a execução do plano de testes do ponto de vista do testador.

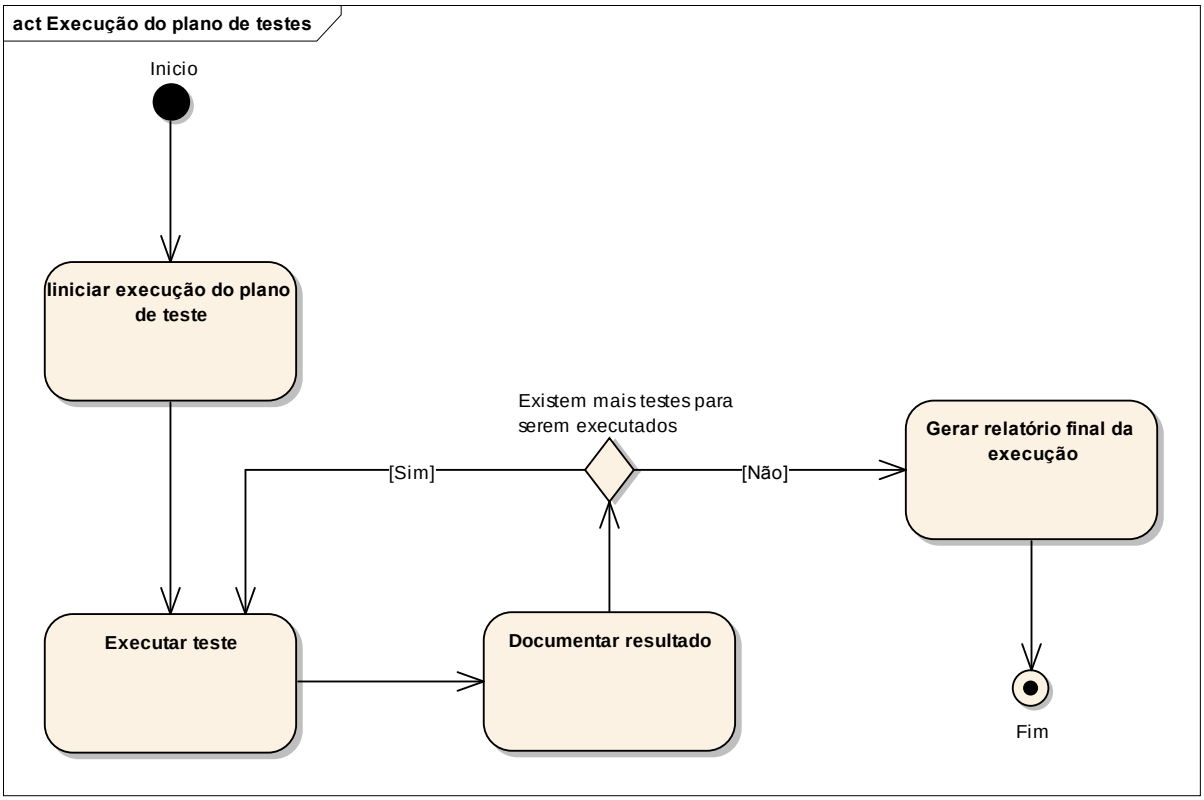


Figura 10 – Execução do plano de testes

Durante a etapa de verificação de um projeto de desenvolvimento ou manutenção, o executor de testes dá inicio a execução do plano de testes. Cada caso de teste é executado e o seu resultado é armazenado pela ferramenta, até que todo o plano de teste seja esgotado. Ao final da execução do plano, é gerado um relatório final com os resultados obtidos para cada teste executado.

3.2.3 Diagrama de casos de uso

As figuras 11 e 12 a seguir documentam as funcionalidades disponibilizadas para os atores Analista de sistemas e Analista de testes.

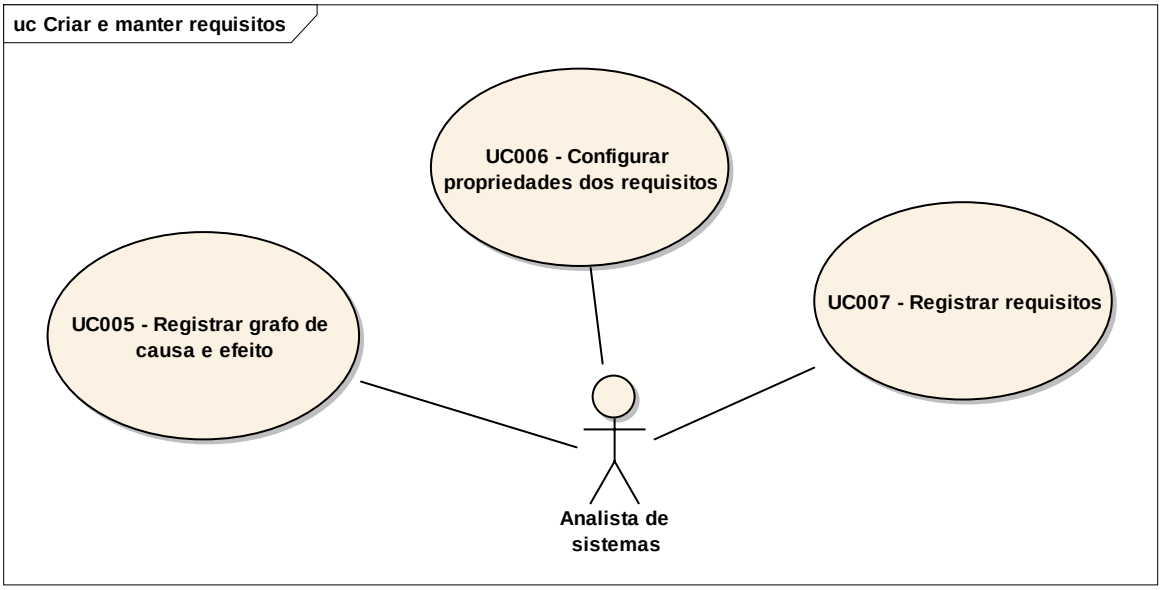


Figura 11 – Casos de uso do ator Analista de Sistemas

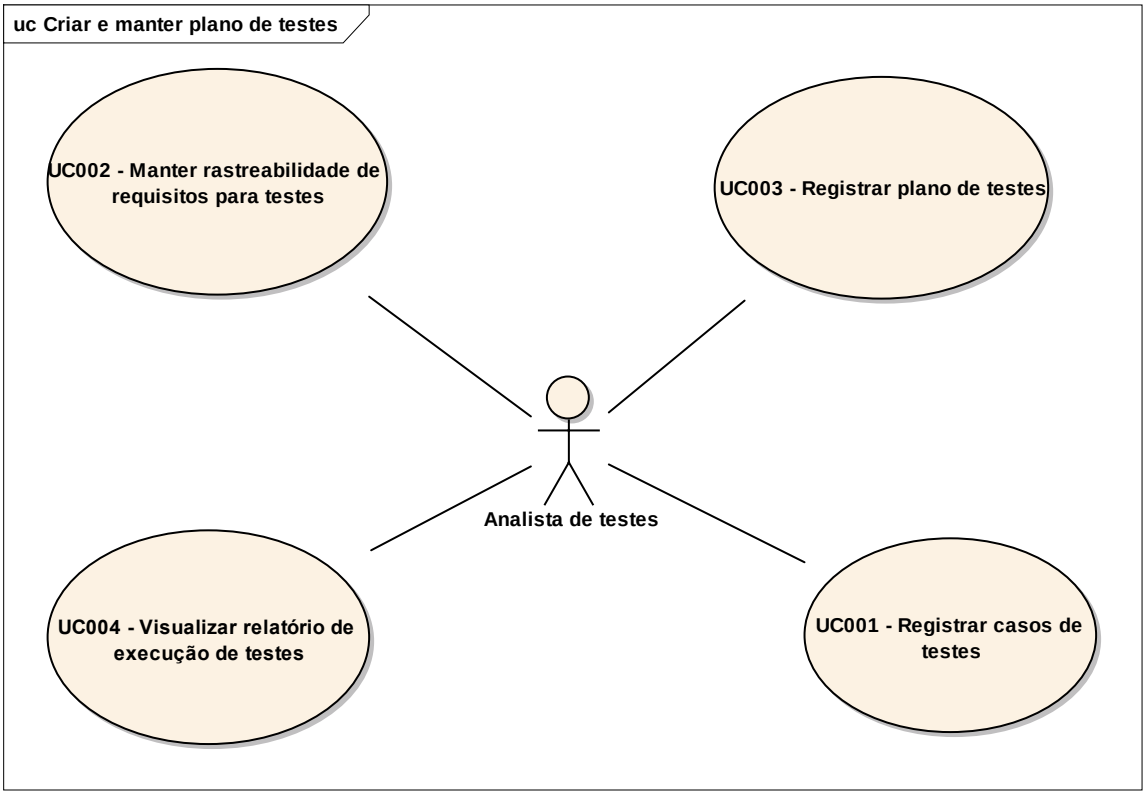


Figura 12 – Casos de uso do ator Analista de testes

A seguir é apresentada uma descrição para cada um dos casos de uso.

a) `UC001 – Registrar casos de testes`: permite que o analista de testes realize a geração dos casos de testes para os requisitos desenvolvidos, além de descrever o procedimento de teste;

b) `UC002 - Manter rastreabilidade de requisitos para testes`: permite ao analista de testes visualizar a rastreabilidade dos requisitos para os casos de testes,

assim como identificar os requisitos que ainda não possuem testes;

c) `UC003 - Registrar plano de testes`: permite ao analista de testes criar e manter o plano de testes, adicionando e removendo casos de testes que devem ser executados a cada nova versão;
d) `UC004 - Visualizar relatório de execução de testes`: permite que o analista de testes visualize o relatório de execução dos casos de testes com o status de cada teste;
e) `UC005 - Grafo de causa e efeito`: permite ao analista de sistemas definir a relação entre as palavras chaves de um requisito, bem como as restrições entre cada uma delas. Permite também que seja feito o relacionamento entre requisitos;
f) `UC006 - Configurar propriedades dos requisitos`: permite ao analista de sistemas criar as palavras chaves para os requisitos desenvolvidos e configurar suas propriedades;
g) `UC007 - Registrar requisitos`: permite ao analista de sistemas criar e manter requisitos do software desenvolvido.

3.2.4 Diagrama de classes

O diagrama de classes da figura 13 apresenta a estrutura de propriedades construída para atender a necessidade do projeto. A partir de uma classe base abstrata foram criadas as propriedades que o usuário poderá configurar para o requisito.

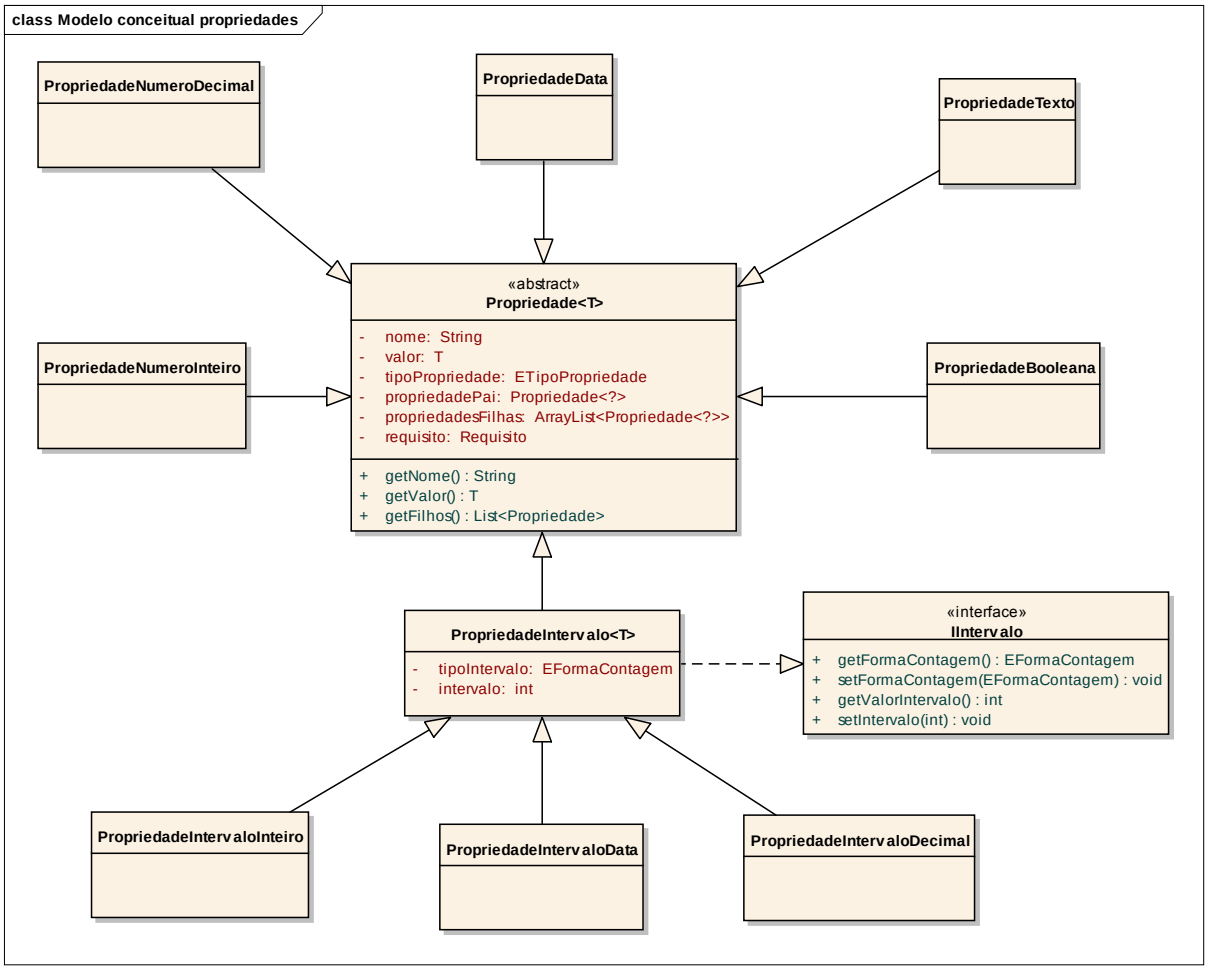


Figura 13 – Relação entre o requisito e as suas propriedades

As propriedades disponíveis para o usuário são listadas a seguir:

a) `PropriedadeBooleana`: representa uma propriedade que pode receber apenas os valores sim ou não. Útil quando a palavra chave trata de algum assinalamento que deverá existir na aplicação desenvolvida ou quando houver alguma condição que seja apenas verdadeira ou falsa;

b) `PropriedadeTexto`: representa uma propriedade que pode receber informações textuais, tais como nomes, descrições, observações, etc. ;

c) `PropriedadeData`: representa uma propriedade que pode receber um valor do tipo data. Esta propriedade permite que o usuário defina o intervalo de datas que o sistema deve aceitar e como o incremento de datas é realizado;

d) `PropriedadeNumeroInteiro`: representa uma propriedade que pode receber um valor numérico inteiro (sem casas decimais). Esta propriedade permite que o usuário defina o intervalo de números que o sistema deve aceitar e como o incremento é realizado;

e) `PropriedadeNumeroDecimal`: representa uma propriedade que pode receber um

valor numérico decimal (com casas decimais). Esta propriedade permite que o usuário defina o intervalo de números que o sistema deve aceitar e como o incremento é realizado;

