UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA WEB PARA TESTES DE FÓRMULAS MATEMÁTICAS

KAUÊ DA SILVA VIEIRA

BLUMENAU
2011

2011/2-13

KAUÊ DA SILVA VIEIRA

FERRAMENTA WEB PARA TESTES DE FÓRMULAS MATEMÁTICAS

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. Aurélio Faustino Hoppe – Orientador

BLUMENAU
2011

2011/2-13

FERRAMENTA WEB PARA TESTES DE FÓRMULAS MATEMÁTICAS

Por

KAUÊ DA SILVA VIEIRA

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

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Joyce Martins, Mestre – FURB
	_____
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB

Blumenau, 14 de dezembro de 2011

Dedico este trabalho a meus pais Antônio e Cleonice, a minha namorada Josileide e a todos que colaboraram para sua realização.

AGRADECIMENTOS

À minha família, que sempre me incentivou e me aconselhou nos momentos difíceis.

À minha namorada, que foi paciente e que sempre esteve ao meu lado durante o desenvolvimento do trabalho.

Ao meu orientador, Aurélio Faustino Hoppe, que acreditou no desenvolvimento deste trabalho e que soube me instruir corretamente para que o trabalho pudesse ser finalizado.

A todos meus amigos, que direta ou indiretamente colaboraram para o desenvolvimento deste trabalho.

A minha empresa, Fácil Informática, que permitiu que eu faltasse ao trabalho quando foi necessário.

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta que permite a execução de testes de métodos de classes. A ferramenta executa testes comparando resultados, realizando testes de funções incorretas, que é um dos tipos de testes de caixa preta. A partir da ferramenta desenvolvida é possível informar o nome das classes e dos métodos que o usuário pretende testar e as fórmulas matemáticas cujo resultado da avaliação é comparado com o resultado da execução dos métodos. A ferramenta executa os métodos e as fórmulas para identificar se o que foi desenvolvido está de acordo com o que foi solicitado. Os testes são descritos de forma textual. Para isso foi criada uma linguagem que é interpretada por analisadores léxico, sintático e semântico. Para o desenvolvimento da ferramenta foi utilizado o ambiente de desenvolvimento Visual Studio 2010, com a linguagem de programação C# e o gerenciador de banco de dados MySQL.

Palavras-chave: Teste de software. Compiladores.

ABSTRACT

This paper presents the development of a tool that allows you to perform classes methods testing. The tool performs tests comparing results, tests of functions incorrect, which is a type of black box testing. From the tool developed you can enter the name of the classes and methods that you want to test and mathematical formulas whose evaluation result is compared to the result of the methods execution. The tool performs the methods and formulas to determine if what was developed is consistent with what was requested. The tests are described in textual form. For it was created a language that is interpreted by lexical analysis, syntactic and semantic. For the development of the tool was used the Visual Studio 2010, with the C # programming language and database manager MySQL.

Key-words: Software testing. Compilers.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

CMM – *Capability Maturity Model*

DLL – *Dynamic Link Library*

GALS – Gerador de Analisadores Léxicos e Sintáticos

IEC – *International Electrotechnical Commission*

ISO – *International Organization for Standardization*

PSP – *Personal Software Process*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SQL – *Structured Query Language*

UC – *Use Case*

UML – *Unified Modeling Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A disputa e a concorrência por espaço no mercado aumentam constantemente. O surgimento acelerado de empresas permite que muitos produtos sejam criados com grande variedade, de forma que o consumidor possa efetuar escolhas de acordo com sua preferência. Com a opção de escolha, surge a necessidade de evolução. Para que não fiquem para trás, as empresas cada vez mais necessitam investir na qualidade de seus produtos. Nas empresas de softwares isto não é diferente.

Para o desenvolvimento de softwares de qualidade é necessário a realização de testes para evitar e prevenir os erros nos sistemas. Conforme Koscianski e Soares (2006, p. 18), "o objetivo do teste é encontrar defeitos revelando que o funcionamento do software em uma determinada situação não está de acordo com o esperado".

A necessidade de realização de testes dos softwares fez com que diversas técnicas fossem criadas para facilitar este processo. Começam a ser identificados diversos tipos de testes que precisam ser realizados para que um padrão de qualidade possa ser mantido. "O teste de software a ser realizado deve utilizar de técnicas de teste bem estabelecidas, visto que apenas estratégias de organização e distribuição de tarefas não são suficientes para a realização de teste com confiabilidade" (INTHURN, 2001, p. 55).

Em consequência dos softwares estarem automatizando processos em todas as áreas, muitas vezes um sistema precisa utilizar fórmulas matemáticas complexas. Isto torna mais difícil a realização de testes que realmente garantam o funcionamento do software.

Sendo assim, neste trabalho é apresentada uma ferramenta que procura facilitar o processo de verificação e consistência de métodos de classes que executam fórmulas matemáticas para programas desenvolvidos na linguagem C#. A partir da ferramenta é possível informar o nome das classes e dos métodos que o usuário pretende testar e as fórmulas matemáticas cujo resultado da avaliação é comparado com o resultado da execução dos métodos. O software executa os métodos e as fórmulas para identificar se o que foi desenvolvido está de acordo com o que foi solicitado.

As informações de entrada da ferramenta, os métodos e as fórmulas, são escritas em uma linguagem formal específica. Para validação e interpretação dessa linguagem foram desenvolvidos analisadores léxico, sintático e semântico. Para a execução dos métodos das classes foram utilizadas técnicas de reflexão.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que facilite a identificação de erros em métodos de classes que realizam cálculos matemáticos na linguagem C#.

Os objetivos específicos do trabalho são:

a) desenvolver uma ferramenta Web para definição de testes que:
   - execute testes e retorne os resultados das verificações através de relatório,
   - agende testes definindo data, hora e periodicidade de sua execução;
b) implementar analisadores léxico e sintático para validação dos métodos de classes que executam fórmulas matemáticas escritos na linguagem C#;
c) implementar analisadores léxico, sintático e semântico para validação e interpretação das fórmulas matemáticas;
d) importar/exportar testes no formato *eXtensible Markup Language* (XML).

1.2 ESTRUTURA DO TRABALHO

O trabalho está divido em quatro capítulos. No primeiro capítulo é feita uma introdução do trabalho, no segundo capítulo é abordada a fundamentação teórica necessária para o desenvolvimento do trabalho junto com os trabalhos correlatos, no terceiro capítulo é descrito o desenvolvimento da ferramenta de testes e experimentos realizados. Por fim, no quarto capítulo são apresentadas as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são abordados testes de software. Na seção 2.2 é apresentada uma visão geral sobre interpretação de linguagem. Na seção 2.3 são demonstradas técnicas de reflexão. Na seção 2.4 é apresentada técnica de execução de cálculos matemáticos. Na seção 2.5 são descritos os trabalhos correlatos.

2.1 TESTES DE SOFTWARE

Conforme Mecenas e Oliveira (2005, p. 44), desde o início da criação de softwares, o processo de testes não tem obtido uma devida importância. Os testes são simplesmente voltados à revisão básica do código-fonte, correção de problemas conhecidos e são realizados com foco nos acertos deixando de lado a procura por erros. É importante ressaltar que para o desenvolvimento de software com qualidade a procura de novos erros é essencial.

Segundo Mecenas e Oliveira (2005, p. 44), em 1979 o processo de testes ganhou uma nova abordagem, voltado para a intensa procura por erros, com isso tornando o processo muito mais trabalhoso, mas também muito mais eficaz. O que gerou uma grande modificação nas concepções de teste de softwares.

As principais razões para se realizar testes nos softwares, segundo Molinari (2003, p. 60), são retorno de investimento e melhora da qualidade do produto final. Com a realização de testes é possível reduzir a quantidade de erros e evitar problemas futuros que podem comprometer a confiança do usuário final, que normalmente é quem pagou pelo software.

As técnicas de testes conhecidas e utilizadas são (MECENAS; OLIVEIRA, 2005, p. 43):

a) teste de unidade: é utilizado para verificação de componentes ou classes;
b) teste de integração: tem o objetivo de testar as interfaces e verificar como se comportam em conjunto;
c) teste de sistema: visa a verificação do sistema de forma geral. Abrange vários tipos, desde testes de segurança do software até de desempenho;
d) teste de aceitação: é uma verificação formal para identificar se os requisitos definidos pelo usuário foram atendidos;

e) teste de caixa branca: tem o objetivo de verificar trechos do código-fonte que não foram testados;
f) teste de caixa preta: reflete a ótica do usuário, que tem o interesse em verificar a funcionalidade do software. Nesse tipo de teste não é verificada a estrutura interna do código fonte. É focado nos resultados e não no que está sendo realizado para se chegar ao resultado. Tem por objetivo a verificação de funções incorretas ou omitidas, erros de interface, erros de comportamento ou desempenho e erros de iniciação e término;
g) teste funcional: visa verificar se as regras de negócio especificadas funcionam corretamente;
h) teste de regressão: é a verificação de um teste que já foi realizado para identificar se alguma modificação no sistema não afetou as funcionalidades antigas.

Segundo Molinari (2010, p. 38), o principal motivo para a automação dos testes de software é a necessidade cada vez maior de se realizar mais testes em menos tempo. Os sistemas estão se tornando cada vez mais complexos e junto está aumentando a exigência por qualidade. Com isso é preciso que sejam criadas ferramentas que agilizem o processo de cada tipo de teste necessário para o desenvolvimento de um produto.

> As ferramentas de teste são instrumentos para facilitar o processo de teste. Elas podem realizar as tarefas de desenvolvimento e execução dos testes, manuseio das informações de resultado e a comunicação para os interessados. (MOREIRA FILHO; RIOS, 2003, p. 153).

Cortês e Chiossi (2001, p. 29) dizem que os testes não devem ser considerados apenas mais uma etapa no processo de desenvolvimento, que caso não sobre tempo para realizá-los por completo possam ser simplificados, para que não seja comprometido o prazo de entrega do software. Cortês e Chiossi (2001, p. 29) também afirmam que para a etapa de testes precisa ser dada tanta importância quanto para as outras etapas.

2.2 INTERPRETAÇÃO DE LINGUAGEM

Segundo Price e Toscani (2001, p. 1), o meio de comunicação mais prático entre as pessoas é a língua ou idioma. Para os computadores a linguagem de programação serve como meio de comunicação entre as pessoas que necessitam resolver algum problema e o computador. A linguagem de programação precisa interpretar as instruções informadas pelo

ser humano.

A necessidade de resolver um problema pode ser descrita para o computador através de uma linguagem de alto nível, que são as linguagens mais semelhantes à linguagem natural. As linguagens de alto nível buscam tornar mais fácil a informação dos problemas para o computador e é ela quem se comunica com as linguagens de baixo nível ou linguagem de máquina, traduzindo as informações para que possam ser executadas pelo computador. Para que seja realizada a tradução da linguagem de alto nível para a linguagem de máquina são utilizados compiladores ou interpretadores.

Para a interpretação da linguagem ou sintaxe informada foram elaboradas técnicas de compiladores.

> Compiladores são programas de computador que traduzem de uma linguagem para outra. Um compilador recebe como entrada um programa escrito na linguagem-fonte e produz um programa equivalente na linguagem-alvo. (LOUDEN, 2004, p. 1).

Conforme Grune et al. (2001, p. 5), os compiladores ou interpretadores analisam a linguagem de entrada e geram uma estrutura semântica que é utilizada para sintetizar a saída na linguagem alvo.

Como estão sendo citados compiladores e interpretadores, é importante ressaltar que existe uma diferença entre um compilador e um interpretador. Segundo Louden (2004, p. 4), os interpretadores executam o programa fonte direto sem a necessidade de gerar um código que será executado após o término da tradução e os compiladores sempre realizam a geração do código executável.

A interpretação de linguagens pode ser divida em três principais fases, que, segundo Price e Toscani (2001, p. 1), são:

a) análise léxica: identifica sequências de caracteres que constituem unidades léxicas, chamadas de *tokens*;
b) análise sintática: verifica se os *tokens* estão posicionados conforme a estrutura gramatical da linguagem;
c) análise semântica: identifica se o programa faz sentido, verificando, por exemplo, se existe compatibilidade entre operandos e operadores.

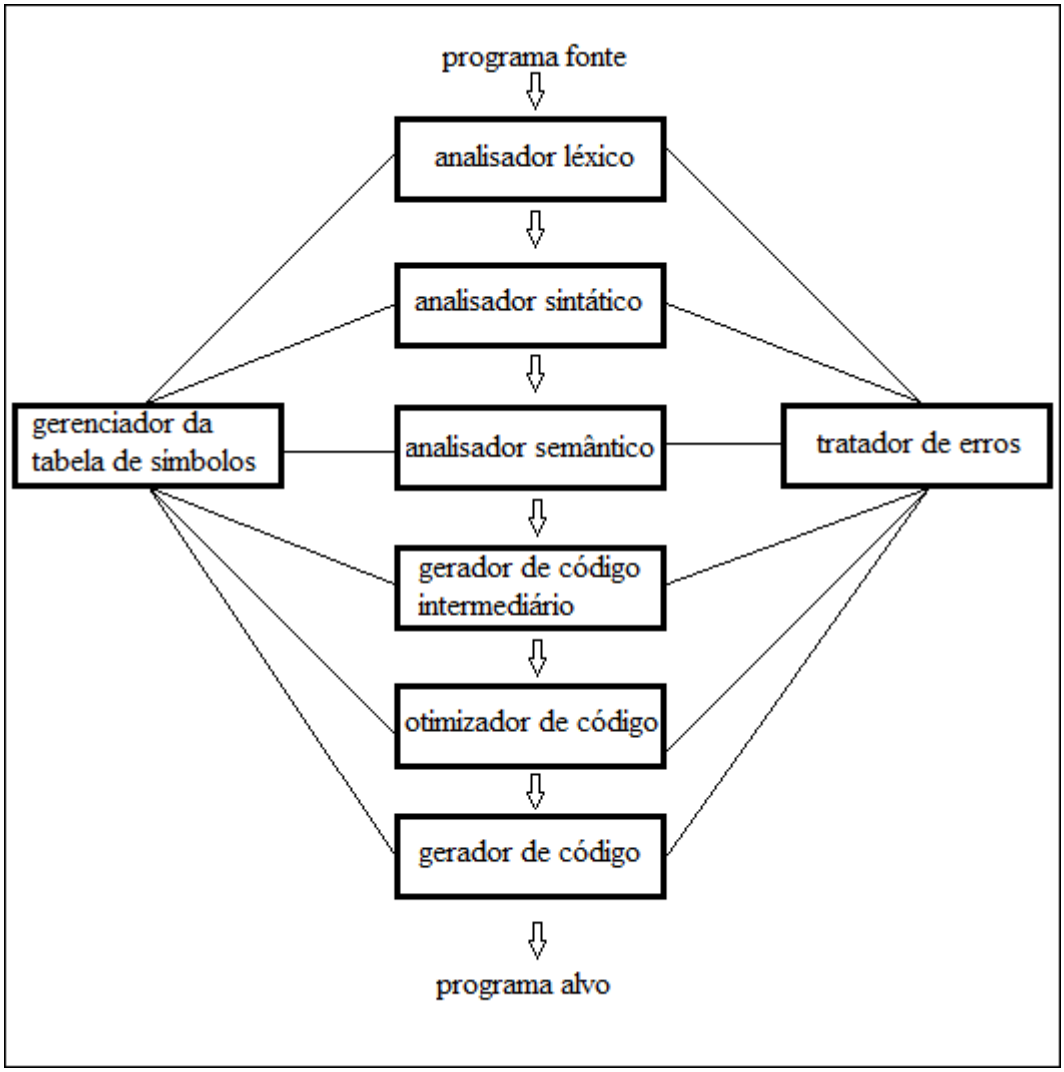
Na análise léxica, conforme Louden (2004, p. 31), é efetuada uma varredura que lê e percorre todo o programa fonte procurando identificar as palavras conhecidas ao compilador ou interpretador. Por exemplo, caso o compilador esteja definido para reconhecer a palavra `if`, então ao efetuar a varredura o analisador precisa identificar a palavra e armazená-la.

Após a identificação das palavras ou expressões conhecidas pelo compilador, é necessário verificar se elas estão posicionadas corretamente, procedimento que, conforme

Louden (2004, p. 145), é realizado pelo analisador sintático. O analisador verifica, por exemplo, se a palavra identificada `if` é seguida por uma condição e se após a condição é informada a palavra reservada `then`. Caso a sequência de entrada não esteja na ordem definida, o analisador identifica erro sintático.

Após o reconhecimento da entrada através dos analisadores léxico e sintático é realizada a análise semântica, que segundo Louden (2004, p. 259), efetua a computação da linguagem informada. É responsável por encontrar o significado da sintaxe identificada. Por exemplo, ao reconhecer (val = 34 * 10 + 5) o analisador é responsável por interpretar a operação encontrando o resultado do cálculo (`345`) e armazenar a informação na variável `val`.

Ao término das analises léxica, sintática e semântica, são realizadas as etapas de geração de código intermediário, otimização de código e geração de código antes da realização programa alvo. São etapas relacionadas com o preparo e geração do código da linguagem alvo. Na Figura 1 é possível visualizar a ordem de realização das etapas de um compilador.



Fonte: adaptado Ferreira (2011).

Figura 1 - Fases de um compilador

As etapas da Figura 1 são realizadas uma após a outra conforme sequência de execução, entrada do "programa fonte", "analisador léxico", "analisador sintático", "analisador semântico", "gerador de código intermediário", "otimizador de código", "gerador de código" e saída do "programa alvo", sendo que cada etapa depende do resultado da execução das etapas anteriores para que seja executada. Na Figura 1 também são apresentadas as etapas "gerenciador da tabela de símbolos" e "tratador de erros" que são responsáveis pelo controle das informações comuns a todas as etapas.

Com a utilização dos analisadores é possível definir o escopo de uma linguagem, determinando o que será permitido informar e como será reconhecido pelo interpretador.

A utilização de compiladores e interpretadores conforme Louden (2004, p. 2) é muito importante porque facilita a utilização das linguagens de programação, sem os mesmos seria necessário utilizar linguagem de máquina para desenvolver softwares, o que tornaria muito mais demorado o desenvolvimento dos sistemas complexos que existem atualmente.

2.3 TÉCNICAS DE REFLEXÃO

Segundo Golm e Kleinoder (1998), reflexão computacional é a atividade que permite aos objetos obterem informações sobre sua estrutura, por exemplo, atributos, métodos e outras informações.

As principais utilidades da reflexão, conforme Repass (2007), são:

a) acessar atributos em metadados de programas;

b) examinar e instanciar tipos em um conjunto;

c) criar novos tipos em tempo de execução;

d) permitir acessar métodos em tipos criados em tempo de execução.

A utilização da reflexão permite que sejam exploradas e acessadas, por exemplo, classes, métodos, atributos de classes, parâmetro de construtores, parâmetros de métodos e tipo do retorno de métodos, todos em tempo de execução.

O Quadro 1 apresenta funcionalidades possíveis com a utilização de reflexão, o exemplo foi desenvolvido na linguagem C#.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Reflection;
6
7   namespace ProgramaExemplo
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              System.Reflection.Assembly assembly = Assembly.LoadFile("C:/DLLs/ClassesBase.dll");
14              object classe = assembly.GetType("ClassesBase.Pessoa");
15
16              ConstructorInfo [] contrutoresDaClasse = classe.GetType().GetConstructors();
17              MethodInfo [] metodosDaClasse = classe.GetType().GetMethods();
18
19              MethodInfo metodo = metodosDaClasse[0];
20              ParameterInfo [] parametrosMetodo = metodo.GetParameters();
21              Type tipoRetornoMetodo = metodo.ReturnType;
22          }
23      }
24  }
```

Quadro 1 - Exemplo de utilização de propriedades com reflexão

O código-fonte exemplo no Quadro 1 efetua os seguintes procedimentos:

a) é carregada uma DLL de um local qualquer do computador (na linha 13, `ClassesBase.dll`);

b) a partir da DLL é carregada a classe `Pessoa` (na linha 14);

c) são carregados os construtores da classe (na linha 16);

d) são carregados os métodos da classe (na linha 17);

e) é identificado e carregado o primeiro método da classe (na linha 19);

f) são pegos os parâmetros do método carregado (na linha 20);

g) é carregado o tipo do retorno do método (na linha 21).

Entre as funcionalidade da reflexão é possível que a partir do nome da classe de um objeto possa ser feita a sua instância. O mesmo ocorre com os métodos dos objetos. A partir da instância do objeto é possível efetuar a execução dos seus métodos informando o nome e os parâmetros do método.

O Quadro 2 apresenta um exemplo prático de utilização de reflexão que foi desenvolvido na linguagem C# para a instanciação de classe e execução de método.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Reflection;
6
7   namespace ProgramaExemplo
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              System.Reflection.Assembly assembly = Assembly.LoadFile("C:/DLLs/FuncoesMatematicas.dll");
14
15              object gerar = assembly.CreateInstance("FuncoesMatematicas.Calculadora", true,
16                  System.Reflection.BindingFlags.CreateInstance, null, new object[0] {}, null, new object[0]);
17
18              int resultado = (int) gerar.GetType().InvokeMember("multiplicar", BindingFlags.InvokeMethod,
19                  null, gerar, new object[2]{4, 5});
20
21              Console.WriteLine("resultado = {0}", resultado);
22
23          }
24      }
25  }
```

Quadro 2 – Exemplo de execução de método com reflexão

O código-fonte exemplo no Quadro 2 efetua os seguintes procedimentos:

a) é carregada a DLL que possui a classe a ser instanciada (na linha 13, `FuncoesMatematicas.dll`);

b) a partir da DLL carregada é feita a instanciação de um objeto da classe `Calculadora` (na linha 15);

c) é executado o método `multiplicar` da classe (na linha 18). O método efetua multiplicação entre dois números inteiros;

d) é listado o resultado da operação (na linha 21).

Conforme Repass (2007) com a utilização de técnicas de reflexão é possível executar funções do sistema sem ter acesso ao código-fonte. É preciso apenas ter as DLLs compiladas com o código e conhecer a assinatura das classes e dos métodos que se deseja executar.

2.4 EXECUÇÃO DE CÁLCULOS MATEMÁTICOS

Segundo Diverio e Menezes (2003, p. 65-68), qualquer problema pode ser representado por uma máquina universal. Uma das máquinas universais é a máquina de pilhas que utiliza a estrutura de dados do tipo pilha, que serve como memória para o controle e execução das informações.

