ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

RBT Tool: Uma Ferramenta para Identificação de Riscos no Teste de Software

Trabalho de Conclusão de Curso

Engenharia da Computação

Keldjan Alves de Oliveira
Orientadora: Prof. Drª. Cristine Gusmão

UNIVERSIDADE
DE PERNAMBUCO

Keldjan Alves de Oliveira

RBT Tool: Uma ferramenta para Identificação de Riscos no Teste de Software

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro 2008.

Dedico este trabalho a todos que depositaram em mim confiança na execução deste trabalho e também dedico à minha mãe que conseguiu agüentar vários períodos difíceis durante esses cinco últimos anos a fim de me apoiar na minha formação como engenheiro da computação.

Agradecimentos

Primeiramente a Deus por tudo que tem me proporcionado.

Aos meus pais, Hosana Jacinta Alves de Oliveira e Luiz Gonzaga de Oliveira Filho, os quais me apóiam e incentivam em todos os instantes da minha vida.

A Ellen Polliana Ramos Souza pela oportunidade de trabalho e pelo suporte durante toda a jornada de elaboração deste trabalho. Sua inteligência, paciência e ensinamentos foram de suma importância para a concretização da *RBT Tool*.

A Júlio Venâncio de Menezes Júnior pelo apoio nesta empreitada e pela paciência em momentos difíceis. Seu apoio é intangível para este trabalho e para o grupo PROMISE ao qual nos juntamos para construir a *RBT Tool*.

A minha orientadora, Prof Drª Cristine Martins Gomes de Gusmão, pela confiança, paciência e compreensão dedicada em todos os momentos. Sua orientação foi fundamental para tornar este trabalho uma realidade.

A Rosely Bandim Medeiros de Almeida pelo apoio em momentos de difíceis me fornecendo segurança e confiança. Deus fez com que anjos cruzassem meu caminho durante minha vida. Graças a esses anjos esse trabalho pôde se concretizar.

A todos os meus colegas que de várias formas ajudaram na elaboração da *RBT Tool* de forma mais direta ou indiretamente. César Oliveira, Renata Wanderley, Danilo Carvalho, Christian Liedtke, Vicente Bezerra, Lúcio Ribeiro, entre vários outros.

Resumo

A atividade de testes tem se mostrado fundamental e de bastante relevância no processo de desenvolvimento de software, evitando retrabalho e que falhas cheguem aos clientes. No entanto, esta atividade é extremamente difícil e demorada, devido ao enorme domínio de entradas e saídas de um sistema, além das diversas possibilidades de caminhos possíveis a serem testados num software. Nesse âmbito, teste baseado em riscos (*Risk-based Testing* - RBT) tem surgido como uma abordagem para minimizar alguns desses problemas de testes de software priorizando esforços e direcionando recursos para os componentes de software que necessitam ser testados cuidadosamente. Dentro deste contexto, o propósito deste trabalho é a construção de uma ferramenta que auxilie as equipes de desenvolvimento de software, em especial os engenheiros de teste, na etapa de identificação de riscos nos requisitos para a realização do teste de software. As tecnologias utilizadas no desenvolvimento desta solução visaram tornar o sistema flexível e de fácil utilização contribuindo nos alcance dos objetivos propostos.

Abstract

Testing has been fundamental and very relevant in a software development process avoiding rework and fails delivering to the customers. However, this activity is extremely difficult and lengthy, due to huge amount of inputs and outputs from a system, in addition to the various possibilities for possible paths to be tested in a software. In this context, Risk-based Testing (RBT) has emerged as an approach to minimize some of these issues on software testing designating resources and prioritizing efforts for software components which need to be tested carefully. In this context, the purpose of this work is a tool development which helps the whole software development team, specially test engineers in the stage of risk identification upon requirements on software test. When developing the tool, technologies were used aiming a flexible and easy system which contributes to objective attainment.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

RBT – *Risk Based Testing*

JVM – Java *Virtual Machine*

XML – *eXtensible Markup Language*

REM – *Requirements Management*

RCP – *Rich Client Plataform*

IDE – *Intregrated Development Enviroment*

PDE – *Plug-in Development Environment*

UML – *Unified Modeling Language*

SVN – *Subversion*

Capítulo 1

Introdução

Nos últimos anos, são notáveis os avanços tecnológicos alcançados pela sociedade moderna. Esse crescimento tem gerado um ambiente competitivo que exige uma adaptação por parte das empresas as quais desejam manter-se na liderança de seus mercados. Dessa forma, novas abordagens são desenvolvidas como forma de se reduzir custos, além de melhorar o aproveitamento do tempo e de recursos. Porém, o foco geralmente está ligado a qualidade oferecida por estes produtos.

Neste cenário, testes de software surgem como uma das formas para encontrar e remover defeitos e para avaliar o grau de qualidade de um produto e dos seus componentes de acordo com as necessidades de cada cliente ou usuário [1]. A execução de testes é utilizada, além disso, para avaliar de forma quantitativa a qualidade de um produto que está sendo desenvolvido.

A atividade de testes tem se mostrado fundamental e de extrema relevância no processo de desenvolvimento de software evitando retrabalho e, principalmente, que falhas cheguem aos clientes deixando-os insatisfeitos ou impedindo-os de realizar suas atividades. No entanto, esta atividade é extremamente difícil e demorada, uma vez que, o domínio de entradas e saídas de um sistema é enorme, bem como, as possibilidades de caminhos possíveis a serem testados num software.

A técnica de testes baseada em análise de riscos tem surgido como uma abordagem para minimizar alguns desses problemas de testes de software de forma geral. O Teste baseado em Risco (*Risk-based Testing* - RBT) consiste em um conjunto de atividades que favorecem a identificação de fatores de riscos associados aos requisitos do produto de software [2]. Uma vez identificados, os riscos são priorizados e casos de testes gerados com base nas estratégias para tratamento e acompanhamento dos fatores de riscos identificados. O controle da execução dos casos de testes permite uma mitigação dos fatores de riscos

associados. Dessa forma, o uso da abordagem permite priorizar esforços e alocar recursos para os componentes de software que necessitam ser testados cuidadosamente, fazendo com que o processo de testes deixe de ser apenas um processo robótico [3].

1.1 Motivação

Risco, em um projeto de software, pode ser representado como uma medida da probabilidade e da perda relacionada à ocorrência de um evento negativo (desvio com relação aos requisitos inicialmente solicitados) que afete o próprio projeto, seu processo ou o seu produto [4], em outras palavras, qualquer evento que possa acontecer e ameaçar o bom andamento do projeto é considerado um risco.

Então, é importante que sejam adotadas ações de prevenção que pemitam um controle maior destes eventos desfavoráveis. Neste contexto, o RBT assume um papel bastante promissor devido às suas características intrínsecas que toma por base a prioridade de componentes ou parte do sistema que apresentam maior probabilidade de falhas.

Embora, as empresas de forma geral tenham despertado para a importância da atividade de teste de software como forma de melhorar a qualidade dos seus produtos e manterem-se competitivas no mercado, esta atividade representa um custo bastante significativo, chegando a custar até 45% do valor inicial de um produto de software [1].

Dessa forma, torna-se indispensável a utilização de processos, métodos, técnicas e ferramentas que permitam a realização da atividade de teste de software de maneira sistematizada e com fundamentação teórica, com o propósito de aumentar a qualidade do produto de software com o menor custo possível [5].

Apesar da abordagem RBT parecer simples, os engenheiros de testes ainda encontram dificuldades em aplicá-la na prática, por não conhecerem técnicas para identificação e análise de riscos para teste de software e também pela ausência de ferramentas específicas que auxiliem na utilização dessas técnicas[6].

1.2 Objetivos gerais e específicos

O objetivo principal deste trabalho é a construção de uma ferramenta para identificação de riscos no teste de software.

A abordagem de testes baseados em riscos é utilizada, mais fortemente, no gerenciamento e estratégia de execução de testes. No entanto, é recomendado o seu uso também na construção dos casos de testes como forma de aperfeiçoar o processo de testes.

Para alcance deste objetivo é importante, de forma específica: (i) Identificar os principais requisitos para uma ferramenta de auxílio aos engenheiros de teste na etapa de identificação de riscos, (ii) elaborar estudo das diferentes técnicas para teste de software e (iii) identificação de riscos a fim de selecionar uma abordagem para implementar na ferramenta.

1.3 Metodologia

Este trabalho foi realizado em fases buscando em cada uma delas estabelecer estratégias que ajudaram no aprofundamento e ampliação de certos conhecimentos tidos como imprescindíveis à consecução dos objetivos enunciados na seção anterior.

As fases foram organizadas da seguinte forma:

Fase 1 – Revisão Bibliográfica de Fundamentação Teórica

Definidos os objetivos, foi realizada a revisão da literatura que buscou sintetizar os principais conceitos, tendências e abordagens referentes ao tema. Os assuntos são: testes de software, riscos de software e *Risk-based Testing* (testes baseados em riscos).

Em seguida, partiu-se para uma fundamentação teórica, que baseou-se na revisão de literatura.

Fase 2 – Definição do Escopo

De posse do conhecimento teórico mudou-se para a fase seguinte que tratou de definir o escopo do projeto. A primeira parte dessa definição consistiu em estabelecer quais os requisitos genéricos seriam implementados na ferramenta.

Fase 3 – Modelagem e Construção da Ferramenta

Definido o escopo da implementação, iniciou-se o processo de implementação da ferramenta. Houve a preparação do ambiente de desenvolvimento além da definição que quais tecnologias seriam adotadas na construção do software.

Fase 4 – Teste da Ferramenta

Nesta fase a ferramenta foi testada a fim de verificar a consistência das funcionalidades implementadas.

1.4 Estrutura do Documento

Após este capítulo introdutório e para uma melhor explanação das ações realizadas para alcançar os objetivos, este documento está formado pela seguinte estrutura:

Capitulo 2 – Teste de Software e Riscos – este capítulo apresenta os conceito de teste de software bem como os níveis e as técnicas existentes. Apresenta, também, os conceitos relacionados a requisitos e riscos descrevendo as abordagens existentes. Além disso, ele procura analisar as lacunas existentes em cada uma destas áreas que fazem parte da engenharia de software.

Capitulo 3 – Modelagem e Construção da Ferramenta – este capítulo apresenta as tecnologias que foram adotadas durante a implementação da ferramenta e mostra todos os elementos que compuseram o processo de construção como um todo, desde a elicitação de requisitos aos testes de validação das funcionalidades implementadas. A arquitetura definida também é mostrada nesta seção.

Capitulo 4 – Considerações Finais e Trabalhos Futuros – este capítulo tem como objetivo apresentar resultados e considerações finais sobre a ferramenta, além de apresentar sugestões de trabalhos futuros. Neste capítulo, ainda serão mostradas as dificuldades e trabalhos relacionados.

Capítulo 2

Teste de Software e Riscos

Ao adquirirmos um produto ou serviço a idéia mais natural é de que este funcione corretamente, satisfazendo as exigências que possuímos com um nível de qualidade aceitável por nós. Na Engenharia de Software o mesmo padrão é seguido. O cliente ao solicitar um software deseja que este possua um nível de qualidade que atenda às suas expectativas e necessidades, isto é, ele deseja um sistema realize o que foi solicitado e acordado sem falhas.

Aguiar [4] define que risco em um projeto de software pode ser representado como uma medida da probabilidade e da perda relacionadas à ocorrência de um evento negativo que afete o próprio projeto, seu processo ou o seu produto. Em outras palavras, qualquer coisa que possa acontecer e ameaçar o bom andamento do projeto - um problema em potencial, o fracasso de um ou mais componentes: prazo, custo, atividades - é um risco. Não é certo que ele ocorrerá, mas se ocorrer, poderá trazer prejuízos.