As propriedades do tipo intervalo (que aparecem no diagrama de classes da figura 13) são utilizadas pelas propriedades listadas para definir os intervalos de valores aceitos por elas.

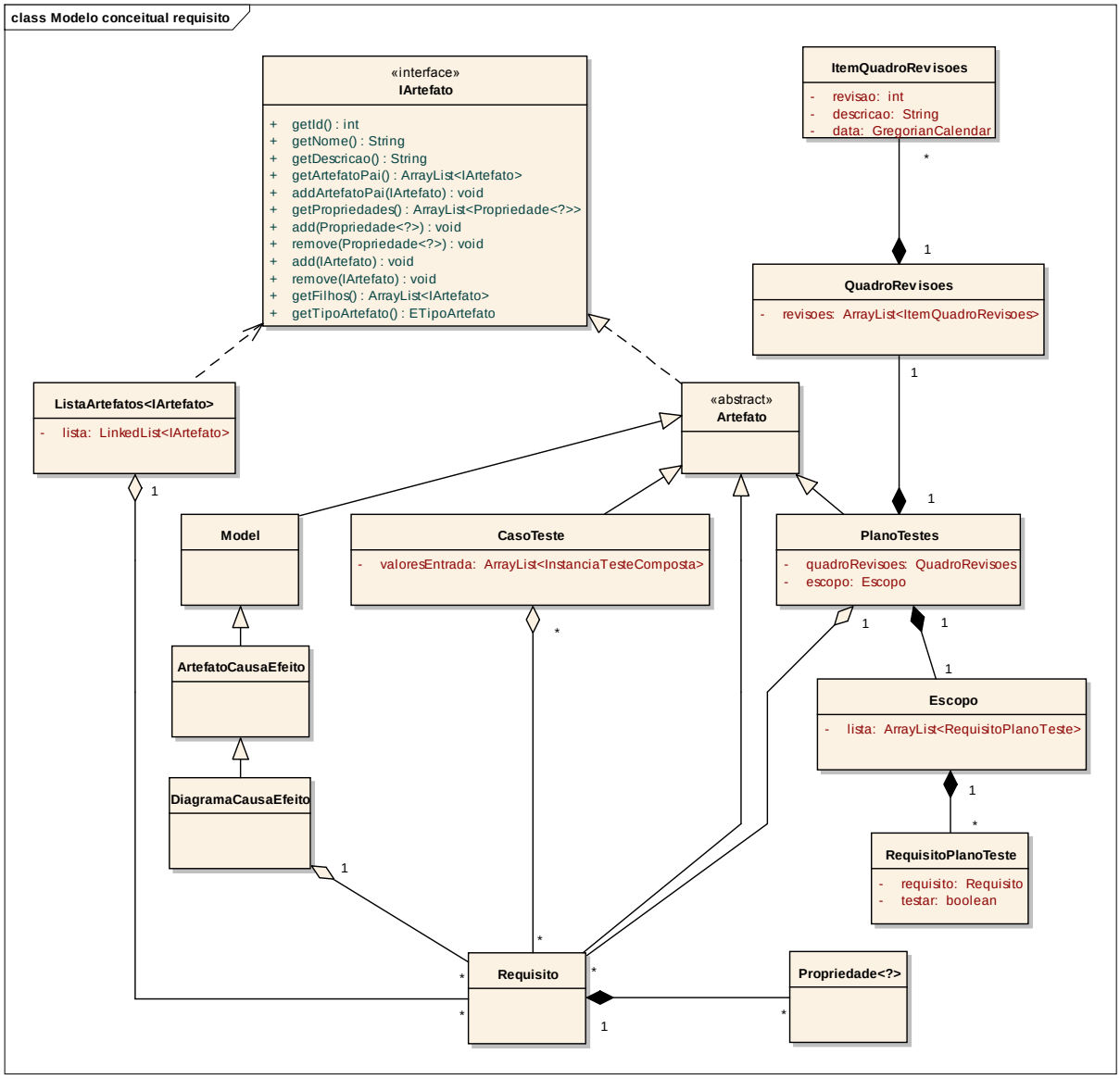


Figura 14 – Modelo conceitual de classes para os artefatos

A figura 14 apresenta o modelo de classes para os requisitos. A interface IArtefato foi definida para servir de base para todos os artefatos gerenciados pela ferramenta: Requisito, CasoTeste, PlanoTestes e DiagramaCausaEfeito.

A figura 15 ilustra as classes e métodos envolvidos no processo de geração de casos de testes. No modelo desenvolvido, o diagrama de causa e efeito tem papel importante no repasse de informações para a classe responsável pela geração dos testes.

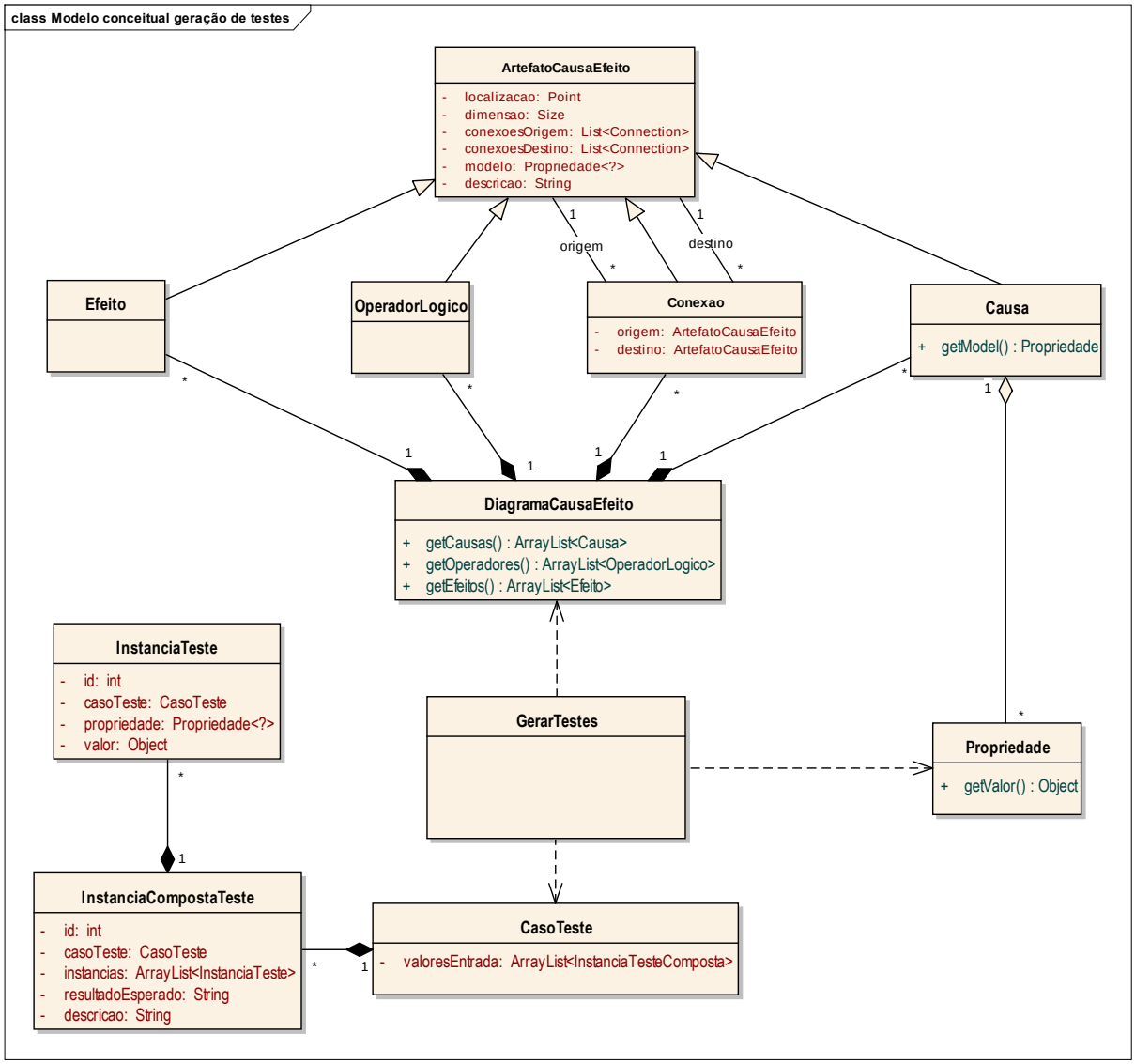


Figura 15 – Diagrama de classes para a geração de testes

A classe Artefato (figura 14) traz uma implementação genérica para os métodos comuns a todos os artefatos. Existem também diversas agregações dentro do modelo, onde o Requisito está presente na maioria deles. O Requisito agrega às classes:

a) `CasoTeste`: todo caso de teste construído na ferramenta é composto necessariamente por requisitos. Se não houver um requisito, não há razão para o caso de teste existir;

b) `DiagramaCausaEfeito`: o diagrama de causa e efeito é composto por causas, que por sua vez são compostas por Propriedades, que por sua vez são originadas nos requisitos. As associações entre os requisitos e os diagramas de causa e efeito auxiliam na identificação de impactos gerados ao alterar um requisito. Caso um requisito, pertencente a um diagrama de causa e efeito for alterado, todos os demais requisitos pertencentes a este diagrama são marcados para teste;

c) `ListaArtefatos`: possui uma relação com todos os requisitos registrados na ferramenta. Esta lista é utilizada pela visualização de requisitos;

d) `PlanoTestes`: O plano de testes é composto por todos os requisitos registrados na ferramenta. Cada requisito pode estar marcado para ser testado ou não.

A figura 16 ilustra o modelo desenvolvido para abrigar regras de validação para as conexões do grafo de causa e efeito. Devem ser utilizadas no diagrama de causa e efeito quando existir alguma restrição de navegação entre um nó e outro do grafo. Por exemplo, caso houver duas causas, sendo que uma representa uma data inicial e a outra a data final, a navegação dessas duas causas para um nó chamado período pode ser restringida por uma regra que informa que a data inicial deve ser menor do que a final e outra regra para que a data final seja maior do que a inicial.

As regras são utilizadas durante o processo de geração de dados para o teste, onde os dados gerados respeitam as regras definidas para cada conexão.