Diverio e Menezes (2003, p. 111) afirmam que a máquina de pilhas pode ser divida em três partes, que são:

a) variável X: é a variável de entrada;
b) variável Y: representa as pilhas, que são utilizadas como memórias de trabalho;
c) programa: é a sequência de instruções.

Uma necessidade que pode ser atendida utilizando a máquina de pilhas é a de efetuar cálculos de fórmulas matemáticas. A estrutura de pilhas permite, por exemplo, que a operação `(4 + 2) * 3` seja processada (variável X). Para isso, os operandos e operadores são empilhados (variável Y) respeitando a ordem correta de execução dos cálculos. Primeiro são empilhadas as constantes e avaliada a operação que está entre parênteses `4 + 2`. Depois é realizada a multiplicação do resultado da primeira operação com a constante `3`, resultando no valor `18`.

Para a realização de cálculos através de máquinas de pilhas é preciso utilizar notações matemáticas que definam como as operações deverão ser efetuadas (BRUNI, 2008, p. 8). Existem três tipos de notações que podem ser utilizadas:

a) notação infixa;
b) notação pré-fixada;
c) notação pós-fixada.

A notação infixa é a notação mais convencional, onde o operador é representado entre os operandos, conforme é mostrado no Quadro 3.

Operação	Notação Infixa
A + B	(A + B)
$\frac{A + B}{C}$	(A + B) / C

Quadro 3 - Exemplo de notação infixa

A notação pré-fixada ou notação polonesa apresenta o operador seguindo pelos dois

operandos que serão calculados, conforme é mostrado no Quadro 4.

Operação	Notação Pré-fixada
A + B	+ A B
$\frac{A + B}{C}$	/ + A B C

Quadro 4 - Exemplo de notação pré-fixada

A notação pós-fixada ou notação polonesa inversa apresenta o operador após os dois operandos que serão calculados, conforme é mostrado no Quadro 5.

Operação	Notação Pós-fixada
A + B	A B +
$\frac{A + B}{C}$	A B + C /

Quadro 5 - Exemplo de notação pós-fixada

Conforme Bruni (2008, p. 8), as notações pré-fixada e pós-fixada foram criadas com o objetivo de permitir a representação de expressões matemáticas sem o uso de parênteses. A realização das operações são realizadas empilhando e desempilhando os operandos e operadores de acordo com a notação utilizada. Observa-se que a utilização das notações aplicadas em máquinas de pilhas torna mais simples a realização das operações matemáticas através de computadores.

2.5 TRABALHOS CORRELATOS

Foram verificados trabalhos acadêmicos que também tem por objetivo a automatização dos testes, os quais são: ferramentas de apoio a geração de testes de Marcos (2007) que será apresentada na seção 2.5.1, sistema para gerenciamento de testes funcionais de software de Kolm (2001) que será abordado na seção 2.5.2 e ferramenta para testes de programação utilizando componentes da biblioteca CLX de Santiago (2002) mostrado na seção 2.5.3. Na seção 2.5.4 é feito um comparativo das características dos trabalhos.

2.5.1 Ferramentas de apoio a geração de testes

A ferramenta desenvolvida por Marcos (2007) tem o objetivo facilitar os testes dos programas desenvolvidos na linguagem Delphi. Para isso, o aplicativo interpreta formulários da linguagem extraindo informações necessárias para a geração dos *scripts* de teste de criação, alteração e exclusão de dados de registros para os formulários.

A interpretação dos formulários é feita utilizando os analisadores léxico, sintático e semântico que analisam o código fonte para a geração dos *scripts* e identificam a assinatura dos métodos para permitir o acesso aos formulários.

Os *scripts* são gerados para DelphiScript que é uma linguagem utiliza pela ferramenta de automação de testes TestComplete. A ferramenta utiliza testes funcionais para que possam ser testadas as entradas e as saídas da aplicação e também utiliza testes de regressão para que os testes possam ser executadas quantas vezes for necessário.

2.5.2 Sistema para gerenciamento de testes funcionais de software

O sistema desenvolvido por Kolm (2001) tem o objetivo de facilitar o processo de testes para os sistemas da empresa Benner Sistemas S/A. Foi criado um módulo de qualidade integrado aos sistemas da empresa, onde é possível informar os módulos, as versões e as rotinas a serem verificadas.

A ferramenta utiliza listas de checagem. São informadas listas de checagem de entrada, que são testadas pela equipe de qualidade responsável pelos testes, onde os erros são identificados. Os testes realizados que não ocorrerem erros são armazenados em uma lista de checagem de saída que pode ser utilizada para outros testes posteriormente. Os testes realizados que ocorreram erros são armazenados em um cadastro de mensagens de erro, que é passado para o programador verificar os erros ocorridos.

Para efetuar o controle das listas de checagem de entrada e saída e controle de mensagens de erro, o sistema utiliza gerenciamento de testes funcionais, o que pode ser considerado uma técnica de automatização de testes.

O sistema foi desenvolvido utilizando o gerenciador de banco de dados MsSQL versão 7.0, utilizou a ferramenta Benner Builder para criação das tabelas na base de dados, utilizou a ferramenta Benner Runner que gera códigos na linguagem Delphi para a construção da

interface.

2.5.3 Ferramenta para testes de programação utilizando a ferramenta CLX

A ferramenta desenvolvida por Santiago (2002) utilizando a linguagem Delphi, tem o objetivo de facilitar a passagem de exercícios de programação para os alunos. O sistema permite que um professor defina uma lista de exercícios de programação com os seus enunciados e permite que o professor informe possíveis soluções para as questões.

O sistema é dividido em dois módulos, um módulo que é visualizado pelo professor, onde pode determinar os exercícios e os alunos que poderão resolvê-los, um módulo que é visualizado pelos alunos, onde o aluno tem acesso aos exercícios disponibilizados pelo professor e pode resolver e enviar os problemas resolvidos.

Após o envio dos exercícios a ferramenta valida o que foi feito, comparando o resultado esperado com o resultado da execução dos programas e retorna para o aluno os problemas encontrados, permitindo que o aluno verifique os problemas e reenvie a solução.

Para a resolução dos exercícios pode-se utilizar qualquer linguagem de programação que gere arquivos, pois a validação é feita nos resultados que são gerados pelos programas em arquivos em formato de texto.

Para a correção dos exercícios utiliza testes funcionais onde são verificados os resultados esperados pelo professor em comparação com o resultado informado pelo aluno.

2.5.4 Comparativo das características dos trabalhos

Esta seção apresenta uma síntese dos trabalhos relacionados que se deu pela análise e comparação dos diferentes sistemas em relação às seguintes características:

a) armazena histórico de testes realizados: o armazenamento dos históricos de testes realizados é importante para que sejam feitas comparações entre os resultados obtidos com os testes;
b) possui interface Web: o fato de uma ferramenta executar na Web aumenta a mobilidade do sistema, tornando o sistema acessível a uma maior quantidade de pessoas sem a necessidade de possuir o sistema instalados e seus computadores para que possam ser utilizados;

c) disponibiliza relatórios de resultados: os relatórios de resultados é apenas uma maneira de facilitar a visualização do resultado da execução dos testes;
d) realiza testes de funções incorretas: identifica se a ferramenta realiza testes de funções das classes do sistema, que é um dos tipos de testes realizados pela técnica de teste de caixa preta;
e) efetua os testes na própria ferramenta: a característica identifica se é a própria ferramenta quem realiza os testes definidos sem a utilização de ferramentas de testes automatizados;
f) gera testes que são executados por outras ferramentas: a característica identifica se a ferramenta gera código-fonte que são executados por outras ferramentas de automatização de testes;
g) permite exportar/importar testes: a importação e exportação dos testes tem o objetivo de facilitar a realização dos testes tornando mais prático a reutilização de testes já executados;
h) utiliza reflexão para realizar os testes: característica que permite à ferramenta efetuar os testes sem ter acesso ao código-fonte do sistema testado.

O Quadro 6 mostra de forma resumida as principais características desses sistemas tendo como base critérios considerados importantes (descritos acima) e, de conceitos relacionados no decorrer deste capítulo.

Características dos trabalhos	**Marcos (2007)**	**Kolm (2001)**	**Santiago (2002)**
Armazena histórico de testes realizados	Não	Sim	Sim
Possui interface Web	Não	Não	Não
Disponibiliza relatórios de resultados	Não	Sim	Sim
Realiza testes de funções incorretas	Não	Não	Não
Efetua os testes na própria ferramenta	Não	Não	Sim
Gera testes que são executados por outras ferramentas	Sim	Não	Não
Permite exportar/importar testes	Sim	Não	Sim
Utiliza reflexão para realizar os testes	Não	Não	Não

Quadro 6 - Características dos trabalhos correlatos

Através da visualização do Quadro 6 é possível identificar as características que cada trabalho correlato possui.

O software de apoio a geração de testes de Marcos (2007) tem o objetivo de, a partir de formulários em Delphi, gerar *scripts* de criação, alteração e exclusão de registros para os formulários. É voltado para testes do tipo funcional e de regressão.

A ferramenta de gerenciamento de testes funcionais de Kolm (2001) é voltada para a organização dos testes dos módulos de um sistema em específico. Não realiza os testes

efetivamente, apenas os gerencia.

O sistema de testes de programação de Santiago (2002) tem o objetivo de facilitar o trabalho dos professores na passagem de exercícios de programação para seus alunos. Utiliza testes funcionais de comparação. Os exercícios e suas possíveis soluções são cadastrados no sistema e são comparados com resultados informados pelos alunos.

Os trabalhos correlatos são voltados principalmente para testes funcionais ou para gerência de testes, mas nenhum deles é voltado para os testes de funções incorretas que é a verificação se as funções executadas estão retornando o que é esperado.

3 DESENVOLVIMENTO

A apresentação do desenvolvimento do trabalho está divida em quatro seções. Na primeira seção (3.1) são abordados os principais requisitos definidos para o desenvolvimento do trabalho. Na segunda seção (3.2) é apresentada a especificação da ferramenta desenvolvida. Na terceira seção (3.3) é mostrada a implementação da ferramenta. Na quarta seção (3.4) são abordados os resultados e discussões do trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta para testes de fórmulas matemáticas foi desenvolvida utilizando como base os seguintes requisitos:

a) cadastrar testes a serem executados, informando o nome do teste, o método de classe que será verificado e a fórmula que será comparada ao resultado da execução do método (RF);
b) agendar a execução dos testes, informando a data (dia, mês e ano), a hora (hora e minuto) e a periodicidade que o teste deverá ser executado (somente uma vez, diariamente, semanalmente, mensalmente, anualmente) (RF);
c) manter histórico dos testes com o resultado gerado caso já tenha sido executado. Deverá controlar a situação do teste (aguardando execução, executando, concluído com sucesso, concluído com erro) (RF);
d) emitir relatório de testes executados. Será listado o nome do teste, o nome da classe e do método verificado e a fórmula utilizada para validação (RF);
e) validar a sintaxe dos testes definidos utilizando os analisadores léxico, sintático e semântico (RF);
f) permitir a importação/exportação dos testes utilizando arquivos no formato XML (RNF);
g) interpretar e executar métodos de classes da linguagem C# .NET utilizando técnicas de reflexão para permitir a realização de testes de funções incorretas (RNF);
h) ser implementada no ambiente de desenvolvimento Visual Studio 2010, utilizando

a linguagem C# .NET (RNF);

i) utilizar o gerenciador de banco de dados MySQL para armazenar os testes e os agendamentos de testes (RNF);
j) ser compatível com os sistemas operacionais Windows XP, Windows Vista e Windows 7 (RNF);
k) utilizar a ferramenta GALS (GESSER, 2003) para geração dos analisadores léxico, sintático que interpretarão a sintaxe dos testes definidos (RNF).

3.2 ESPECIFICAÇÃO

A especificação da ferramenta foi representada através de diagramas da *Unified Modeling Language* (UML) utilizando a ferramenta *Enterprise Architect* (SPARXSYSTEMS, 2000). Os diagramas especificados foram o de caso de uso, de classes e de sequência.

Na seção 3.2.1 é apresentado o diagrama de casos de uso, na seção 3.2.2 os diagramas de classes e na seção 3.2.3 são apresentados os diagramas de sequência.

3.2.1 Diagrama de casos de uso

Na especificação do diagrama (Figura 2) são apresentadas as funções que são exercidas pelo testador, que é o usuário da ferramenta.

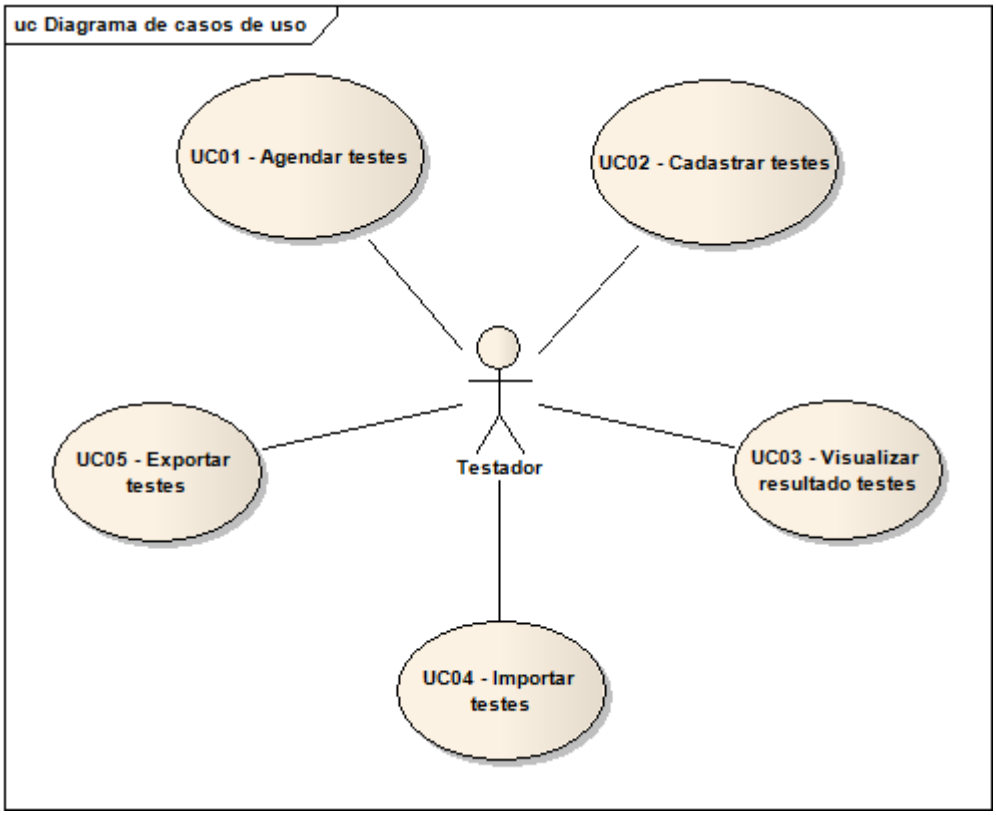


Figura 2 - Diagrama de casos de uso realizado pelo testador

No diagrama de casos de uso da Figura 2 são exibidos os casos, cadastrar agendamento de testes que é onde é definida a data de execução dos testes a periodicidade que os testes devem ser executados, cadastrar testes que é o momento onde são definidos os testes a serem executados, visualizar testes que é uma função do testador após os testes terem sido executados, importar testes que é uma funcionalidade que permite ao testador importar testes exportados para os agendamentos cadastrados no sistema e exportar testes que a funcionalidade que permite ao testador exportar os testes criados no sistema.

A seguir são demonstrados os detalhamentos de cada caso de uso identificado no diagrama de casos de uso da Figura 2. No Quadro 7 é demonstrado o detalhamento do cadastro dos agendamentos de testes, no Quadro 8 é apresentado o detalhamento do cadastro de testes, no Quadro 9 é exibido o detalhamento da visualização dos resultados dos testes, no Quadro 10 é apresentado o detalhamento da importação de testes e no Quadro 11 é demonstrado o detalhamento da exportação de testes.

UC01 - Agendar testes	
Cenário principal	01) O sistema apresenta o cadastro de agendamento 02) O testador informa a identificação do agendamento 03) O testador informa a data em que o agendamento deverá ser executado 04) O testador seleciona a opção ativo igual a sim 05) O testador seleciona a periodicidade somente uma vez 06) O sistema verifica se todas as informações foram preenchidas 07) O sistema grava o registro do agendamento 08) O sistema executa o agendamento ao atingir a data definida para execução
Cenário alternativo 1	No passo 08 caso a opção ativa esteja preenchida como não 08.1) O sistema não irá executar o agendamento
Cenário alternativo 2	No passo 08 caso a data de execução seja inferior a data atual 08.1) O sistema não irá executar o agendamento
Cenário exceção 1	No passo 06 caso algum campo não tenha sido preenchido 06.1) O sistema informa o não preenchimento do campo 06.2) O sistema volta para o passo 01
Cenário exceção 2	No passo 06 caso a data informada esteja em um formato inválido 06.1) O sistema informa que a data está em um formato incorreto 06.2) O sistema volta para o passo 01
Pós-condição	O agendamento estará cadastrado

Quadro 7 - Caso de Uso 01

No caso de uso do Quadro 7 são demonstrados os procedimentos que ocorrem ao cadastrar um agendamento de testes. A execução do agendamento comentada no item oito do caso de uso não ocorre necessariamente na sequência do cadastramento. Os testes serão realizados apenas na data de execução definida no agendamento.

UC02 - Cadastrar testes	
Pré-condições	O agendamento de testes deve estar cadastrado
Cenário principal	01) O sistema apresenta o cadastro do teste 02) O testador informa a identificação do teste 03) O testador informa a conexão de banco de dados que será utilizada no teste 04) O testador informa conteúdo do teste utilizando a linguagem criada para definição dos testes 05) O sistema verifica se os campos obrigatórios foram preenchidos 06) O sistema grava o registro do teste
Cenário exceção 1	No passo 05 caso algum campo obrigatório não tenha sido preenchido 05.1) O sistema informa o não preenchimento do campo 05.2) O sistema volta para o passo 01
Pós-condição	O teste estará cadastrado

Quadro 8 - Caso de Uso 02

No caso de uso do Quadro 8 são demonstrados os procedimentos que ocorrem ao cadastrar um testes.

UC03 - Visualizar resultado testes	
Pré-condições	O teste deve ter sido executado sem erro
Cenário principal	01) O sistema apresenta um *link* para *download* do arquivo no cadastro de teste 02) O testador clicará no link para abrir o arquivo 03) O sistema apresentará o relatório no formato XLS com o resultado da execução do teste
Pós-condição	O resultado da execução do teste terá sido visualizado

Quadro 9 - Caso de Uso 03

No caso de uso do Quadro 9 são demonstrados os procedimentos para visualizar o relatório com o resultado da execução do teste. É gerado um relatório para cada teste cadastrado no agendamento após o agendamento ter sido executado.

UC04 - Importar testes	
Pré-condições	Possuir ao menos um agendamento criado no sistema
Cenário principal	01) O sistema apresenta o cadastro de importação de testes 02) O testador informa a identificação da importação 03) O testador seleciona o tipo importador 04) O testador seleciona o agendamento onde os testes serão importados 05) O testador informa qual arquivo será importado 06) O sistema verifica se todos os campos foram preenchidos 07) O sistema verifica se o arquivo informado existe 08) O sistema efetua a importação dos testes para o agendamento 09) O sistema cria o registro de importação
Cenário alternativo 1	No passo 03 caso o testador selecione o tipo exportar 03.1) O sistema exibe as informações para o cadastro da exportações ao invés de exibir as informações para o cadastro da importação
Cenário exceção 1	No passo 07 caso o arquivo informado não exista 07.1) O sistema avisa que o arquivo informado não existe 07.2) O sistema volta para o passo 01
Cenário exceção 2	No passo 08 caso ocorra algum erro na importação do arquivo 08.1) O sistema registra o erro ocorrido no campo de *log* 08.2) O sistema volta para o passo 01
Cenário exceção 3	No passo 06 caso algum campo não tenha sido preenchido 06.1) O sistema informa o não preenchimento do campo 06.2) O sistema volta para o passo 01
Pós-condição	Os testes serão importados no agendamento

Quadro 10 - Caso de Uso 04

No caso de uso do Quadro 10 são demonstrados os procedimentos para realizar a importação de testes em um agendamento já criado no sistema.

UC05 - Exportar testes	
Pré-condições	Possuir ao menos um agendamento criado no sistema
Cenário principal	01) O sistema apresenta o cadastro de exportação de testes 02) O testador informa a identificação da exportação 03) O testador seleciona o tipo exportador 04) O testador seleciona o agendamento onde os testes serão exportados 05) O testador seleciona os testes do agendamento que deverão ser exportados 06) O testador informa para onde o arquivo deverá ser exportado 07) O sistema verifica se todos os campos foram preenchidos 08) O sistema efetua a exportação dos testes para o arquivo informado 09) O sistema cria o registro de exportação
Cenário alternativo 1	No passo 03 caso o testador selecione o tipo importar 03.1) O sistema exibe as informações para o cadastro da importação ao invés de exibir as informações para o cadastro da exportação
Cenário exceção 1	No passo 08 caso o diretório do arquivo informado não exista 08.1) O sistema avisa que o arquivo informado não existe 08.2) O sistema volta para o passo 09
Cenário exceção 2	No passo 08 caso ocorra algum erro na exportação do arquivo 08.1) O sistema registra o erro ocorrido no campo de *log* 08.2) O sistema volta para o passo 01
Cenário exceção 3	No passo 07 caso algum campo não tenha sido preenchido 07.1) O sistema informa o não preenchimento do campo 07.2) O sistema volta para o passo 01
Pós-condição	Os testes serão exportados para um arquivo no formato XML

Quadro 11 - Caso de Uso 05

No caso de uso do Quadro 11 são demonstrados os procedimentos para realizar a exportação dos testes de um agendamento criado no sistema.

3.2.2 Diagrama de classes

O diagrama de classes foi dividido em quatro partes para facilitar a visualização e compreensão das classes desenvolvidas. No diagrama de classes geral da Figura 3 são apresentadas as classes que armazenam as informações dos cadastros do sistema, junto com algumas classes de execução dos testes. No diagrama de classes analisador da Figura 4 são apresentadas as classes que foram geradas através da ferramenta GALS (GESSER, 2003), junto com algumas classes criadas para complementar as classes geradas. No diagrama de classes controlador semântico da Figura 5 são apresentadas as classes utilizadas na execução da analise semântica dos testes. No diagrama de classes interface WEB da Figura 6 são apresentadas as classes das telas de cadastro da ferramenta e as principais classes utilizadas para controle dos cadastros.