Inserido nesse contexto, esse capítulo destina-se a introduzir os conceitos de teste e de identificação de risco de software (Seção 2.1), mostrando os principais aspectos referentes à atividade de testes e algumas maneiras de se identificar riscos (Seção 2.2) dentro do processo de desenvolvimento de software, bem como suas vantagens e desvantagens.

2.1 Teste de Software

Teste de software é uma das atividades cujo objetivo é verificar e validar um produto que foi desenvolvido. Pressman [7] argumenta que a fase de testes é aquela onde é executada uma série de atividades que consistem em avaliar o produto de software criado, através do código desenvolvido e/ou das funcionalidades disponibilizadas, a procura de qualquer tipo de erro ou falha, que possa vir a interferir no correto funcionamento do aplicativo. O teste deve detectar o maior

número de erros possível para que eles possam ser corrigidos antes da entrega final do produto.

2.1.1 Níveis de Teste de Software

Os níveis de teste definem o momento do ciclo de vida do software em que são realizados os testes. Em cada nível do processo de software, desde o levantamento dos requisitos até o desenvolvimento do programa, testes estáticos ou dinâmicos podem ser realizados. A Figura 2.1 apresenta os principais níveis e estágios.

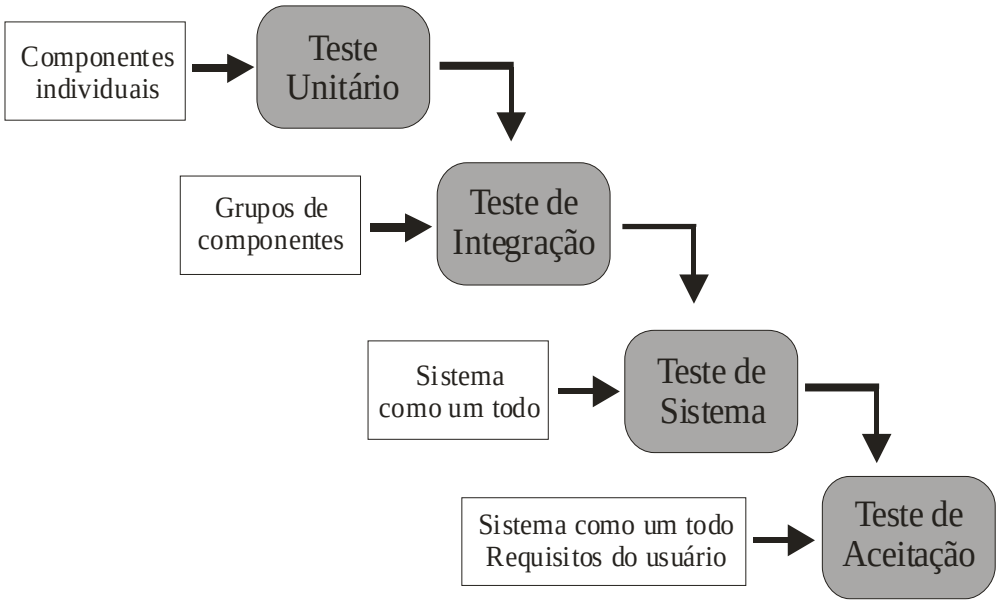

Figura 2.1. Níveis de Testes [8].

Abaixo, estão descritos os níveis apresentados na Figura 2.1:

- **Teste Unitário** - Essa etapa do processo de teste é responsável por verificar as pequenas partes do software desenvolvido, ou seja, se concentra na verificação da menor unidade do projeto de software. Tem como alvo: uma função ou um procedimento, em um sistema procedural; uma classe ou um método, em um sistema orientado a objeto; ou qualquer trecho pequeno de código.

- **Teste de Integração** – É uma técnica sistemática para formar a estrutura do programa, realizando testes para descobrir falhas provenientes da

integração interna dos componentes de um sistema. A integração entre sistemas distintos não faz parte dessa etapa de teste. Essas interfaces são testadas na fase seguinte, a de teste de sistema.

- **Teste de Sistema** – Essa fase de teste busca testar o sistema de um modo geral, a partir da visão do usuário final. Todas as funcionalidades são testadas em busca de falhas. Assim, os testes são executados em condições reais e semelhantes às que o usuário final utilizará em seu cotidiano ao manipular o sistema.

- **Teste de Aceitação** - É a fase de teste realizada pelo usuário final, que simula operações rotineiras do sistema para verificar se as funcionalidades do software estão de acordo com o que fora solicitado. A validação é bem sucedida quando o software funciona de uma maneira razoavelmente esperada pelo cliente [7].

2.2 Técnicas e Métodos de Identificação de Riscos

Na literatura, de uma forma geral, os autores consideram a fase de identificação de risco como uma das mais importantes em todo processo do gerenciamento de riscos, pois apresenta um impacto maior na acuracidade das avaliações de riscos, já que a forma como os riscos são identificados e coletados constituem-se na questão central para a efetividade de todo este processo [9]. Segundo o Guia PMBOK [10], a fase de identificação de riscos compreende a determinação de quais riscos podem afetar o projeto e em documentar as suas características.

Através do levantamento bibliográfico realizado observa-se a existência de várias técnicas propostas para a identificação de riscos. A seguir, serão apresentadas de forma resumida, as técnicas de identificação de risco mais comuns.

2.2.1 Listas de critério de qualidade (Checklists)

As listas de critério de qualidade, conhecidas também como *checklists*, consiste em uma lista de itens, que vão sendo marcados podendo ser utilizada por um membro da equipe, em grupo ou em uma entrevista. A Tabela 2.1 relaciona três riscos com três requisitos. A identificação dos riscos através desta técnica se dá pela marcação dos campos de junção entre o risco e o requisito. Os campos onde não há uma marcação considera-se que não há este risco no requisito correspondente.

Tabela 2.1. Identificação de Riscos com Listas

	Clareza	Completude	Validação
Criar Projeto	X	X	
Remover Lista	X		X
Inserir Questão	X	X	X

Neste exemplo da Tabela 2.1, o requisito “Criar Projeto” e “Inserir Questão” possuem o risco “Completude” associados uma vez que os campos correspondentes estão marcados. Porém, o requisito “Remover Lista” não possui este risco atrelado.

Assim, verificamos que a identificação dos riscos através deste tipo de abordagem requer um conhecimento prévio dos riscos que atigem um determinado requisito do sistema. Esta é sua principal desvantagem na identificação dos riscos.

2.2.2 Questionário

Os questionários são compostos de perguntas cuja finalidade é identificar os riscos que estão relacionados a um determinado requisito no projeto. O seu preenchimento pode consistir de respostas binárias, sim ou não, e através destas é possível saber qual risco atinge um determinado requisito. Existem outros tipos de questionários cujas respostas podem ser baseada em múltiplas escolhas ou ainda respostas abertas. A Tabela 2.2 mostra um exemplo de uso dessa técnica.

Tabela 2.2. Identificação de Riscos com Questionário

	Este requisito é capaz de fazê-lo entender todo o comportamento da funcionalidade?	Este requisito foi validado com o cliente?	Este requisito está mudando enquanto a funcionalidade está sendo desenvolvida?
Criar Projeto	**Sim**	**Não**	**Sim**
Remover Lista	**Sim**	**Não**	**Não**
Inserir Questão	**Sim**	**Sim**	**Sim**

Neste exemplo, todas as perguntas são respondidas com um sim ou um não em relação a um determinado requisito. Assim, na Tabela 2.2 a pergunta “Este requisito foi validado com o cliente?” foi respondida com um “NÃO” em relação ao requisito “Criar Projeto”. Dessa forma, se um determinado risco, por exemplo “Completude”, estiver atrelado a resposta “NÃO” nesta pergunta, o requisito “Criar Projeto” e “Remover Lista” terá este risco como identificado. Já o requisito “Inserir Questão” não apresenta este risco uma vez que sua resposta foi “SIM”.

A vantagem desta técnica está na não necessidade de conhecimento prévio sobre riscos por parte da pessoa que está respondendo ao questionário. Assim, neste trabalho, foi escolhida esta técnica como a que seria implementada na ferramenta *RBT Tool*.

2.2.3 Metodologia Heurística

Um método heurístico para encontrar uma solução é um método útil que não funciona sempre. Bach [3] mostra como conduzir uma análise de riscos através de

heurísticas no processo de teste baseado em riscos. O autor define o processo RBT (*Risk Based Testing*) em três atividades distintas:

1) Criar uma lista de riscos priorizadas;

2) Realizar testes que explora cada risco;

3) Ajustar o esforço de teste focando nos riscos.

Em seu trabalho, o autor mostra duas abordagens diferentes para identificação de riscos: a abordagem “de dentro para fora” (*inside-out approach*) e a abordagem “de fora para dentro” (*outside-in approach*).

Na abordagem “de dentro para fora” o produto é estudado e se pergunta repetidamente o que pode dar errado no mesmo, isto é, quais as vulnerabilidades, ameaças e vítimas existentes. Nessa abordagem, é elaborada uma lista de riscos e, também, estratégias para mitigar estes riscos. Ainda, é exigido um conhecimento da técnica por parte do entrevistador, visto que fatores como segurança, confusões e ambigüidades são analisadas para entrevistar o especialista.

Já na abordagem “de fora para dentro” é utilizada uma lista de riscos potenciais pré-definidos e verificado se os mesmos se aplicam a uma determinada funcionalidade. Neste enfoque, Bach mostra três tipos de listas de riscos normalmente utilizadas:

1) Listas de Categorias de Critérios de Qualidade – agrupam categorias para atenderem aos diferentes tipos de requisitos.

2) Listas Genéricas de Riscos – são aquelas que possuem riscos universais a qualquer sistema.

3) Catálogos de Riscos – são listas de riscos mais retritas pertencendo a um domínio em particular.

A principal dificuldade dessa abordagem está no conhecimento prévio do negócio a fim de realizar a análise dos riscos da forma mais fiel possível. Na

abordagem “de dentro para fora” é utilizada uma lista de riscos potenciais pré-definidos.

2.2.4 Metodologia Baseada em Uso

Besson [11] define uma estratégia que aperfeiçoa os esforços da garantia da qualidade minimizando os riscos. Segundo o autor, qualquer atividade de teste implica numa diminuição de riscos. No entanto, independente da quantidade de teste, sempre haverá um risco. A idéia é investir o mínimo de esforço em teste possível para maximizar a redução dos riscos.

A metodologia proposta por Besson é dividida em cinco passos. Para identificar riscos são procuradas as funcionalidades vitais que podem previnir o usuário de utilizar o sistema se um defeito for encontrado, ou seja, um defeito com alta severidade. Uma forma eficiente de listar essas funcionalidades é a partir de pesquisas com os usuários finais do sistema, experts no dominío do sistema ou através de dados estátisticos de uso de versões anteriores. Uma vez que o risco aumenta com a frequência de uso, as funcionalidades mais usadas terão maior risco.

2.2.5 Metodologia para Código Fonte Orientado a Objetos

Rosenberg [12] fornece uma metodologia para identificação de classes que estejam mais propensas a erros. Segundo o autor foi comprovado anteriormente que o código que é mais complexo tem uma maior incidência de erros ou problemas. Através da aplicação de métricas de complexidade de software orientado a objetos se pode chegar a classes que possuem maior probabilidade de falha. Seis métricas de medição de projetos orientadas a objeto são utilizadas, identificadas e aplicadas pelo *Software Assurance Technology Center* (SATC) do NASA *Goddard Space Flight Center*.

Por mais de três anos, o SATC tem coletado e analisado código orientado a objetos escritos tanto em C++ e em Java. Mais de 20.000 classes foram analisadas, de mais de 15 programas. A análise das classes é fácil de ser aplicada, inclusive pode ser realizada de forma automática. No entanto, os autores não propõem

estratégias de testes para o código, tão pouco, formas de rastreamento das funcionalidades impactadas por classes com alto risco de falhas.