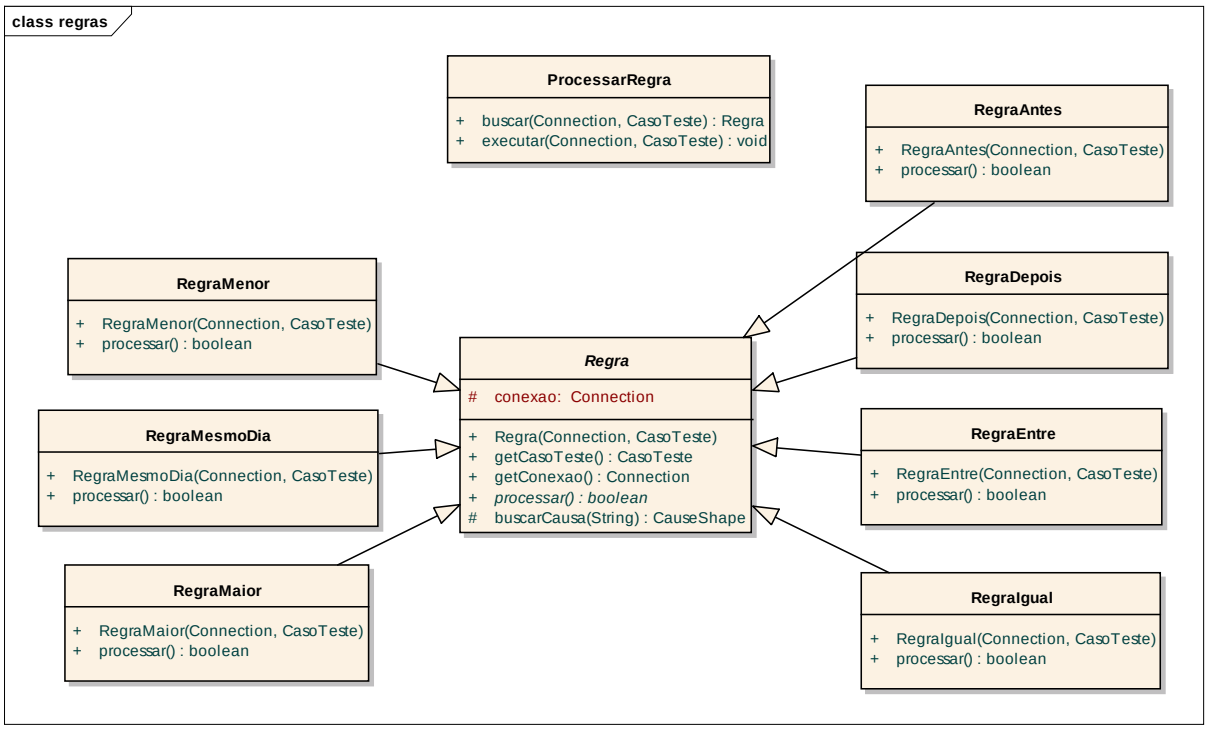


Figura 16 – Modelo de regras

Além das regras, foram desenvolvidos também operadores, que permitem calcular valores enviados por conexões, quando não é possível fixá-los no modelo. Por exemplo, em um diagrama onde o valor do imposto depende de qual conexão de taxa chegou até ele, não é possível determinar previamente o valor do imposto. Neste caso, é possível utilizar uma das operações para realizar o cálculo no momento da geração dos testes. O modelo de classes desenvolvido para atender a esta funcionalidade está representado na figura 17.

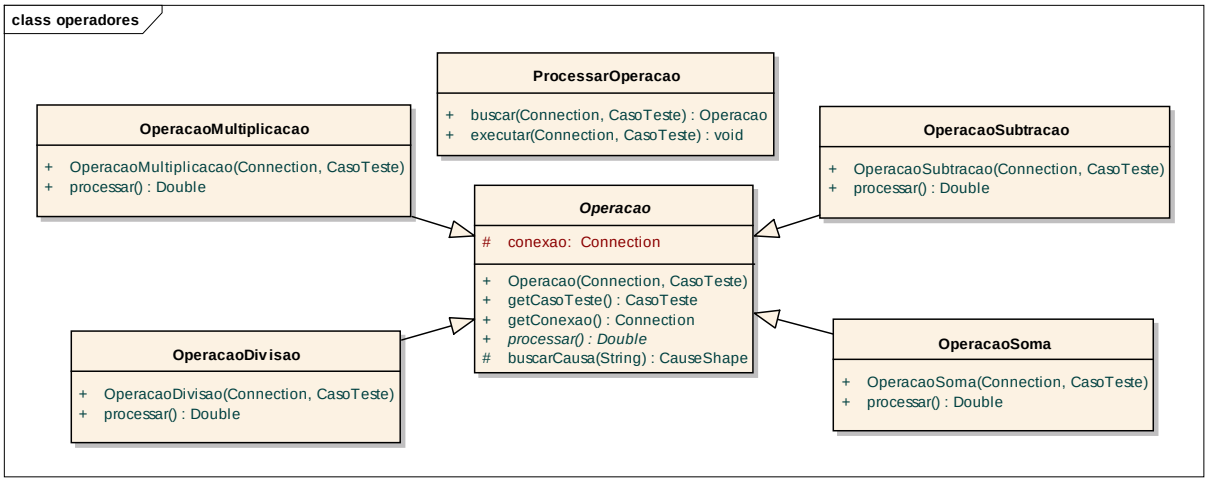


Figura 17 – Modelo de operações aritméticas

3.2.5 Diagrama de seqüência

A figura 18 ilustra a seqüência de eventos que ocorrem durante a geração de casos de testes. A geração utiliza dados do diagrama de causa e efeito para realizar a geração de testes. Durante este processo, são instanciados objetos InstanciaCompostaTeste, que por sua vez pode abrigar um ou mais objetos InstanciaTeste.

A classe DiagramaCausaEfeito possui todos os dados necessários para que o processo de geração de casos de testes seja executado.

Neste processo, um dos passos mais importantes é identificar as causas que não tem conexão de chegada. Identificando-as, é possível determinar quais são os dados de entrada que o usuário terá disponível. Os valores de todas as outras causas são calculadas através de regras alocadas nas conexões, que por sua vez ligam duas causas ou uma causa a um operador lógico ou ainda uma causa a um efeito, no diagrama de causa e efeito.

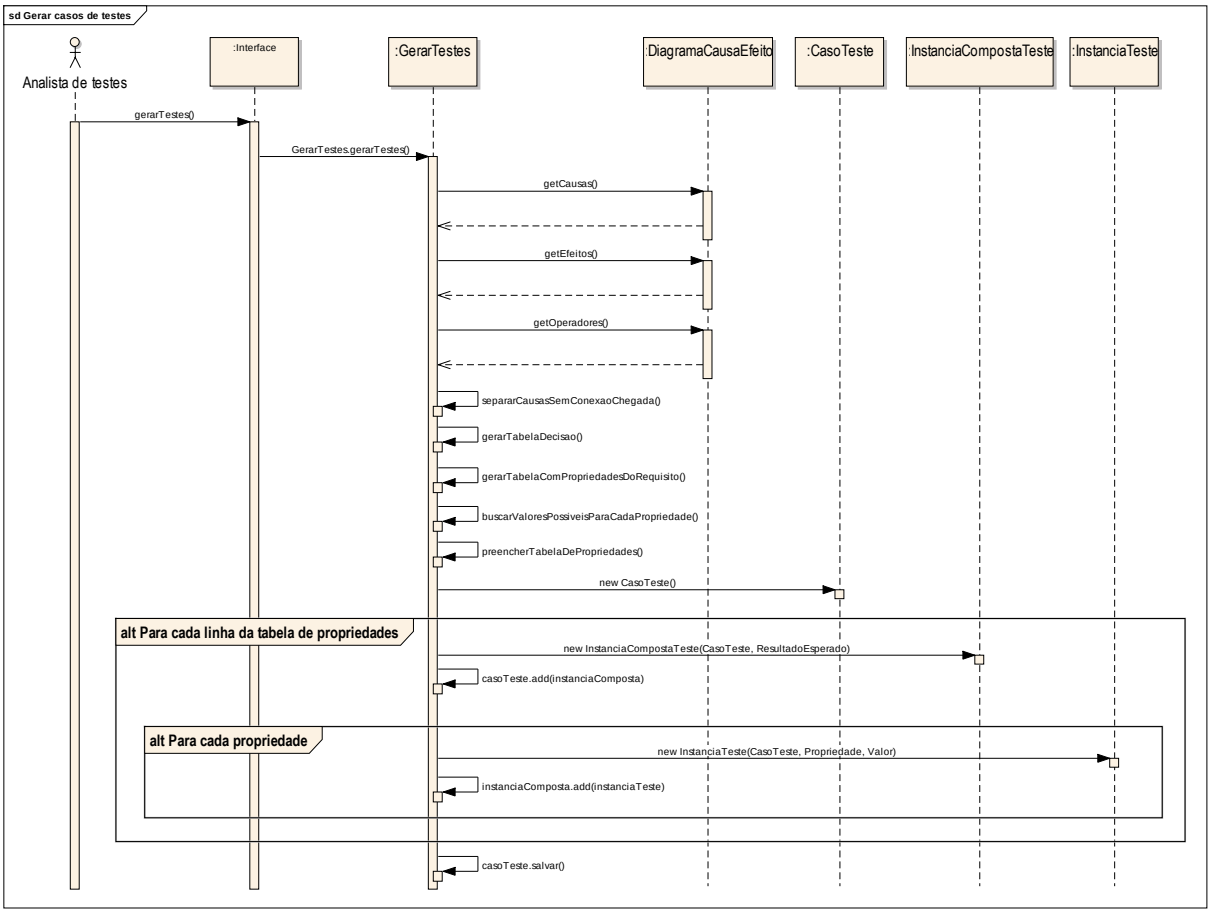


Figura 18 – Diagrama de seqüência para geração de testes

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A ferramenta proposta foi desenvolvida utilizando a plataforma *Eclipse RPC*. A figura 19 ilustra a interface apresentada ao executar a ferramenta pela primeira vez.

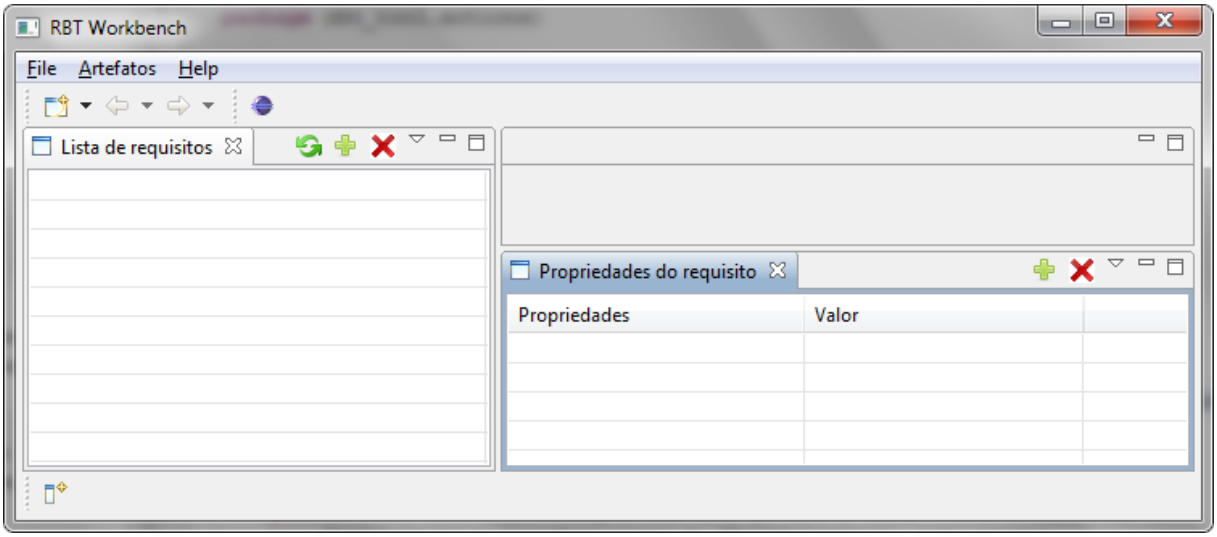


Figura 19 – Interface da ferramenta

A implementação desta plataforma consiste em estender *extensions*[2] adicionando as funcionalidades desejadas.

```
package qaworkbench.requirements.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IViewActionDelegate;
import org.eclipse.ui.IViewPart;
import rbt_tool.views.RequirementsTreeView;

public class ActionRequirementsListViewAdd implements IViewActionDelegate
{

	@Override
	public void init(IViewPart view) {
	}
	@Override
	public void run(IAction action) {
		RequirementsListView.getModel().createRequirement(
				RequirementsListView.getModel().get().size() + 1L,
				"Novo requisito");
		RequirementsListView.refresh();
	}
	@Override
	public void selectionChanged(IAction action, ISelection selection) {
	}
}
```

Quadro 4 – *Action* responsável pela inclusão de requisitos

O quadro 4 apresenta um exemplo de extensão viabilizada pela plataforma. Ao implementar a interface *IViewActionDelegate*, sempre que o botão "Adicionar requisito" for pressionado, o método *run* será chamado pela ferramenta. A associação desta classe com o botão é feita através do arquivo *plugin.xml*, apresentado no quadro 5.

[2] *Extensions* são componentes da ferramenta que podem ser estendidos adicionando o comportamento da lógica de negócio necessária.

O *Eclipse RCP* provê uma série de *interfaces* e classes abstratas que facilitam o desenvolvimento das extensões para a ferramenta. Estas extensões são associadas à plataforma através de um arquivo de configuração chamado *plugin.xml*, que contém a descrição das *extensions* utilizadas e quais as classes que as estendem. A quadro 5 apresenta o trecho do *plugin.xml* que associa a *action* de inclusão de requisitos à plataforma.

```
<plugin>

   <extension
         id="application"
         point="org.eclipse.core.runtime.applications">
      <application>
         <run
               class="rbt_tool.Application">
         </run>
      </application>
   </extension>
   <extension
         point="org.eclipse.ui.perspectives">
      <perspective
            name="RBT Perspective"
            class="rbt_tool.Perspective"
            id="RBT_Tool.Perspective">
      </perspective>
   </extension>
   <extension
         point="org.eclipse.ui.views">
      <category
            id="RBT_Tool.ViewCategory"
            name="RBT Tool">
      </category>
      <view
            allowMultiple="true"
            category="RBT_Tool.ViewCategory"
            class="rbt_tool.views.RequirementsTreeView"
            fastViewWidthRatio="0.35f"
            id="rbt_tool.views.RequirementsTreeView"
            name="Lista de requisitos"
            restorable="true">
      </view>
   <extension
         point="org.eclipse.ui.viewActions">
      <viewContribution
            id="RBT_Tool.view.artefatos"
            targetID="rbt_tool.views.RequirementsTreeView">
         <action
               class="rbt_tool.actions.ActionRequirementsListViewRemove"
               icon="icons/actions/remove_x_16.png"
               id="RBT_Tool.requirements.ViewAction.RemoveRequirement"
               label="Remover requisito"
               menubarPath="additions"
               state="true"
               style="push"
               toolbarPath="additions"
               tooltip="Remover requisito">
         </action>
         <action
               class="rbt_tool.actions.ActionRequirementsListViewAdd"
```

```
                icon="icons/actions/add.png"
                id="rbt_tool.actions.ActionRequirementsListViewAdd"
                label="Adicionar requisito"
                menubarPath="additions"
                state="true"
                style="push"
                toolbarPath="additions"
                tooltip="Adicionar requisito">
          </action>
       </viewContribution>
   </extension>
</plugin>
```

Quadro 5 – Trecho do arquivo *plugin.xml*

3.3.2 Operacionalidade da implementação

A seguir será apresentada a operacionalidade da ferramenta desenvolvida através de um estudo de caso utilizando requisitos desenvolvidos com o propósito de validar o funcionamento da ferramenta.