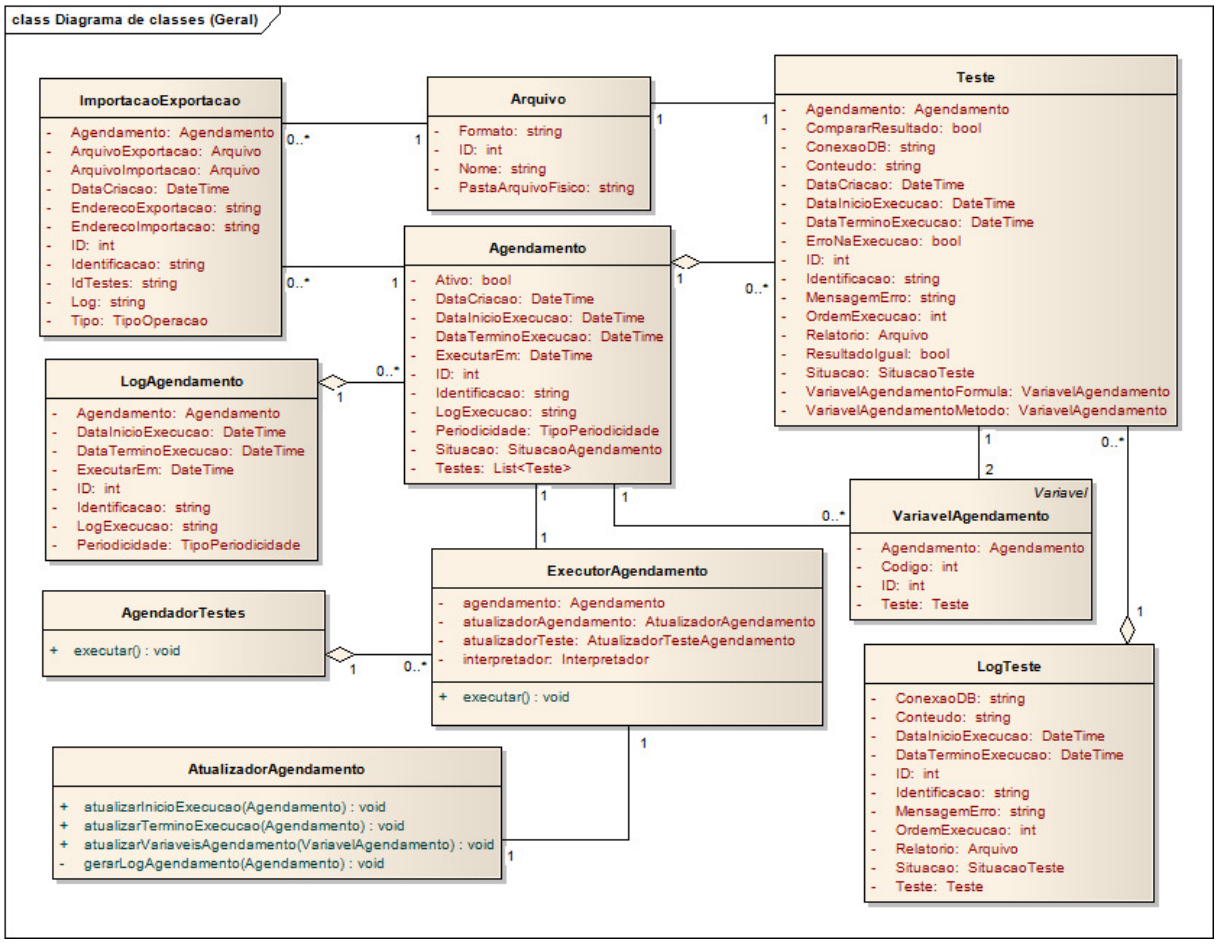


Figura 3 - Diagrama de classes geral

Segue o detalhamento das classes apresentadas no diagrama de classes geral da Figura 3:

a) classe `Agendamento`: responsável por armazenar informações dos agendamentos. A classe armazena informações sobre se o teste está ativo ou não, data de criação, data de início de execução, data de término de execução, data em que o agendamento deve ser executado, identificação, informações do *log* de execução, periodicidade de execução, código de identificação, situação e testes do agendamento;

b) classe `Teste`: responsável por armazenar informações dos testes. A classe armazena informações sobre a identificação do teste, o agendamento ao qual o teste está vinculado, a data de criação, data de início de execução, data de término de execução, código de execução, se houve erro ou não na execução, se no teste deverão ser comparados os resultados ou não, a conexão de banco de dados utilizada no teste, a ordem de execução, a mensagem de erro na execução do teste, o relatório de resultados, se o resultado da comparação está igual ou não, a situação e as variáveis de resultado de execução do método e da fórmula;

c) classe `Arquivo`: responsável por armazenar informações de todos os arquivos da ferramenta, relatórios, arquivos de importação e arquivos de exportação. A classe armazena informações do formato do arquivo, o nome do arquivo, o código de identificação e o endereço físico do arquivo. Os arquivos não são armazenados fisicamente na base de dados do sistema, o que é armazenado é apenas o caminho onde o arquivo está fisicamente;

d) classe `LogAgendamento`: responsável por armazenar os históricos de execução dos agendamentos. A classe armazena as informações do agendamento para que possam ser comparadas em um próxima execução do agendamento;

e) classe `LogTeste`: responsável por armazenar os históricos de execução dos testes. A classe armazena as informações do teste para que possam ser comparadas em uma próxima execução do teste;

f) classe `ImportacaoExportacao`: responsável por armazenar informações sobre importações e exportações realizadas. A classe armazena informações sobre a identificação da operação, o tipo de operação (se é uma importação ou uma exportação), o agendamento para o qual os testes serão importados de onde serão exportados, os testes que serão exportados caso seja uma exportação, o endereço do arquivo de importação, o endereço do arquivo de exportação, a data de criação do registro, o *log* de execução da operação, o código de identificação dos testes que serão exportados e o arquivo físico dos arquivos de importação e de exportação;

g) classe `VariavelAgendamento`: responsável por armazenar as informações dos resultados dos testes do agendamento. A classe é uma classe filha da classe `Variavel`, com isso herda as informações da mesma. Além das informações da variável, armazena o agendamento a qual está vinculada;

h) classe `AgendadorTestes`: responsável por monitorar e identificar os agendamentos que devem ser executados. A classe inicia a execução dos agendamentos;

i) classe `ExecutorAgendamento`: responsável por executar os agendamentos iniciados pela classe `AgendadorTestes`. Utiliza a classe `AtualizadorAgendamento` e a classe `AtualizadorTesteAgendamento` para realizar a atualização das informações dos agendamentos e dos testes dos agendamentos executados no banco de dados;

j) classe `AtualizadorAgendamento`: responsável por atualizar as informações referentes a execução do agendamento no banco de dados.

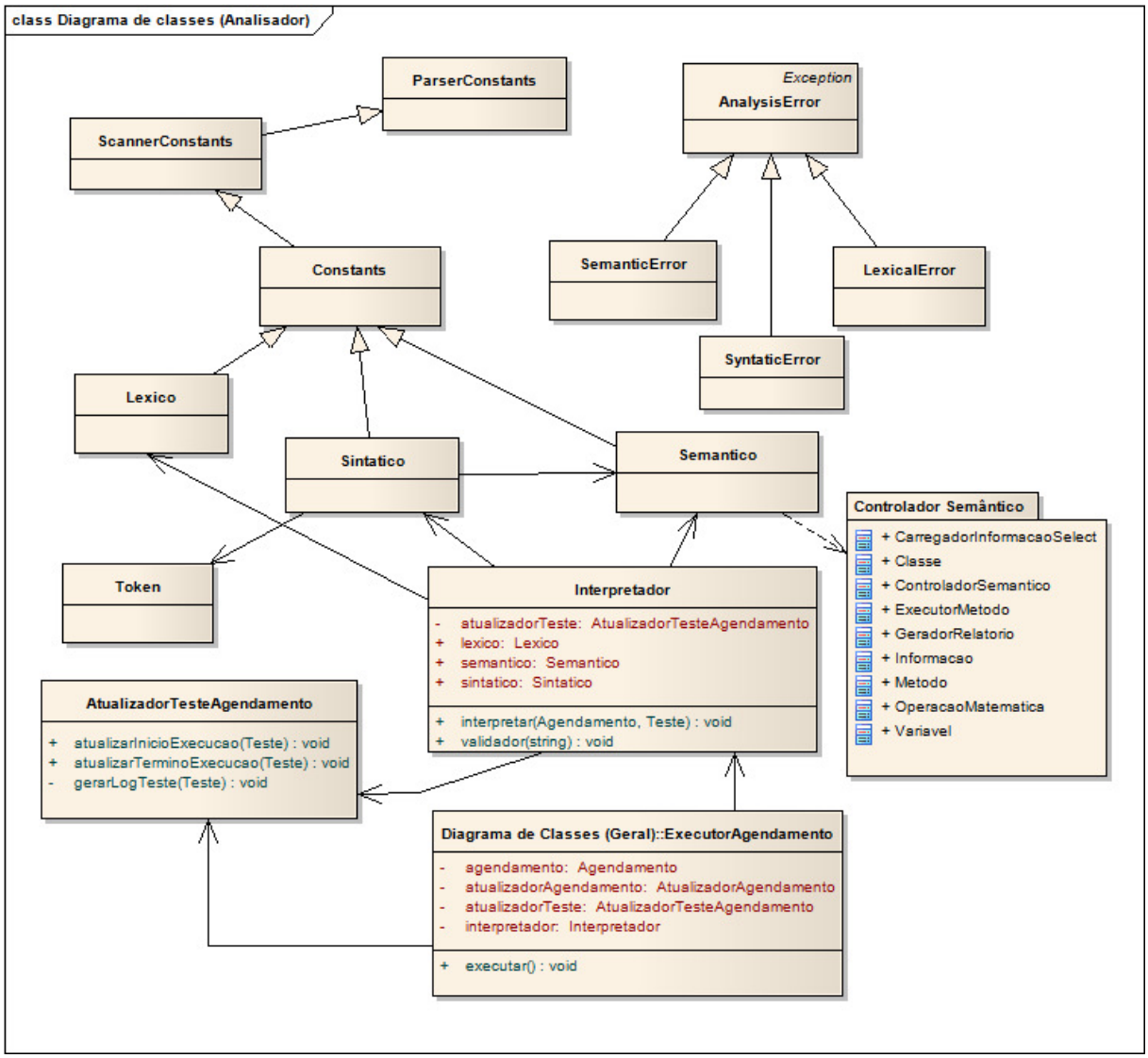


Figura 4 - Diagrama de classes analisador

Segue o detalhamento das classes apresentadas no diagrama de classes analisador da Figura 4:

a) classe `Interpretador`: responsável por iniciar a execução dos analisadores léxico, sintático e semântico. A classe além de instanciar os analisadores através do método interpretar, utiliza a classe `AtualizadorTesteAgendamento` para realizar a atualização das informações dos testes executados no banco de dados;

b) classe `AtualizadorTesteAgendamento`: responsável por atualizar as informações referentes a execução dos testes do agendamento no banco de dados;

c) classe `ExecutorAgendamento`: classe descrita no detalhamento do diagrama de classes geral;

d) demais classes são responsáveis pelo controle dos analisadores. Classes geradas pela ferramenta GALS (GESSER, 2003).

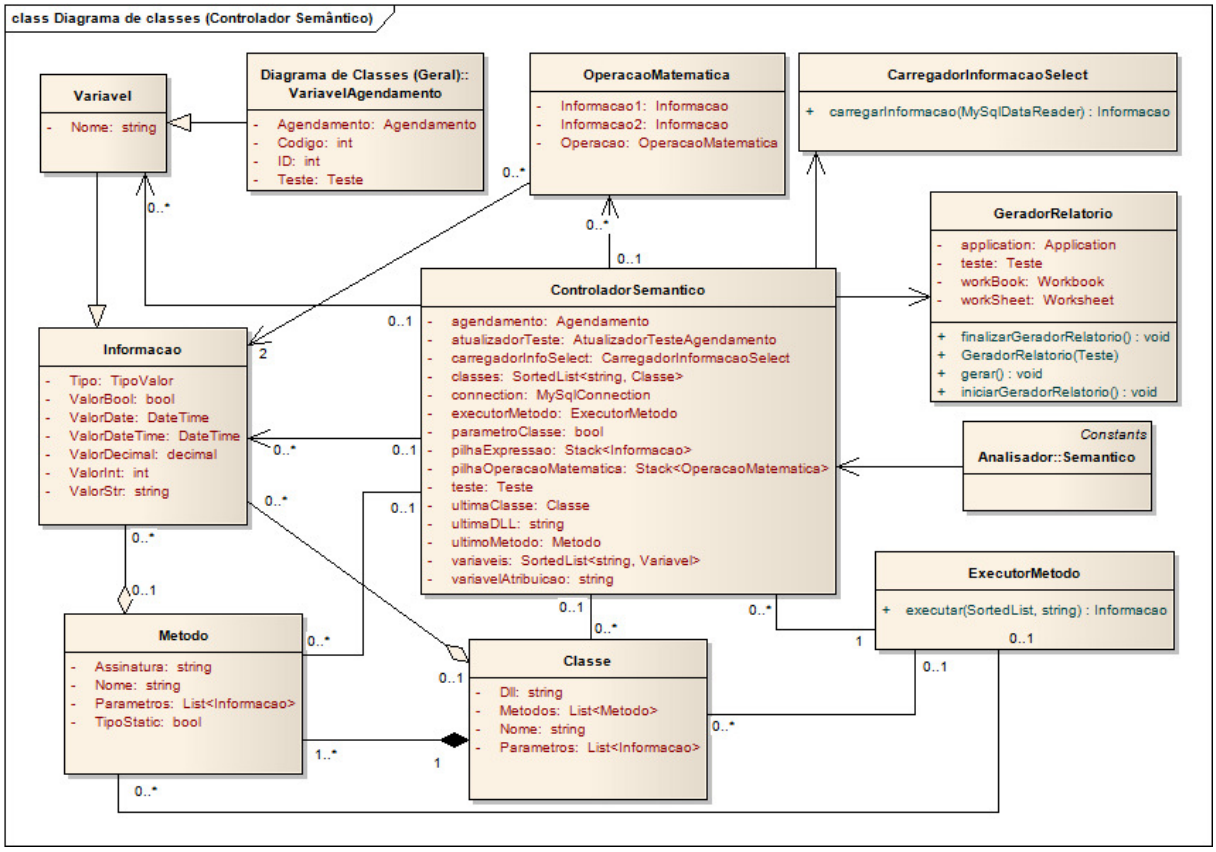


Figura 5 - Diagrama de classes controlador semântico

Segue o detalhamento das classes apresentadas no diagrama de classes controlador semântico da Figura 5:

a) classe `Semantico`: classe descrita no detalhamento do diagrama de classes analisador;

b) classe `VariavelAgendamento`: classe descrita no detalhamento do diagrama de classes geral;

c) classe `ControladorSemantico`: responsável pela execução do analisador semântico. Classe que controla o analisador semântico. Armazena todas as informações da execução e efetua a chamada de todas as classes utilizadas na execução;

d) classe `Variavel`: responsável por armazenar as variáveis utilizadas na análise semântica. A classe é uma classe filha da classe `Informacao`, com isso herda as informações da mesma. Além das informações armazena o nome das variáveis para que possam ser utilizadas no conteúdo dos testes;

e) classe `Informacao`: responsável por armazenar os tipos primitivos utilizados na

análise semântica. A classe armazena o tipo de valor e o valor das informações, permitindo que sejam armazenadas informações dos tipos `Bool`, `Date`, `DateTime`, `Decimal`, `Int` e `String`;

f) classe `OperacaoMatematica`: responsável por armazenar as operações matemáticas utilizadas na analise semântica. A classe armazena as informações necessárias para a execução das funções `Round`, `Square`, `Absolute` e `Pow` que serão mostradas posteriormente no Quadro 13;

g) classe `Classe`: responsável por armazenar informações das classes onde estão os métodos que serão executados. A classe armazena o nome da DLL a qual a classe pertence, os métodos da classe e os parâmetros do construtor da classe;

h) classe `Metodo`: responsável por armazenar informações dos métodos que serão executados através de reflexão. A classe armazena informações da assinatura do método, nome, parâmetros e identificação se o método é estático ou não;

i) classe `GeradorRelatorio`: responsável por gerar o relatório de resultados da execução. A classe gera o relatório utilizando informações da classe `Teste`. Após a geração do relatório a classe armazena o relatório no cadastro do teste;

j) classe `ExecutorMetodo`: responsável por executar os métodos definidos através de técnicas de reflexão. A classe possui o método `executar` que através da assinatura informada do método carrega a DLL, a classe e o método através de reflexão. Após o carregamento efetua a execução do método identificado retornando a informação de resultado da execução do método;

k) classe `CarregadorInformacaoSelect`: responsável por carregar da base de dados as consultas utilizadas na análise semântica. A classe possui o método `carregarInformacao` que identifica o tipo de dado retornado pela base de dados e armazena esta informação em objeto instanciado a partir da classe `Informacao`, o objeto de informação é retorna do pelo método que é utilizado na execução semântica.

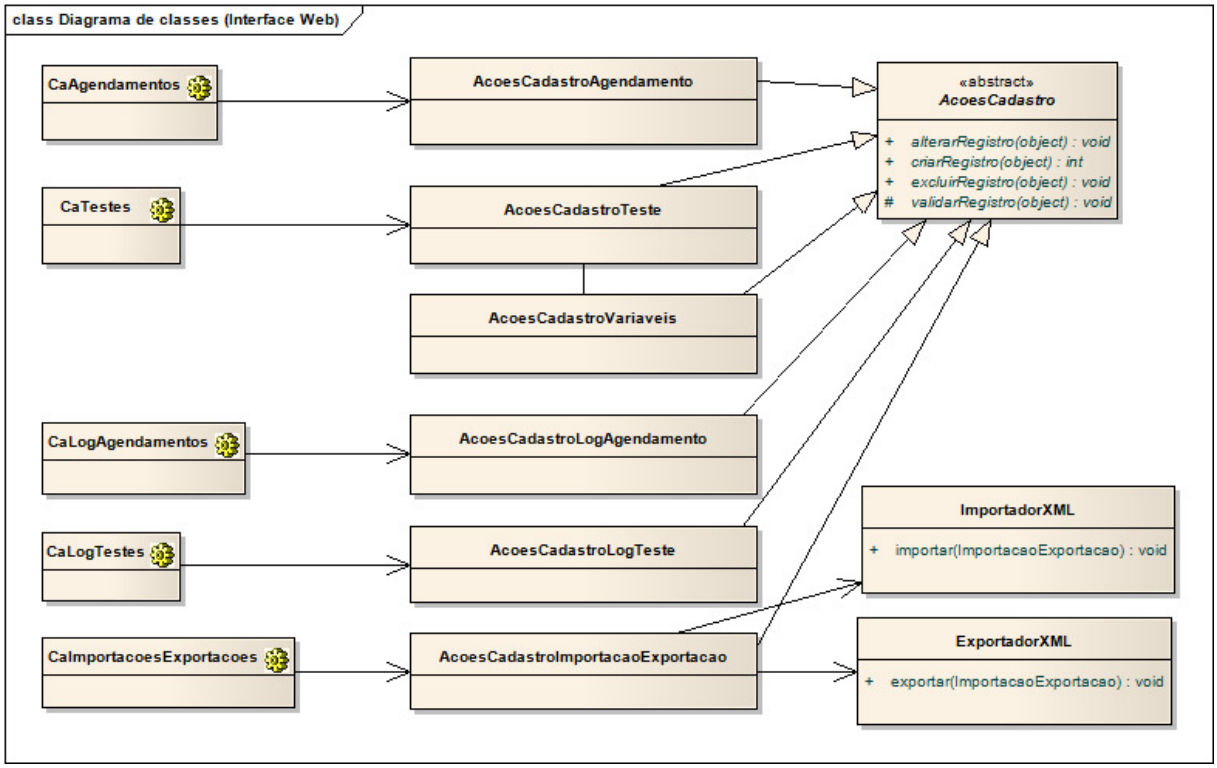


Figura 6 - Diagrama de classes interface Web

Segue o detalhamento das classes apresentadas no diagrama de classes interface Web da Figura 6:

a) classe `CaAgendamentos`: tela responsável pelo cadastro de agendamentos. A classe utiliza a classe `AcoesCadastroAgendamento` para realizar as operações de criação, alteração, exclusão e validações dos registros de agendamento;

b) classe `CaTestes`: tela responsável pelo cadastro de testes. A classe utiliza a classe `AcoesCadastroTeste` para realizar as operações de criação, alteração, exclusão e validações dos registros de teste;

c) classe `CaLogAgendamentos`: tela responsável pelo *log* de agendamentos. A classe utiliza a classe `AcoesCadastroLogAgendamento` para realizar as operações de criação, alteração e exclusão de registros de *log* de agendamentos;

d) classe `CaLogTestes`: tela responsável pelo *log* de testes. A classe utiliza a classe `AcoesCadastroLogTeste` para realizar as operações de criação, alteração e exclusão dos registros de *log* de testes;

e) classe `CaImportacoesExportacoes`: tela responsável pelo cadastro de importações e exportações de testes. A classe utiliza a classe `AcoesCadastroAgendamento` para realizar as operações de criação, alteração, exclusão e validações dos registros de importações e exportações de testes;

f) classe abstrata `AcoesCadastro`: responsável por definir as funções que serão

realizadas pelos cadastros, a classe não efetua nenhuma operação ela apenas define os métodos que deverão ser implementados pelas classes que lhe implementarem;

g) classe `AcoesCadastroAgendamento`: responsável pelas ações do cadastro de agendamentos. A classe realiza as operações de criação, alteração, exclusão e validações das informações da tela de cadastro dos agendamentos;
h) classe `AcoesCadastroTeste`: responsável pelas ações do cadastro de testes. A classe realiza as operações de criação, alteração, exclusão e validações das informações da tela de cadastro dos testes;
i) classe `AcoesCadastroVariaveis`: responsável pelas ações de controle das variáveis de resultado dos agendamento. A classe é utilizada após a execução de cada teste para criar as informações de resultados dos testes realizados. A classe realiza operações de criação, alteração e exclusão das variáveis de resultados dos testes;
j) classe `AcoesCadastroLogAgendamento`: responsável pelas ações do cadastro de *log* de agendamentos. A classe realiza as operações de criação e exclusão das informações da tela de cadastro de históricos dos agendamentos;
k) classe `AcoesCadastroLogTeste`: responsável pelas ações do cadastro de *log* de testes. A classe realiza as operações de criação e exclusão das informações da tela de cadastro de históricos dos testes;
l) classe `AcoesCadastroImportacaoExportacao`: responsável pela ações do cadastro de importações e exportações. A classe realiza as operações de criação, alteração, exclusão e validações das informações da tela de cadastro de importações e exportações. A classe utiliza as classes `ImportacaoXML` e `ExportacaoXML` para realizar a importação ou exportação dos testes;
m) classe `ImportacaoXML`: responsável pela importação dos testes. A classe possui o método `importar` que realiza importação dos testes, identificando os testes informados no arquivo XML e criando os testes vinculados ao agendamento informado na tela de importação;
n) classe `ExportacaoXML`: responsável pela geração dos arquivos de exportações dos testes. A classe possui o método `exportar` que realiza a exportação dos arquivos no formato XML contendo informações sobre os testes exportados.