2.2.6 *Brainstorming*

Técnica de geração de idéias em grupo dividida em duas fases: a primeira, onde os participantes apresentam o maior número possível de idéias; e a segunda, onde cada participante defende sua idéia com o objetivo de convencer os demais membros do grupo. Na segunda fase são filtradas as melhores idéias, permanecendo somente aquelas aprovadas pelo grupo.

A meta dessa técnica é obter uma lista abrangente de possíveis riscos. Normalmente é realizada com um conjunto multidisciplinar de especialistas que não fazem parte da equipe de desenvolvimento.

2.2.7 Técnica Delphi

Delphi é uma técnica para a busca de um consenso de opinião de um grupo de especialistas a respeito de eventos futuros. Baseia-se no uso estruturado do conhecimento, da experiência e da criatividade de um painel de especialistas, pressupondo-se que o julgamento coletivo, quando organizado adequadamente, é melhor que a opinião de um só indivíduo. Esta técnica de criação de consenso utiliza respostas escritas ao invés de reunir pessoalmente os membros do grupo, ou ainda método para a sistemática coleta e comparação crítica de julgamentos, de participantes anonimamente isolados, sobre um tópico, através de um conjunto de questionários cuidadosamente desenvolvidos, intercalados com informações sumarizadas e "*feedback*" das opiniões, derivadas das respostas anteriores.

2.2.8 Entrevistas

Entrevista é uma das principais fontes de coleta de dados que podem ser usadas para identificar riscos. As entrevistas são realizadas com os participantes mais experientes do projeto, com partes interessadas e com especialistas. Porém, estas podem ser conduzidas individualmente de forma livre, semi-estruturada ou estruturada.

2.3 Resumo do Capítulo

Este capítulo tratou de introduzir o conceito de testes e riscos de software. Também foram apresentadas algumas técnicas utilizadas na identificação desses riscos. A partir disso, mostramos os questionários e sua importância na identificação dos riscos definindo o motivo pelo qual escolhemos esta técnica.

Capítulo 3

Modelagem e Construção da Ferramenta

Este capítulo fornece uma visão geral das tecnologias empregadas, requisitos do sistema, arquitetura, bem como testes de avaliação utilizados na construção do software. Através da Seção 3.1 serão vistas as tecnologias que foram utilizadas como subsídio na implementação da técnica de identificação escolhida na *RBT Tool*. A Seção 3.2 mostra os requisitos que foram implementados no sistema. Já a Seção 3.3 dá uma visão geral do processo que foi adotado no desenvolvimento da ferramenta. A Seção 3.4 delimita o módulo que foi desenvolvido. A Seção 3.5 discute a arquitetura adotada na implementação do sistema. Finalmente, na Seção 3.6 são apresentados os testes de avaliação das funcionalidades implementadas na ferramenta *RBT Tool*.

3.1 Tecnologias

Esta subseção agrupa as tecnologias que foram selecionadas e utilizadas na contrução da *RBT Tool.*

3.1.1 JAVA

Java [13][14][15] é uma linguagem de programação orientada a objetos, desenvolvida por equipe de pessoas na *Sun Microsystems*. Inicialmente, elaborada para ser a linguagem-base de projetos de software para produtos eletrônicos, Java teve seu grande reconhecimento em 1995.

É uma linguagem muito utilizada atualmente já que possui características muito desejadas desde o seu surgimento. Dentre as suas características podemos destacar [14][15]:

- **Java é independente de arquitetura** – uma das características de Java que a tornou ideal para uso na elaboração de aplicativos distribuídos foi a sua independência de plataforma. Java consegue essa característica devido ao compilador Java não gerar instruções específicas a uma plataforma, mas, sim um programa em um código intermediário, denominado “bytecode”, que pode ser descrito como uma linguagem de máquina destinada a um processador virtual. O código Java compilado pode então ser executado por um interpretador de bytecodes, a JVM - Java *Virtual Machine*, que é um emulador de processador.

- **Java é portável** – a característica de neutralidade da arquitetura Java é o grande motivo pelo qual os programas em Java são portáveis. Outro aspecto da portabilidade envolve a estrutura ou os tipos de dados inerentes da linguagem, como inteiro, string e ponto flutuante.

- **Java é robusto** – quanto mais robusto um aplicativo, mais confiável ele será. Isso é desejável tanto para os desenvolvedores de software quanto aos consumidores. A maioria das linguagens Orientadas a Objetos, como o Java, possui tipos bastante fortes. Isso significa que a maior parte da verificação de tipos de dados é realizada em tempo de compilação, e não em tempo de execução. Isso evita muitos erros e condições aleatórias nos aplicativos.

Java concilia todas estas características oferecendo a tradução dos códigos de bytes para o código de máquina nativo em tempo de execução.

3.1.2 XStream

XStream é uma biblioteca para serialização de objetos para XML (*eXtensible Markup Language*) e para deserialização também. Esta também possui algumas características que foram interessantes no desenvolvimento da *RBTTool*. Abaixo relacionamos algumas destas características:

- **Facilidade de Uso** – É fornecida uma fachada de alto nível que simplifica os casos de uso mais comuns.

- **Não requer mapeamentos** – A maioria dos objetos podem ser serializados sem a necessidade de se especificar mapeamentos.

- **Desempenho** – Velocidade e baixo consumo de memória são partes cruciais do projeto XStream tornando-o adequado para objetos com grafos enormes ou sistemas com uma grande quantidade de troca de mensagens.

- **XML claro** – Nenhuma informação é duplicada e pode ser obtida via reflexão. Isto resulta num XML mais fácil de ler e mais compacto que a serialização nativa de Java.

- **Não requer modificações nos objetos** – Serializa campos internos até mesmo *private* e *final.*Também suporta classes aninhadas e não-públicas.

- **Mensagens de Erro** – Quando uma exceção ocorre devido a um XML mal formatado, um diagnóstico detalhado é fornecido para ajudar a isolar e corrigir o problema.

3.1.3 REM – *Requirement Management*

No ínicio do projeto de implementação da ferramenta foi adotada a ferramenta REM (*Requirements Management*) para a elicitação dos requisitos necessários à ferramenta.

REM é uma ferramenta de gerenciamento de requisitos experimental projetada para suportar a fase de engenharia de requisitos num projeto de desenvolvimento de software de acordo com a metodologia definida na tese de doutorado de Amador Durán [16].

3.1.4 Eclipse

Eclipse é uma comunidade de código-aberto cujos projetos estão focados na construção de uma plataforma de desenvolvimento aberto formada de estruturas

extensíveis, ferramentas e ambientes para construir, distribuir e gerenciar software por todo seu ciclo de vida.

A comunidade Eclipse tem mais de 60 projetos de código aberto. Estes projetos podem ser conceitualmente organizado em sete categorias diferentes:

1. Desenvolvimento de Aplicações Empresariais (*Enterprise Development*)
2. Desenvolvimento de Dispositivos e Sistemas Embarcados (*Embedded and Device Development*)
3. Plataforma de Cliente Ricas (*Rich Client Platform* - RCP)
4. Aplicações de Internet Ricas (*Rich Internet Applications*)
5. Estruturas de Aplicações (*Application Frameworks*)
6. Gerenciamento do Ciclo de Vida da Aplicação (*Application Lifecycle Management* - ALM)
7. Arquitetura Orientada a Seviço (*Service Oriented Architecture* - SOA)

A comunidade Eclipse é suportada por uma grande quantidade de fornecedores de solução em Tecnologia da Informação, universidades, instituições de pesquisas e pessoas que extendem, suportam e complementam a Plataforma Eclipse.

No projeto de desenvolvimento da ferramenta *RBT Tool* o IDE (*Integrated Development Enviroment*) Eclipse foi utilizado com ambiente de desenvolvimento.

3.1.5 RCP – *Rich Client Plataform*

Aplicações RCP são baseadas na arquitetura de *plug-in* comum da plataforma de desenvolvimento Eclipse. O próprio Eclipse provê um ambiente para desenvolvimento de *plug-ins.* Este é conhecido como PDE (*Eclipse's Plug-in Development Environment*).

Essencialmente, RCP fornece um ambiente Eclipse genérico que os desenvolvedores podem herdar para construir suas próprias aplicações. Uma aplicação consiste de no mínimo um *plug-in* padrão e usa os mesmos elementos de interface com o usuário os quais o IDE Eclipse utiliza. A Figura 3.1 apresenta os elementos básicos da interface com usuário do Eclipse.

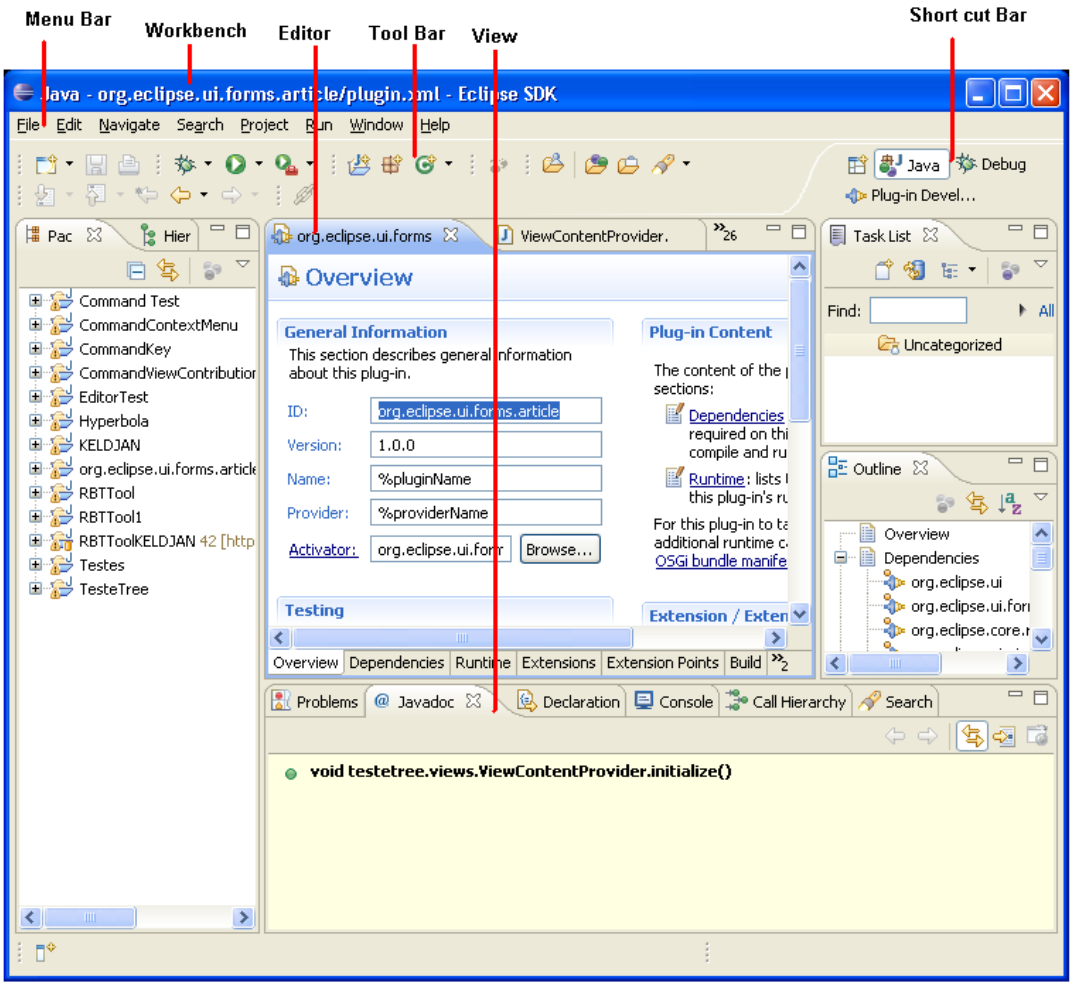

Figura 3.1. Elementos básicos da interface com usuário do Eclipse.

Os elementos básicos do ambiente incluem:

1. ***Workbench* –** o container que abarca todas a janelas.