O estudo de caso consiste em desenvolver um requisito de uma ferramenta de registro de laudos de testes, onde são cadastradas informações do laudo e dos testes realizados para ele seguindo os passos descritos na figura 8.

3.3.2.1 Registrar requisitos

O registro de novos requisitos é iniciado através de um botão na parte superior da lista de requisitos, ou então clicando com o botão direito do mouse sobre a área reservada para isto e selecionando a opção “Novo requisito”, conforme demonstrado na figura 19.

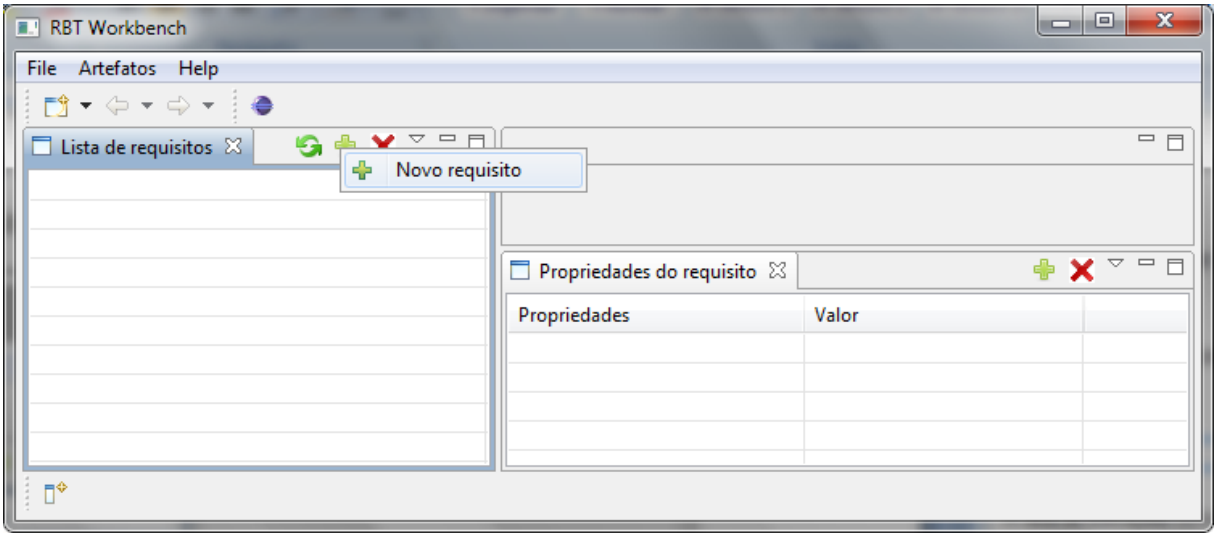


Figura 20 – Registro de novo requisito

Ao incluir o novo requisito, um duplo clique com o mouse sobre seu identificador na lista de requisitos abre um editor, onde é possível alterar seu identificador, nome e a descrição. A figura 21 apresenta o requisito recém criado aberto para edição.

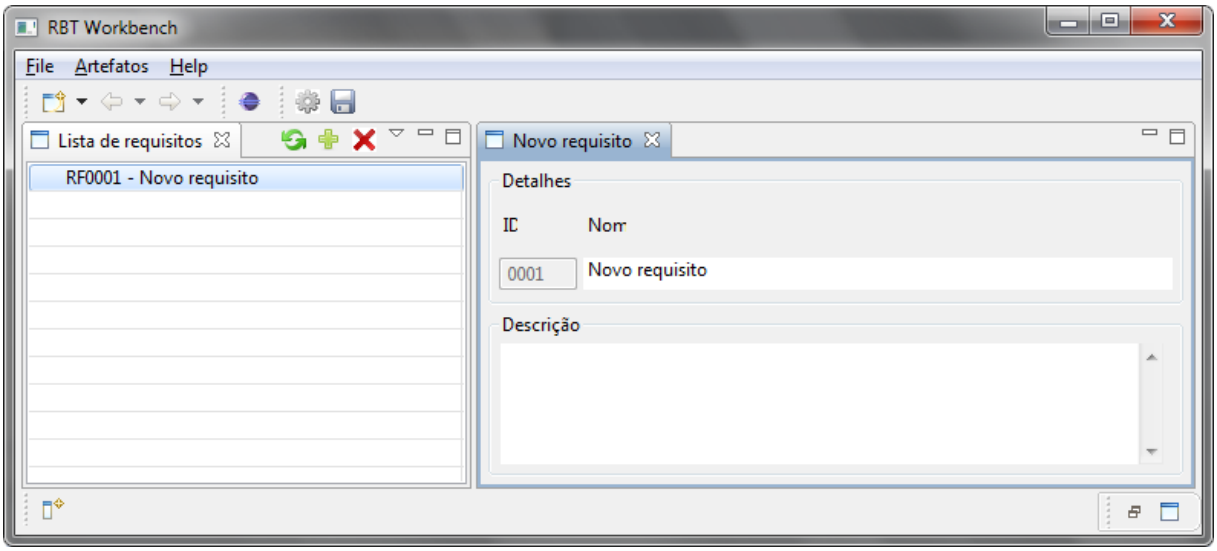


Figura 21 – Requisito aberto para edição

Utilizando o campo descrição, o analista de sistemas documenta a necessidade que precisa ser atendida de forma clara e concisa. A figura 22 apresenta o requisito utilizado no estudo de caso já redigido na ferramenta.

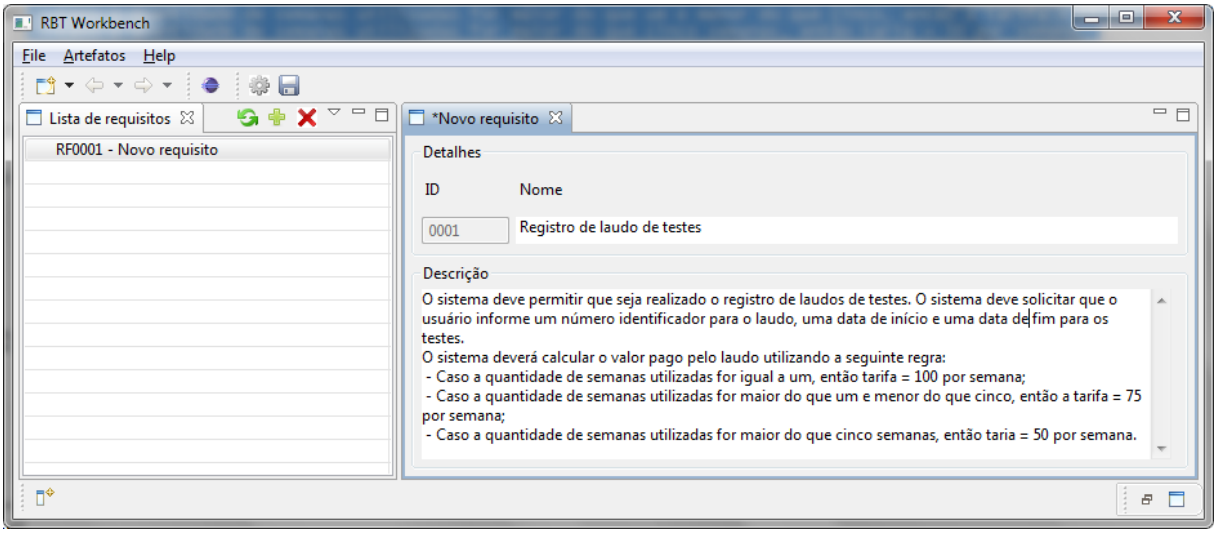


Figura 22 – Definição do requisito

3.3.2.2 Definir as palavras chaves

Após o requisito ser descrito, seja completa ou parcialmente, é possível marcar as palavras chaves que deverão ser utilizadas mais tarde para a construção do grafo de causa e efeito e para a geração de testes.

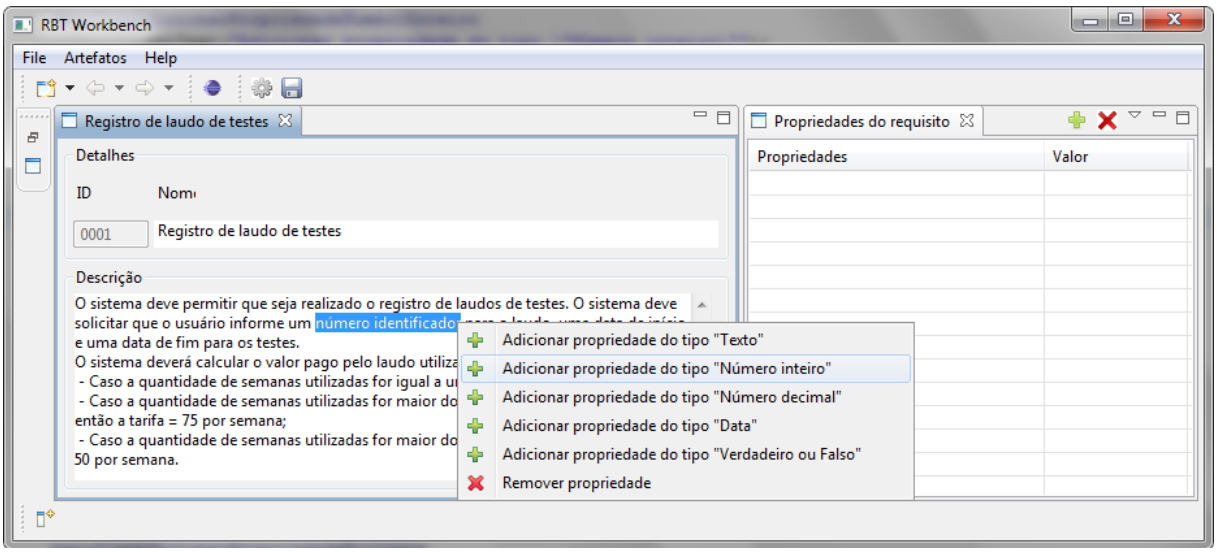


Figura 23 – Definição das palavras chaves

Para marcar uma palavra como chave, basta selecioná-la no editor, clicar com o botão direito do mouse e selecionar a opção adequada no menu de contexto apresentado. Nesta etapa é muito importante que todas as palavras que tenham relevância para o domínio do problema sejam marcadas. Essas palavras chaves quantificarão o requisito no passo seguinte.

O menu de contexto apresentado ao clicar com o botão direito do mouse (vide figura 23) apresenta uma lista com as opções de tipos de palavras chaves disponíveis. É preciso que

neste passo seja escolhido o tipo mais adequado para a palavra selecionada. No exemplo acima, a palavra chave é “número identificador”, então será escolhida a opção “Número inteiro”.

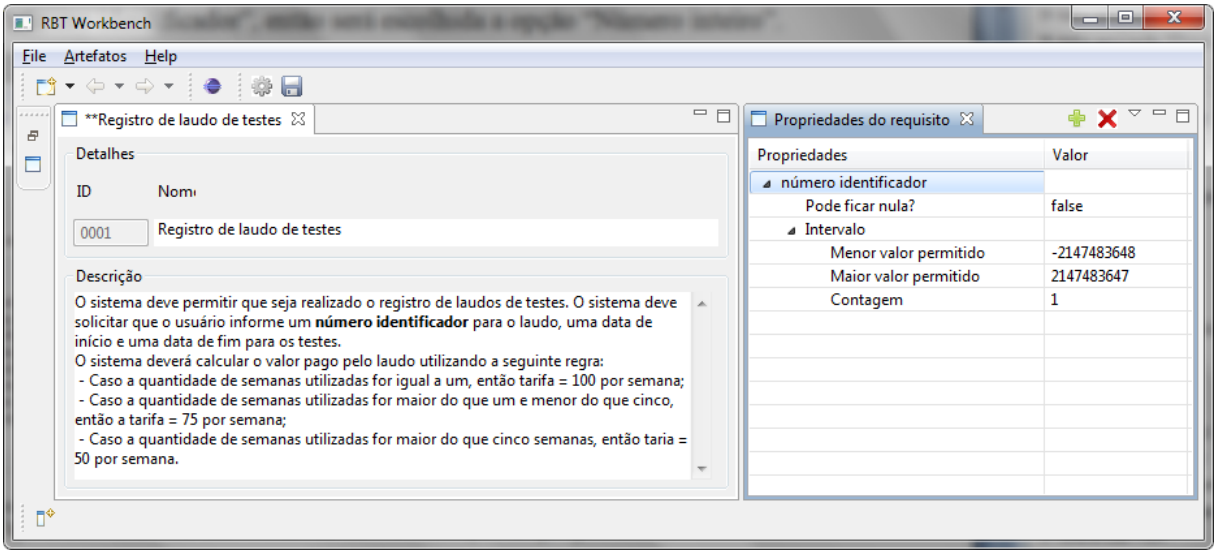


Figura 24 – Palavra chave exibida na visualização de propriedades

Após marcar uma palavra como chave para o requisito, ela passará a ser exibida em negrito na descrição, conforme apresentado na figura 24. As palavras chaves marcadas no requisito são apresentadas na visualização de propriedades do requisito. Cada palavra chave é a raiz de uma estrutura representada por uma árvore, onde são apresentadas as suas propriedades.