3.2.3 Diagramas de sequência

Na especificação dos diagramas é representada a sequência de operações que ocorrem quando o testador realiza suas atividades. São apresentados três diagramas de sequência, um diagrama que representa a sequência de operações ao criar o agendamento na Figura 7, um diagrama que representa a criação de um teste na Figura 8 e um diagrama que representa a importação e exportação dos testes na Figura 9.

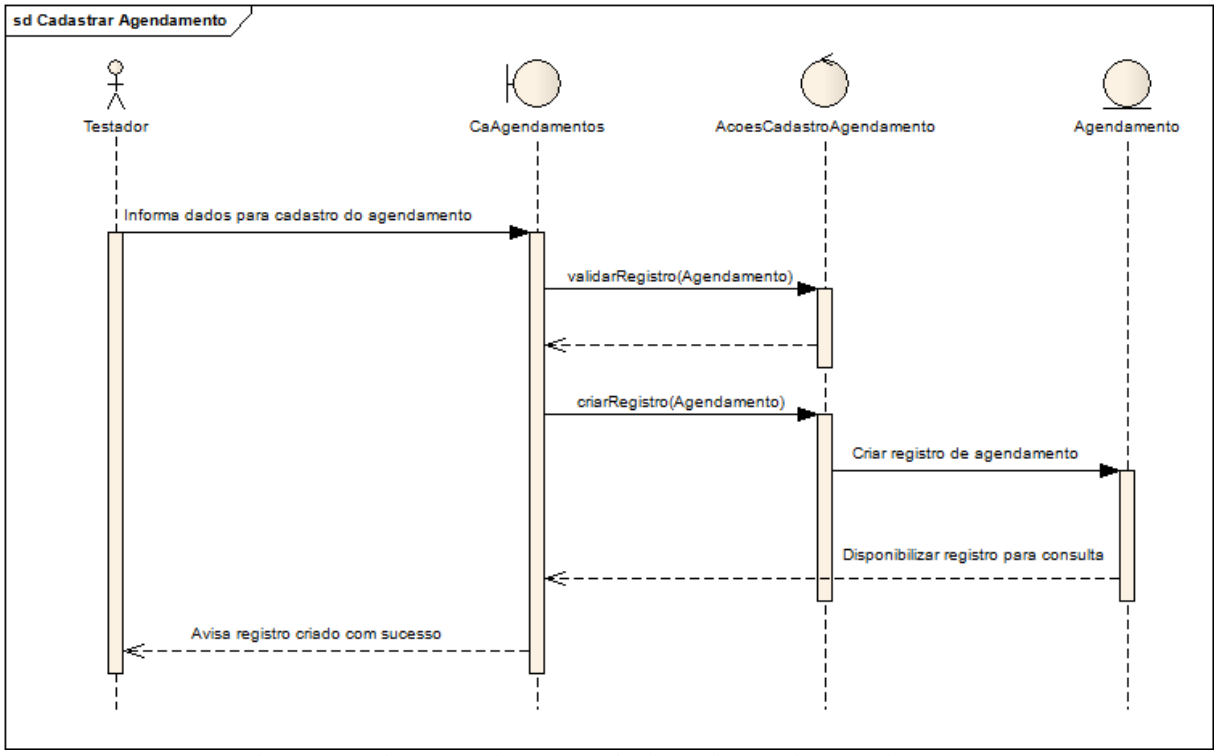


Figura 7 - Diagrama de sequência do cadastro do agendamento

Na Figura 7 é mostrado o processo realizado a partir do momento que o testador informa os dados do agendamento. Os dados são informados através da interface Web `CaAgendamentos` são enviados a classe `AcoesCadastroAgendamento` que responsável pelo controle das ações da tela e que utiliza a classe `Agendamento` para armazenar as dados informados pelo testador. Após a validação é criado o registro na base de dados e é alertado ao testador que o procedimento foi realizado com sucesso. A organização das classes é feita da maneira apresentada, pois está sendo utilizado o padrão de projeto *Façade*. O padrão busca facilitar a utilização de classes para isso trabalha criando classes de controle. Na Figura 7 a classe de controle é a `AcoesCadastroAgendamento`. A classe disponibiliza os comandos que serão utilizados na tela de cadastro dos agendamentos.

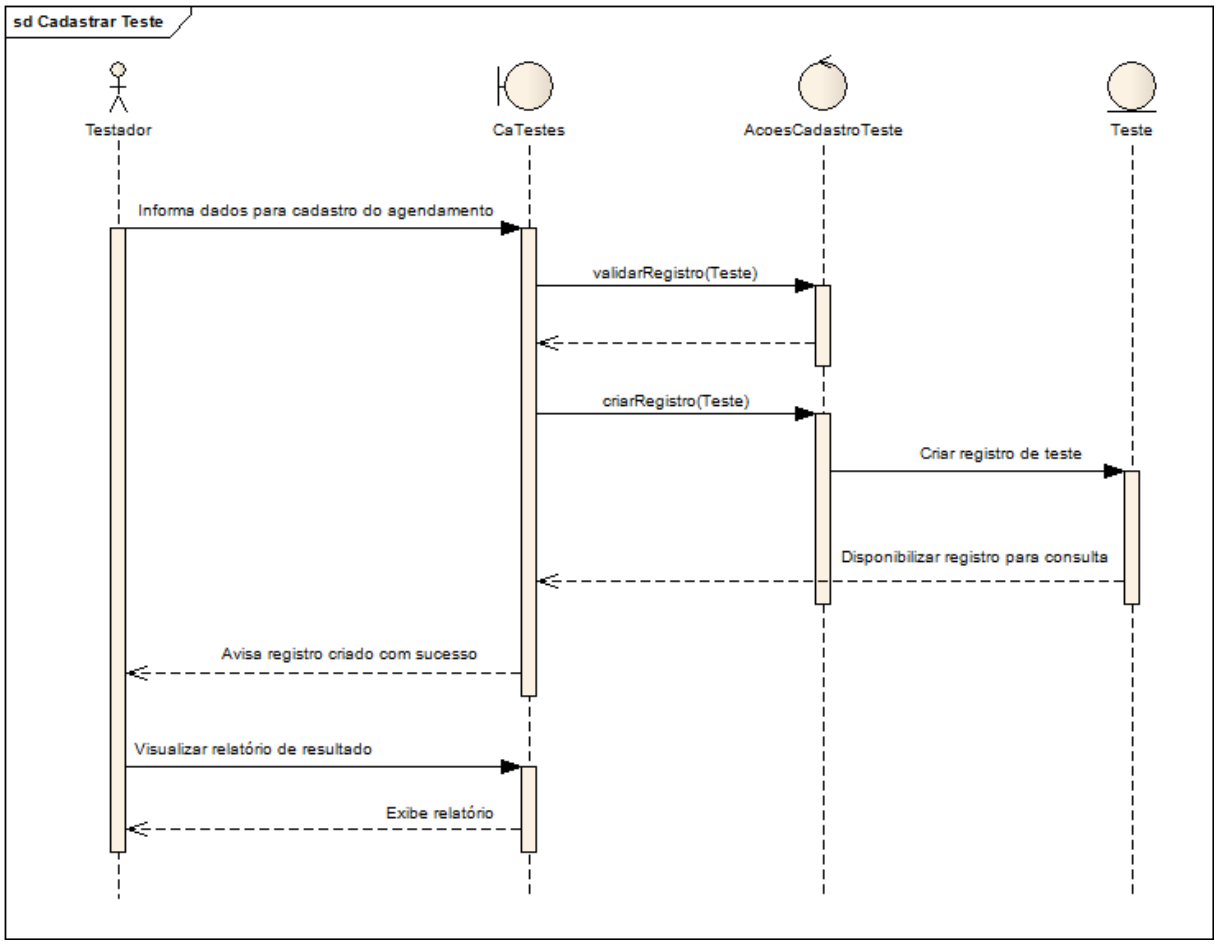


Figura 8 - Diagrama de sequência do cadastramento do teste

Na Figura 8 é mostrado a criação do teste através da tela `CaTestes` que utiliza uma classe de controle (`AcoesCadastroTeste`) assim como no cadastro do agendamento e utiliza também a classe base `Teste`. Após o cadastro do teste e execução do agendamento, é possível visualizar na tela o relatório com o resultado do teste realizado. No cadastro do teste, assim como em todas as telas do sistema, também é utilizado o padrão de projeto *Façade*.

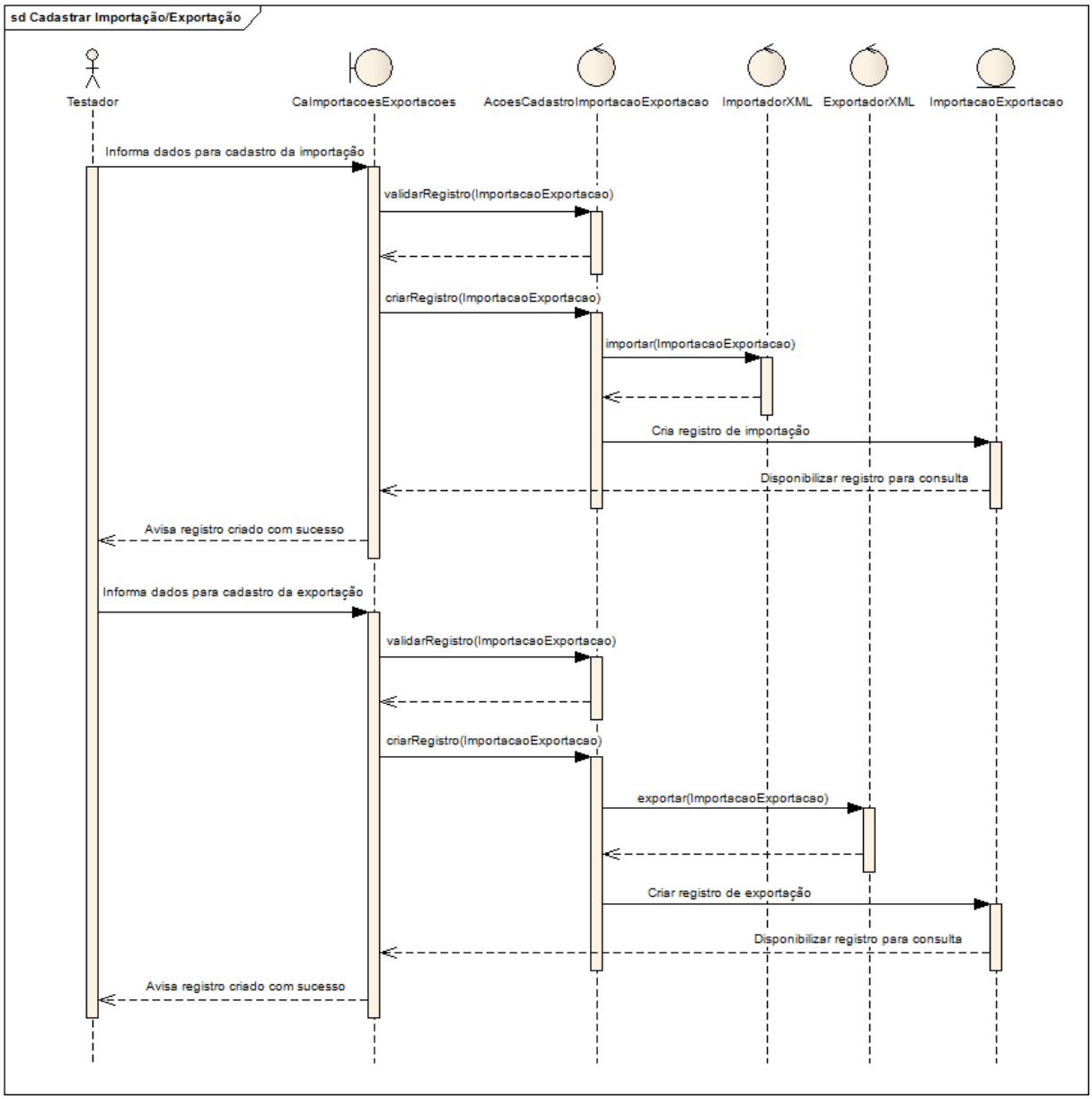


Figura 9 - Diagrama de sequência do cadastramento da importação/exportação de testes

Na Figura 9 é mostrado a criação da importação ou exportação através da tela `CaImportacoesExportacoes` que utiliza uma classe de controle (`AcoesCadastroImportacaoExportacao`) assim como no cadastro do teste e utiliza também a classe base `ImportacaoExportacao`. A tela permite que seja realizada uma das rotinas. Pode ser definida a realização da importação que utiliza a classe `ImportadorXML` para efetuar a importação, ou pode ser definida a realização da exportação que utiliza a classe `ExportadorXML` para executar a exportação.

3.2.4 Diagrama de entidade-relacionamento

No diagrama de entidade-relacionamento da Figura 10 são apresentadas as tabelas utilizadas na base de dados da ferramenta.

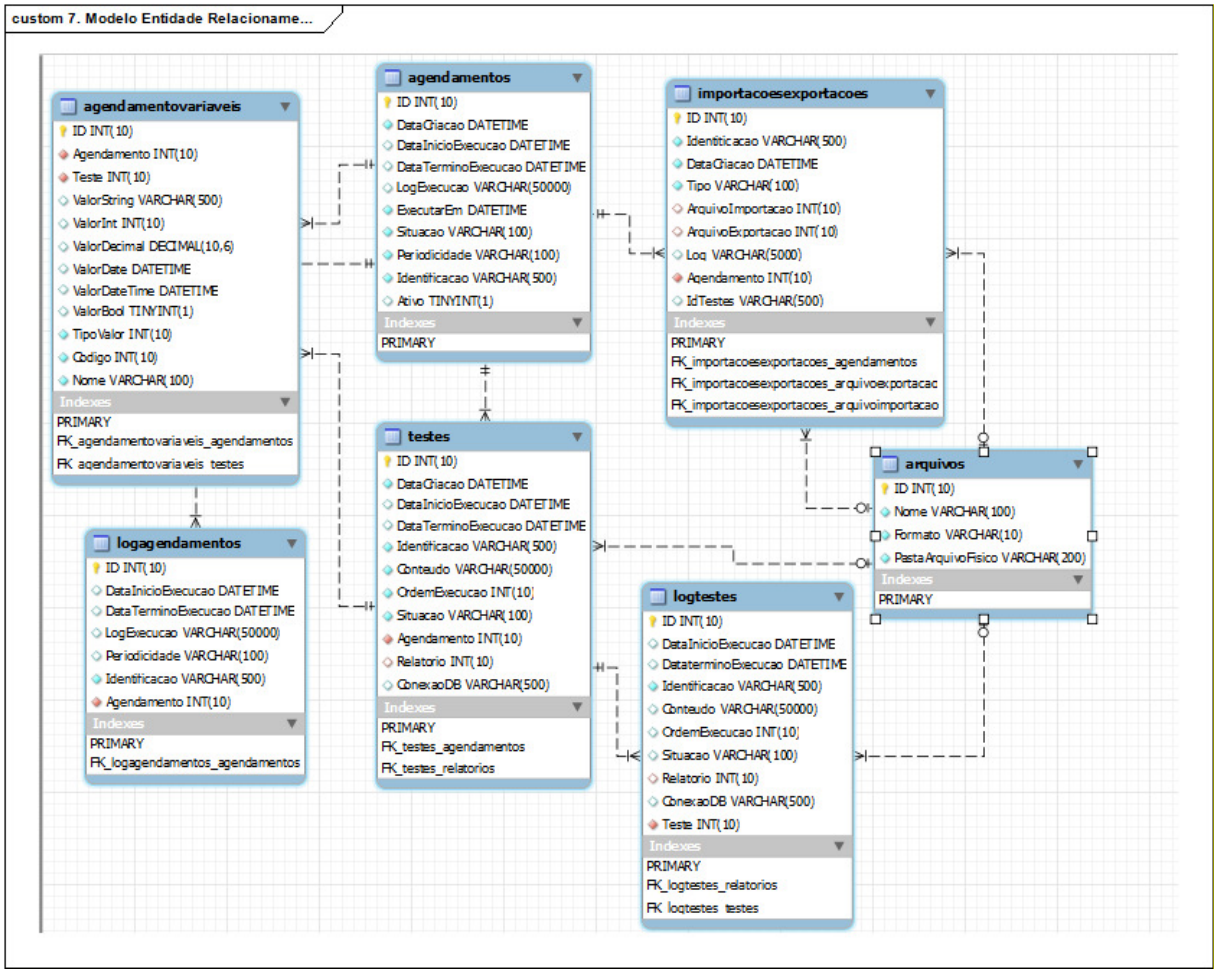


Figura 10 - Tabelas e seus relacionamentos

A seguir é descrito o detalhamento das tabelas apresentadas na Figura 10:

a) tabela `Agendamentos`: local onde são armazenadas as informações dos agendamentos do sistema;

b) tabela `Testes`: local onde são armazenadas as informações dos testes realizados no sistema. A tabela está associada à tabela de `Agendamentos`, os testes são criados somente após a criação do agendamento;

c) tabela `LogAgendamentos`: local onde são armazenadas os históricos das execuções dos agendamentos. A tabela está associada à tabela de `Agendamentos`, os históricos são criados somente após a execução dos agendamentos;

d) tabela `LogTestes`: local onde são armazenadas os históricos das execuções dos

testes dos agendamentos. A tabela está associada à tabela de `Testes`, os históricos são criados somente após a execução dos testes;

e) tabela `AgendamentoVariaveis`: local onde são armazenados os resultados da execução dos métodos e das fórmulas informadas nos testes dos agendamentos;
f) tabela `Arquivos`: local onde são armazenados todos os arquivos do sistema, que são relatórios, arquivos de importação e arquivos de exportação;
g) tabela `ImportacoesExportacoes`: local onde são armazenados os registros de importações e os registros de exportações do sistema.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas para implementação da ferramenta, assim como o processo de implementação.

3.3.1 Técnicas e ferramentas utilizadas

A ferramenta foi implementada na linguagem de programação C# utilizando ambiente de desenvolvimento Visual Studio 2010 com o Framework .Net 4.0. Foi utilizado o gerenciador de banco de dados MySQL 5. Foi utilizado a ferramenta GALS (GESSER, 2003) para a geração dos analisadores léxico e sintático na linguagem de programação Java. O código-fonte gerado para a linguagem Java foi convertido para a linguagem de programação C#. Foram utilizadas técnicas de compiladores (vide seção 2.2 da fundamentação teórica) e de reflexão (vide seção 2.3 da fundamentação teórica) para o desenvolvimento da ferramenta.

3.3.2 Operacionalidade da implementação

A operacionalidade da implementação está divida em três seções. Na seção 3.3.2.1 é mostrado como os testes podem ser definidos através da ferramenta. 3.3.2.2 é apresentada a ferramenta mostrando como o testador irá utilizá-la. Na seção 3.3.2.3 são apresentados alguns

trechos mais importantes de código-fonte utilizados na implementação da ferramenta.

3.3.2.1 Definição do conteúdo dos testes na ferramenta

Foi desenvolvida uma linguagem para definição do conteúdo dos testes que permite a realização de diversos testes. A linguagem é validada e interpretada pelos analisadores léxico, sintático e semântico (vide seção 2.2 da fundamentação teórica). O interpretador irá utilizar a notação matemática pré-fixada para a realização dos cálculos matemáticos (vide seção 2.4 da fundamentação teórica). A linguagem permite que sejam criadas variáveis do tipo `decimal`, `int`, `string`, `bool`, `date` e `datetime`. A declaração das variáveis é realizada conforme Quadro 12.

```
decimal info1 = 4.56;
int info2 = 3;
string info3 = "Texto";
bool info4 = true;
bool info5 = false;
date info6 = CreateDate("30/10/2011");
datetime info7 = CreateDateTime("30/10/2011 14:00:00");
```

Quadro 12 - Declaração de variáveis

Permite que sejam realizadas funções matemáticas para realizar o arredondamento de valores, função para achar o valor absoluto de valores, função para efetuar a raiz quadrada e para efetuar a exponenciação de valores, conforme Quadro 13.

```
decimal info1 = Round(10.341567, 2); #retorna o valor arredondado#
decimal info2 = Square(9); #retorna a raiz quadrada do valor#
decimal info3 = Absolute(8); #retorna o valor absoluto#
decimal info4 = Pow(3, 2); #retorna o resultado da exponenciação#
```

Quadro 13 - Funções matemáticas

O caractere `#` é utilizado para determinar o início e término do comentário. No exemplo do Quadro 13, o valor `10.341567` será arredondado com duas casas decimais e armazenado na variável `info1`. Será efetuada a raiz quadrada do valor `9` e armazenado na variável `info2`. Será encontrado o valor absoluto do valor `8` e armazenado na variável `info3`. Será elevado o valor `3` a potência `2` e armazenado na variável `info4`.

A linguagem permite que sejam realizadas consultas a um banco de dados conforme Quadro 14.

```
decimal info1 = SqlCommand("select valor from cobrancas where id = 10");
int info2 = SqlCommand("select Count(id) qtde from cobrancas");
```

Quadro 14 - Consulta a banco de dados

Os comandos são executados e têm seus resultados armazenados nas variáveis. Para efetuar as consultas é preciso informar o texto de conexão com o banco de dados. O texto de conexão deve ser informado ao criar o teste no campo “Conexão BD”, conforme Figura 11.

