ANTONIO MENEGHETTI FACULDADE - AMF

CURSO DE SISTEMAS DE INFORMAÇÃO

JOÃO ORLANDO MISSIO

AUTOMAÇÃO DE TESTES NO PROCESSO DE PRODUÇÃO DE SOFTWARE PARA REDUÇÃO DO INCIDENTE DE ERROS

RESTINGA SECA/RS

2016

JOÃO ORLANDO MISSIO

AUTOMAÇÃO DE TESTES NO PROCESSO DE PRODUÇÃO DE SOFTWARE PARA REDUÇÃO DO INCIDENTE DE ERROS

Trabalho de Conclusão de Curso para a obtenção de grau de Bacharel em Sistemas de Informação.
Orientador: Ms. Fábio Sarturi Prass

RESTINGA SECA/RS

2016

FACULDADE ANTONIO MENEGHETTI

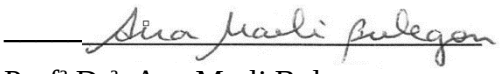
João Orlando Missio

AUTOMAÇÃO DE TESTES NO PROCESSO DE PRODUÇÃO DE SOFTWARE PARA REDUÇÃO DO INCIDENTE DE ERROS

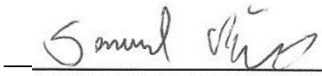
Trabalho de Conclusão de Curso-Monografia, apresentado como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação, Curso de Graduação em Sistemas de Informação, Faculdade Antonio Meneghetti-AMF.

Orientador: Prof. Ms. Fábio Sarturi Prass

Prof. Ms. Fábio Sarturi Prass
Orientador do Trabalho de Conclusão de Curso
Antonio Meneghetti Faculdade



Profª Drª. Ana Marli Bulegon
Membro da Banca Examinadora
Antonio Meneghetti Faculdade



Prof Ms. Samuel Vizzotto
Membro da Banca Examinadora
Antonio Meneghetti Faculdade

Restinga Sêca, RS, Novembro de 2016

AGRADECIMENTOS

Quero agradecer, em primeiro lugar, a Deus, pela força e coragem durante toda esta longa caminhada.

A minha família, principalmente aos meus pais Darci Missio e Nilza Teresinha de Marco Missio pela educação dada, incentivo e esforços realizados para que eu chegasse a esta etapa tão importante da minha vida.

A minha namorada Katlin Alice Rampelotto pelo carinho, apoio e compreensão, por estar ao meu lado neste momento tão importante, sem dúvidas você foi um grande pilar para a realização deste sonho.

Ao professor Fábio Sartiru Prass pelas orientação e ajuda na elaboração deste trabalho, sem dúvidas um grande exemplo de profissional, agradeço pela sua dedicação e esforço oferecido nesta trajetória.

A professora Ana Marli Bulegon pelas enormes contribuições concedidas durante toda essa fase de elaboração deste projeto e aos demais professores do curso de Sistemas de Informação por dedicarem seu tempo a transmitir seus conhecimentos que possuem, nos instigando a ser mais e sempre buscar ser o melhor naquilo que fazemos.

A todos, meu muito obrigado!

“Se você não ama algo, não vai andar um quilômetro extra,
não vai trabalhar um fim de semana extra e nem desafiar tanto o comum.”.

Steve Jobs

RESUMO

Com o crescimento da utilização de softwares no dia-a-dia na gestão de empresas, os sistemas estão ficando cada vez mais complexos para atender aos requisitos propostos pelos clientes. Desse modo, para que atenda às necessidades, sem apresentar problemas causados por erros no desenvolvimento, é necessário que o software seja submetido há um processo de teste de qualidade de software. Este processo é fundamental para se obter qualidade no produto final, o qual é disponibilizado aos usuários para utilização. No entanto, com a utilização da automação no processo de testes é possível reduzir o seu índice de ocorrências de erros, contribuindo, assim, para uma melhor qualidade e redução de erros aos seus usuários. Este trabalho busca demonstrar as vantagens em utilizar a execução de testes automatizados no processo de produção do software. Diante disso, é utilizado a ferramenta Visual Studio para desenvolvimento de um sistema para realizar a automação de testes. Durante a execução da automação é possível realizar uma coleta dos erros obtidos e, desta forma, comprovar a importância da utilização de testes automatizados para complementar os demais tipos de testes realizados no sistema.

Palavras-chave: Automação de Teste de Software; Visual Studio; C#; Coded UI Tests.

ABSTRACT

With the increasing use of software on a day-to-day business management, systems are becoming more complex to meet the requirements proposed by the customers. Thus, to meet the needs, without presenting problems caused by errors in development, it is necessary that the software undergoes a process of software quality test. This process is crucial to achieving quality in the final product, which is available to users for use. However, with the use of automation in the testing process can reduce its index of error occurrences, thus contributing to better quality and reduced errors to their users. This work aims to demonstrate the advantages of using the execution of automated tests in the software production process. Therefore, it uses the Visual Studio development tool for a system to perform the test automation. During the execution of automation it is possible to perform a collection of errors obtained and thus prove the importance of using automated tests to complement other types of tests performed on the system.