No exemplo da figura 24, a palavra chave “número identificador” é apresentado com uma estrutura onde pode ser definido, por exemplo, se a palavra chave pode ficar com valor nulo ou sempre deve ter um valor atribuído. Neste mesmo exemplo, pode ser definido o intervalo de valores possíveis para a palavra chave.

Esta estrutura de propriedades é fixa, e inserida para a palavra chave de acordo com o tipo escolhido no momento da sua criação.

3.3.2.3 Quantificar as palavras chaves

A quantificação das palavras chaves ocorre ao configurar valores adequados para elas. As suas propriedades visam tornar o requisito mais sólido, e devem ser utilizadas inclusive nas fases de projeto e desenvolvimento do produto.

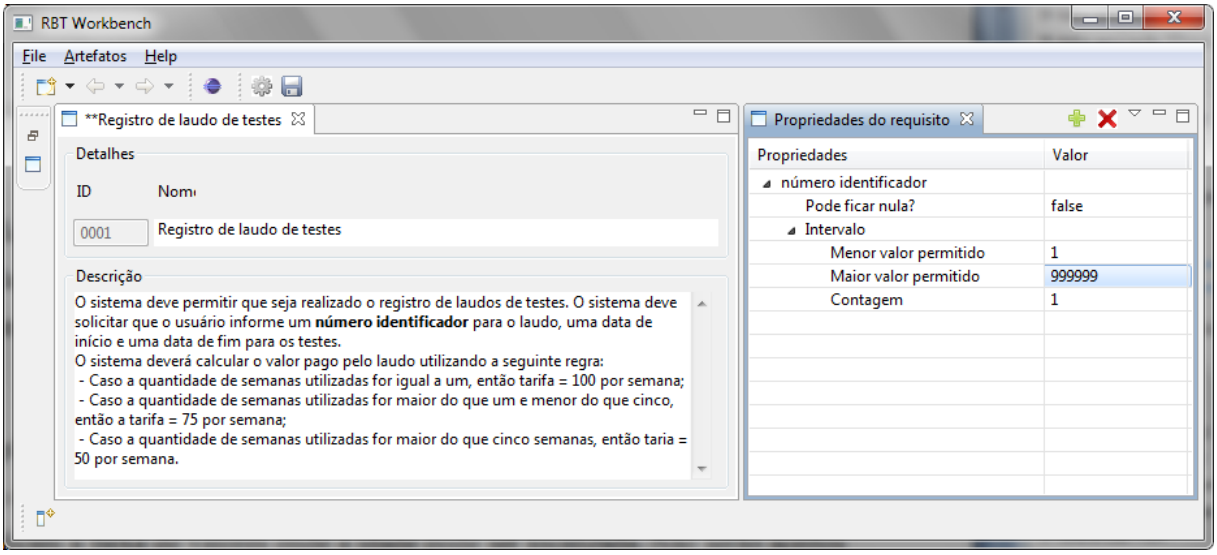


Figura 25 – Quantificando uma palavra chave

A figura 25 apresenta a palavra chave “número identificador” já quantificada. Como, ao criar a palavra chave, foi selecionado o tipo “Número inteiro”, as propriedades apresentadas são relacionadas a este tipo de palavra chave.

No exemplo da figura 25, a palavra chave recebe quatro parâmetros. O primeiro informa que este será um dado obrigatório no registro de laudos, pois não pode ficar nulo. A segunda e a terceira propriedades indicam a faixa de valores onde o número do laudo pode estar contido. O último parâmetro indica como é feito o incremento do intervalo. Se o exemplo estivesse tratando de anos contados em décadas, por exemplo, poderíamos informar o valor 10 neste campo.

As propriedades para cada palavra chave são pré-definidas, bem como os tipos de palavra chaves disponíveis, e foram definidos no capítulo 3.2.4.

3.3.2.4 Montar o grafo de causa e efeito

A montagem do grafo de causa e efeito busca um melhor aproveitamento das palavras chaves selecionadas no requisito, de forma que o relacionamento entre elas seja documentado e possa ser utilizado como restrição na geração de casos de testes. Estas restrições ajudam a reduzir o número de testes gerados pela ferramenta, além de excluir condições que seriam impossíveis de serem testadas na prática.

Para incluir um novo grafo de causa e efeito, deve-se acessar o menu Arquivo/Novo, e selecionar o item “Novo grafo de causa e efeito”, conforme demonstrado na figura 26.

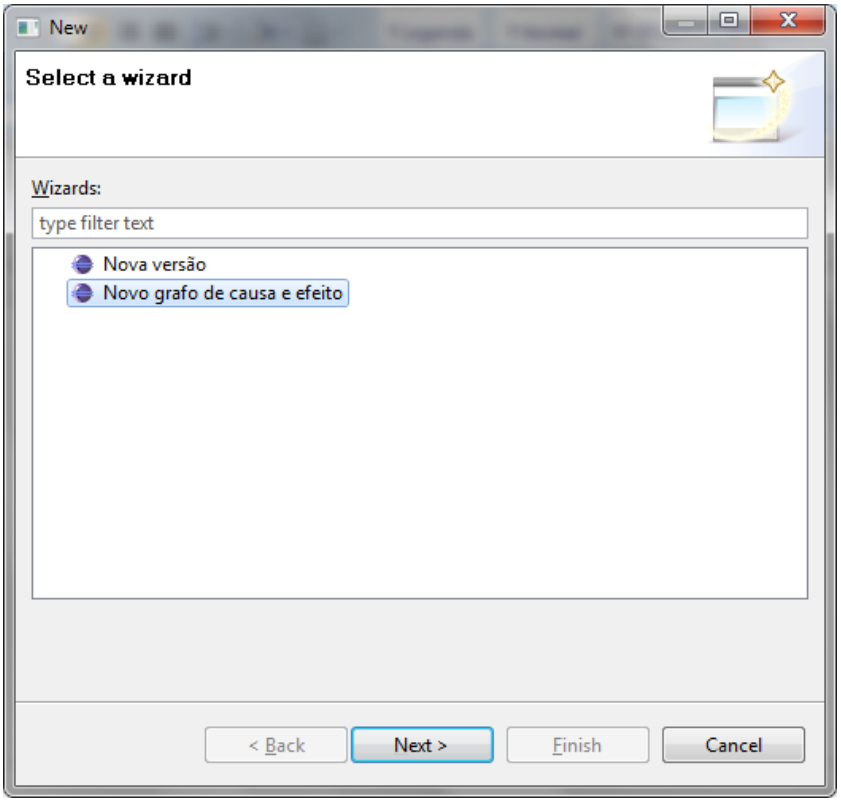


Figura 26 – Novo grafo de causa e efeito

Na figura 27 é apresentado um espaço destinado ao nome do grafo e outro destinado à seleção dos requisitos que terão suas propriedades relacionadas no grafo. Enquanto nenhum requisito for selecionado, o assistente não permitirá que o processo seja concluído. Para selecionar um requisito para o grafo, é necessário dar um duplo clique com o mouse sobre ele.

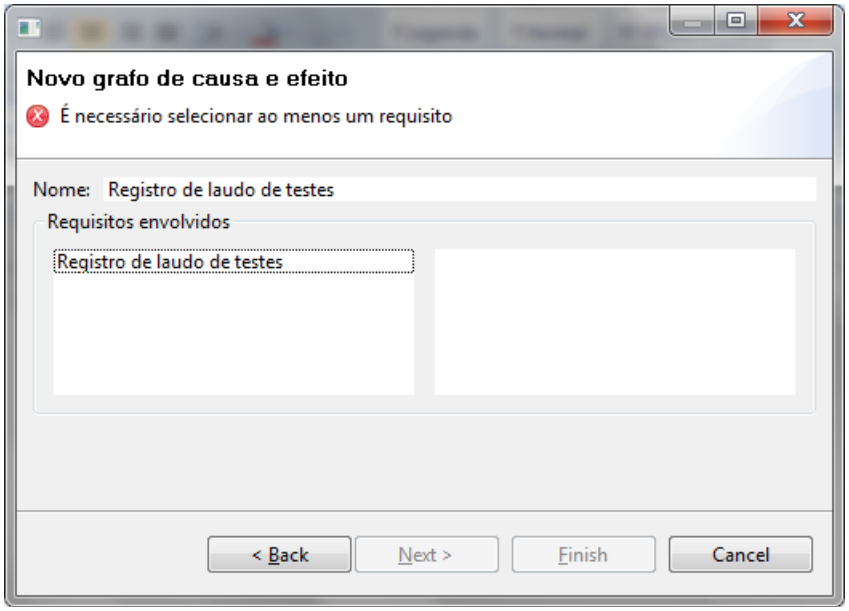


Figura 27 – Seleção dos requisitos utilizados no grafo

Após selecionar o requisito, o botão *Finish* é habilitado, permitindo a conclusão do assistente (ver figura 28).

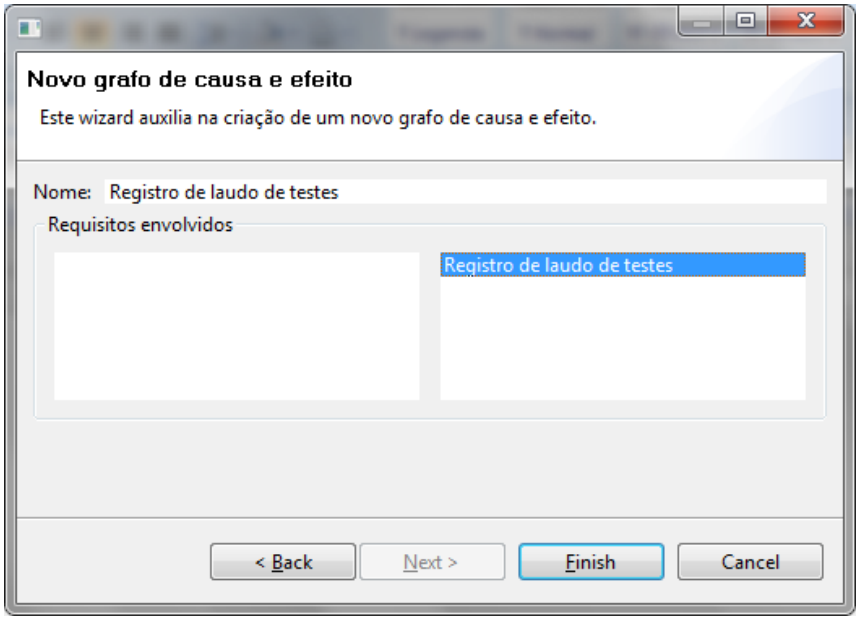


Figura 28 – Requisito selecionado e botão *Finish* habilitado

Um grafo pode ser constituído por palavras chaves de um ou mais requisitos. A inclusão de mais de um requisito no grafo de causa e efeito permite documentar e testar o relacionamento entre as características que o cliente espera que o software atenda.

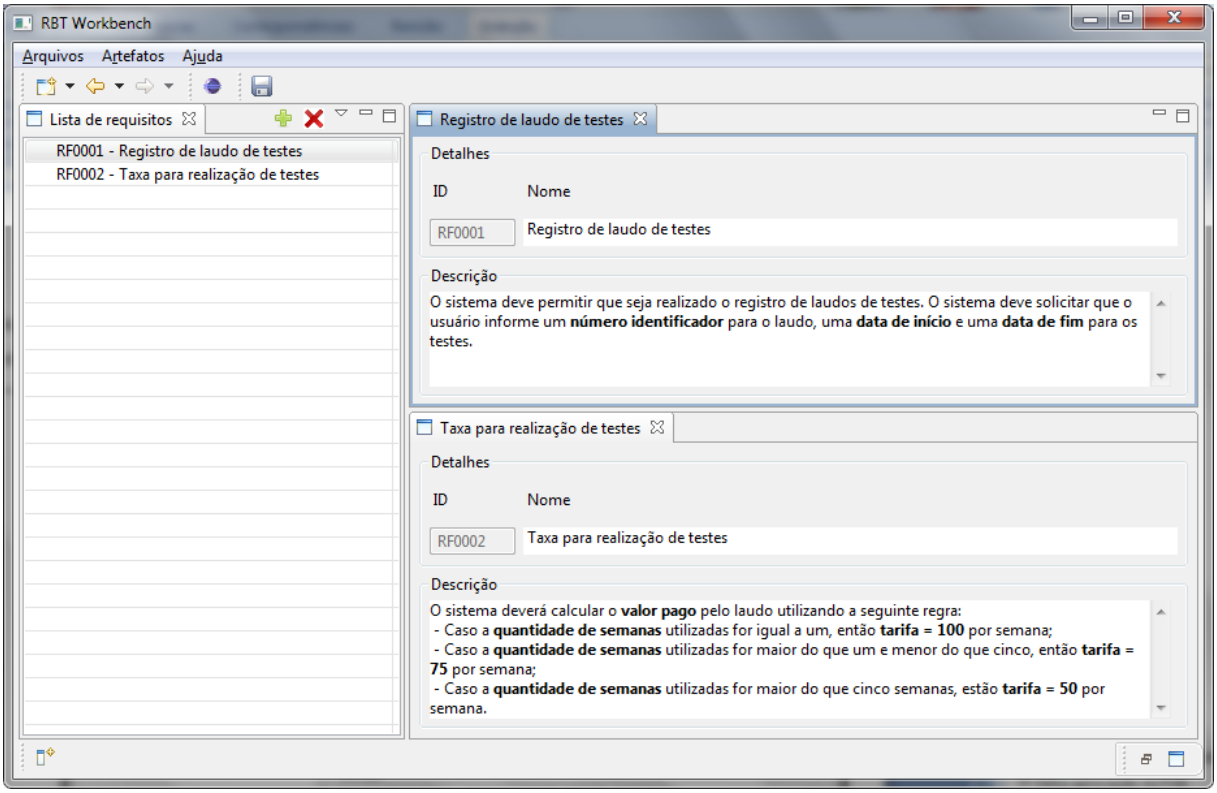


Figura 29 – Requisito dividido em dois

Para ilustrar esta funcionalidade, o requisito apresentado anteriormente foi dividido em dois, conforme demonstrado na figura 29.