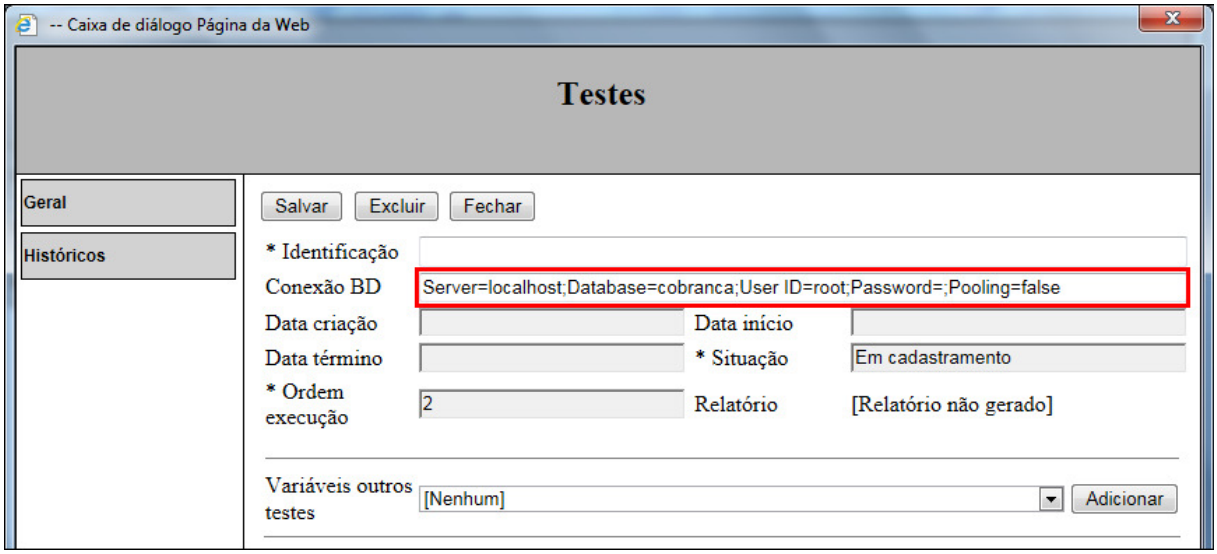


Figura 11 - Preenchimento do texto de conexão

A linguagem permite que sejam realizados cálculos matemáticos com diversos operandos e operadores, conforme Quadro 15.

```
decimal info1 = 10.45;
int info2 = 5;
decimal info3 = ((info1 * 4)/(info2 + 7.5) - info1);
```

Quadro 15 - Operações matemáticas complexas

Permite que sejam carregados métodos das classes definidos em DLLs, conforme Quadro 16.

```
DefineMethod
[
Dll: "C:/CalculadorMatematico.dll"
Class: "CalculadorMatematico.OperacaoMatematica" ("Multiplicação")
Method: "multiplicar" (3.1, 2.0)
Static: false
Signature: "multiplicarValores"
];
```

Quadro 16 - Definição de método de classe

Para a declaração do método de classe é necessário informar as seguintes informações, a DLL `C:/CalculadorMatematimo.dll` a qual a classe pertence, a classe e seus parâmetros (`“CalculadorMatematico.OperacaoMatematica” (“Multiplicação”)`), o método e seus parâmetros (`“multiplicar” (3.1, 2.0)`), se o método é estático ou não (`false`) e a assinatura `multiplicarValores` que é utilizada para identificar o método no momento de executá-lo.

A linguagem permite a execução dos métodos de classes conforme Quadro 17.

```
decimal info1 = ExecuteMethod("multiplicarValores");
```

Quadro 17 - Execução de método de classe

O comando `ExecuteMethod` executa o método de classe utilizando a assinatura definida e armazena o resultado da execução na variável `info1`.

A linguagem possui três palavras reservadas que precisam obrigatoriamente ser informadas, que são "`CheckMethod`", "`CheckFormula`" e "`Test`". Elas são responsáveis por definir o que deverá ser comparado na execução do teste, conforme Quadro 18.

```
CheckMethod: ExecuteMethod("multiplicarValores");
CheckFormula: ((3.0 * 4.0));
Test: true;
```

Quadro 18 - Informações para comparação

O comando `CheckMethod` serve para definir qual método de classe deverá ser executado na comparação. O comando `CheckFormula` serve para definir qual fórmula matemática deverá ser executada na comparação. O comando `Test` serve para determinar se o teste deverá ser resultado ou não. Caso o teste seja definido para não resultar não é gerado relatório de execução. Isto pode ser utilizado para criar testes que serão utilizados apenas como base para outros testes e estes sim irão listar o resultado.

3.3.2.2 Utilização da ferramenta

Ao acessar a ferramenta é apresentada a tela inicial que exibe um menu com as opções "Agendamentos" e "Importações/Exportações", conforme (Figura 12 – item A). Por padrão é exibida a tela de agendamentos que mostra uma lista com os agendamentos já cadastrados na (Figura 12 – item B).

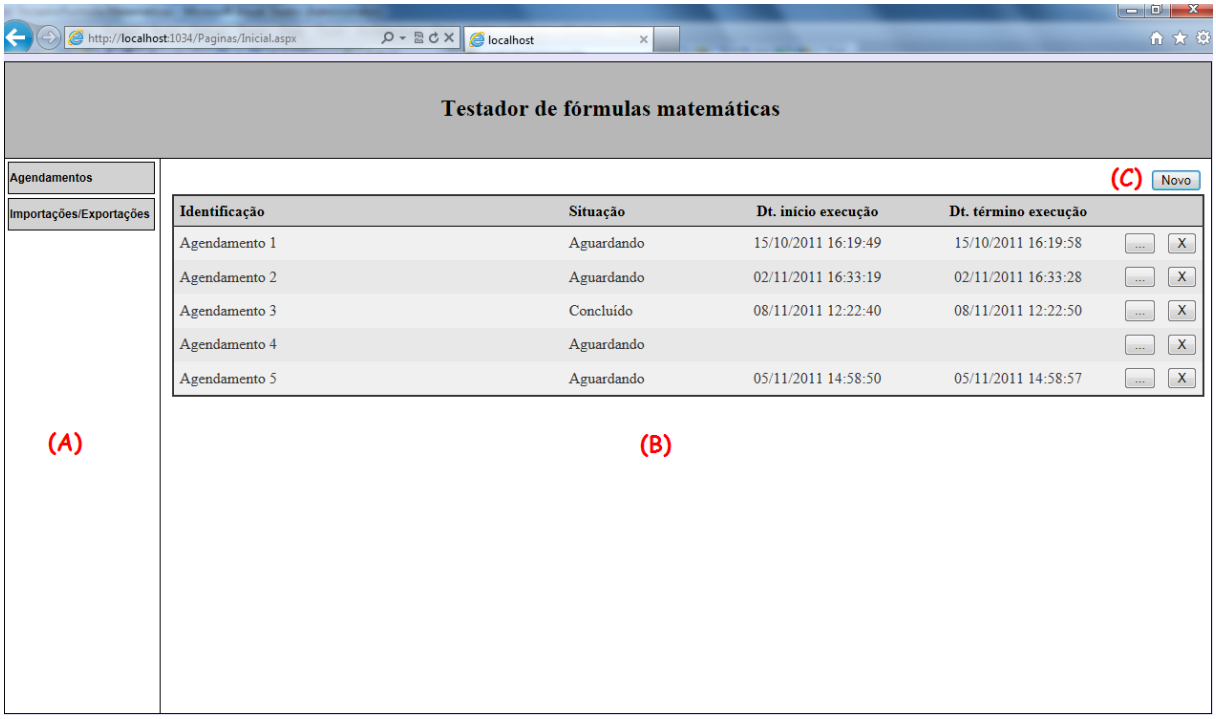


Figura 12 - Tela inicial da ferramenta

Para que seja efetuado um teste é preciso definir quando deverá ser executado e com que periodicidade. Para isso foi definido o cadastro de agendamento dos testes. Para criar um novo teste é preciso clicar no botão “Novo” (Figura 12 – item C), botão que está localizado acima da lista de agendamentos já criados. Ao clicar no botão é carregada a tela de cadastro do agendamento, verificar na (Figura 13).

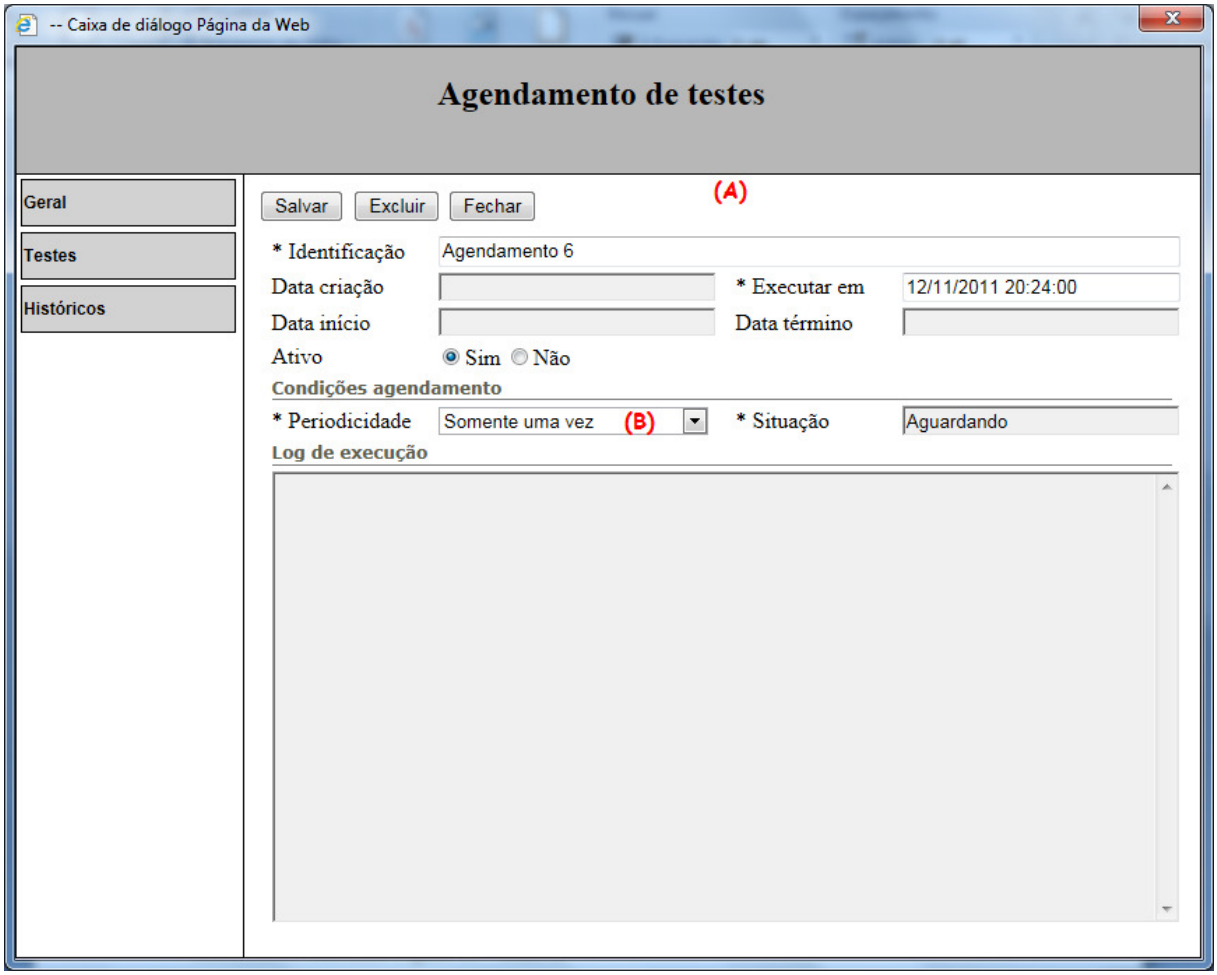


Figura 13 - Tela de cadastro de agendamento

Na tela do cadastro de agendamento é preciso informar uma identificação, a data em que os testes do agendamento serão executados, se o agendamento está ativo ou não e a periodicidade que os testes serão executados (Figura 13 – item A).

A data definida para execução deve ser posterior à data atual, caso contrário o agendamento não será executado. O campo “Ativo” determina se o agendamento pode ser executado ou não.

No campo de periodicidade (Figura 13 – item B) é possível selecionar as opções (Somente uma vez; Diariamente; Semanalmente; Mensalmente; Anualmente). Caso seja definida a opção “Somente uma vez”, após a execução dos testes o agendamento será concluído. Caso seja definida uma das outras opções, após a execução dos testes o agendamento é atualizado para a próxima data de execução. Por exemplo, se for utilizada a opção “Diariamente” e o agendamento for executado hoje, após a execução o agendamento será atualizado para a data de amanhã.

Após o cadastro do agendamento são criados os testes que serão executados no agendamento. Para criar um teste é preciso acessar a opção “Testes” no menu e clicar no

botão “Novo”. Ao clicar no botão é carregada a tela de cadastro do teste conforme Figura 14.

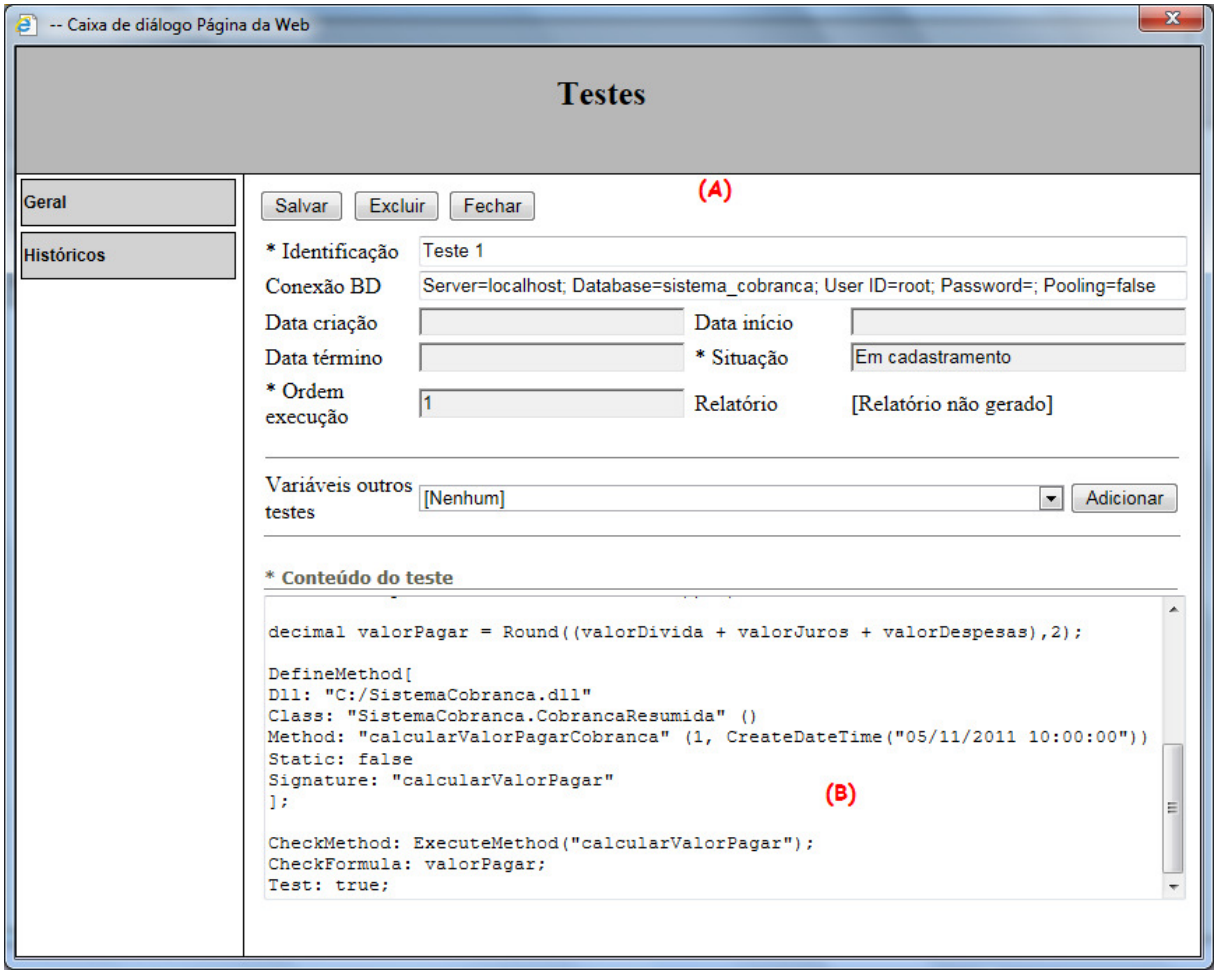


Figura 14 - Tela de cadastro de teste

Na tela de cadastro de teste é preciso informar a identificação e o conteúdo do teste. A linguagem utilizada para definição do conteúdo do teste permite que sejam feitas consultas em banco de dados. Caso seja utilizada uma consulta, é preciso preencher o campo de conexão informando o texto de conexão com o banco de dados que será feita a consulta.

No conteúdo (Figura 14 – item B) do teste criado é realizado o cálculo do valor a pagar de dívida de uma cobrança. Para a realização do cálculo são considerados os valores de juros e de despesas. A partir do conteúdo definido para o teste será efetuada a comparação entre o resultado da operação `(valorDivida + valorJuros + valorDespesas)` e o resultado da execução do método `“calcularValorPagarCobranca”` utilizando os parâmetros `(1, CreateDateTime(“05/11/2011 10:00:00”))`, que servem para identificar o registro de cobrança que será calculado e a data que o mesmo está sendo pago.

O campo de ordem de execução é utilizado para definir a ordem que os testes do agendamento deverão ser executados (Figura 14 – item A). A ordem permite que um teste utilize o resultado de outro teste do mesmo agendamento com ordem de execução inferior. A

utilização pode ser feita através do campo de variáveis outros testes, conforme Figura 15 – item A. No campo são carregadas apenas as variáveis de resultado dos testes anteriores, resultado de execução do método e variável de execução da fórmula. É preciso selecionar a variável que será utilizada e clicar no botão “Adicionar”, então a variável é adicionada ao conteúdo do teste.

Caso seja criado um segundo teste para o agendamento, são carregadas as variáveis de resultado do teste 1, conforme Figura 15 – item A.

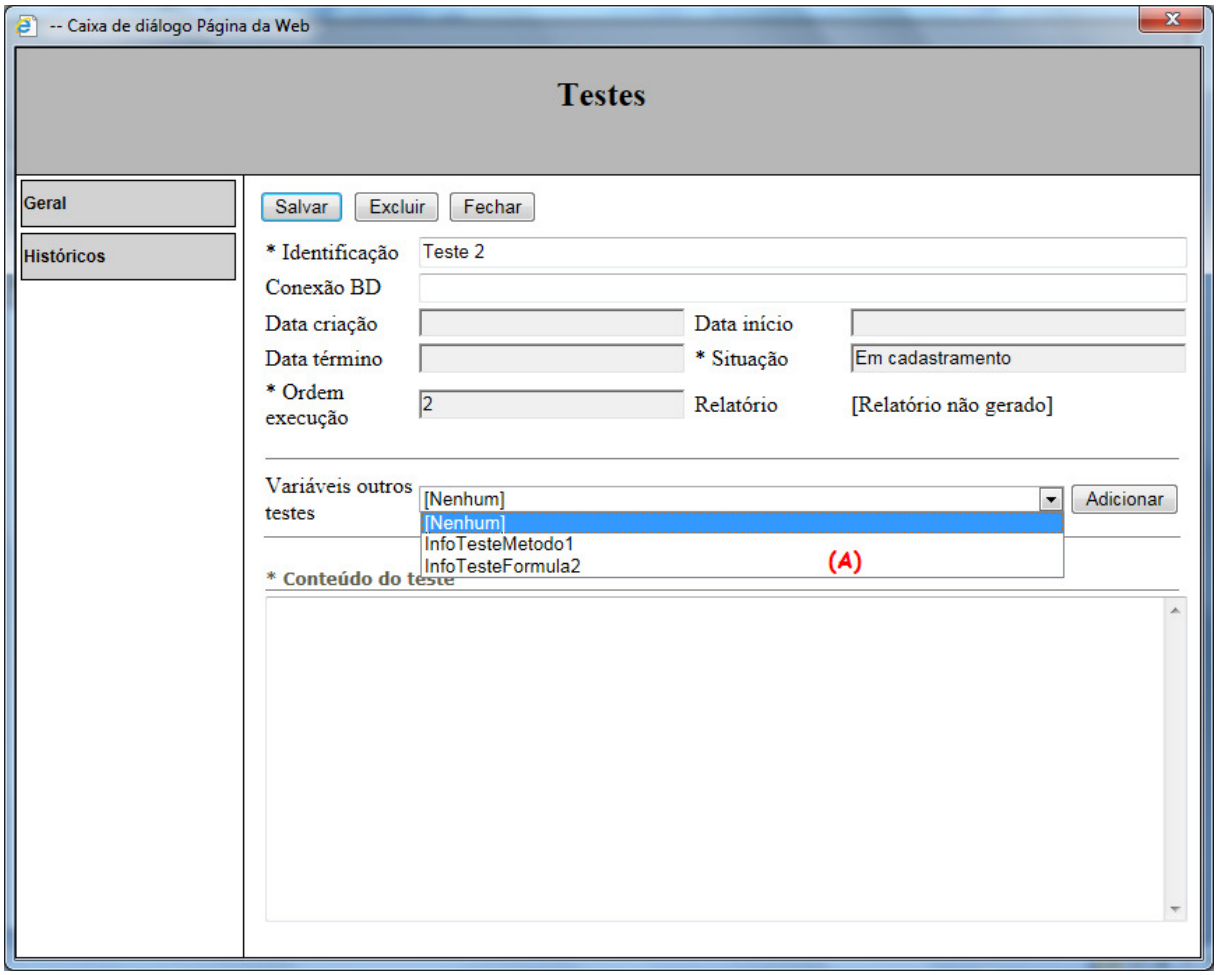


Figura 15 - Utilização de variáveis de outros testes

Após o cadastro dos testes no agendamento, no momento que for atingida a data de execução definida no agendamento, os testes são efetuados.

Os agendamentos de testes são identificados e executados através de um aplicativo que monitora o cadastro de agendamentos. O aplicativo executa independente da aplicação Web, ele fica monitorando constantemente a cada sessenta segundos. Com isso mesmo que a conexão com a internet seja interrompida, o aplicativo pode continuar executando. Outra vantagem é que podem ser definidas execuções para horários em que o servidor está sendo menos utilizado.

No exemplo foi criado um teste em um agendamento que foi executado na data "12/11/2011 20:24:00". Na Figura 16 é exibido o aplicativo de monitoramento executando o teste definido.