Keywords: Software Test Automation; Visual Studio; C#; Coded UI Tests.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

TI - Tecnologia de Informação

IDE - Integrated Development Environment

MSDN - Microsoft Developer Network

SUMÁRIO

1 INTRODUÇÃO

Os softwares são de fundamental importância para as pessoas na atualidade, pois, necessita-se de sua utilização no dia-a-dia para a gestão de empresas, uso pessoal, entretenimento, entre outros. Eles estão cada vez mais presentes em todos os lugares, seja na Medicina, Engenharia, Contabilidade, etc. Porém, devido à demanda do mercado de trabalho e ao avanço da tecnologia estes estão se tornando cada vez mais complexos, assim, exigindo-se cada vez mais qualidade em seu desenvolvimento.

Essa complexidade também impacta no seu desenvolvimento e muitos softwares podem não funcionar como o esperado pelo usuário final. Isso pode ocorrer caso exista uma má comunicação durante o levantamento de requisitos realizado juntamente ao usuário ou, até mesmo, problemas ocorridos na construção do código.

Nesse sentido, em sua fase de desenvolvimento é importante que se elaborem testes de software a fim de identificar falhas decorrentes dessa complexidade, para que estas sejam corrigidas antes da entrega do produto. Isso reduziria o índice de falhas e evitar-se-ia a geração de prejuízos para seus usuários.

Diante disso, decorre o seguinte problema: de que modo pode-se detectar os erros em um software antes do uso pelo consumidor final?

1.1 Objetivos

A fim de solucionar o problema acima descrito, esta pesquisa tem como objetivos:

1.1.1 Objetivo principal

Demostrar as vantagens em utilizar a execução de testes automatizados no processo de produção do software.

1.1.2 Objetivos específicos

- Estudar diferentes tipos de testes de software existentes;
- Analisar as vantagens da utilização de testes de software automatizados;
- Implementar uma ferramenta para automação de testes de um sistema.

1.2 Justificativa

Este projeto justifica-se, pois, a utilização de testes automatizados em softwares pode contribuir para a obtenção de uma melhor qualidade do mesmo e redução de prejuízos a seus usuários.

Com o sistema de automação de testes pode-se reduzir a ocorrência de erros, que podem passar despercebidos durante a fase de desenvolvimento e serem descobertos em produção; também auxilia na redução do esforço nos testes e em tarefas repetitivas necessárias de serem executadas durante o teste de software. Tudo isso, sendo de extrema importância para reduzir gastos com tempo de retrabalho para correção de erros, também, tendo-se assim um cliente mais satisfeito com o serviço no qual foi contratado.

Neste sentido, esta pesquisa propõe o desenvolvimento de uma ferramenta para automatização de testes de um sistema a fim de minimizar possíveis falhas decorrentes de erros de implementação no produto antes de sua utilização pelos clientes.

2 ASPECTOS GERAIS DA QUALIDADE E TESTE DE SOFTWARE

Nesse item será abordado conceitos importantes sobre testes de software e automação de testes necessários para o desenvolvimento da ferramenta de testes automatizados.

2.1 Testes de Software

Teste de software é o processo de execução de um programa ou sistema com a finalidade de encontrar erros. Também pode ser definido como o processo de validação de funcionalidades do sistema a fim de verificar se o seu comportamento está conforme os requisitos especificados no documento, os quais guiaram seu desenvolvimento funcionam conforme o esperado. O processo de teste de software está relacionado diretamente com a garantia de qualidade do produto (VICCARI, 2009).

2.1.1 O que é um erro de programa?

Segundo Izabel (2014), um erro de programa é um desvio do que foi especificado no documento de requisitos. Esse erro de implementação no código pode gerar defeitos que podem ser encontrados no software apenas em sua entrega final. Dessa forma, apenas um erro na implementação do código fonte pode vir a gerar vários defeitos (*bugs*) no funcionamento do sistema.

Na Figura 1 é possível observar que o erro é gerado pela ocorrência de um defeito na implementação do produto, caso este erro seja obtido pelo usuário final, o mesmo é tratado como uma falha.

Figura 1. Defeito *vs* erro *vs* falha.
Fonte:http://goo.gl/RGKvdL

2.2 Tipos de Teste de Software

Neste item serão abordados os principais tipos (ou níveis) de teste de software, os quais podem ser aplicados em qualquer sistema.

2.2.1 Teste Unitário

Segundo Rios e Moreira (2013), teste unitário tem a finalidade de verificar o funcionamento de parte do sistema ou software de forma independente visando garantir que estes atendem as especificações em termos de características e funcionalidades. Esses testes são geralmente realizados por desenvolvedores.