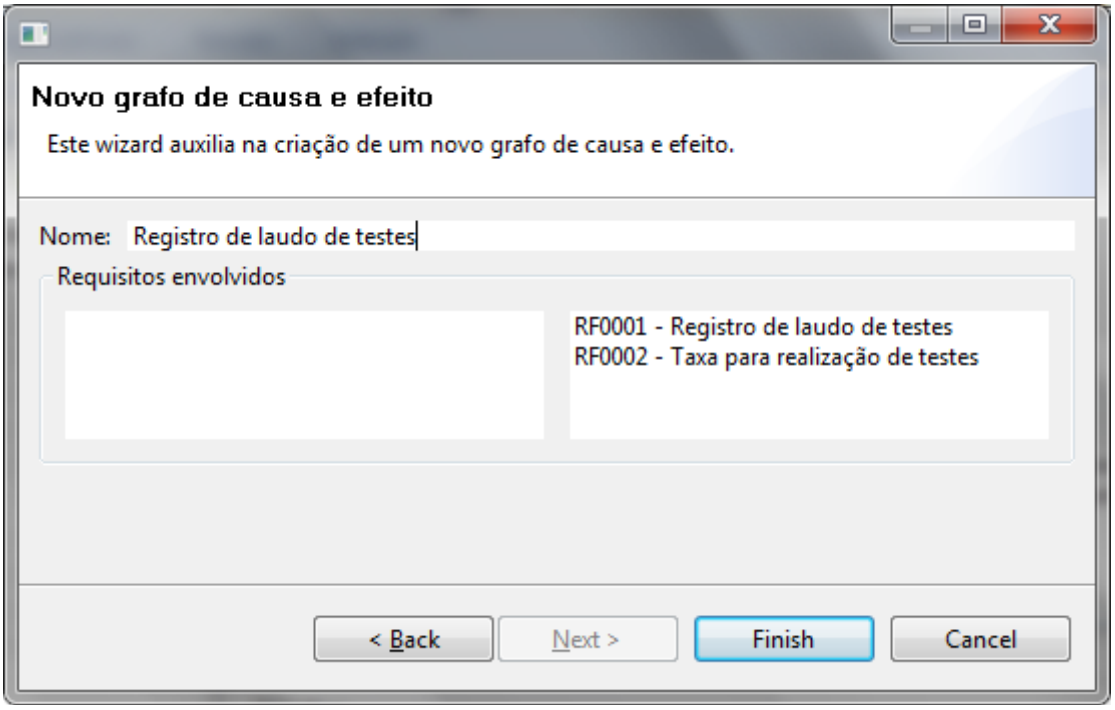


Figura 30 – Dois requisitos selecionados para um mesmo grafo

A figura 30 apresenta como a ferramenta possibilita que os dois requisitos sejam adicionados ao grafo, permitindo o teste do relacionamento do relacionamento entre eles. Tanto o grafo gerado para o requisito único quanto para os dois requisitos agrupados em um único grafo seriam exatamente os mesmos.

As palavras chaves dos requisitos são inseridas de forma automática, bastando apenas organizá-las e relacioná-las umas com as outras.

O grafo de causa e efeito é composto por cinco tipos de entidades:

a) causas: representam as entradas mais básicas do sistema, as condições mais elementares. São representados por círculos;
b) efeitos: representam as saídas possíveis do sistema, e não são utilizadas como causas para outras partes do grafo. São representados por caixas azuis;
c) operadores AND, OR e XOR: utilizados para testar condições lógicas no grafo;
d) conexões: indicam o relacionamento entre duas palavras chaves;

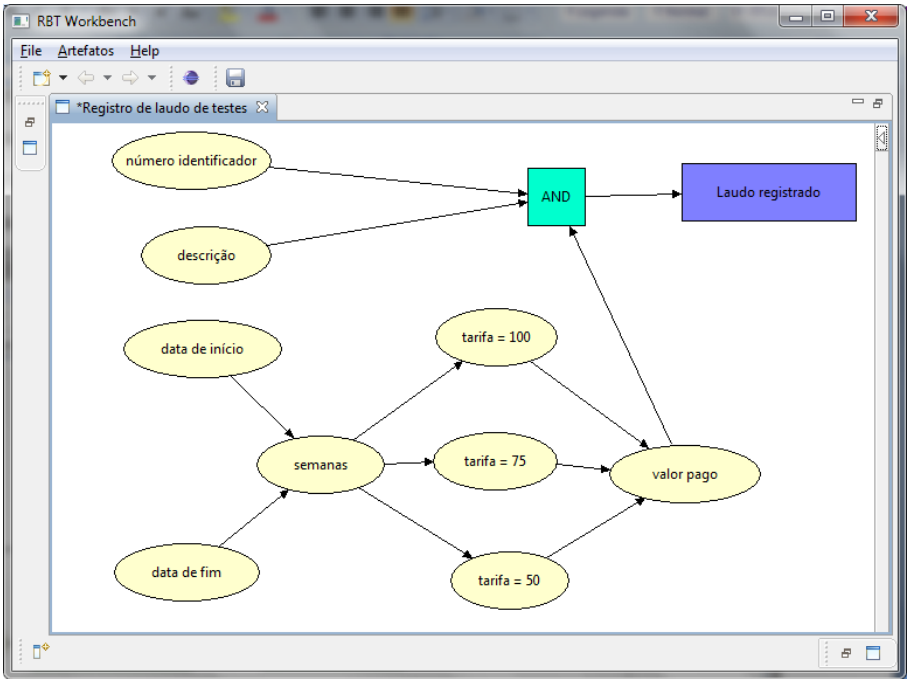


Figura 31 – Grafo de causa e efeito para o registro de um laudo de testes

A figura 31 apresenta o grafo para registro de um laudo de testes já organizado, porém sem regras e operações definidas para as conexões. Definir regras para as conexões restringe a navegabilidade pelo grafo, auxiliando no momento da geração dos respectivos testes. Ao valores para os testes não são calculados em nenhum momento pela ferramenta levando em consideração estas regras e operações. A ferramenta apenas aponta nos casos de testes gerados que estas regras devem ser aplicadas.

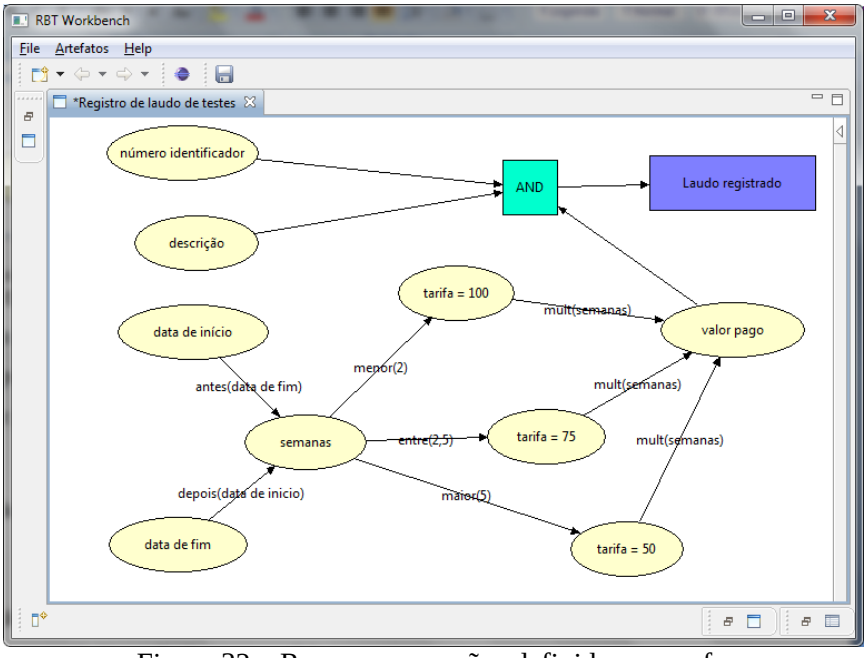


Figura 32 – Regras e operações definidas no grafo

A figura 32 apresenta o grafo para registro do laudo de testes com as regras e operações já definidas. As regras desenvolvidas estão listadas no Quadro 6.

Regras	Parâmetros	Restrições	Resultado
Antes	Nome de uma propriedade do tipo “Data” ou “aaaa,mm,dd”, onde aaaa representa o ano, mm representa o mês e dd representa o dia	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Data”.	Verdadeiro caso a causa da qual a conexão partiu for anterior a data do parâmetro
Depois	Nome de uma propriedade do tipo “Data” ou “aaaa,mm,dd”, onde aaaa representa o ano, mm representa o mês e dd representa o dia	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Data”.	Verdadeiro caso a causa da qual a conexão partiu for posterior a data do parâmetro
Entre	Nome de duas propriedades do tipo “Numero”, separadas por vírgula ou dois número separados por vírgula	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Numero”.	Verdadeiro quando o valor da causa da qual a conexão partiu estiver entre os valores especificados no parâmetro
Igual	Nome de uma propriedade do tipo “Numero” ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Numero”.	quando o valor da causa da qual a conexão partiu for igual ao valor especificado no parâmetro
Maior	Nome de uma propriedade do tipo “Numero” ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Numero”.	quando o valor da causa da qual a conexão partiu for maior do que o valor especificado no parâmetro
Menor	Nome de uma propriedade do tipo “Numero” ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Numero”.	quando o valor da causa da qual a conexão partiu for menor do que o valor especificado no parâmetro
MesmoDia	Nome de uma propriedade do tipo “Data” ou “aaaa,mm,dd”, onde aaaa representa o ano, mm representa o mês e dd representa o dia	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo “Data”.	Verdadeiro caso a causa da qual a conexão partiu for igual a data do parâmetro

Quadro 6 - Regras disponíveis para o grafo de causa e efeito

As operações estão listadas na Quadro 7.

Operação	Parâmetros	Restrições	Resultado
Soma	Nome de uma propriedade do tipo "Número" ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo "Número".	Soma do parâmetro com o valor da propriedade de origem da conexão
Sub	Nome de uma propriedade do tipo "Número" ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo "Número".	Subtração do parâmetro com o valor da propriedade de origem da conexão
Mult	Nome de uma propriedade do tipo "Número" ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo "Número".	Multiplicação do parâmetro com o valor da propriedade de origem da conexão
Divisao	Nome de uma propriedade do tipo "Número" ou um número literal	Pode ser chamado quando a origem da conexão é de uma palavra chave do tipo "Número".	Divisão do valor da propriedade de origem da conexão pelo valor do parâmetro

Quadro 7 - Operações disponíveis para o grafo de causa e efeito

As regras definidas para propriedades numéricas servem tanto para as propriedades do tipo "Número inteiro" quanto para as propriedades do tipo "Número decimal".

3.3.2.5 Geração de casos de testes

A geração dos casos de testes pode ocorrer de duas formas: a partir do requisito e suas propriedades ou a partir do grafo de causa e efeito.

A geração consiste basicamente em sugerir os dados que devem ser verificados durante a execução dos testes e o resultado esperado para cada um. A maneira como o teste deve proceder (o procedimento de teste) deve ser documentada manualmente pelo analista de testes no campo descrição. Os campos ID e Nome devem ser preenchidos de forma que estes possam ser referenciados em outros documentos do projetos de teste (fora do escopo da ferramenta).

Através da geração do caso de teste a partir do requisito é possível verificar rotinas triviais, como por exemplo, cadastros básicos do sistema. A figura 32 ilustra um caso de teste gerado para o registro de laudos de testes.

Os testes gerados pela ferramenta incluem verificações de valores limítrofes, consistências de obrigatoriedade e valores inválidos. Para isto, são utilizadas as técnicas de particionamento de equivalência e valores limítrofes.

Por exemplo, para o caso do registro de laudo de testes, foi definida a propriedade “identificador único”. Essa propriedade foi definida com o tipo numérico e com valor mínimo e máximo iguais a 1 e 999999. Para esta propriedade a ferramenta aplica a técnica de particionamento de equivalência associada a de valores limite, gerando os testes listados na figura 33, com os respectivos resultados esperados.