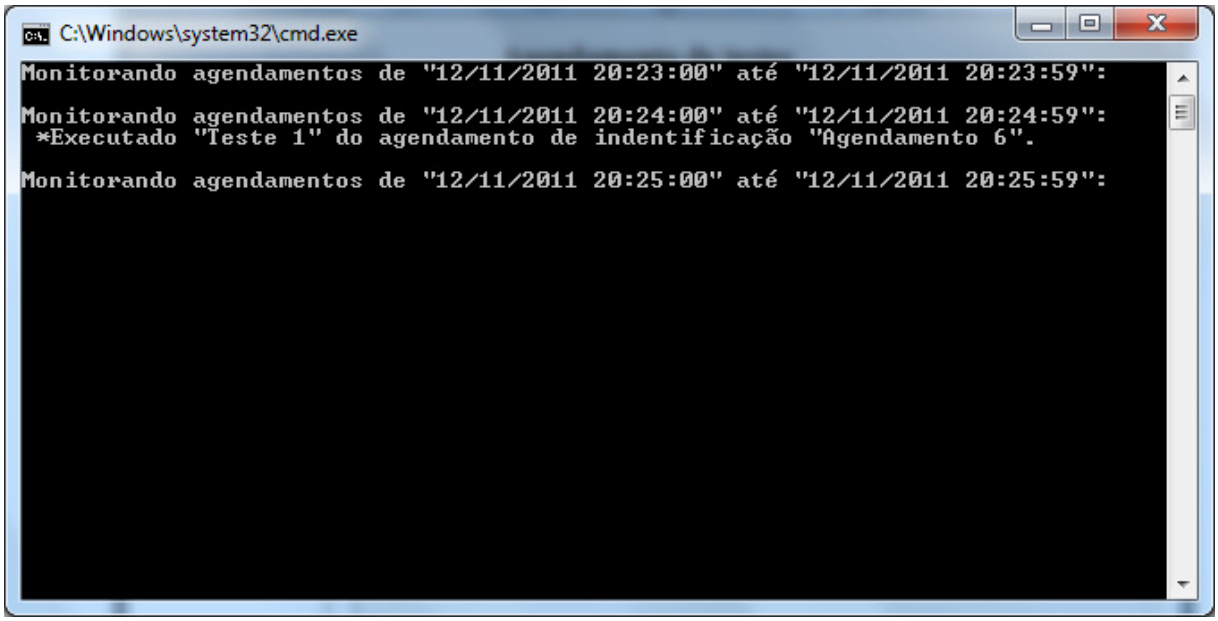


Figura 16 - Execução do aplicativo de monitoramento

Após a execução do agendamento e seus testes, são realizados os seguintes procedimentos:

a) é atualizado o cadastro do agendamento registrando a data de início de execução, a data de término de execução e o *log* de execução que irá identificar o que ocorreu na execução;
b) é atualizado também no cadastro do agendamento as informações referentes a próxima execução do agendamento, caso possua uma próxima execução. Neste caso como foi definido que o agendamento deveria executar somente uma vez, o agendamento é concluído após a execução;
c) é atualizado o cadastro do teste registrando a data de início de execução, a data de término da execução e a situação do teste;
d) é disponibilizado no cadastro do teste o relatório com o resultado da execução;
e) é gerado um registro de histórico para registrar as informações do agendamento executado;
f) é gerado um registro de histórico para cada teste executado, registrando as informações do teste realizado.

O relatório com o resultado da execução do teste pode ser visualizado através do cadastro de teste. O relatório está disponível através do *link* "*Download* do arquivo" (Figura 17 – item A). Para visualizá-lo é preciso clicar no *link* e efetuar o *download* do arquivo.

Figura 17 - Acesso ao relatório de resultado do teste

O relatório apresenta informações sobre o teste efetuado (Figura 18 – item A), sobre o agendamento que executou o teste (Figura 18 – item B) e sobre a comparação de resultados efetuada no teste (Figura 18 – item C). Nas informações sobre a comparação são exibidas à fórmula e o método que foram definidos para serem comparados e o resultado da execução da fórmula e do método, o item D da Figura 18 mostra se o resultado da execução é igual ou diferente do esperado.

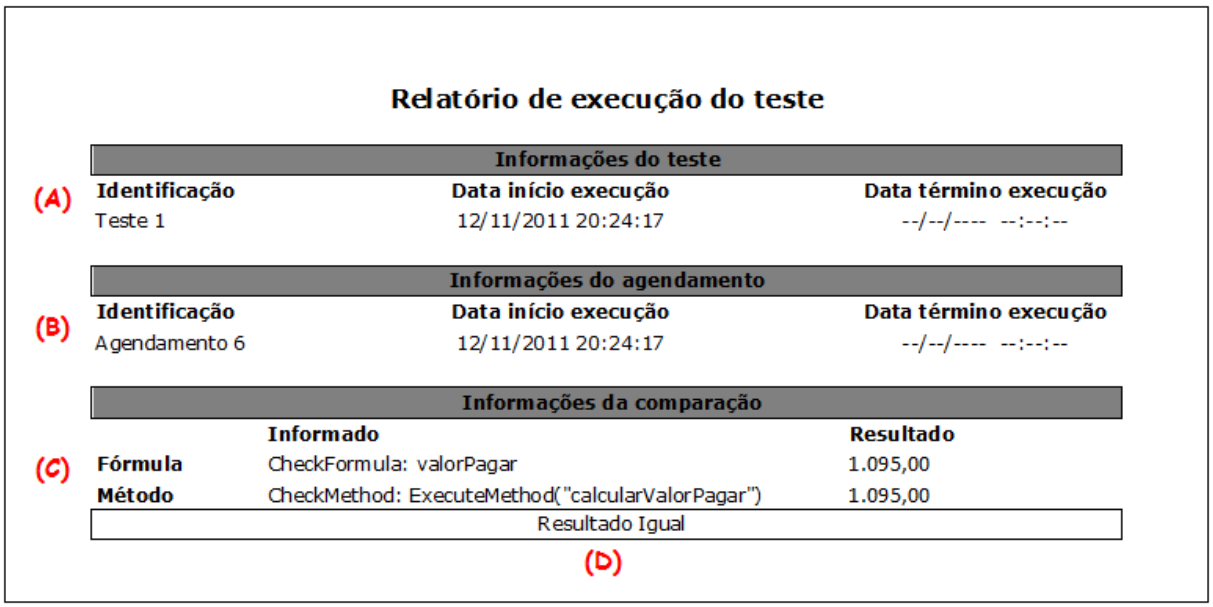


Figura 18 - relatório de resultado do teste

O histórico de agendamento pode ser visualizado através da opção “Históricos” no menu do cadastro de agendamento, (Figura 19 – item A). Ao clicar na opção é exibida uma lista de históricos gerados, neste caso a lista possuirá apenas um registro. Para acessá-lo é preciso clicar no botão “...” (Figura 19 – item B) que está disponível ao lado das informações do registro. Ao clicar no botão é carregada a tela de histórico do agendamento, conforme

Figura 20.

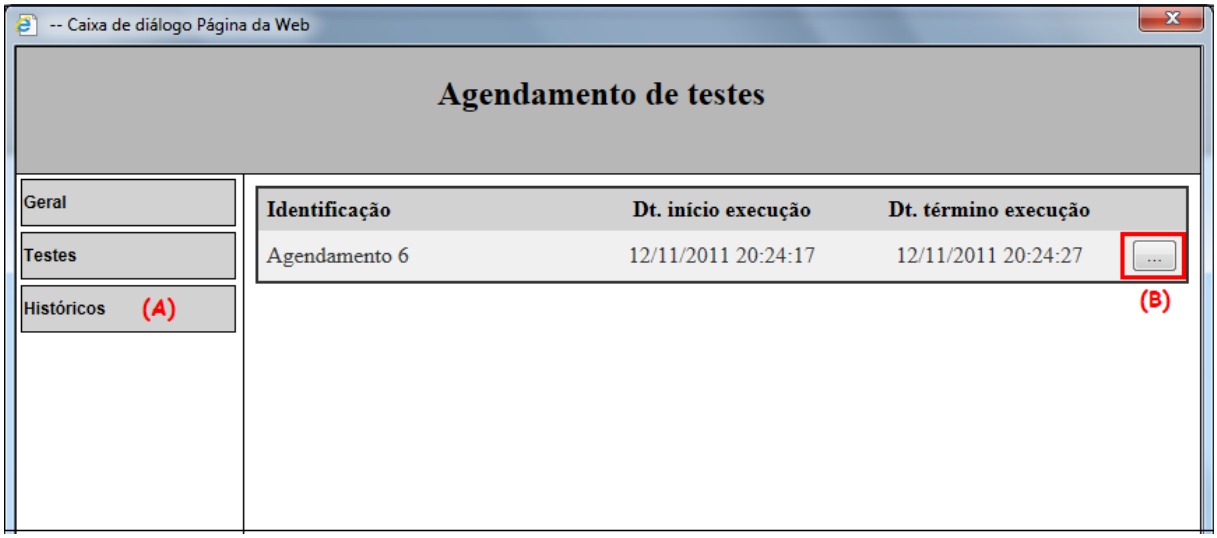


Figura 19 – Grid de históricos do agendamento

O histórico armazena as informações atuais do agendamento para que caso o agendamento seja executado novamente possam ser comparadas as execuções.

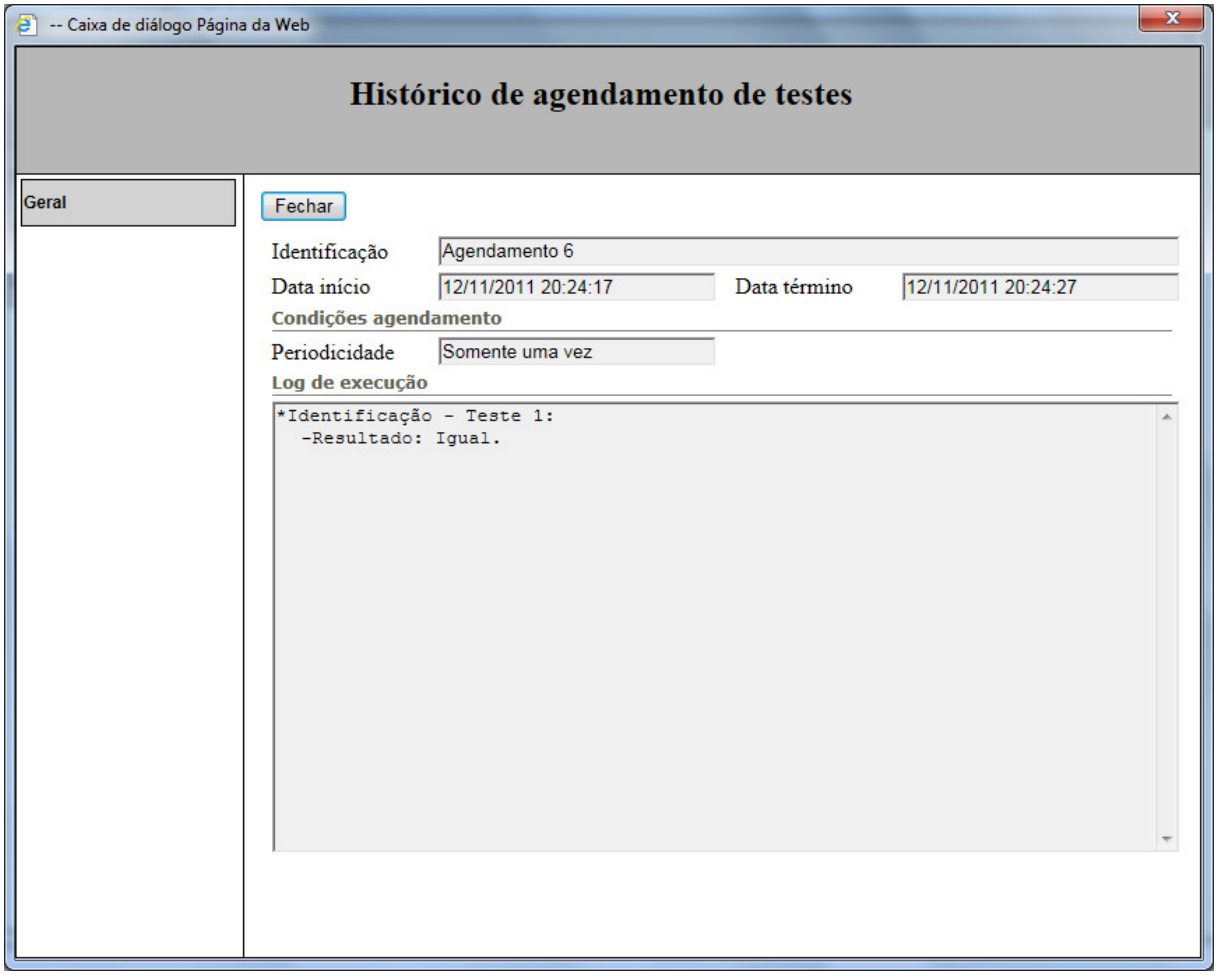


Figura 20 - Tela de histórico de agendamento

Além dos históricos de execução do agendamento são armazenados também os históricos de execução dos testes. O histórico de teste pode ser visualizado através da opção

“Históricos” no menu do cadastro de teste (Figura 21 – item A). Para acessar o registro do histórico é preciso proceder da mesma maneira que foi acessado o histórico do agendamento, é preciso clicar no botão “...” (Figura 21 – item B) que está disponível ao lado das informações do registro. Ao clicar no botão é carregada a tela de histórico do teste, conforme Figura 22.

Figura 21 - Grid de históricos do teste

O histórico (Figura 22) armazena as informações atuais do teste para que caso o teste seja executado novamente possam ser comparadas as execuções. Por exemplo, caso seja identificada a necessidade de modificar o conteúdo do teste, a modificação pode ser comparada utilizando o histórico de testes.

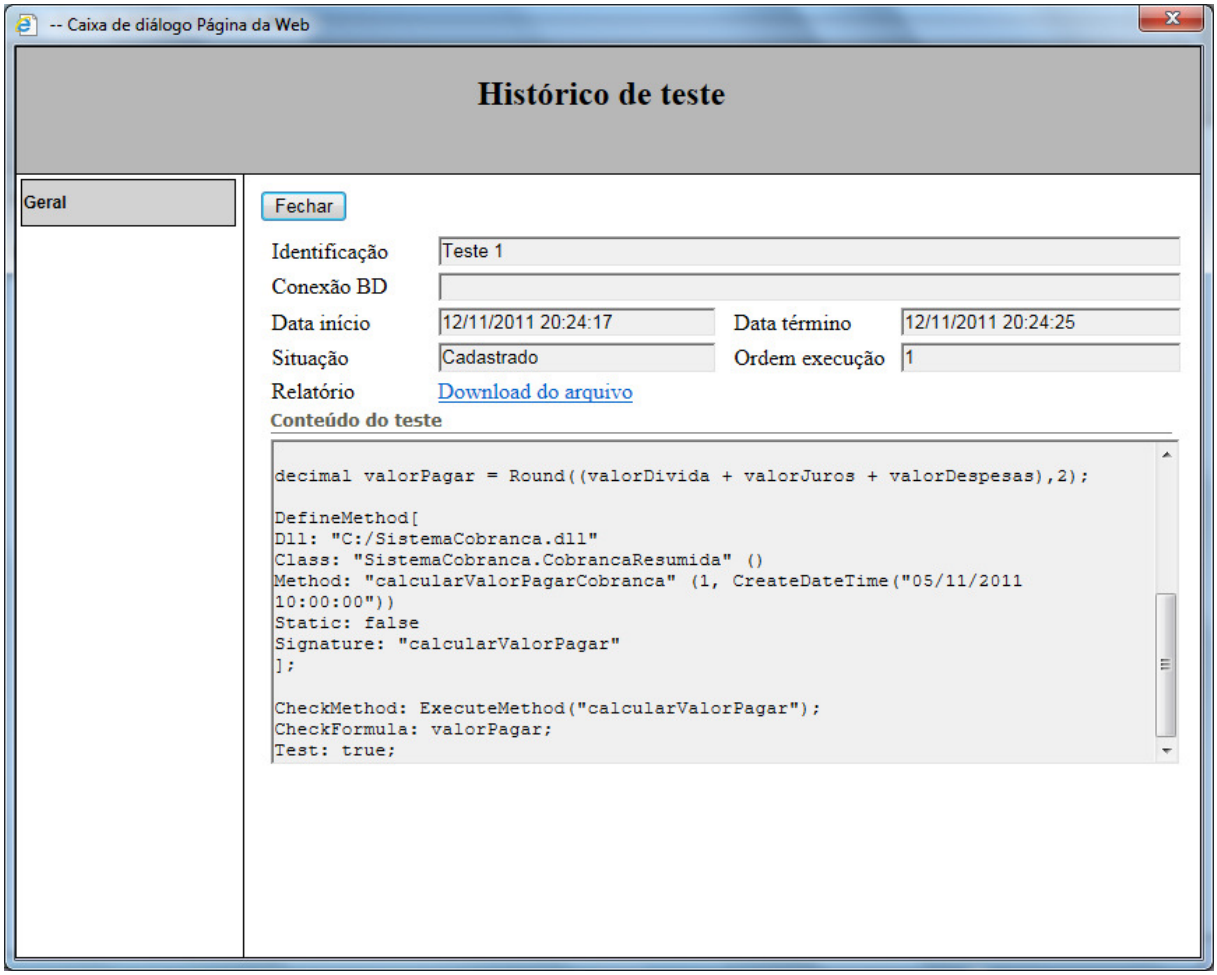


Figura 22 - Tela de histórico de teste

A ferramenta permite que sejam efetuadas exportações e importações de testes para os agendamentos. Para utilizar as funcionalidades é preciso acessar tela inicial da ferramenta, e acessar a opção "Importações/Exportações" do menu conforme (Figura 23 – item A). A opção irá exibir uma lista de importações e exportações já realizadas. Para criar uma nova importação ou exportação é preciso clicar no botão "Novo" (Figura 23 – item B) que está localizado acima da lista de registros. Ao clicar no botão é carregada a tela de cadastro de importações/exportações, conforme Figura 24.

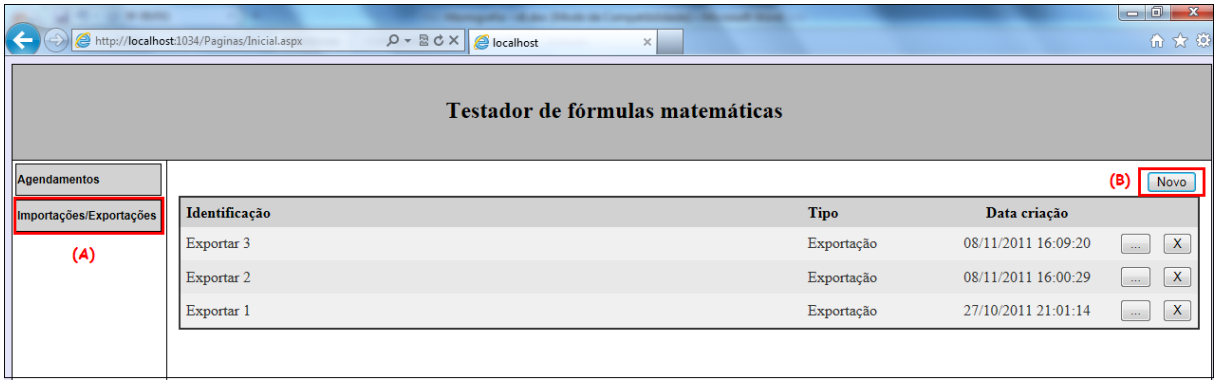


Figura 23 - Grid de importações e exportações

No cadastro de importações e exportações é preciso informar uma identificação, o tipo de operação que deverá ser realizada e o endereço físico do arquivo conforme Figura 24. Caso seja selecionado o tipo como exportação é habilitado para seleção o agendamento e os testes que devem ser exportados. Caso seja selecionado o tipo como importação é habilitado para seleção apenas o agendamento para onde os testes devem ser importados.

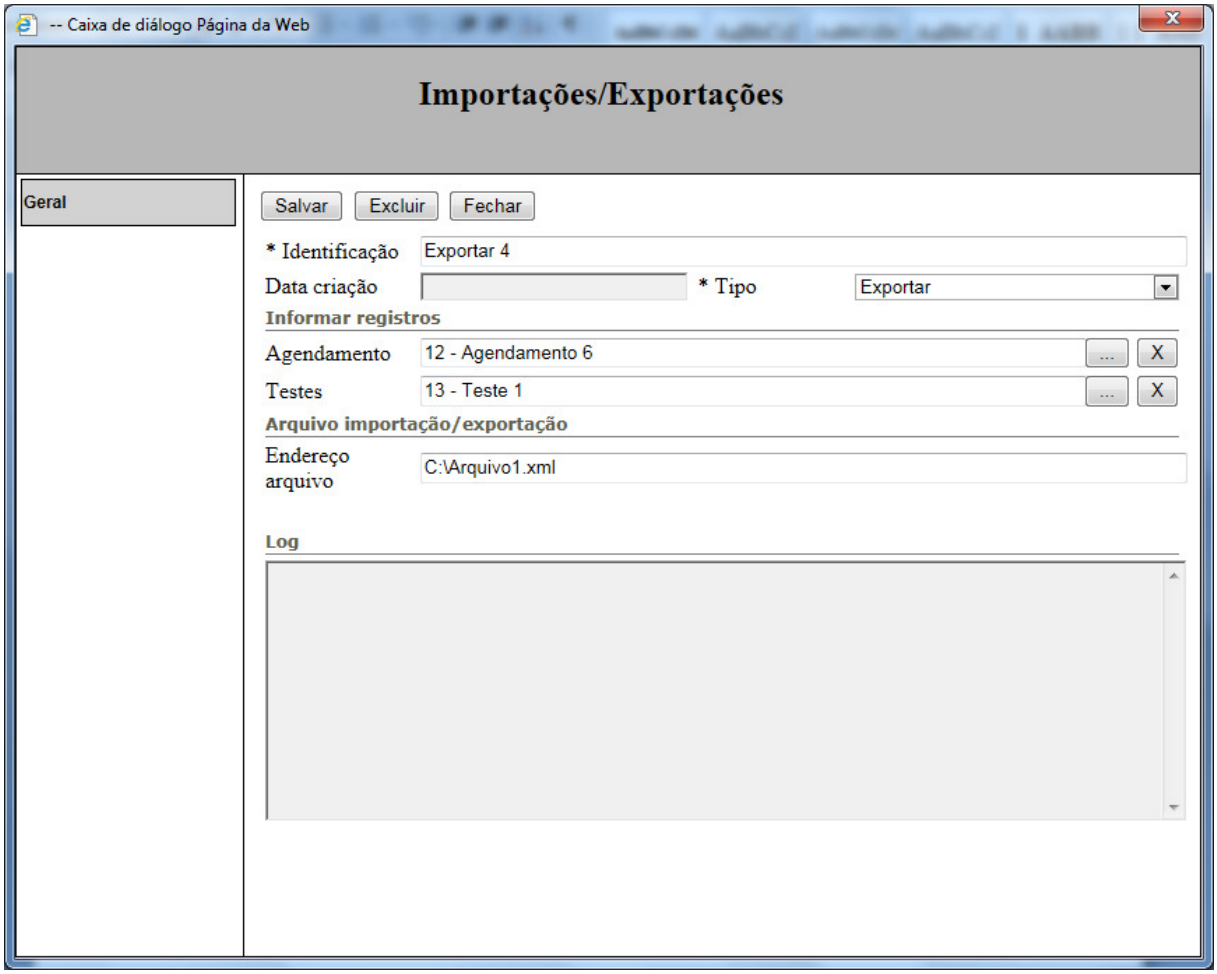


Figura 24 - Exportação de teste

A funcionalidade permite que um teste possa, por exemplo, ser exportado de um agendamento e importado em outro.