2. ***Perspective* –** um container visual para todos as *views* e *editors* abertos.

3. ***View* –** um container visual para mostrar os recursos de um tipo particular. Tipicamente, uma *view* contém uma grade de dados ou uma estrutura em árvore.

4. ***Short Cut Bar* –** uma série de ícones que permitem ao usuário acessar rapidamente diferentes perspectivas.

5. ***Menu Bar* –** uma série de *actions* sensíveis ao conteúdo que dá a habilidade ao usuário de executar algumas funções pré-definidas.

6. ***Tool Bar* –** uma série de *actions* sensíveis ao contexto que permitem ao usuário executar algumas funções pré-definidas.

7. ***Editor* –** *Editors* são as ferramentas primárias que os usuários empregam para mostrar e manipular dados.

A Figura 3.2 apresenta os elementos utilizados na ferramenta *RBT Tool*. A *view* agrupa os projetos que foram criados numa estrutura em árvore. O editor é utilizado para agrupar informações que são gerenciadas durante o projeto. A barra de menus agrupam as funcionalidades que são utilizadas durante todo o decorrer da identificação de riscos no projeto.

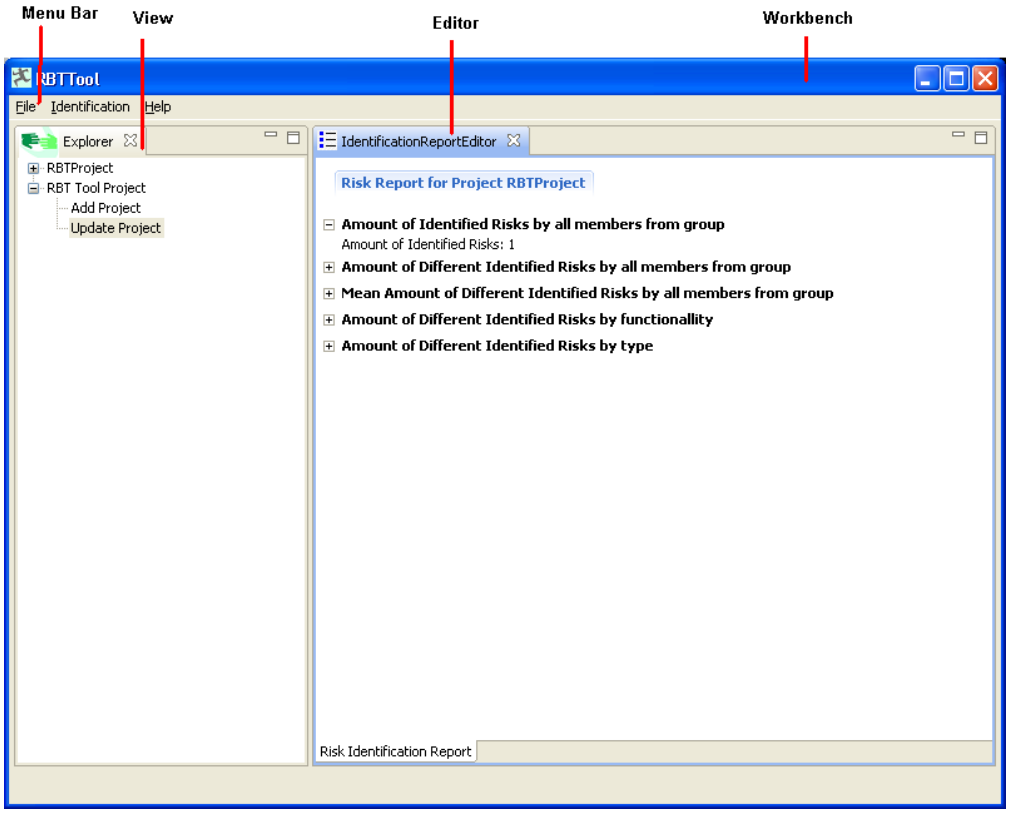

Figura 3.2. Elementos básicos da interface com usuário do *RBT Tool*.

3.1.6 SVN – *Subversion*

SVN é um sistema de controle de versão usado para manter versões atuais e históricas de arquivos como código-fonte, páginas web e documentação. Este sistema é bastante conhecido na comunidade de código aberto, além de ser utilizado em diversos projetos de código fontes abertos como Apache Software Foundation, GNOME, KDE, Python, entre outros.

3.1.7 UML – *Unified Modeling Language*

A *Unified Modeling Language* (UML) [17] é uma linguagem gráfica universal para visualização, especificação, construção e documentação de artefatos de um sistema intensivo de software. A construção de um modelo visual facilita a comunicação e faz com que os membros de um grupo tenham a mesma idéia do sistema.

UML possui algumas vantagens como a inclusão de artefatos como deliverables, que são documentos como especificação de requisitos, especificações funcionais, planos de testes. E também, materiais que são importantes para controlar, medir, e refletir sobre um sistema durante o seu desenvolvimento e implantação.

Na versão 2.0 da UML existem 13 tipos de diagramas que são responsáveis por uma parcial representação do sistema. Eles representam graficamente um conjunto de elementos com seus respectivos relacionamentos usados para visualizar o sistema de perspectivas diferentes. Existem dois tipos de diagramas: os diagramas estáticos e os diagramas dinâmicos. Os estáticos são utilizados para visualizar as partes estáticas do sistema e os dinâmicos para modelar o seu comportamento. No projeto em questão foi utilizado apenas diagramas estáticos UML, o diagrama de classes e diagramas de caso de uso [18].

3.2 Requisitos do Sistema

Para construção da *RBT Tool*, foi adotada a elicitação de requisitos como ponto inicial. Nesta fase identificou-se os papéis e quais as funcionalidades necessárias nesta versão da identificação de riscos.

Existem vários atores para aplicação da abordagem RBT num projeto. A Figura 3.3 mostra a visão geral entre os casos de uso principais que deverão estar implementados na *RBT Tool* para versão completa.

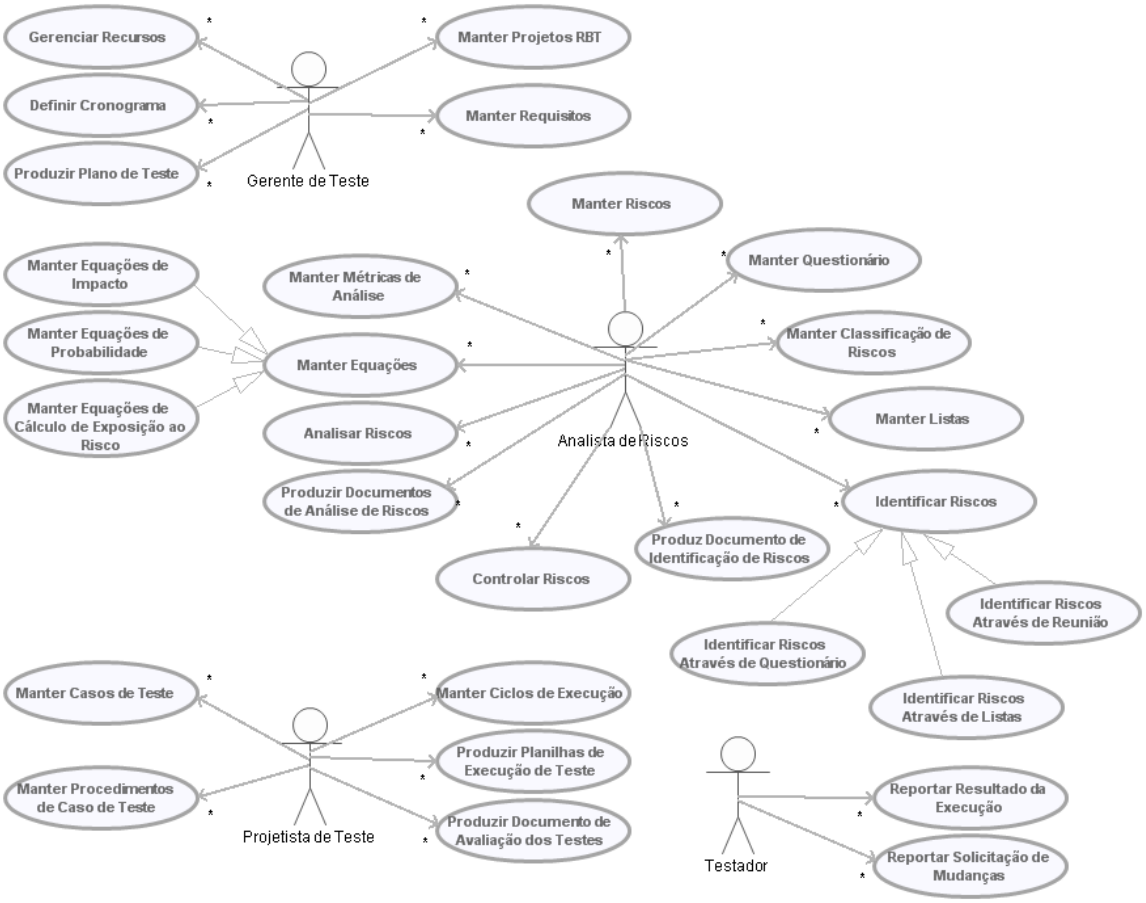

Figura 3.3. Visão Geral de Todos os casos de uso para a RBT Tool.

A seguir, são brevemente descritos os casos de uso implementados nesta primeira versão da *RBT Tool*. Os detalhes e descrição dos demais casos de uso estão disponíveis na página da ferramenta em [19].

- **Manter Projetos RBT:** Torna possível a inclusão, exclusão e alteração dos projetos de Teste de Software baseado em Riscos pelo Gerente de Teste. Dentre outras informações, é possível informar quais requisitos que fazem parte do projeto, a versão do software que será testado, os recursos necessários e datas de início e fim da atividade de teste.
- **Manter Requisitos:** Permite cadastrar os requisitos que estão no escopo dos testes pelo Gerente de Teste.
- **Manter Riscos:** O Analista de Riscos inclui na ferramenta riscos que podem ser identificados no produto de software.
- **Manter Questionário:** O Analista de Riscos cadastra questionários que podem ser utilizados na etapa de identificação. Ele cadastra uma ou mais

perguntas no questionário e associa estas perguntas a um risco cadastrado na ferramenta.

- **Identificar Riscos:** O Analista de Riscos cadastra o questionário dentro de um projeto utilizando a ferramenta, exporta para um documento e envia para os participantes da identificação. Após respondidos, os questionários são importados pela ferramenta, os dados são processados e sumarizados, resultando numa lista de riscos identificados para cada requisito do software em teste.

A partir dessa visão mais geral, foram definidos os casos de uso básicos para implementação da identificação de riscos, conforme a Figura 3.3.