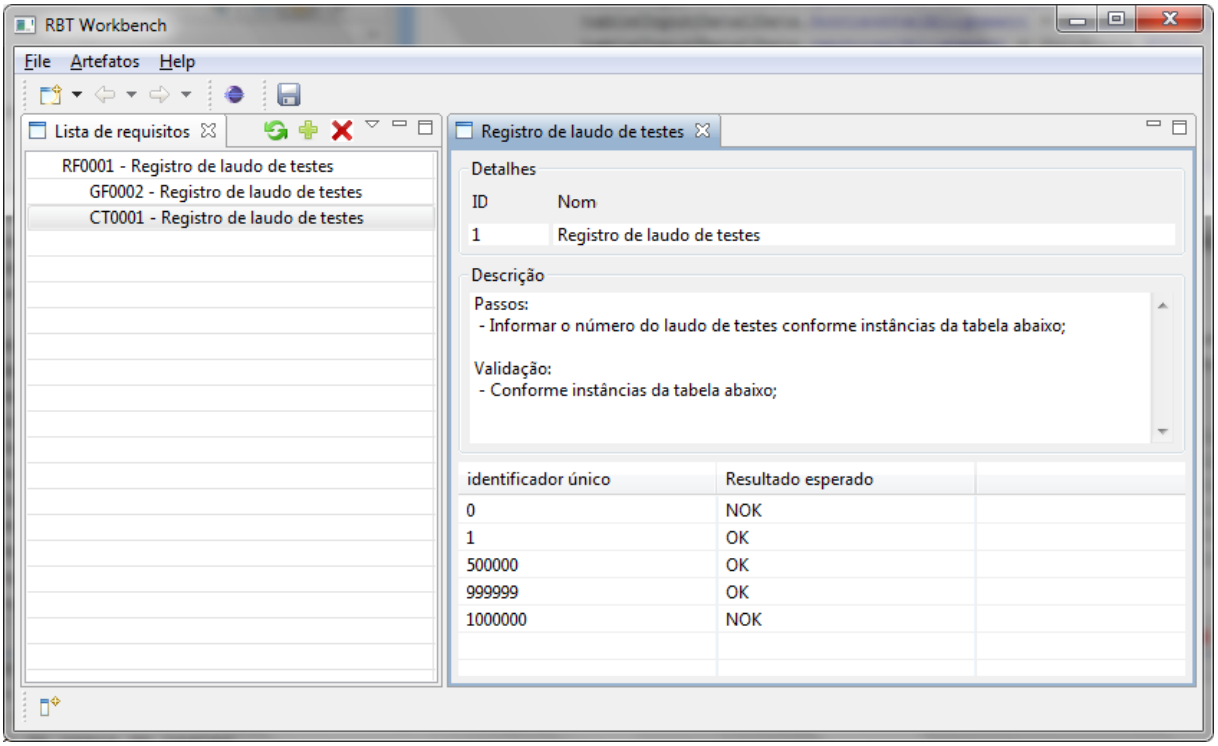


Figura 33 – Caso de teste para cadastramento de funcionários

A geração realizada a partir de um grafo de causa e efeito apresenta os testes de uma maneira um pouco diferente, onde são consideradas as ligações entre as propriedades, definidas através do grafo, além das regras e operações definidas para as conexões.

A figura 33 apresenta também a matriz de rastreabilidade montada pela ferramenta. Ela é representada através de uma estrutura de árvore, onde é possível visualizar todos os artefatos que estão relacionados a um requisito. No exemplo, o requisito RF0001 – Registrar laudo de testes possui um grafo de causa e efeito (GF002) e um caso de teste (CT001) associado.

Esta matriz é utilizada pela ferramenta para saber quais os testes que devem ser executados por um plano de testes.

3.3.2.6 Execução dos casos de testes

A execução dos casos de testes é iniciada através da seleção do plano de testes. Ao

clicar no ícone que representa o início da execução, um assistente guia o testador através das entradas, apresentando quais os valores que devem ser utilizados no teste corrente e qual o resultado esperado. Para avançar ao próximo teste, o testador deve informar qual o resultado obtido com a entrada atual.

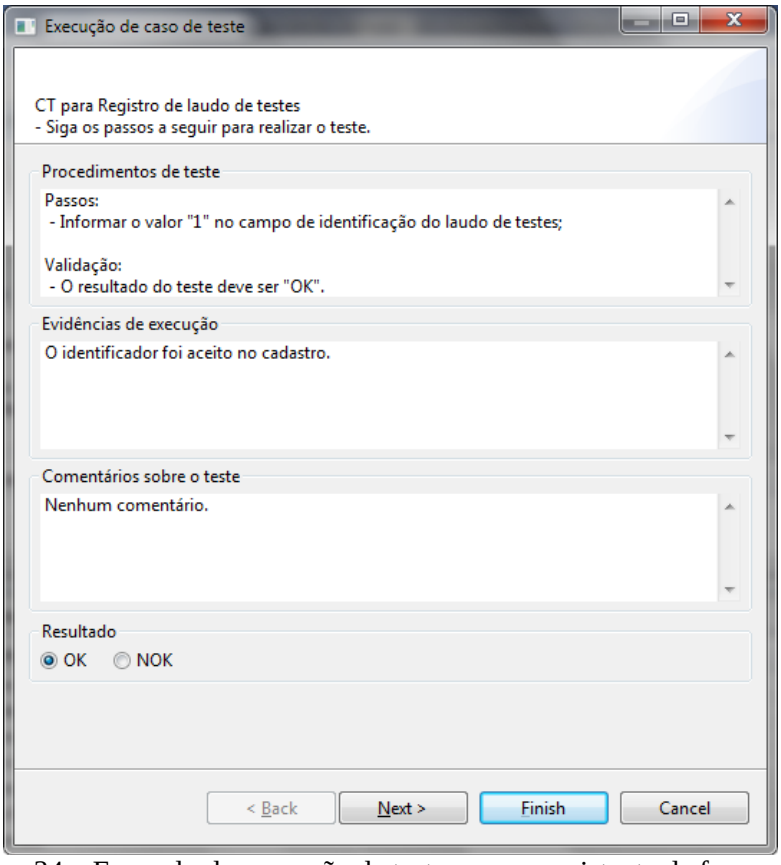


Figura 34 – Exemplo de execução de testes com o assistente da ferramenta

A figura 34 apresenta o assistente e execução de testes, onde os dados do caso de teste são apresentados ao usuário. O resultado obtido na execução do teste deve ser informado no painel “Resultado”. Os campos de “Evidência de execução” e “Comentários sobre o teste” são utilizados para prover mais informações sobre a execução dos testes para uma posterior análise de eficiência dos testes.

Ao final da execução, é possível utilizar uma das visualizações disponibilizadas para verificar qual o resultado dos testes executados. A figura 35 ilustra o resultado de uma bateria de testes.

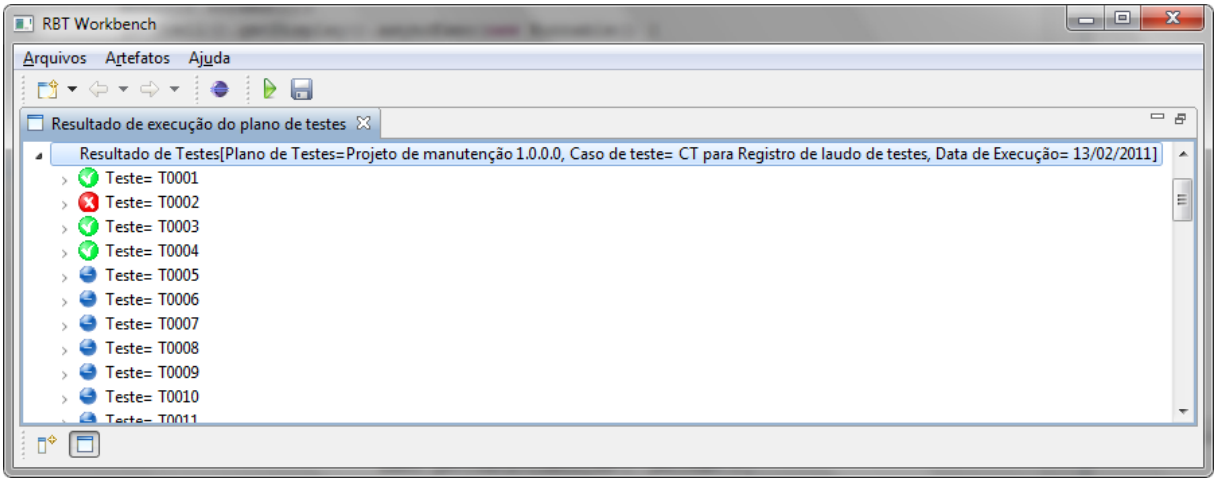


Figura 35 – Resultado de execução de testes

Os testes com ícone verde representam aqueles cujo resultado do teste foi *OK*. Aqueles com ícone vermelho indicam que o resultado do teste foi Não *OK*. Os testes que não foram executados, no caso de interrupção da execução dos testes, são representados pelo ícone azul.

3.3.2.7 Gerar nova versão do projeto

Através do menu Arquivo/Novo é possível configurar uma nova versão para o projeto. Uma nova versão consiste basicamente em um novo número de versão imutável, gerado pela ferramenta, um nome de identificação, uma descrição sucinta (opcional) e uma data de início. A figura 36 apresenta a tela de cadastro de novas versões.

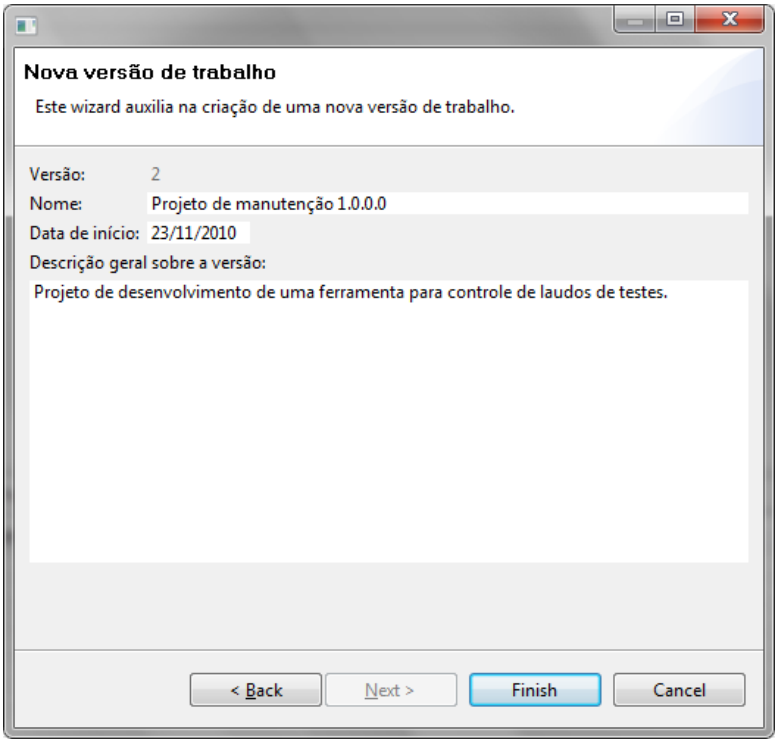


Figura 36 – Registro de nova versão do projeto

Cada vez que uma nova versão é criada, a ferramenta trata de gerar um novo plano de testes e fechar a versão anterior, além de passar a registrar neste novo plano os requisitos e testes que sofreram algum tipo de alteração no decorrer da versão.

O retorno a versões anteriores do projeto não é suportado pela ferramenta. Para isto, é necessário utilizar uma ferramenta de controle de versão externa, onde os arquivos anteriores possam ser recuperados.

3.3.2.8 Exibir e manter o plano de testes

O plano de testes é criado de forma automática pela ferramenta toda vez que uma nova versão do projeto é iniciada. Os testes ou requisitos alterados pela ferramenta são prontamente incluídos no plano de testes. Através dos grafos de causa e efeito, a ferramenta consegue identificar também outros requisitos que podem ser impactados pela alteração, e também trata de incluir seus testes no plano. A figura 37 apresenta a aparência do plano de testes disponibilizado na ferramenta.

É possível também incluir testes de forma arbitrária no plano ou então todos os testes registrados na ferramenta, com o intuito de realizar testes de regressão, assegurando que as alterações realizadas não impactaram no funcionamento dos demais requisitos.