No exemplo é exportado o teste criado para o arquivo "`C:\Arquivo1.xml`" gerando um arquivo XML com a as informações do teste, conforme Quadro 19.

```
<?xml version="1.0" encoding="UTF-8"?>
<testes>
  <teste>
    <dataCriacao>12/11/2011 20:22:47</dataCriacao>
    <identificacao>Teste 1</identificacao>
    <conteudo>
      decimal valorDivida = SqlCommand("select valordivida from cobrancas where id = 1;");

      int diasAtraso = SqlCommand("select datediff(|2011-11-05 10:00:00|, datavencimento) from cobrancas where id = 1;");
      decimal pctJuros = Round(SqlCommand("select pctjuros from cobrancas where id = 1;"), 4);
      int mesesAtraso = diasAtraso / 30;
      decimal valorJuros = Round((valorDivida * ((pctJuros * mesesAtraso) / 100.00)),2);

      decimal valorDespesas = Round(SqlCommand("select sum(valor) from cobrancadespesas where cobranca = 1;"), 2);

      decimal valorPagar = Round((valorDivida + valorJuros + valorDespesas),2);

      DefineMethod[
      Dll: "C:/SistemaCobranca.dll"
      Class: "SistemaCobranca.CobrancaResumida" ()
      Method: "calcularValorPagarCobranca" (1, CreateDateTime("05/11/2011 10:00:00"))
      Static: false
      Signature: "calcularValorPagar"
      ];

      CheckMethod: ExecuteMethod("calcularValorPagar");
      CheckFormula: valorPagar;
      Test: true;
    </conteudo>
    <conexaoDB>Server=localhost; Database=sistema_cobranca; User ID=root; Password=; Pooling=false</conexaoDB>
  </teste>
</testes>
```

Quadro 19 - Arquivo XML gerado na exportação

Após a exportação é criado um outro registro com o tipo “Importar” que efetua a importação do teste que foi exportado para o arquivo XML. A importação é realizada para um agendamento que já estava criado no sistema, conforme Figura 25.

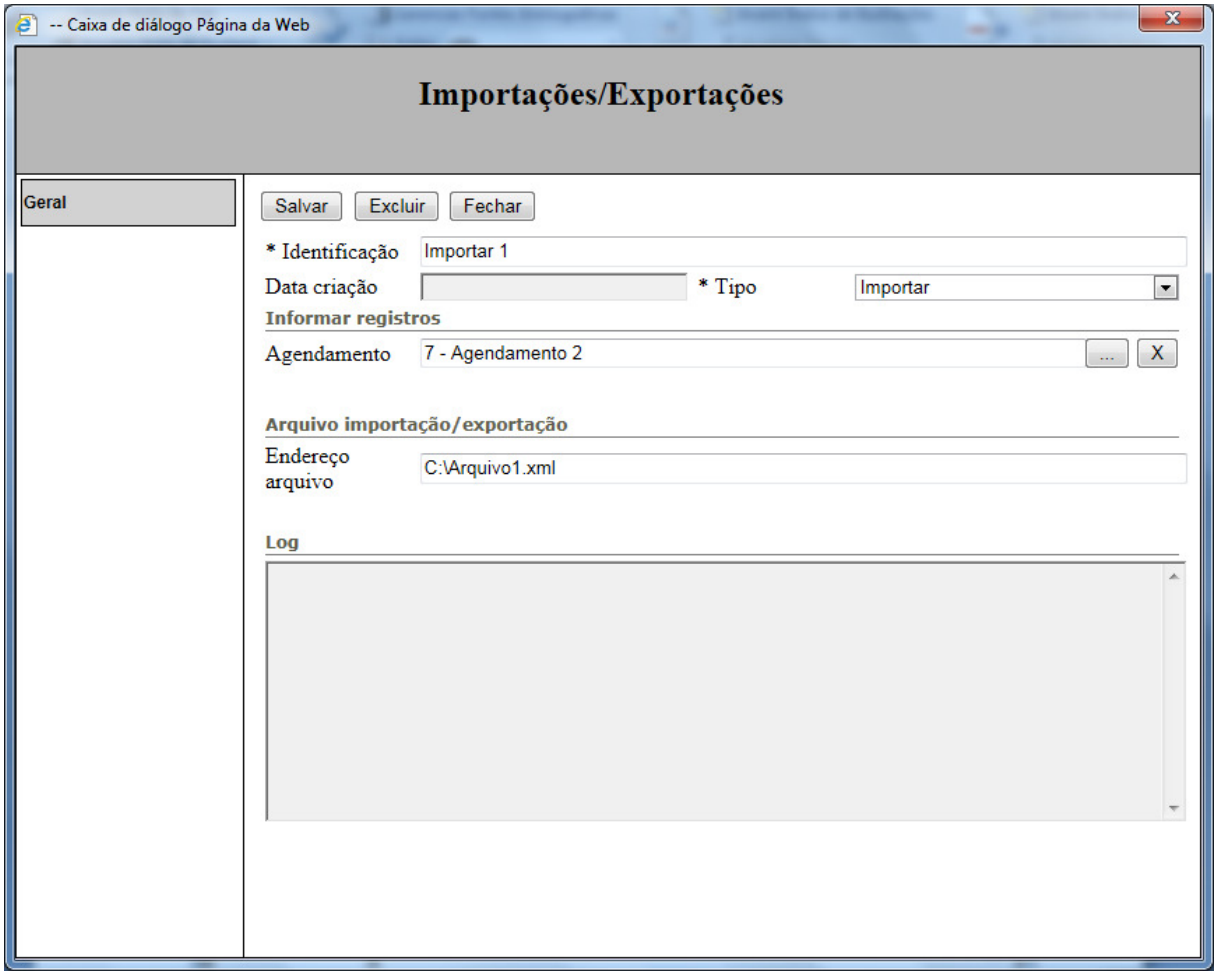


Figura 25 - Importação de testes

Na importação não é disponibilizado o campo para informação dos testes, pois a importação será de todos os testes que estiverem no arquivo informado.

3.3.2.3 Realização da implementação

Para realizar a execução dos agendamentos de teste foram criadas duas classes (`AgendadorTestes` e `ExecutorAgendamento`) que identificam e iniciam a execução dos testes do agendamento.

A classe `AgendadorTestes` é uma *Thread*[1] que executa a cada sessenta segundos verificando se existe algum agendamento em condições de execução, conforme Quadro 20.

[1] *Thread*: "threads são fluxos de execução que rodam dentro de um processo (aplicação)." (CESTA, 1996).

```
11  public class AgendadorTestes
12  {
13      private CarregarAgendamento carregadorAgendamento;
14
15      public AgendadorTestes()
16      {
17          this.carregadorAgendamento = new CarregarAgendamento();
18      }
19
20      public void executar()
21      {
22          bool pularLinha = false;
23          while (true)
24          {
25              DateTime dataInicial = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day,
26                  DateTime.Now.Hour, DateTime.Now.Minute, 0);
27              DateTime dataFinal = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day,
28                  DateTime.Now.Hour, DateTime.Now.Minute, 59);
29              List<Agendamento> agendamentos = carregadorAgendamento.carregarAgendamentosData(dataInicial, dataFinal);
30              Console.WriteLine("{0}Monitorando agendamentos de \"{1}\" até \"{2}\":", (pularLinha?"\r\n":""),
31                  dataInicial.ToString("dd/MM/yyyy HH:mm:ss"), dataFinal.ToString("dd/MM/yyyy HH:mm:ss"));
32
33              foreach (Agendamento agendamento in agendamentos)
34              {
35                  ExecutorAgendamento executorAgendamento = new ExecutorAgendamento(agendamento);
36                  Thread thread = new Thread(new ThreadStart(executorAgendamento.executar));
37                  thread.Start();
38              }
39              pularLinha = true;
40              Thread.Sleep(60000);
41          }
42      }
43  }
```

Quadro 20 - Rotina que monitora os agendamentos

O carregamento dos agendamentos que são executados é feito através do método `carregarAgendamentosData` da classe `CarregarAgendamento` que é executado na linha 29. O método efetua consulta na base de dados verificando se existe algum agendamento definido para executar dentro do minuto que está sendo monitorado, se o agendamento está ativo e se está com situação “Aguardando”. Os agendamentos que atendem as condições são carregados para uma lista de agendamentos a executar.

Após carregar os agendamentos é instanciado um executor para cada agendamento. O executor (classe `ExecutorAgendamento`) é uma *Thread* que é executada a cada dois segundos, conforme Quadro 21.

```
14      public class ExecutorAgendamento
15      {
16          private Agendamento agendamento;
17          private Interpretador interpretador;
18          private AtualizadorAgendamento atualizadorAgendamento;
19          private AtualizadorTesteAgendamento atualizadorTeste;
20          public ExecutorAgendamento(Agendamento agendamento)
21          {
22              this.agendamento = agendamento;
23              this.interpretador = new Interpretador();
24              this.atualizadorAgendamento = new AtualizadorAgendamento();
25              this.atualizadorTeste = new AtualizadorTesteAgendamento();
26          }
27          public void executar()
28          {
29              atualizadorAgendamento.atualizarInicioExecucao(agendamento);
30              while (true)
31              {
32                  foreach (Teste teste in agendamento.Testes)
33                  {
34                      interpretador.interpretar(agendamento, teste);
35                      atualizadorTeste.atualizarTerminoExecucao(teste);
36                      Console.WriteLine(" *Executado \"{0}\" do agendamento de indentificação \"{1}\".",
37                          teste.Identificacao, agendamento.Identificacao);
38                      Thread.Sleep(2000);
39                  }
40                  break;
41              }
42              agendamento.montarLogExecucao();
43              atualizadorAgendamento.atualizarTerminoExecucao(agendamento);
44          }
45      }
```

Quadro 21 - Rotina executor agendamento

Durante a realização dos testes quando é identificado um método de classe a ser executado é utilizada a classe `ExecutorMetodo` que carrega a DLL, a classe, os parâmetros da classe, o método a ser executado e os parâmetros do método utilizando reflexão, conforme Quadro 22.

```
10  public class ExecutorMetodo
11  {
12      private Classe classeAtual = null;
13      private Metodo metodoAtual = null;
14
15      public Informacao executar(String assinaturaMetodo, SortedList<String, Classe> classes)...
19      private void carregarMetodoEClasseDeExecucao(String assinaturaMetodo, SortedList<String, Classe> classes)...
36      private Informacao efetuarReflexao(){
37          Informacao info = null;
38          System.Reflection.Assembly assembly = null;
39          object instanciaClasse = null;
40          MethodInfo instanciaMetodo = null;
41          object [] parametrosClasse = null;
42          object[] parametrosMetodo = null;
43          System.Type tipoResultado = null;
44          object resultadoMetodo = null;
45
46          try
47          {
48              assembly = System.Reflection.Assembly.LoadFile(classeAtual.Dll);
49          }
50          catch
51          {
52              throw new Exception(String.Format("Não foi possível carregar a dll '{0}'.", classeAtual.Dll));
53          }
54          try
55          {
56              parametrosClasse = carregarParamentrosClasse();
57          }
58          catch
59          {
60              throw new Exception(String.Format("Não foi possível carregar os parâmetros da classe '{0}'.",
61                  classeAtual.Nome));
62          }
```

Quadro 22 - Rotina de execução do método

A reflexão da DLL é realizada na linha 48. Quando a rotina não consegue carregar a DLL, o sistema avisa ao usuário qual DLL não foi possível carregar.

Após executar o teste é gerado o relatório com o resultado da execução, procedimento que é realizado através da classe `GeradorRelatorio`. O relatório é gerado no formato XLS utilizando a biblioteca `Microsoft.Oficce.Interop.Excel` conforme Quadro 23.

```
14    public class GeradorRelatorio
15    {
16        private Teste teste;
17        private Application application;
18        private Workbook workBook;
19        private Worksheet workSheet;
20        private ControleImpressao controleImp;
21        private int linhaInicial;
22        private int colunaInicial;
23        private int linha;
24        private int coluna;
25        private string nomeFonte = "Verdana";
26        private int tamanhoFonte = 9;
27        private CultureInfo cultureInfoUS = new CultureInfo("en-US");
28        public GeradorRelatorio(Teste teste)...
33        public void iniciarGeradorRelatorio()
34        {
35            application = new Microsoft.Office.Interop.Excel.Application();
36            workBook = application.Workbooks.Add(XlWBATemplate.xlWBATWorksheet);
37            workSheet = (Worksheet)workBook.Worksheets.Add(Type.Missing, Type.Missing, 2, Type.Missing);
38            for (int i = 1; i <= workBook.Worksheets.Count; i++){
39                workSheet = (Worksheet) workBook.Worksheets[i];
40                workSheet.Name = "Planilha " + i.ToString();
41            }
42            workSheet = (Worksheet) workBook.Worksheets[1];
43        }
44        public void gerar()...
140       private void imprimirResultadoMetodo()...
164       private void imprimirResultadoFormula()...
188       private string buscaInformacaoFormula()...
197       private string buscaInformacaoMetodo()...
240       public void finalizarGeradorRelatorio()...
256   }
```

Quadro 23 - Rotina de geração de relatório

A ferramenta permite que os testes sejam exportados. Procedimento que é realizado utilizando a classe `ExportadorXML`. O testador informa os testes que deseja exportar e a ferramenta gera arquivo no formato XML com informações dos testes identificados para exportação. A rotina efetua a exportação das informações dos testes selecionados na tela de exportação. A exportação é realizada utilizando a biblioteca `System.Xml` conforme Quadro 24.

```
16      public class ExportadorXML
17      {
18          public void exportar(ImportacaoExportacao impExp)...
22          private void criarArquivoFisico(ImportacaoExportacao impExp)
23          {
24              string caminho = @"" + impExp.EnderecoExportacao;
25              if (File.Exists(caminho))
26                  File.Delete(caminho);
27              XmlDocument doc = new XmlDocument();
28              XmlDeclaration declaration = doc.CreateXmlDeclaration("1.0", "UTF-8", null);
29              doc.InsertBefore((XmlNode)declaration, (XmlNode) doc.DocumentElement);
30              XmlNode noRaiz = doc.CreateElement("testes");
31              doc.AppendChild(noRaiz);
32              List<Teste> testes = carregarTestesInformados(impExp.IdTestes);
33              foreach (Teste testeAtual in testes)
34              {
35                  XmlNode noTeste = doc.CreateElement("teste");
36                  XmlNode noDataCriacao = doc.CreateElement("dataCriacao");
37                  noDataCriacao.InnerText = testeAtual.DataCriacao.ToString();
38                  XmlNode noIdentificacao = doc.CreateElement("identificacao");
39                  noIdentificacao.InnerText = testeAtual.Identificacao;
40                  XmlNode noConteudo = doc.CreateElement("conteudo");
41                  noConteudo.InnerText = testeAtual.Conteudo;
42                  XmlNode noConexaoDB = doc.CreateElement("conexaoDB");
43                  noConexaoDB.InnerText = testeAtual.ConexaoDB;
44                  noTeste.AppendChild(noDataCriacao);
45                  noTeste.AppendChild(noIdentificacao);
46                  noTeste.AppendChild(noConteudo);
47                  noTeste.AppendChild(noConexaoDB);
48                  noRaiz.AppendChild(noTeste);
49              }
50              doc.Save(caminho);
51              gravarArquivoNoSistema(doc, impExp);
52          }
```

Quadro 24 - Rotina de exportação de testes

Após realizar a exportação é gerado o arquivo XML com as informações dos testes. O arquivo gerado pode ser importado no sistema para que os testes sejam reutilizados. Procedimento que é realizado utilizando a classe `ImportadorXML`. Classe que efetua a importação de arquivo no formato XML. A rotina lê as informações dos testes registradas no arquivo e cria registros de testes vinculados ao agendamento selecionado na tela de importação. A importação é efetuada utilizando a biblioteca `System.Xml` conforme Quadro 25 e Quadro 26.

```
18      public class ImportadorXML
19      {
20          private CultureInfo cultureInfoBR = new CultureInfo("pt-BR");
21          public void importar(ImportacaoExportacao impExp)
22          {
23              string arquivoFisico = impExp.EnderecoImportacao;
24              if (!File.Exists(arquivoFisico))
25                  throw new Exception("Não foi possível identificar o arquivo informado.");
26              List<Teste> testes = new List<Teste>();
27              XmlDocument doc = new XmlDocument();
28              doc.Load(arquivoFisico);
29              XmlNode noTestes = doc.GetElementsByTagName("testes")[0];
```

Quadro 25 - Rotina de importação de testes (parte 1)

A rotina verifica se o registro informado na tela de importação realmente existe fisicamente.

```
30              foreach (XmlNode noTeste in noTestes.ChildNodes){
31                  Teste teste = new Teste();
32                  foreach (XmlNode noInfo in noTeste.ChildNodes){
33                      switch(noInfo.Name){
34                          case "dataCriacao":{
35                              if (noInfo.ChildNodes.Count > 0)
36                                  teste.DataCriacao = DateTime.ParseExact(noInfo.ChildNodes[0].Value,
37                                      "dd/MM/yyyy HH:mm:ss", cultureInfoBR);
38                              break;
39                          }
40                          case "identificacao":{
41                              if (noInfo.ChildNodes.Count > 0)
42                                  teste.Identificacao = noInfo.ChildNodes[0].Value;
43                              break;
44                          }
45                          case "conteudo":{
46                              if (noInfo.ChildNodes.Count > 0)
47                                  teste.Conteudo = noInfo.ChildNodes[0].Value;
48                              break;
49                          }
50                          case "conexaoDB":{
51                              if (noInfo.ChildNodes.Count > 0)
52                                  teste.ConexaoDB = noInfo.ChildNodes[0].Value;
53                              break;
54                          }
55                      }
56                  }
57                  CarregarTeste carregarTeste = new CarregarTeste();
58                  teste.OrdemExecucao = carregarTeste.buscaMaiorOrdemTestes(impExp.Agendamento.ID, -1) + 1;
59                  testes.Add(teste);
60              }
61              criarRegistrosTestes(testes);
62              gravarArquivoNoSistema(doc, impExp);
63          }
```

Quadro 26 - Rotina de importação de testes (parte 2)

A rotina (linha 58) cria o registro de teste com a ordem posterior à ordem do último teste criado para o agendamento.

3.4 RESULTADOS E DISCUSSÃO

Os resultados e discussões estão divididos em cinco seções. Na seção 3.4.1 é apresentada a avaliação realizada sobre a utilização da ferramenta, na seção 3.4.2 são abordadas a amostragem e os instrumentos utilizados para a coleta de dados, na seção 3.4.3 é exibido o procedimento de coleta dos dados, na seção 3.4.4 é descrita a análise dos resultados obtidos e na seção 3.4.5 é mostrada a comparação entre as funcionalidades da ferramenta desenvolvida e os trabalhos correlatos.

3.4.1 Avaliação da ferramenta

Esta seção descreve as avaliações realizadas na ferramenta desenvolvida, tendo como objetivo principal apresentar a metodologia e os resultados obtidos durante este processo.

O fator principal do experimento é avaliar a ferramenta dentro do contexto específico (testes de funções incorretas), visando identificar problemas que podem englobar aspectos funcionais e de utilização, para eventuais modificações na ferramenta.

Para a condução do experimento, optou-se pela utilização do estudo de caráter descritivo. O método descritivo "busca descobrir situações, eventos, atitudes e opiniões que ocorrem em uma população amostral. Objetiva-se verificar alguns fatos, mas não testar alguma teoria" (RIBEIRO, 2000, p. 25). O método foi aplicado a partir de um questionário aberto e de outro fechado, que contribuíram para uma análise quantitativa e qualitativa, respectivamente. No questionário aberto deixou-se um espaço onde o usuário pudesse registrar sua opinião sobre a ferramenta e o questionário fechado correspondeu a uma série de questões específicas sobre a satisfação do usuário em relação à experiência realizada.

3.4.2 Amostragem e instrumentos de coleta de dados

A amostra do experimento foi composta por cinco usuários adultos e com perfil misto de programadores e validadores de software. A média de idade dos avaliadores é de 23 anos, 100% são do sexo masculino, 20% possuem ensino superior completo e 80% estão cursando ensino superior. Os avaliadores foram submetidos a um questionário conforme Apêndice A.

Embora pequena, a amostra foi considerada adequada pelo fato de possibilitar um *feedback* destes usuários em todas áreas que constituem este trabalho. Contudo, os participantes vinculados a área de qualidade de software podem contribuir mais fortemente pelo fato de trabalhar com testes de software, fornecendo assim dicas importantes para o uso da ferramenta.

Para tanto, foram utilizados os seguintes instrumentos de coleta de dados:

a) anotações do avaliador que apontam aspectos sobre a realidade do experimento e registram o parecer do usuário no momento do experimento;
b) questionário aberto com a pergunta: "Qual é a sua opinião sobre a ferramenta quanto ao seu uso e funcionalidades? (críticas e sugestões)";

c) questionário fechado, com questões específicas. O questionário foi composto por sete perguntas referentes à eficiência da ferramenta, sua usabilidade e funcionalidades, conforme apresentado no Quadro 27 - Respotas da avaliação da ferramenta.

3.4.3 Procedimentos para coleta de dados

Os procedimentos de coleta de dados foram realizados individualmente entre os cinco participantes e tiveram uma duração média de 40 minutos.