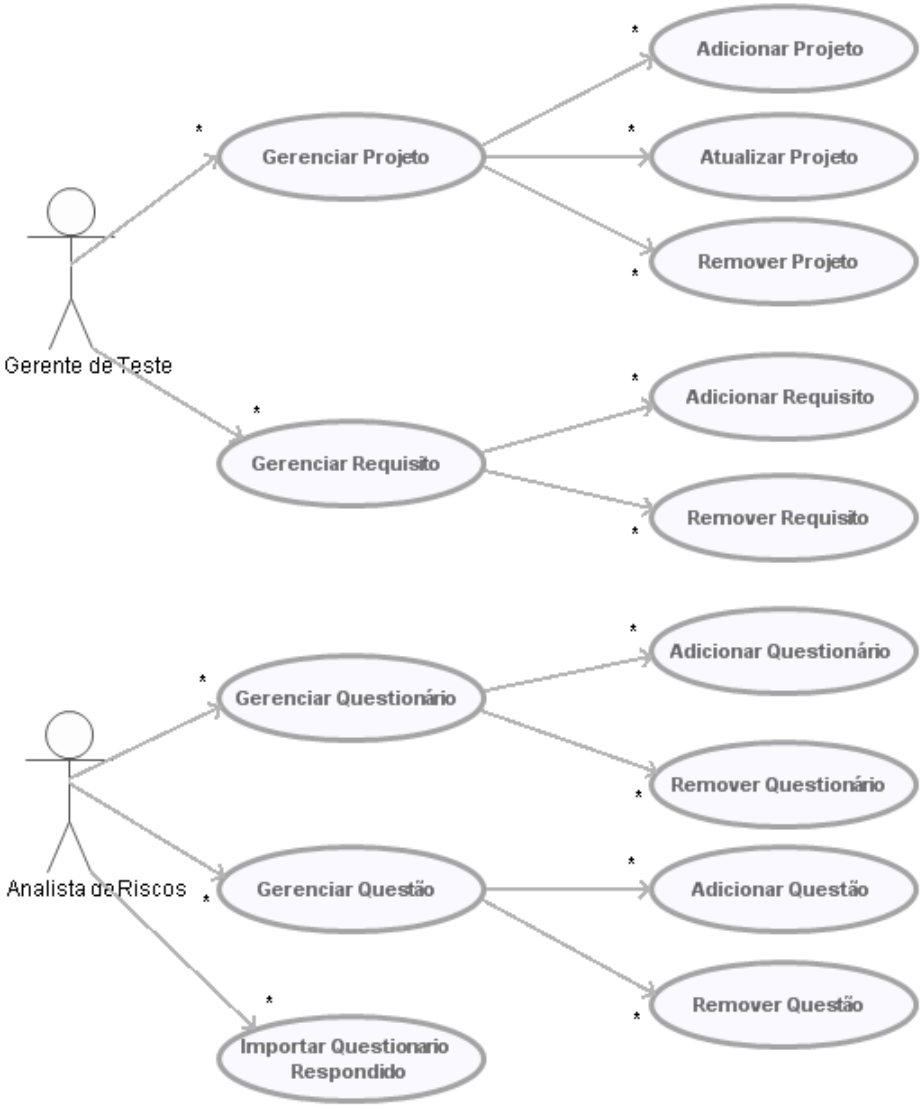

Figura 3.4. Casos de Uso implementados para Identificação de Riscos.

A implementação dos casos de uso presentes na Figura 3.4 abrange os requisitos inerentes a identificação de riscos objetivada na *RBT Tool*. Neste diagrama é mostrado o Gerente de Teste como responsável pelo gerenciamento dos projetos e dos requisitos adicionados à *RBT Tool*. Já o Analista de Riscos é responsável pelo gerenciamento dos questionários, questões e pela importação dos questionários respondidos pelos usuários envolvidos na identificação dos riscos.

Ainda no diagrama da Figura 3.4 é mostrado que o gerenciamento ocorre através da inclusão e remoção em geral dos elementos que compõe a identificação de riscos.

3.3 Processo de Desenvolvimento

O processo utilizado no desenvolvimento da *RBT Tool* foi iterativo incremental dirigida a caso de uso, os quais foram descritos na seção 3.2. Devido a falta de familiaridade com os elementos que compõe a interface gráfica, este processo teve seu cronograma ajustado diversas vezes

Na tabela 3.1 é mostrada a relação entre as iterações e caso de uso que foram implementados na ferramenta. Ainda nesta tabela, é apresentado o tempo gasto em cada iteração utilizado na implementação da *RBT Tool*.

Tabela 3.1. Iterações de desenvolvimento.

Iteração	Caso de Uso	Tempo Utilizado
1ª Iteração	Adicionar Projeto	2 semanas
	Atualizar Projeto	
	Remover Projeto	
2ª Iteração	Adicionar Requisito	2 semanas
	Remover Requisito	
	Adicionar Questionário	
	Remover Questionário	
	Adicionar Quesão	
	Remover Questão	
3ª Iteração	Exportar Respostas	2 semanas
	Importar Respostas	
4ª Iteração	Informar Riscos Identificados	1 semana

Através da Tabela 3.1, percebemos que a quantidade de casos de uso da 1ª iteração foi menor se comparado à 2ª iteração uma vez que, neste período, elementos como arquitetura e tecnologias estavam sendo definidos. A 3ª e 4ª iteração possui uma quantidade pequena de casos de uso implementados por que neste outro período coincidiu a realização de outros projetos e provas provenientes de disciplinas da graduação.

3.4 Módulo do Sistema

O sistema é composto basicamente por um módulo, o qual é o módulo cliente. Neste módulo é possível realizar atividades concernentes a identificação de riscos junto a todos os *stakeholders*.

Este módulo é utilizado tanto pelos analistras de risco, quanto pelo gerente de testes visto que é necessária uma equipe responsável pela aplicação do processo de testes baseado em riscos como um todo.

3.5 Arquitetura do Sistema

A arquitetura do sistema foi baseada no padrão de projeto arquitetural Camadas (*Layers*) [20]. Camadas é um padrão de arquitetura de software que tem como objetivo separar dados ou lógica de negócios (Persistência) da interface do usuário (Apresentação) e do fluxo da aplicação (Negócio), de modo que seja possível obter uma maior :

- **Modularidade** – dividir a aplicação em módulos deixando-os o mais independentes possível,
- **Manutenibilidade** – reduzir o custo de manutenção no sistema,
- **Extensibilidade** – permitir a inclusão de novas funcionalidades sem grandes impactos na estrutura já existente, entre outras características.

Uma das vantagens em se usar esse padrão é a fácil manutenção e atualização da aplicação que o implementa, o que será de grande ajuda na implementação dos trabalhos futuros, que serão integrados a essa aplicação, além de tornar a aplicação escalável.

A arquitetura lógica do sistema está representada no diagrama de pacotes na Figura 3.5 [21]. O sistema possui três pacotes, o pacote Apresentação que contém todas as classes necessárias à interface com o usuário, o pacote Negócio responsável por implementar a lógica de negócio da aplicação que invocam alterações nos dados e o pacote Persistência que contém as classes que são responsáveis pela persistência das informações fornecidas, no caso da RBT Tool, em arquivos XML.

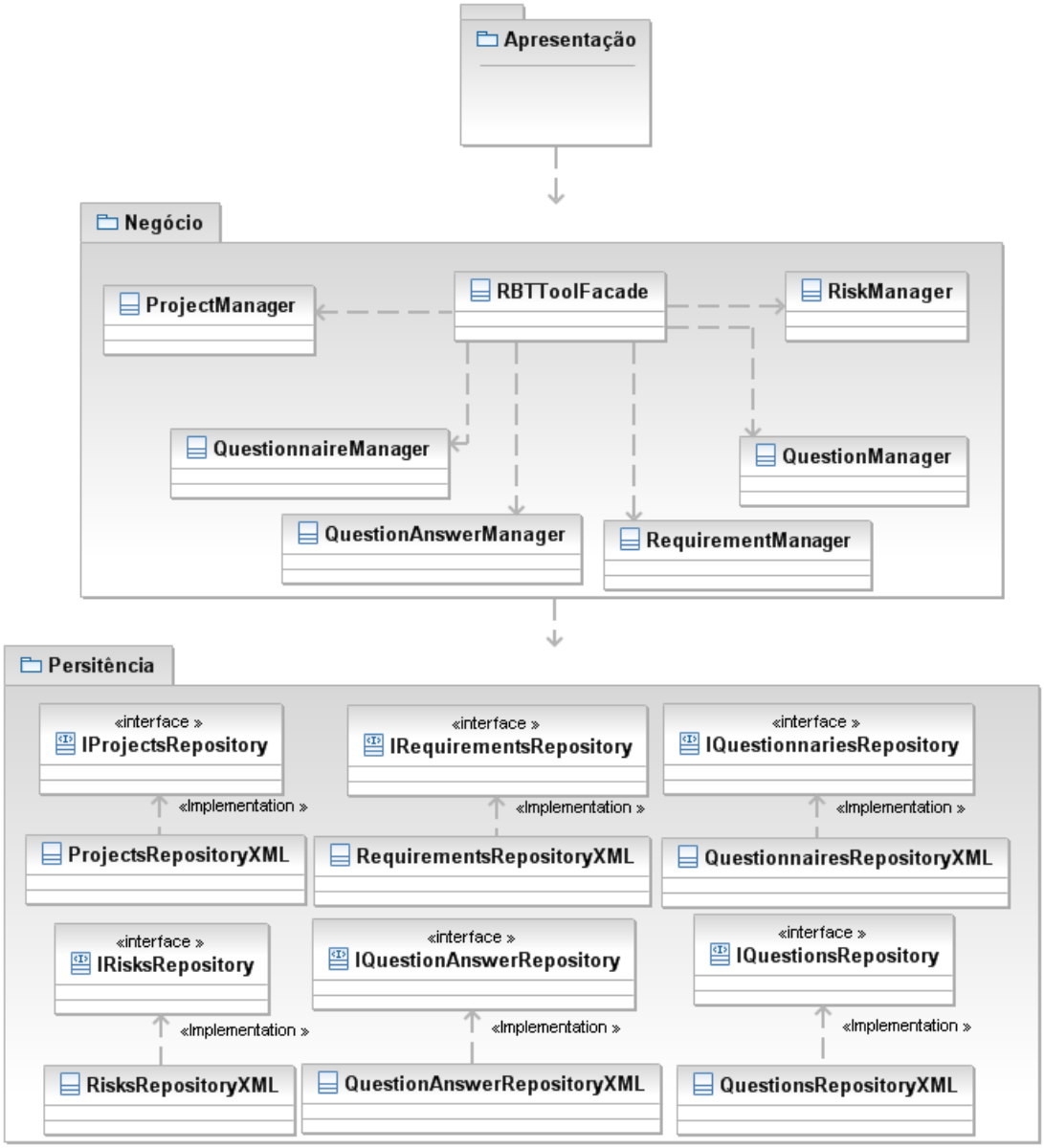

Figura 3.5. Diagrama de Pacotes da Visão Lógica

A implementação dessa arquitetura foi bastante útil, pois facilitará a inclusão dos módulos subseqüentes. Ainda, vale ressaltar que o pacote Apresentação possui uma arquitetura particular baseada na arquitetura da plataforma Eclipse a qual é facilmente extensível. A Figura 3.6 ilustra os componentes principais da arquitetura Eclipse junto a *RBT Tool*.

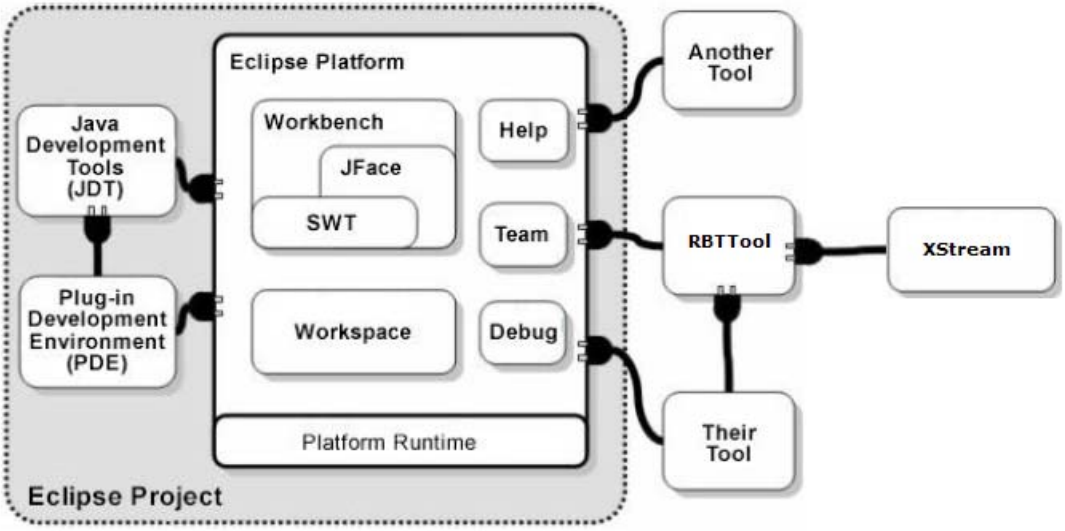

Figura 3.6. Principais componentes da arquitetura Eclipse

Excetuando-se os arquivos que formam o Eclipse *Plataform Runtime*, todas as funcionalidades do Eclipse são implementados através do uso de *plug-ins*. *Plug-in* é o bloco básico de construção que os desenvolvedores usam para adicionar novas capacidades e funcionalidades ao ambiente. O *runtime* do Eclipse é responsável por gerenciar o ciclo de vida do *plug-in* dentro do *workbench*. Todos os *plug-ins* para um ambiente em particular estão localizados numa pasta pasta do *plug-in* que contém a estrutura de uma aplicação RCP. Quando em execução, o *runtime* do Eclipse encontra todos os *plug-ins* disponíveis e usa essas informações para criar um registro de *plug-in* global, no nosso caso, a *RBTTool*.