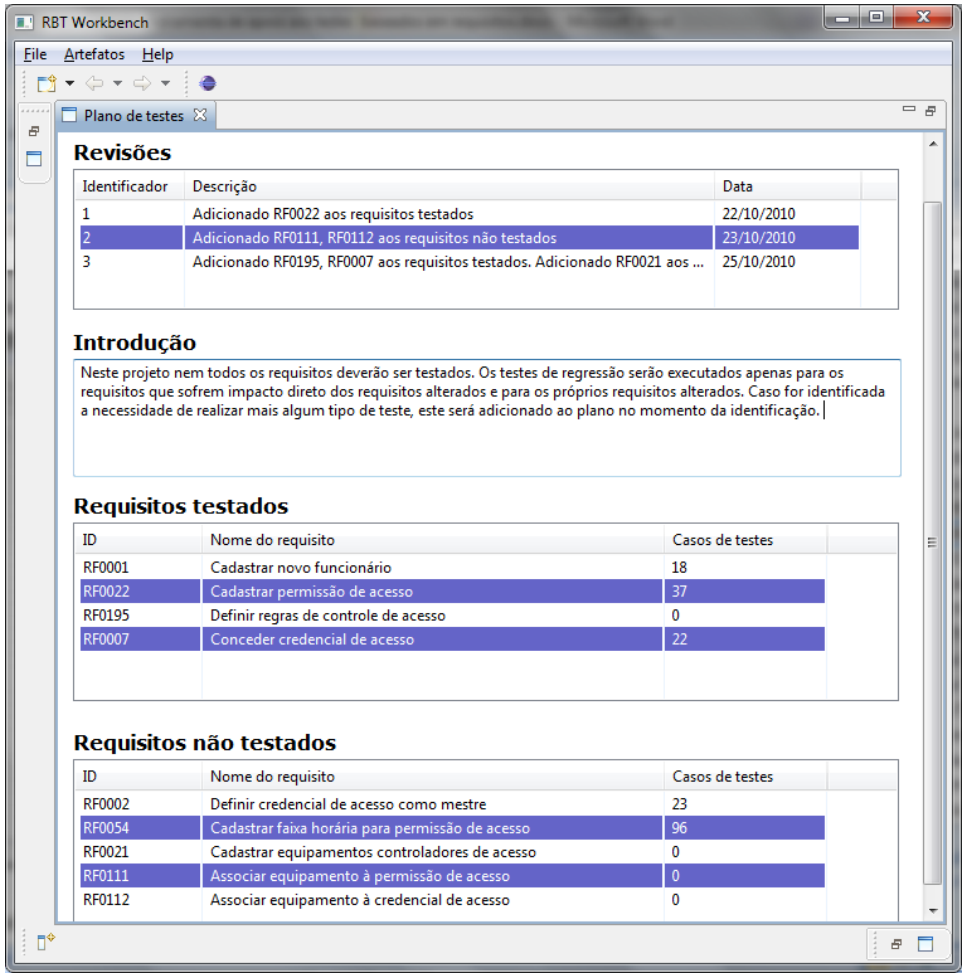


Figura 37 – Plano de testes

3.4 RESULTADOS E DISCUSSÃO

Partindo do princípio da necessidade de construção de uma ferramenta que trabalhe os testes a partir dos requisitos, foi desenvolvida uma ferramenta que permite desde o registro de requisitos até a execução dos casos de testes relacionados a ele.

O desenvolvimento contribuiu com a melhoria da concisão dos requisitos ao permitir que eles fossem quantificados, através da marcação de palavras chaves e da adição de propriedades para elas.

Através desta contribuição, foi possível também criar uma ferramenta para modelagem de grafos de causa e efeito, utilizando como base as palavras chaves do requisito. O grafo permite construir uma visão lógica do que está descrito no requisito, além de ajudar a melhorar a descrição dele. Ao montar o grafo de causa e efeito é possível identificar novas palavras chaves não visualizadas no requisito, onde é possível então retornar e melhorar a sua

descrição.

Tanto as palavras chaves do requisito quanto o grafo de causa e efeito são utilizados na montagem dos casos de testes. Esta montagem pode ocorrer tanto de forma manual quanto de forma semi-automatizada. Ao montar um caso de teste manualmente, as definições dos dados de entrada do mesmo ficam sob responsabilidade do usuário. Caso optar pela geração semi-automatizada, os dados de entrada são gerados pela ferramenta, tendo como base as palavras chaves definida para os requisitos e o grafo de causa e efeito.

Os requisitos registrados na ferramenta são monitorados para que qualquer mudança realizada faça com que ele seja adicionado ao escopo de testes do projeto. Esta adição é realizada no plano de testes, no qual é possível visualizar quais os requisitos que estão dentro do escopo de testes e quais não estão além de revisões e uma introdução para o plano.

O plano de testes possui também um assistente que auxilia na execução dos casos de testes. Este assistente guia o usuário através dos casos de testes que devem ser executados para cada requisito que estiver no escopo da versão.

Ao final é possível visualizar quais os requisitos que possuem testes que falharam, possibilitando uma visão bastante prática dos pontos do sistema que precisam de ajustes antes da entrega da versão para o mercado.

O quadro 8 apresenta uma comparação com outras ferramentas que também possuem propósito de auxiliar no gerenciamento dos testes baseados em requisitos. Na tabela foram adicionadas apenas as funcionalidades consideradas importantes para a ferramenta desenvolvida no presente trabalho. Todas as demais ferramentas da comparação possuem funcionalidades que o RBT *Workbench* não atende.

A principal limitação da ferramenta ficou por conta da falta de mais relatórios gerenciais, tais como relatórios de incidentes de testes, relatório resumo de testes, relatório de *log* de testes, entre outros.

Funcionalidade	*SilkCentral Test Manager*	*TarGeT*	*TestLink*	*RBT Workbench*
Realizar o acompanhamento de alterações em requisitos	Atende parcialmente	Não atende	Não atende	Atende
Provê rastreabilidade do requisito para os testes	Atende parcialmente	Atende	Atende	Atende
Registrar planos de testes	Não atende	Não atende	Atende	Atende parcialmente
Registrar propriedades para quantificar os requisitos	Não atende	Não atende	Não atende	Atende
Gerar casos de testes para os requisitos	Não atende	Atende	Não atende	Atende
Criar grafos de causa e efeito para utilização em testes	Não atende	Não atende	Não atende	Atende
Registrar *suítes* de testes e distribuir testes entre elas	Não atende	Atende	Atende	Não atende
Auxilia a manutenção do plano de testes durante o projeto	Atende parcialmente	Atende	Não atende	Atende
Possui assistente para a execução dos casos de testes	Atende	Não atende	Atende	Atende
Gera relatórios de testes	Atende	Atende	Atende	Não Atende

Quadro 8 – Comparativo de funcionalidades dos trabalhos correlatos

4 CONCLUSÕES

O principal objetivo da ferramenta construída é dar suporte ao desenvolvimento de testes baseados em requisitos. Este suporte é alcançado ao realizar a rastreabilidade dos requisitos para os casos de testes e vice-versa. Utilizando a ferramenta para realizar o registro de requisitos e seus respectivos casos de testes, é possível obter uma boa produtividade ao garantir que os requisitos alterados estão dispostos no escopo de testes da versão do software em produção.

Ao utilizar a técnica de testes baseados em requisitos, unindo a construção de grafos de causa e efeito, a definição de classes de equivalência e valores limite, foi possível desenvolver uma ferramenta que auxilie na geração dos casos de testes, tornando mais rápida a produção de testes que validem o funcionamento da característica desenvolvida.

A forma como a ferramenta foi desenvolvida, utilizando palavras chaves selecionadas pelo usuário dentro do texto do requisito, permite que este texto continue sendo escrito em linguagem natural, sem a necessidade de um vocabulário específico para o fim de geração de testes.

As propriedades que são adicionadas ao requisito possibilitaram a geração dos casos de testes bastante abrangentes, de forma que cada requisito possua 100% de cobertura com o menor esforço de testes possível.

O apoio ao desenvolvimento de testes utilizando particionamento de equivalência e valor limite ocorreu utilizando propriedades definidas para as palavras chaves do requisito. Ao definir limites superiores e inferiores a ferramenta realiza a geração de testes utilizando tais técnicas para limitar a quantidade de testes gerados.

Quanto aos requisitos da ferramenta, ela permite realizar o registro de requisitos, porém sem realizar nenhum tipo de tratamento para verificar se o requisito é funcional ou não funcional.

A rastreabilidade dos casos de testes para os requisitos é realizada através de um vínculo criado entre as duas entidades. Qualquer alteração no requisito gera uma pendência de execução do teste. A matriz de rastreabilidade é apresentada através da lista de requisitos, onde são apresentados os grafos de causa e efeito e os casos de testes relacionados a cada requisito.

O agrupamento de testes em *suítes* não foi atendido pela ferramenta. Optou-se por manter apenas o agrupamento de testes por requisito.

O plano de testes foi criado de modo que seja possível visualizar todos os requisitos que serão ou não testados. Foi realizada também a automação da inclusão de requisitos para serem testados no plano de testes.

A ferramenta foi desenvolvida utilizando a linguagem Java e por fim, desenvolvida na forma de um *plug-in* para a plataforma *Eclipse*. Ela pode ser distribuída em dois formatos: Standalone, onde a ferramenta é uma aplicação totalmente isolada do ambiente Eclipse IDE, e também na forma de plug-in, onde basta copiar o arquivo com extensão "jar" para a pasta *plugins* do ambiente *Eclipse*.

4.1 EXTENSÕES

A ferramenta desenvolvida pode ser estendida para abrigar mais funcionalidades relacionadas à engenharia de software.

a) desenho do projeto do software, com diagrama de classes que utilizem as propriedades definidas no requisito, ou com mapeamento para elas;
b) criar uma biblioteca onde seja possível cadastrar palavras chaves que são comuns a vários requisitos, promovendo a reutilização delas;
c) geração de *scripts* de teste automatizado utilizando os dados gerados pela ferramenta;
d) migração para a plataforma *Eclipse Rich Client Platform (Eclipse RAP)*, possibilitando a sua execução através de um browser;
e) adicionar mais regras e operações para o tratamento de propriedades;

REFERÊNCIAS BIBLIOGRÁFICAS

BORBA, Paulo; SILVA, Michelle. **TaRGeT**: test and requirement generation tool. [Recife?], 2010. Disponível em: <http://twiki.cin.ufpe.br/twiki/bin/view/CInBTCResearchProject/ToolTarget>. Acesso em: 30 mar. 2010.

BORLAND SOFTWARE CORPORATION. **SilkCentral test manager**. [S.l.], [2010]. Disponível em: <http://www.borland.com/br/products/silk/silkcentral_test/index.html>. Acesso em: 30 mar. 2010.

DIAS NETO, Arilo C. **Introdução a teste de software**. Rio de Janeiro, 2010. Disponível em: <http://www.devmedia.com.br/post-8035-Artigo-Engenharia-de-Software-Introducao-a-Teste-de-Software.html>. Acesso em: 30 mar. 2010.

ECLIPSE PLUGIN DEVELOPMENT. **Introduction to Eclipse plugin development**. [S.l.], 2008. Disponível em: <http://www.eclipsepluginsite.com/index.html>. Acesso em: 30 mar. 2010.

ECLIPSE PROCESS FRAMEWORK COMPOSER. **Iniciando com XP**. [S.l.], 2008. Disponível em: <http://epf.eclipse.org/wikis/xppt/xp/guidances/guidelines/equivalence_class_analysis_E178943D.html?nodeId=19be24aa>. Acesso em: 22 nov. 2010.

GOLDSMITH, Robin F. **Pros and cons of requirements-based software testing**. [S.l.], 2009. Disponível em: <http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1347997_mem1,00.html>. Acesso em: 7 abr. 2010.

GUTIÉRREZ, Javier J. et al. **Generation of test cases from functional requirements**: a survey. Seville, [2006]. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.6641&rep=rep1&type=pdf>. Acesso em: 23 maio 2010.

MINOCHA, Sandy. **Desenvolvendo seu primeiro aplicativo Eclipse RCP***:* criar uma rica aplicação cliente está mais fácil do que nunca, graças ao Eclipse V3.1. Toronto, 2006. Disponível em: <http://www.ibm.com/developerworks/br/library/os-ecl-rcpapp/section2.html>. Acesso em: 22 nov. 2010.

MYERS, Glenford J. **The Art of Software Testing**. New York: John Wiley & Sons, 1979.

LEITE, Jair C. **Notas de aula de engenharia de software**. Natal, 2000. Disponível em: <http://www.dimap.ufrn.br/~jair/ES/c8.html>. Acesso em: 22 nov. 2010.

MACORATTI.NET. **A gestão de requisitos**. [S.l.], 1999. Disponível em: <http://www.macoratti.net/vb_conc2.htm>. Acesso em: 7 abr. 2010.

MKS. **Requirements based tesing**: encourage collaboration through traceability. [S.l.], 2009. Disponível em: <http://www.mks.com/requirements_based_testing_RBT>. Acesso em: 30 mar. 2010.

MOGYORODI, Gary. **Requirements-based testing proof of concept**. Ontario, 2008. Disponível em: <http://softestserv.ca/Requirements-Based%20Testing%20Proof%20of%20Concept%20STS.pdf>. Acesso em: 7 abr. 2010.

NURSIMULU, Khenaidoo; PROBERT, Robert L. **Cause-effect graphing analysis and validation of requirement**. Ontario, [1995]. Disponível em: <http://www.cs.ubc.ca/local/reading/proceedings/cascon95/pdf/nursimul.pdf>. Acesso em: 30 mar. 2010.

PARRINGTON, Norman; ROPER, Marc. **Understanding software testing**. Chichester: John Wiley & Sons, 1989.

PERRY, William E. **A structured approach to systems testing**. 2. ed. Wellesley: QED Information Sciences, 1988.

RAMACHANDRAN, Muthu. **Requirements-driven software test**: a process oriented approach. New York, 1996. Disponível em: <http://portal.acm.org/citation.cfm?id=232088&dl=GUIDE&coll=GUIDE&CFID=83158805&CFTOKEN=57295929>. Acesso em: 6 abr. 2010.

RIOS, Emerson et al. **Base de conhecimento em testes de software**. 2. ed. São Paulo: Martins, 2007.

RYSER, Johannes; GLINZ, Martin. **A practical approach to validating and testing software systems using scenarios**. Zurich, 2000. Disponível em: <http://www.ifi.unizh.ch/groups/req/ftp/papers/QWE99_ScenarioBasedTesting.pdf>. Acesso em: 06 abr. 2010.

SHAVOR, Sherry et al. **The Java developer's guide to Eclipse**. Boston: Addison-Wesley, 2003.

SRIVASTAVA, Praveen R.; PATEL, Parshad; CHATROLA, Siddharth. **Cause effect graph to decision table generation**. Pilani, 2009.Disponível em: <http://portal.acm.org/citation.cfm?id=1507195.1507216>. Acesso em: 06 abr. 2010.

TESTLINK COMMUNITY. **TestLink**: open source test management. [S.l.], 2010. Disponível em: <http://www.teamst.org/>. Acesso em: 30 mar. 2010.