Houve uma explicação inicial sobre o objetivo da avaliação, que era focado nas tarefas que levam a realização de testes dos métodos de classes.

Após o primeiro contato foi solicitado aos usuários que lessem o material apresentado na seção 3.3.2.2 referente à linguagem de definição dos testes, antes de realizarem a avaliação.

Depois disso, foi distribuído um formulário impresso contendo tarefas que o usuário deveria realizar na ferramenta (segunda parte do Apêndice A). Estas tarefas tinham o objetivo de direcionar o usuário durante a criação até a conclusão de um teste na ferramenta. Ao final, foi apresentado um formulário com algumas questões relacionadas à satisfação do usuário em relação ao uso da ferramenta (terceira parte do Apêndice A).

Os testes foram realizados em sua maioria utilizando o sistema operacional Windows 7, no navegador Internet Explorer 9.

3.4.4 Descrição e análise dos resultados obtidos

Esta seção foi dividida em análise qualitativa e quantitativa dos resultados. A análise qualitativa das respostas foi realizada a partir das anotações do avaliador e do questionário aberto. Houve poucas contribuições no questionário aberto, embora os usuários tenham sugerido algumas reformulações que foram anotadas e/ou observadas pelo avaliador. A análise quantitativa foi obtida através do questionário fechado. As sugestões foram registradas para reformulação das partes funcionais e de usabilidade da ferramenta. Nas próximas seções é apresentado e discutido os resultados alcançados nos experimentos.

3.4.4.1 Análise qualitativa dos resultados

Dentre as anotações feitas pelo avaliador nas sessões de entrevista, destacam-se os itens a seguir:

a) foram feitos comentários bons e construtivos para a melhoria da ferramenta. Por exemplo, um comentário feito por um dos usuário foi “No geral a ferramenta está muito boa. Poderia ser melhorada um pouco a usabilidade.”;
b) foi identificado um problema no relatório, que estava listando a data “00/01/1900 00:00:00” nas colunas de “Data término execução” do agendamento e do teste;
c) foram feitas sugestões para implementar algumas teclas de atalho, como exemplo, a tecla de atalho “Esc”, para fechar os cadastros;
d) foi identificado que na página de importações e exportações (Figura 24) faltou utilizar o caractere “*” para identificar alguns campos como obrigatórios;
e) foi sugerido que na página inicial do sistema (Figura 12) fosse identificado de alguma maneira se a lista de registros que está sendo exibido no momento é a de “agendamentos” ou a de “importações/exportações”.

No geral, os usuários conseguiram realizar todos os procedimentos solicitados. Alguns dos usuários necessitaram de auxílio com relação à utilização do agendamento, principalmente com relação à identificação de quando terminou sua execução. Após serem instruídos conseguiram identificar sem problemas.

3.4.4.2 Análise quantitativa dos resultados

Os usuários responderam ás sete perguntas do questionário fechado. O resultado deste questionário é apresentado no Quadro 27. As opções de respostas observam uma escala de níveis que segue uma gradiente em termos de concordância a discordância. Em cada questão, os usuários assinalaram o grau de avaliação da ferramenta em termos de "Concordo totalmente", "Concordo parcialmente", "Não concordo nem discordo", "Discordo parcialmente" e "Discordo totalmente". Este tipo de questionário ressalva as respostas positivas, negativas e irrelevantes em relação ao software avaliado. Esse modelo de gradiente foi retirado de Kruger, Fritsch e Viccari (2001, p. 21). As perguntas referem-se a aspectos relativos à usabilidade, funcionalidade eficiência como ferramenta de testes.

A partir das respostas obtidas na avaliação foi formulado o Quadro 27.

Perguntas / Critérios de avaliação	Concordo totalmente	Concordo parcialmente	Não concordo nem discordo	Discordo parcialmente	Discordo totalmente
1. É fácil de encontrar as opções / funcionalidades da ferramenta	40%	60%			
2. A linguagem definida para testar as funções incorretas é de fácil uso	20%	80%			
3. É importante o agendamento de testes	80%	20%			
4. É necessária a integração com outra ferramenta de teste		40%	40%	20%	
5. É importante a funcionalidade de exportar/importar testes	40%	40%	20%		
6. É fácil montar um teste na ferramenta	20%	80%			
7. De uma maneira geral a ferramenta é muito boa	20%	80%			

Quadro 27 - Respotas da avaliação da ferramenta

A partir dos resultados das questões 1 e 2 do Quadro 27 é possível perceber que os participantes não tiveram dificuldade para utilizar as funcionalidades da ferramenta e para criar os testes solicitados.

Com relação às respostas da questão 3 é possível identificar que a grande maioria dos participantes concordam totalmente com a utilização do agendamento para a realização dos testes e, que ele é importante em uma ferramenta de teste.

Na questão 4 é possível observar que as opiniões ficaram divididas com relação a integração ou não de outras ferramentas para a realização dos testes. É possível perceber também que nenhum dos avaliados concordou ou discordou com certeza sobre a questão.

A partir da avaliação da questão 5 é possível perceber que a grande maioria concordou com a utilização das funcionalidades de importação e exportação dos testes, e que apenas um avaliado ficou imparcial.

Com o resultado da questão 6 é possível concluir que os avaliados conseguiram utilizar a linguagem para a realização dos testes, mas que tiveram um pouco de dificuldade, provavelmente em função de ser a primeira vez que estavam utilizando a ferramenta.

A partir dos resultados da questão 7 é possível verificar que ninguém discordou da utilização da ferramenta para a realização dos testes.

Com os resultados obtidos é possível concluir que de maneira geral a ferramenta teve uma boa aceitação e que não apresentou grandes problemas em sua utilização.

3.4.5 Comparação com trabalhos correlatos

Após discussões dos trabalhos relacionados, das considerações sobre interpretação de linguagem, testes de software, técnicas de reflexão, execução de cálculos matemáticos, da especificação, desenvolvimento e operacionalidade da ferramenta de testes e dos resultados obtidos, reformulou-se o quadro das características dos trabalhos relacionados (Quadro 6 da seção 2.5.4) adicionando-se a ferramenta desenvolvida, para fins de comparação. O Quadro 28 apresenta a análise comparativa entre as características da ferramenta desenvolvida com as características existentes nos trabalhos relacionados.

Características dos trabalhos	**Marcos (2007)**	**Kolm (2001)**	**Santiago (2002)**	**Ferramenta desenvolvida**
Armazena histórico de testes realizados	Não	Sim	Sim	Sim
Possui interface Web	Não	Não	Não	Sim
Disponibiliza relatórios de resultados	Não	Sim	Sim	Sim
Realiza testes de funções incorretas	Não	Não	Não	Sim
Efetua os testes na própria ferramenta	Não	Não	Sim	Sim
Gera testes que são executados por outras ferramentas	Sim	Não	Não	Não
Permite exportar/importar testes	Sim	Não	Sim	Sim
Utiliza reflexão para realizar os testes	Não	Não	Não	Sim

Quadro 28 - Comparação com trabalhos correlatos

A partir do Quadro 28 é possível perceber as características comuns e as principais diferenças da ferramenta desenvolvida.

Os trabalhos correlatos são voltados principalmente para testes funcionais ou para gerência de testes. O trabalho desenvolvido também abrange a gerência e organização dos testes, mas tem por objetivo principal testar funções incorretas do sistema, ou seja, verificar se os métodos estão retornando o que é esperado, e não executar testes funcionais.

Entre as características apresentadas um item que é importante ressaltar é o fato de a ferramenta executar na Web, o que os outros trabalhos não são. A execução na Web permite uma maior mobilidade para o sistema, tornando o sistema acessível a uma maior quantidade de pessoas.

Outra característica importante é a utilização de reflexão para a realização dos testes, isso torna possível à ferramenta efetuar os testes sem ter acesso ao código-fonte, o que permite, por exemplo, que a equipe de desenvolvimento e a equipe de testes não necessariamente trabalhem juntas.

Ao analisar o conjunto de características abrangido pela ferramenta com relação aos trabalhos correlatos é possível verificar o como a ferramenta pode ser prática de ser utilizada e

o quanto ela pode ser importante para esta área específica de testes de software.

4 CONCLUSÕES

A ferramenta desenvolvida tem por objetivo facilitar a identificação de erros em métodos de classes na linguagem C#. Os testes efetuados pela ferramenta são realizados através de comparação de resultados. Para isso, permite que sejam informados nomes de classes e de métodos que o usuário pretende testar e as fórmulas matemáticas cujo resultado da avaliação é comparado com o resultado da execução dos métodos. O software executa os métodos e as fórmulas para identificar se o que foi desenvolvido está de acordo com o que foi solicitado.

A ferramenta realiza a validação dos métodos de classes sem que se tenha acesso ao código-fonte, o que é classificado como teste de funções incorretas que é um dos itens que compõe a técnica de testes caixa preta. Para o desenvolvimento deste trabalho fez-se necessário o estudo de técnicas de compiladores e de conceitos sobre testes de software.

Para o desenvolvimento do software foi utilizada a ferramenta de geração de analisadores léxico e sintático GALS (GESSER, 2003), que se mostrou muito útil e prática de ser utilizada. O gerador de analisadores foi utilizado gerando códigos fonte na linguagem de programação Java. Como o trabalho foi desenvolvido utilizando a linguagem de programação C#, foi necessário realizar a conversão de todo o código fonte gerado. Além dos analisadores léxico e sintático gerados pela ferramenta GALS, foi desenvolvido também o analisador semântico.

Com relação às tecnologias utilizadas, neste trabalho foi utilizado o ambiente de desenvolvimento Visual Studio 2010, com a linguagem de programação C# e o gerenciador de banco de dados MySQL, sendo que não foram encontrados problemas com a utilização das mesmas, todas colaboraram para a conclusão deste trabalho.

Foi possível obter bons resultados com o término do trabalho, os requisitos e os objetivos propostos foram todos atingidos. A partir dos resultados e discussões foi possível demonstrar o quanto a ferramenta desenvolvida pode ser útil e o quanto ela pode facilitar o trabalho dos testadores de software.

A partir do desenvolvimento deste trabalho é possível concluir o quanto é importante a realização de testes de software, e para isso quanto mais estes processos de testes forem automatizados mais prático se torna a realização dos mesmos.

4.1 LIMITAÇÕES

As principais limitações da ferramenta são em relação à linguagem para definição dos testes que foi desenvolvida:

a) a linguagem formal limita a utilização de variáveis e parâmetros dos tipos `decimal`, `int`, `string`, `bool`, `date` e `datetime`. A linguagem poderia ser expandida para permitir utilizar outros tipos de informações, como por exemplo, vetores, listas, filas, pilhas, permitir a passagem de parâmetros por referência;
b) a linguagem permite que sejam utilizados comandos SQL, mas está limitando a informação do comando de forma integral, precisa ser passada uma string com o comando inteiro, não permitindo que sejam utilizadas as variáveis definidas no conteúdo do comando.

4.2 EXTENSÕES

Como extensão para o trabalho propõe-se:

a) permitir que a ferramenta efetue a execução de testes de softwares desenvolvidos em linguagens de programação que não gerem DLLs, por exemplo, na linguagem Java. O trabalho atual permite que sejam executados métodos de classe apenas a partir da reflexão de DLLs;
b) permitir que sejam efetuados outros tipos de testes com os métodos de classe, por exemplo, pode ser permitido efetuar testes de performance dos métodos, resultando o tempo gasto com a execução do método. O trabalho atual não aborda este tipo de teste, a ferramenta apenas registra os tempos de início e fim da execução dos testes e dos agendamentos;
c) permitir que sejam feitos testes utilizando vários tipos de bancos de dados. O trabalho permite testes apenas utilizando conexão com o banco de dados MySQL.

REFERÊNCIAS BIBLIOGRÁFICAS

BRUNI, Adriano L. **Matemática financeira para concursos**. São Paulo: Editora Atlas, 2008.

CESTA, André A. **Estudo comparativo de linguagens de programação orientadas a objetos**. [S.l.], 1996. Disponível em: <http://www.ic.unicamp.br/~cmrubira/aacesta/java/javatut12.html>. Acesso em: 20 out. 2011.

CÔRTES, Mario L.; CHIOSSI, Thelma, C. S. **Modelos de qualidade de software**. Campinas: Unicamp, 2001.

DIVERIO, Tiarajú A.; MENEZES, Paulo B. **Teoria da computação**: máquinas universais e computabilidade. 2. ed. Porto Alegre: Sagra Luzzatto, 2003.

FERREIRA, Rodrigo M. **Gerador de código intermediário (código de três endereços) 1ª parte**. [S.l.], 2010. Disponível em: <http://compiladoresnapratica.blogspot.com/>. Acesso em: 5 nov. 2011.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. [S.l.], 2003. Disponível em: <http://gals.sourceforge.net/>. Acesso em: 22 mar. 2011.

GOLM, Michael; KLEINODER, Jurgen. **MetaXa and the future of reflection**. Erlangen, 1998. Disponível em: <http://www.csg.is.titech.ac.jp/~chiba/oopsla98/proc/golm.pdf>. Acesso em: 27 mar. 2011.

GRUNE, Dick et al. **Projeto moderno de compiladores**: implementação e aplicações. Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus, 2001.

INTHURN, Cândido. **Qualidade & teste de software**. Florianópolis: Visual Books, 2001.

KOLM, Everton L. **Sistema para gerenciamento de testes funcionais de software**. 2001. 45 f. Trabalho de Estágio Supervisionado (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

KOSCIANSKI, André; SOARES, Michel S. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

KRUGER, Susana E.; FRITSCH, Eloi; VICCARI, Rosa M. Avaliação pedagógica do software STR. **Revista Brasileira de Informática na Educação**, Florianópolis, p. 21-33, novembro, 2001.

LOUDEN, Kenneth C. **Compiladores**: princípios e práticas. Tradução Flávio Soares Corrêa da Silva. São Paulo: Pioneira Thomson Learning, 2004.

MARCOS, Adriana F. **Ferramenta de apoio à automatização de testes através do TestComplete para programas desenvolvidos em Delphi**. 2007. 75 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MECENAS, Ivan; OLIVEIRA, Vivianne. **Qualidade em software**: uma metodologia para homologação de sistemas. Rio de Janeiro: Alta Books, 2005.

MOLINARI, Leonardo. **Inovação e automação de testes de software**. São Paulo: Érica, 2010.

MOLINARI, Leonardo. **Testes de software**: produzindo sistemas melhores e mais confiáveis. São Paulo: Érica, 2003.

MOREIRA FILHO, Trayahú R.; RIOS, Emerson. **Teste de software**: o ponto de partida para qualquer projeto de teste de software é uma metodologia de testes consistente e adequada ao ambiente de desenvolvimento da empresa. Rio de Janeiro: Alta Books, 2003.

PRICE, Ana M. A.; TOSCANI, Simão S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

REPASS, Mike. **Tudo sobre o CLR**: reflexões sobre a reflexão. [S.l.], 2007. Disponível em: <http://msdn.microsoft.com/pt-br/magazine/cc163408.aspx>. Acesso em: 7 nov. 2011.

RIBEIRO, Vinicius G. **Um estudo sobre métodos de pesquisa utilizados em segurança computacional**. 2000. 69 f. Trabalho Individual (Mestrado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

SANTIAGO, Denise. **Ferramenta para testes de programas utilizando componentes da biblioteca CLX**. 2002. 57 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SPARXSYSTEMS. Creswick, 2000. Disponível em: <http://www.sparxsystems.com/>. Acesso em: 20 jul. 2011.

APÊNDICE A – Questionário de avaliação

Data de realização da avaliação: / /2011.

Observação: As informações recebidas serão mantidas confidenciais.

PRIMEIRA PARTE:
Este questionário possui 6 questões. Por gentileza, responda cada uma delas.
Preencha o seguinte questionário de informações sobre o seu perfil. Seja o mais específico possível nas respostas.

1. Sexo: ( ) Masculino ( ) Feminino
2. Idade: ___ anos.
3. Profissão atual:
4. Tempo na profissão: ___ anos.
5. Já utilizou/utiliza alguma ferramenta de teste: ( ) Sim ( )Não
6. Quais ferramentas já utilizou (Opcional):

SEGUNDA PARTE:
O objetivo desta parte do experimento é identificar se os usuários têm facilidade para utilizar a ferramenta de testes desenvolvida. Para isso foi preparado um roteiro organizado por etapas, instruindo o usuário a realizar procedimentos que exploram as funcionalidades da ferramenta. O roteiro foi separado em duas etapas. Na primeira etapa foi solicitada desde a criação de um agendamento até a exportação do teste do agendamento criado.

Etapa 1: Realize os procedimentos conforme descritos a seguir:

1. acesse o sistema, será apresentada a tela inicial do sistema;
2. identifique onde são criados os agendamentos de testes e solicite a criação de um novo agendamento;
3. para criar o agendamento dê um nome para ele (utilize o nome "Agendamento 1 testador X", onde "X" é o seu número), identifique quando este deverá ser executado e com que periodicidade (Importante: deverá ser informada uma data que defina que agendamento será executado após a criação do teste do agendamento, para que o agendamento não seja executado antes do teste ser criado). Após definir como será o agendamento identifique como salvar o agendamento e salve-o;
4. o registro de agendamento será criado e retornará para a tela inicial do sistema, identifique o registro criado e descubra como acessá-lo;
5. após acessar o registro do agendamento, identifique como criar um teste para o agendamento e solicite a criação de um teste;
6. para criar o teste dê um nome para ele (utilize o nome "Teste 1 testador X", onde "X" é o seu número) e identifique o conteúdo do teste que deverá ser realizado:
   - no conteúdo do teste deverá ser definida uma variável que efetua a soma do valor "`5.0`" com o valor "`4.0`";
   - deverá ser definida a utilização do método "`somar`" da classe "`CalculadorMatematico.OperacaoMatematica`" que se encontra na DLL "`C:/CalculadorMatematico.dll`", considerando que o método não é estático;

- deverá ser informado um parâmetro do tipo "`string`" para a classe identificando a operação que será realizada. Deverão ser informados os parâmetros "`5.1`" e "`4.0`" para o método;
- deverá ser informado que o resultado do cálculo da variável deverá ser comparado ao resultado da execução do método.

7. após definir como será o teste identifique como salvar o teste e salve-o;
8. o registro de teste será criado e retornará para a tela de agendamentos;
9. ao atingir a data e hora definida para execução o executor de agendamentos irá executar o teste do agendamento, é preciso apenas esperar a data ser atingida;
10. para identificar que o agendamento foi executado é preciso verificar se a data de término de execução foi preenchida no registro do agendamento;
11. após o agendamento ser executado identifique o registro de teste criado e descubra como acessá-lo;
12. após acessar o registro de teste descubra como visualizar o relatório de resultado da execução;
13. identifique que o resultado de execução do método ficou diferente do resultado de execução da fórmula;
14. após visualizar o resultado da execução do teste volte para a tela inicial do sistema;
15. identificar onde são criados as exportações de testes e solicite a criação de uma nova exportação;
16. para criar a exportação dê um nome (utilize o nome "Exportar 1 testador X", onde "X" é o seu número), identifique que a operação realizada será uma exportação, selecione o agendamento do qual será feita a exportação, selecione os testes do agendamento que serão exportados e informe o nome do arquivo que deverá ser gerado. Após definir como será a exportação identifique como salvar a exportação e salve-a;
17. o registro de exportação será criado e retornará para a tela inicial do sistema.

Período de realização da etapa 1:

Data de início: / / 2011 Data de fim: / / 2011

Hora de inicio: Hora de fim:

A segunda etapa tem como pré-requisito a realização da primeira etapa.
Na segunda etapa foi solicitada desde a importação do teste exportado até a visualização do teste executado com sucesso.

Etapa 2: Realize os procedimentos conforme descritos a seguir:

1. identifique o registro de exportação criado e descubra como acessá-lo;
2. após acessar o registro descubra como acessar o arquivo XML de exportação e salve-o em um local de sua máquina para poder acessá-lo posteriormente;
3. feche a tela de exportação, você irá retornar para a tela inicial do sistema;
4. crie um novo agendamento (utilize o nome "Agendamento 2 testador X", onde "X" é o seu número) sem definir nenhum teste para o mesmo (seu teste será importado);
5. após a criação do novo agendamento, identifique onde são criados as importações de testes e solicite a criação de uma nova importação;
6. para criar a importação dê um nome para ela (utilize o nome "Importar 1 testador X", onde "X" é o seu número), identifique que a operação realizada será uma importação, selecione o novo agendamento criado e informe o nome do arquivo que deverá ser importado;
7. após definir como será a importação identifique como salvar a importação e salve-a;

8. o registro de importação será criado e retornará para a tela inicial do sistema;
9. acesse o novo agendamento e através deste acesse o teste importado;
10. No teste importado altere o seu conteúdo para que ao invés de passar os parâmetros "`5.1`" e "`4.0`" para o método comparado, passar os parâmetros "`5.0`" e "`4.0`" para o método;
11. após definir a alteração no teste salve-o;
12. o registro de teste será alterado e retornará para a tela de agendamentos;
13. ao atingir a data definida para execução o executor de agendamentos irá executar o teste do agendamento, é preciso apenas esperar a data ser atingida;
14. após o agendamento ser executado acesse o registro do teste;
15. após acessar o registro de teste visualize o relatório de resultado da execução;
16. identifique que o resultado de execução do método desta vez ficou igual ao resultado de execução da fórmula.

Período de realização da etapa 2:

Data de início: / / 2011 Data de fim: / / 2011

Hora de inicio: Hora de fim:

TERCEIRA PARTE:

Após a utilização da ferramenta realizada na segunda parte, você está convidado a responder um questionário de avaliação da ferramenta.

As respostas deverão ser feitas na tabela abaixo observando às impressões obtidas com a utilização da ferramenta. Você deve responder preenchendo uma das alternativas.

Após o questionário com perguntas objetivas é apresentado um espaço para comentários gerais sobre a ferramenta e para sugestões de melhorias.

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Não concordo nem discordo**	**Discordo parcialmente**	**Discordo totalmente**
1. É fácil encontrar as opções / funcionalidades da ferramenta					
2. A linguagem definida para testar as funções incorretas é de fácil uso					
3. É importante o agendamento de testes					
4. É necessário a integração com outra ferramenta de teste					
5. É importante a funcionalidade de exportar/importar testes					
6. É fácil montar um teste na ferramenta					
7. De uma maneira geral a ferramenta é muito boa					

Qual é a sua opinião sobre a ferramenta quanto ao seu uso e funcionalidades?
(críticas e sugestões)

_____
_____
_____

Obrigado pela sua participação!