3.6 Teste de Avaliação

À medida que os casos de uso eram implementados, o sistema era testado. Assim, os testes ficaram concentrados nos aspectos funcionais do sistema.

A fim de apresentar as funcionalidades de maneira menos formal, será mostrado o fluxo que poderia ser seguido na utilização da técnica de questionário para a identificação dos riscos através de *screenshots*. Ainda, relacionamos as funcionalidades com os requisitos propostos inicialmente para a *RBT Tool* e estão disponíveis em [19].

3.6.1 Criando um novo projeto

Na ferramenta, o ínicio do processo é marcado a partir da criação de um novo projeto. Esta funcionalidade relaciona-se diretamente com o caso de uso “Adicionar Projeto” da Figura 3.4, apresentada anteriormente. Esta tarefa seria de responsabilidade do Gerente de Testes. A Figura 3.7 demonstra o diálogo de criação de um novo projeto na ferramenta.

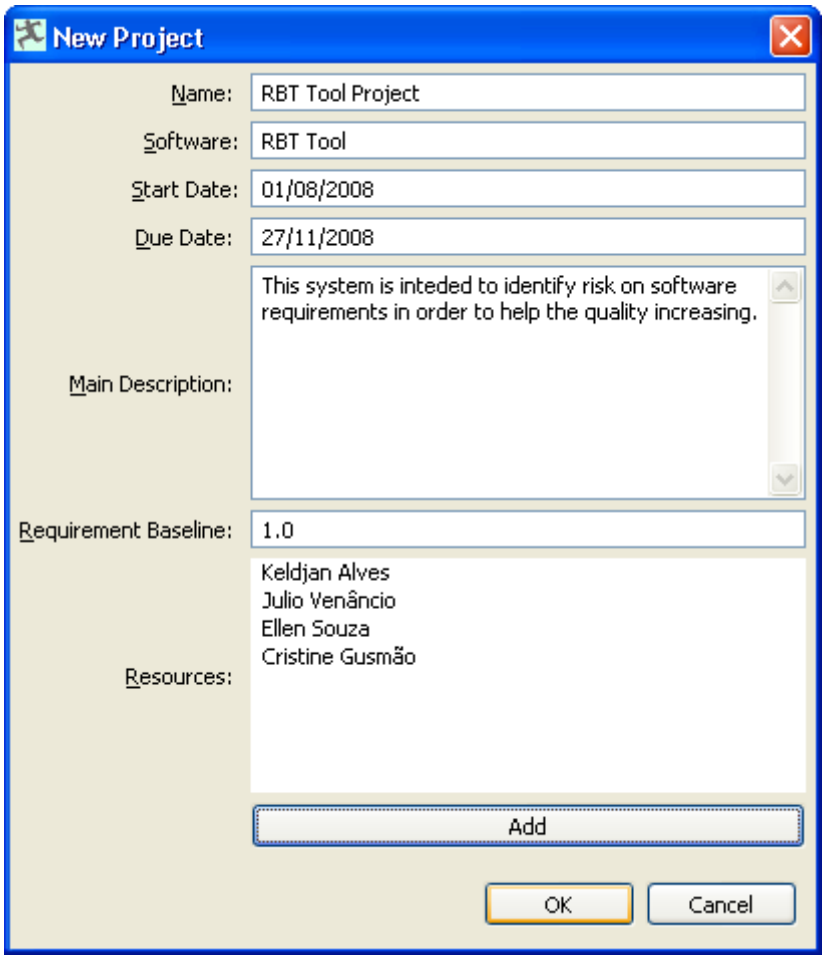

Figura 3.7. Diálogo de criação de um novo projeto

Nesta atividade, é mandatório fornecer um nome para o projeto de forma que ele seja identificado exclusivamente. Outros campos como nome do software, datas de ínicio e de entrega, são solicitados com o intuito de manter informações mais acuradas sobre o projeto.

3.6.2 Adicionando um novo requisito a um projeto

A fim de indentificar quais partes do software são atingidas com algum risco, necessita-se cadastrar quais requisitos pertencem a um determinado software. Esta funcionalidade relaciona-se diretamente com o caso de uso “Adicionar Requisito” da Figura 3.4, apresentada anteriormente. A Figura 3.8 mostra o diálogo utilizado no cadastro de novos requisitos que serão relacionados a um determinado projeto.

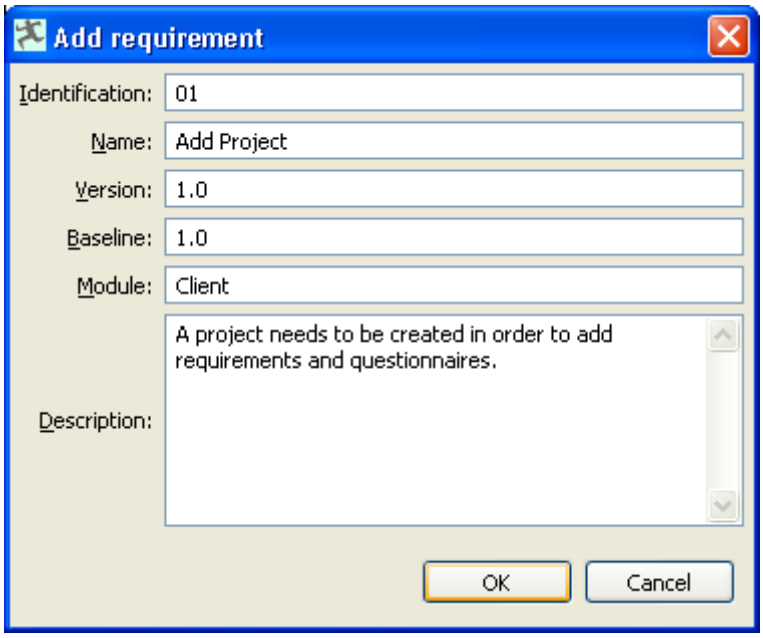

Figura 3.8. Diálogo de criação de um novo requisito

Nesta atividade o campo de identificação é mandatório para identificar o requisito exclusivamente dentro do mesmo projeto. Os demais campos são opcionais, porém importantes na elaboração dos casos de testes que serão criados com base na identificação de riscos realizada.

Após a criação de um determinado requisito, este ficará agrupado num mesmo editor do projeto ao qual ele se integra. A Figura 3.9 demonstra o agrupamento de requisitos para o projeto fictício RBT *Tool Project*.

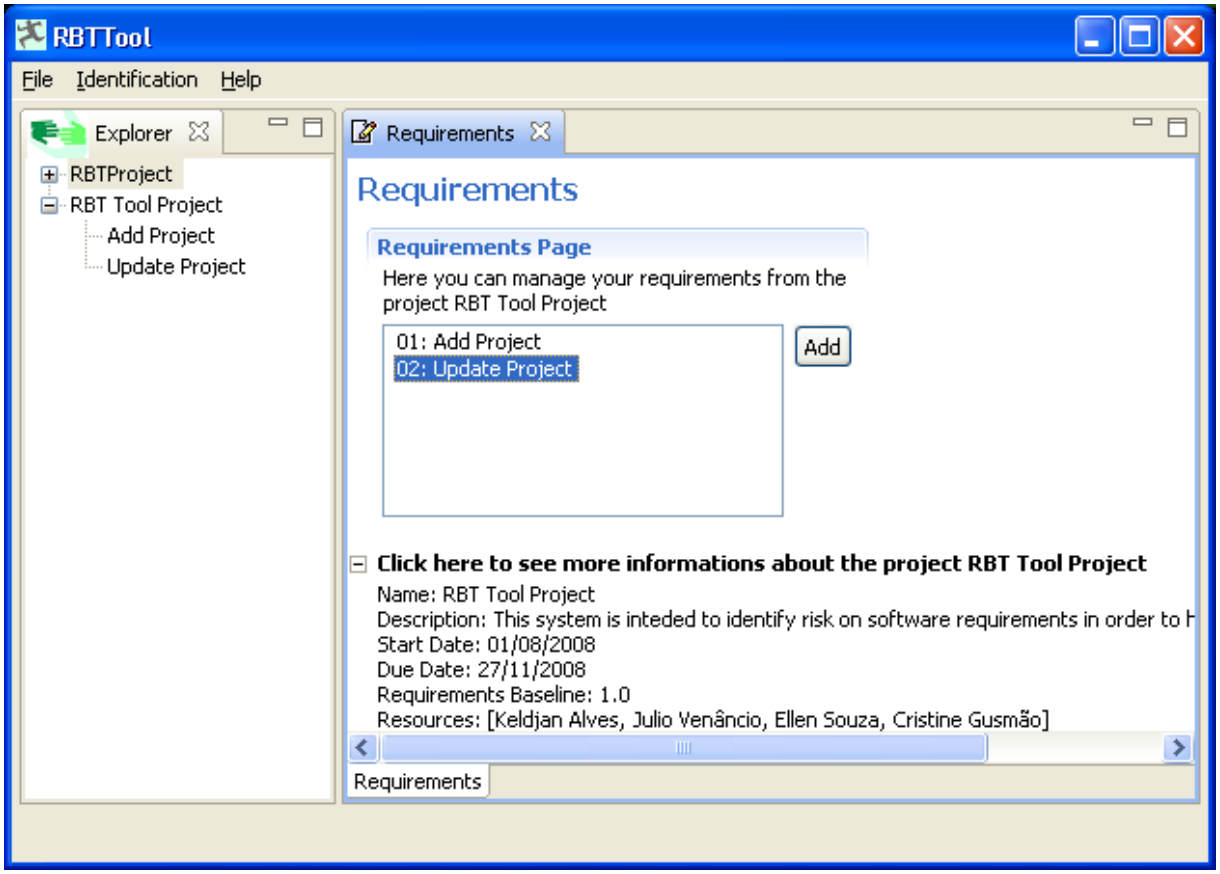

Figura 3.9. Editor de requisitos agrupados de um projeto.

Neste Editor do projeto percebe-se os requisitos dispostos numa mesma área central e um botão de adicionar para facilitar a adição de novos requisitos. Mais abaixo, foi criado uma área retrátil onde pode-se verificar as informações que foram inseridas no período de criação do projeto.

3.6.3 Adicionando um novo questionário

Esta outra funcionalidade foi implementada na ferramenta com a finalidade de criar-se questionários que sejam utilizados num determinado projeto de desenvolvimento de software. A Figura 3.10 apresenta os campos necessários a adição de um novo questionário na ferramenta.

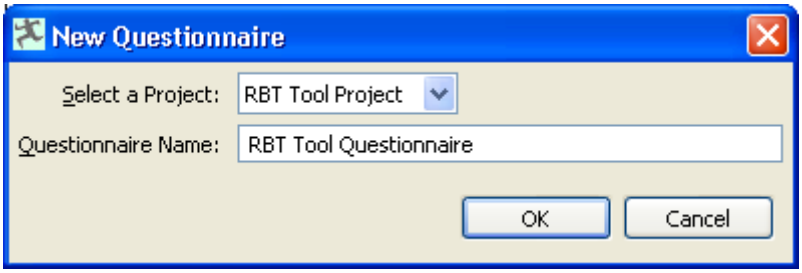

Figura 3.10. Diálogo de Criação de Novo Questionário.

Ao se adicionar um novo questionário, deve-se escolher a qual projeto o questionário irá pertencer e fornecer um nome para o questionário. Nesta funcionalidade, ambos os campos são mandatórios. Esta funcionalidade relaciona-se diretamente com o caso de uso “Adicionar Questionário” da Figura 3.4, apresentada anteriormente.

3.6.4 Adicionando uma nova pergunta

Esta funcionalidade agrupa todas as informações básicas para a identificação do risco na ferramenta e relaciona-se diretamente com o caso de uso “Adicionar Questão” da Figura 3.4, apresentada anteriormente. Dentre suas funções, está inclusa a ligação entre o risco e uma determinada resposta. A Figura 3.11 mostra o diálogo necessário à adição de uma nova pergunta no questionário.

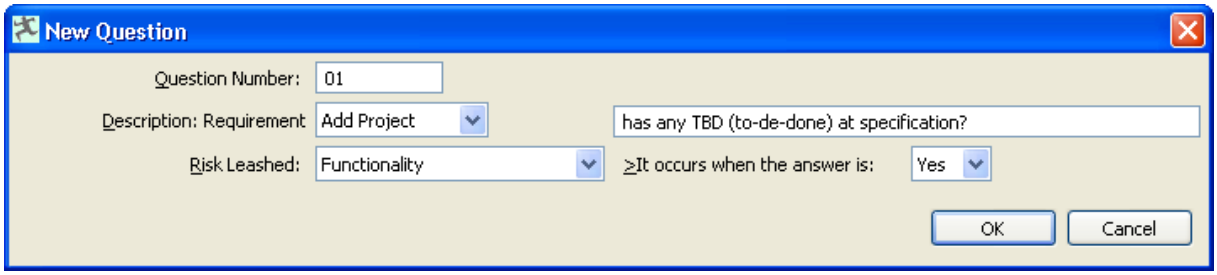

Figura 3.11. Diálogo para inserção de nova pergunta

O diálogo mostrado na Figura 3.11 agrupa vários campos. O número da questão representa o campo de identificação. Através deste campo, a pergunta torna-se distinta das demais incluídas no mesmo questionário. O campo de descrição agrupa a escolha do requisito junto ao corpo da pergunta. A fim de manter-se a coerência é necessário elaborar questões factíveis e inteligíveis ao ser humano. A caixa de seleção, para o risco que será atrelado, possui todos os riscos

pré-cadastrados conforme a lista de taxonomia proposta pelo SEI para os riscos de Engenharia do Produto. O campo remanescente é o responsável pela junção de quando o risco irá ocorrer. No caso, ele é uma caixa de seleção composta apenas de duas escolhas: Sim ou Não.

A fim de agrupar as questões foi denifido um editor com todas as questões referentes a um mesmo questionário. A Figura 3.12 apresenta as questões para o questionário *RBT Tool Questionnaire* utilizado como exemplo. No título destacado em azul percebe-se o nome do questionário ao qual as questões estão atreladas. Na parte central estão dispostas as perguntas que foram adicionadas. Na parte inferior do editor também está a parte retrátil que agrupa as informações referente ao projeto a que o questionário pertence.

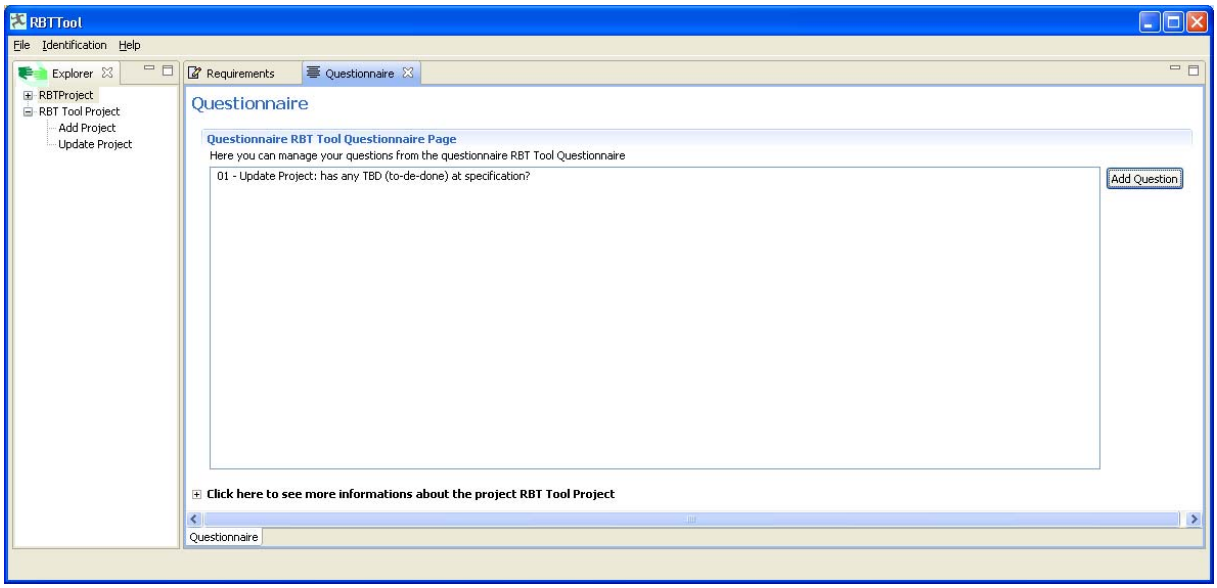

Figura 3.12. Editor com questões agrupadas num mesmo questionário.

3.6.5 Respondendo um questionário

Na técnica de identificação de riscos através de questionário, o usuário da ferramenta tem de responder às questões de forma a subsidiar a elaboração do relatório final.

Para responder às questões, é necessário montar um questionário criado previamente. As respostas consistem apenas de “SIM” ou “NÃO” em relação a uma

determinada pergunta. Ao final das escolhas das respostas é necessário salvá-las atrelando um nome para o usuário que respondeu as questões.

Esta funcionalidade relaciona-se diretamente com o caso de uso “Identificar Riscos Através de Questionários” da Figura 3.3, apresentada anteriormente. A Figura 3.13 mostra o editor no qual são respondidas as questões.

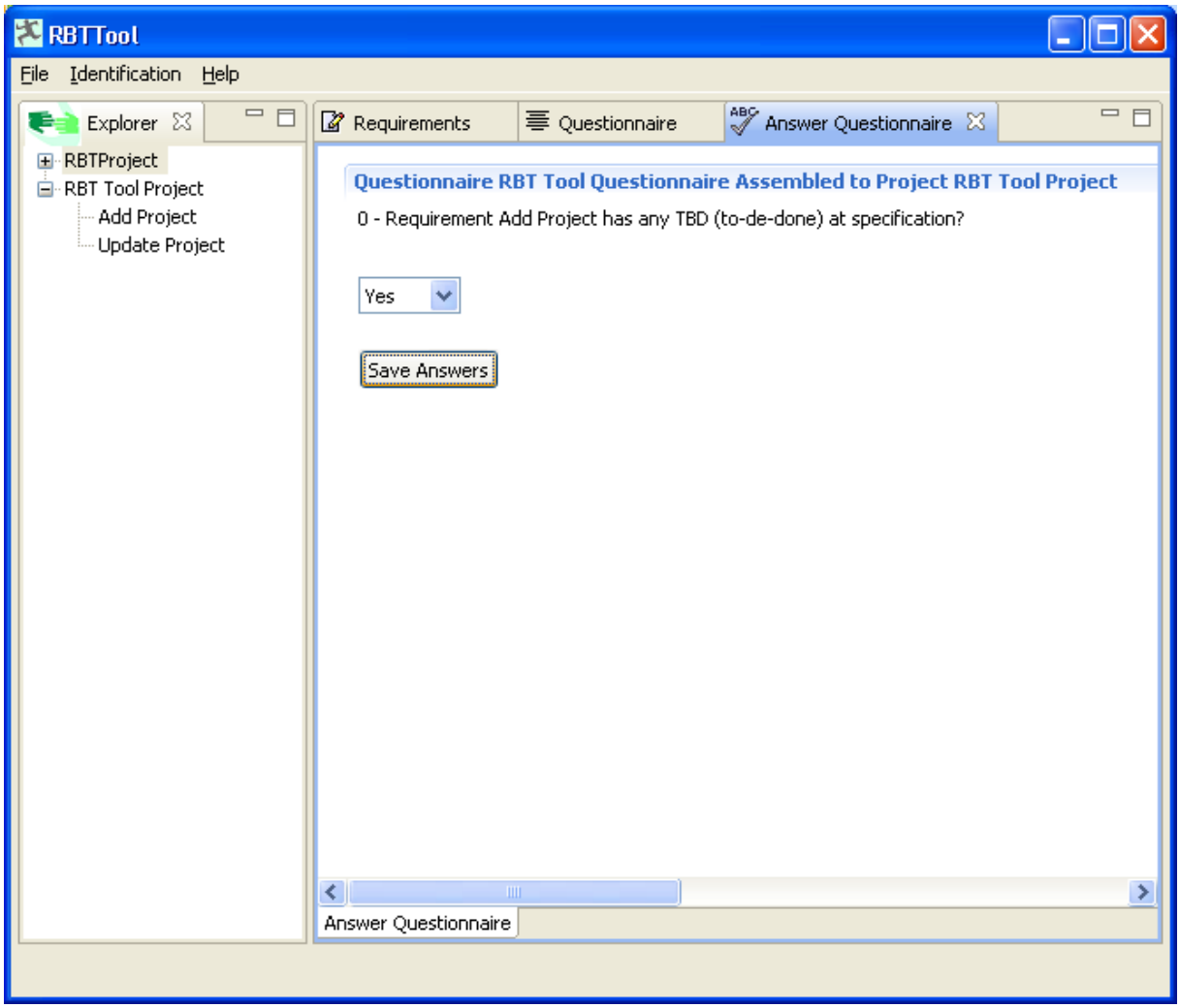

Figura 3.13. Questionário montado para salvar as respostas.

A fim de identificar-se os riscos de fato, é necessário exportar as respostas referentes ao questionário e enviar ao gerente de testes o qual é responsável pela manutenção do projeto. O arquivo exportado é montado no formato XML e identifica unicamente as respostas de um determinado usuário.

3.6.6 Montando o relatório com as informações sobre os riscos identificados

Para iniciar a identificação de riscos, faz-se necessário importar um XML que contenham as respostas de um usuário. Dessa forma, a ferramenta poderá compilar um relatório sumarizando as principais informações relevantes ao projeto que está sendo identificado.

A Figura 3.14 mostra o editor que contém as informações referentes aos riscos identificados no projeto. As informações agrupadas neste editor são as seguintes:

- **Quantidade de riscos identificados por todos os membros do grupo** – Esta seção agrupa a quantidade total de riscos identificados durante todo o processo de identificação através do questionário aplicado.

- **Quantidade de riscos diferentes identificados por todos os membros do grupo** – Similar à seção anterior porém, nesta seção apenas os riscos diferentes são identificados.

- **Quantidade média de riscos diferentes identificados por toda a equipe** – Define um valor médio como métrica de identificação de riscos diferentes por pessoa da equipe.

- **Quantidade de riscos diferentes agrupados por funcionalidade –** Esta outra seção agrupa a quantidade de riscos identificados relacionada com uma funcionalidade específica.

- **Quantidade de riscos identificados por tipo –** Esta seção agrupa a quantidade de riscos identificados relacionados a um determinado tipo.

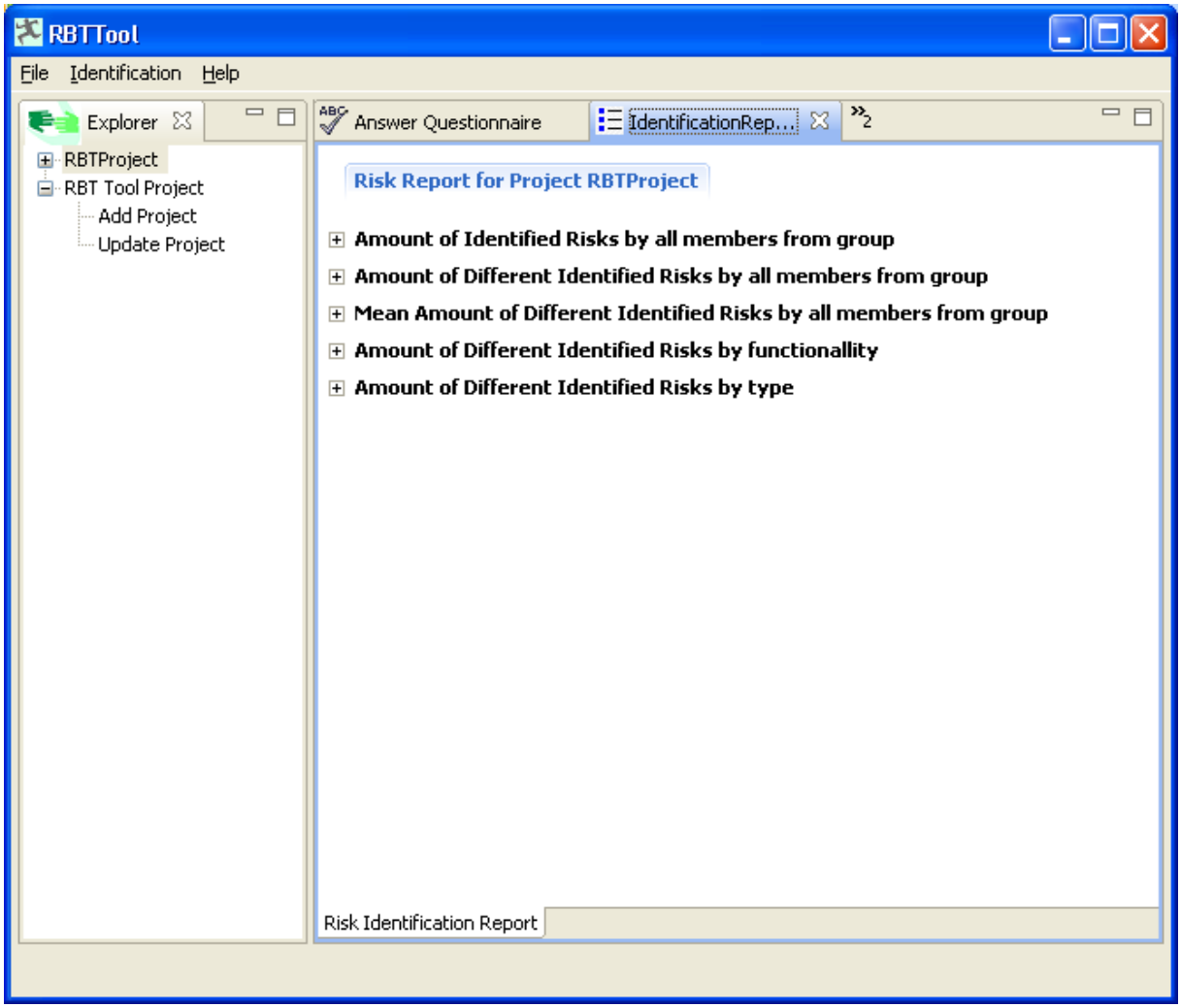

Figura 3.14. Relatório de Riscos Identificados para um determinado projeto.

Ressalte-se, ainda, que esta funcionalidade relaciona-se diretamente com o caso de uso “Produzir Documento de Identificação de Riscos” executado pelo Analista de Risco no diagrama da Figura 3.3 a qual foi apresentada previamente.

3.7 Resumo do Capítulo

Este capítulo tratou de mostrar as tecnologias nas quais foram baseados todo o desenvolvimento da *RBT Tool*. Também foram apresentados os casos de uso, processo de desenvolvimento adotado, a arquitetura, o módulo do sistema e, por fim, os testes de avaliação que foram executados nas funcionalidades implementadas na *RBT Tool*.

Capítulo 4
Considerações Finais e Trabalhos Futuros

Como conclusão deste trabalho destaca-se a relevância de possuirmos uma ferramenta que auxilia na identificação identifica dos riscos que podem vir a atingir o projeto mesmo sem o conhecimento prévio de riscos.

Com a *RBT Tool*, os engenheiros de testes tem um auxílio na utilização da abordagem RBT através da identificação e, posterior análise de riscos para o teste de software. Assim, a abordagem poderá ter uma disseminação maior, a qual já se mostrou bastante eficiente em alguns estudos de caso [22][23] e, ainda, será possível obter uma melhora na qualidade do software criado.

Está em andamento a implementação das funcionalidades referentes à análise dos riscos identificados a fim de tornar a *RBT Tool* mais completa no sentido de apoiar as atividades que necessitam mitigar os riscos em ambientes de desenvolvimento de software.

Atualmente a *RBT Tool* possui apenas um módulo cliente para ser utilizado no processo de identificação de riscos restrigindo, assim, a forma de acesso apenas através do envio do XML que contém o projeto utilizado.

4.1 Problemas Enfrentados

O desenvolvimento deste trabalho não se deu de forma simples. O maior problema enfrentado durante a implementação foi o desconhecimento total dos elementos que compunham a interface gráfica – tecnologia a ser adotada, a RCP.

Outro fator complicador foi a restrição de tempo para desenvolvimento deste trabalho junto às outras disciplinas cursadas durante o período letivo na universidade.

4.2 Trabalhos relacionados

Encontramos na literatura uma tentativa de construção de uma ferramenta para suportar a abordagem RBT. Jørgensen [24] fornece um levantamento dos requisitos necessários para a construção de uma ferramenta para suportar a análise de riscos e execução dos casos de testes. Este trabalho propõe a construção de uma ferramenta que ajude na aplicação da abordagem RBT auxiliando, principalmente, os engenheiros de testes nas fases de identificação, análise e controle de riscos num software.

A *RBT Tool* possui características interessantes no sentido de que implementa de fato os requisitos propostos na identificação de riscos através da aplicação de questionários, no entanto, é importante salientar que apenas esta técnica de identificação de riscos deixa a desejar. É importante que a *RBT Tool* permita, no futuro, seleção de técnicas e métodos para apoiar a atividade de identificação de riscos.

Outro ponto a considerar é que a *RBT Tool* foi construída com base em RCP, na plataforma Eclipse, o que a torna multiplataforma e herda todas a características provenientes deste tipo de desenvolvimento sobretudo de código aberto.

4.3 Trabalhos futuros

Como trabalhos futuros foram identificados alguns pontos de interesse para evolução e melhoria dos serviços disponibilizados através do uso da *RBT Tool*:

1. Necessidade de melhorias na usabilidade e restrições de integridade mais seguras em todo o processo, principalmente através destas melhoras na interface gráfica RCP.

2. Necessidade de implementar uma arquitetura cliente-servidor a fim de realizar troca de informações uma maneira mais segura.

3. Implementar técnicas de inteligência artificial para possibilitar uma ajuda nos esforços de identificação de riscos de uma forma mais rápida.

4. Rever a disponibilização de outras técnicas e métodos de apoio à identificação de riscos; e

5. Criptografar os dados dos usuários que utilizam a *RBT Tool* a fim de manter a privacidade dos arquivos trocados entre máquinas.

Por fim, vale salientar a necessidade de implementação de todos os casos de uso remanescentes relacionados às outras partes concernentes a abordagem RBT necessárias na *RBT Tool*.

Bibliografia

[1] KANER, C.; FALK, J., NGUYEN, H. Q. ***Testing computer software***, 2ª edição". John Wiley & Sons, 1999.

[2] BACH James; **Troubleshooting risk-based testing**. In: Software Testing & Quality Engineering Magazine, 2003.

[3] BACH, J. ***James Bach on Risk-Based Testing: How to conduct heuristic risk analysis***. In: Software Testing & Quality Engineering Magazine, p.23-28, 1999.

[4] AGUIAR, Maurício. **Gerenciando objetivamente seu projeto**. Developers' Magazine, v. 6, n. 66, p. 46-47, fev. 2002.

[5] DELAMARO, M.; MALDONADO, J.; JINO M. ***Introdução ao Teste de Software***. 1a edição, Elsevier, 2007.

[6] SOUZA, E.; GUSMÃO C.; ROCHA, H. **RBTProcess - Proposta de Modelo de Processo de Teste de Software baseado em Riscos**. In: III EBTS – Encontro Brasileiro de Teste de Software, 2008.

[7] PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995. 1056p.

[8] COPSTEIN, Bernardo; OLIVEIRA, Flávio. **Teste Unitário – Introdução**. FACIN – PUCRS, Rio Grande do Sul, 2003. Disponível em: <http://www.inf.pucrs.br/~flavio/tsi/TesteUnitario_Introducao.ppt>. Acesso em: 16 de Novembro 2008.

[9] MARTINS, Claudia Garrido. **Aplicação das Técnicas de Identificação de Risco em Projetos de E & P.** Monografia (Pós-Graduação - MBA em Engenharia Econômica e Financeira), Universidade Federal Fluminense, Rio de Janeiro, Niterói, 2006.

[10] Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos (Guia PMBOK) Terceira edição, 2004, Project Management Institute.

[11] BESSON S. ***A Strategy for Risk-Based Testing***. In StickyMinds.com, agosto de 2004.

[12] ROSENBERG, L. H.; STAPKO, R.; GALLO, A. ***Risk-based object oriented testing*****.** In Twenty-Fourth Annual Software Engineering Workshop, NASA SEW24, 1999.

[13] Sun Developer Network. Disponível em <http://java.sun.com/>. Acesso em: 22 de novembro de 2008.

[14] DEITEL, Harvey M. e DEITEL, Paul J. Java: Como Programar. Quarta edição. Editora Bookman, 2001.

[15] CORNELL, Gary e HORSTMAN, Cay S. Core Java 2: Fundamentos. Primeira edição. Volume 1. Editora Makron Books, 2000.

[16] DURAN, A. **A Methodological Framework for Requirements Engineering of Information Systems,** 2000.

[17] RITTGEN, Peter. **Enterprise Modeling and Computing with UML**. IGI Global, 2006, 314p.

[18] BLAHA, Michael; RUMBAUGH, James. **Modelagem e Projetos Baseados em Objetos com UML**. Editora Campus, 2006, 510p.

[19] Documento de Requisitos da RBT Tool. Disponível em < http://pma.dsc.upe.br/rbttool/requirements/RBTToolReqsV2.html>. Acesso em: 23 de novembro de 2008.

[20] Architecture: Layers and Packages. Disponível em: <http://www.cs.purdue.edu/homes/apm/courses/BITSC461-fall03/uml-slides/design-4-apm.ppt> Acesso em: 23 de novembro de 2008.

[21] LARMAN, C. **Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento**. 3. Ed. Editora Bookman, 2007. 695p.

[22] AMLAND, S. ***Risk-based Testing: Risk analysis fundamentals and metrics for software testing including a financial application case study***. In: 5th International Conference EuroSTAR'99, Espanha, 1999.

[23] CHEN, Y. ***Specification-Based Regression Testing Measurement with Risk Analysis***. Dissertação de mestrado, University of Ottawa/ Canada, 2002.

[24] JØRGENSEN, L. K. U.; MALDONADO, J.; JINO M. **A software tool for risk based testing**. Trabalho de graduação, NorwegianUnivesity of Science and Technology/ Norway, 2004.