2.2.2 Teste de Integração

O teste de integração visa garantir que as *interfaces* funcionem e que os dados estão processados de forma correta, conforme o especificado no documento. Esses testes podem ser realizados de forma incremental, onde cada módulo ou componente é incluído de forma sequencial

até que todos os casos de testes sejam testados. Esse nível de teste geralmente é realizado por desenvolvedores ou por uma equipe de testes (RIOS & MOREIRA, 2013).

2.2.3 Teste Funcional

O teste funcional, também conhecido como teste de caixa preta, são definidos de acordo com os requisitos funcionais do software. Como não há acesso aos detalhes de implementação do mesmo, o analista concentra-se nas funções que o software contemplará. Baseando-se na especificação determinam-se as saídas que são esperadas para um determinado conjunto de dados (NETO, 2010).

2.2.4 Teste de Sistema

Os testes de sistema visam à execução do sistema como um todo ou parte do sistema, nesta etapa são testadas as funções do software de forma mais próxima possível do que ocorrerá no ambiente de produção, onde são realizados testes de carga, performance, usabilidade, compatibilidade, segurança e recuperação. Esses testes são realizados por uma equipe de testes ou também pelo usuário final no caso de serem dependentes do ambiente de produção como os testes de carga, performance e estresse (RIOS & MOREIRA, 2013).

2.2.5 Teste de Aceitação

O teste de aceitação geralmente é realizado pelo usuário final, visando verificar se a solução atende aos objetivos do negócio e aos seus requisitos, no que diz respeito à funcionalidade e usabilidade, antes da utilização do software ou sistema no ambiente de produção (RIOS & MOREIRA, 2013).

2.3 Automação de Testes

Conforme Izabel (2014), a automação de testes consiste em utilizar um software externo que não faça parte do sistema no qual será realizado a automação, para controlar a execução dos testes e a comparação dos resultados encontrados na execução da automação com os resultados esperados. Desse modo, podem-se automatizar tarefas de teste repetitivas, as quais são importantes de serem realizadas no sistema, que fazem parte do fluxo de teste de um software ou também adicionar novos testes que seriam muito trabalhosos e consequentemente custosos de serem realizados manualmente no sistema.

2.3.1 *Framework*

Segundo Nícolas Müller (2008), um *framework* é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica, ou também específica, por configuração, durante a programação de uma aplicação.

2.3.2 Coded Ui Tests

A ferramenta de testes de *interfaces* de usuário codificadas chamada Coded Ui Tests é um *framework* de testes automatizados presente na ferramenta de desenvolvimento de software Visual Studio. Segundo MSDN (2016), esses testes automatizados permitem que seja verificado se todo o aplicativo, inclusive a *interface* de usuário, está funcionando corretamente. Esse tipo de testes de *interface* de usuário são úteis quando há validações ou lógicas na *interface* do usuário, como por exemplo em páginas Web.

Esse *framework* é muito utilizado para automatizar testes que não há necessidade de serem executados manualmente, desse modo, não sendo necessário realizar validações de interface e lógicas presentes no programa, como comportamentos de interface e cálculos de forma manual após novas implementações ou alterações no código fonte da aplicação, assim garantindo que o produto esta funcionando conforme especificação funcional do sistema.

2.4 Linguagem de Programação C#

A linguagem de programação C# é dirigida por eventos e totalmente orientada a objetos, na qual os sistemas são criados usando uma IDE (Integrated Development Environment – ambiente de desenvolvimento integrado). Nesta IDE é possível criar, executar, testar, e depurar programas convenientemente, reduzindo assim o tempo necessário para produzir um programa funcional a uma fração do que levaria sem usar o IDE (DEITEL,2003, p. 8).

Segundo MSDN (2016), a linguagem de programação C# é baseada em linguagens de programação como C, C++ e Java. Sua sintaxe simplifica muitas das complexidades encontradas nestas outras linguagens, pois fornece recursos poderosos, dentre eles, expressões lambda, delegações, enumerações, além de possuir suporte a tipos genéricos, o qual fornece maior segurança de tipagem e desempenho para o código fonte.

2.5 Visual Studio .Net

O Visual Studio .NET é a IDE da Microsoft para a criação, documentação, execução e depuração de programas que podem ser escritos em diversas linguagens de programação .NET como por exemplo C#, Visual Basic, F#, entre outros. O Visual Studio .NET também oferece ferramentas de edição para manipular vários tipos de arquivos, além de ser uma ferramenta poderosa e sofisticada para criar aplicativos de missão e comercialização critica (DEITEL, 2003, p.28).

Na Figura 2 pode ser visualizado a interface "*About*" presente na IDE Visual Studio 2013 em sua versão Ultimate, a qual possui suporte para desenvolvimento de *interfaces* de usuário codificadas, que é utilizada para o desenvolvimento do software para automação de testes.

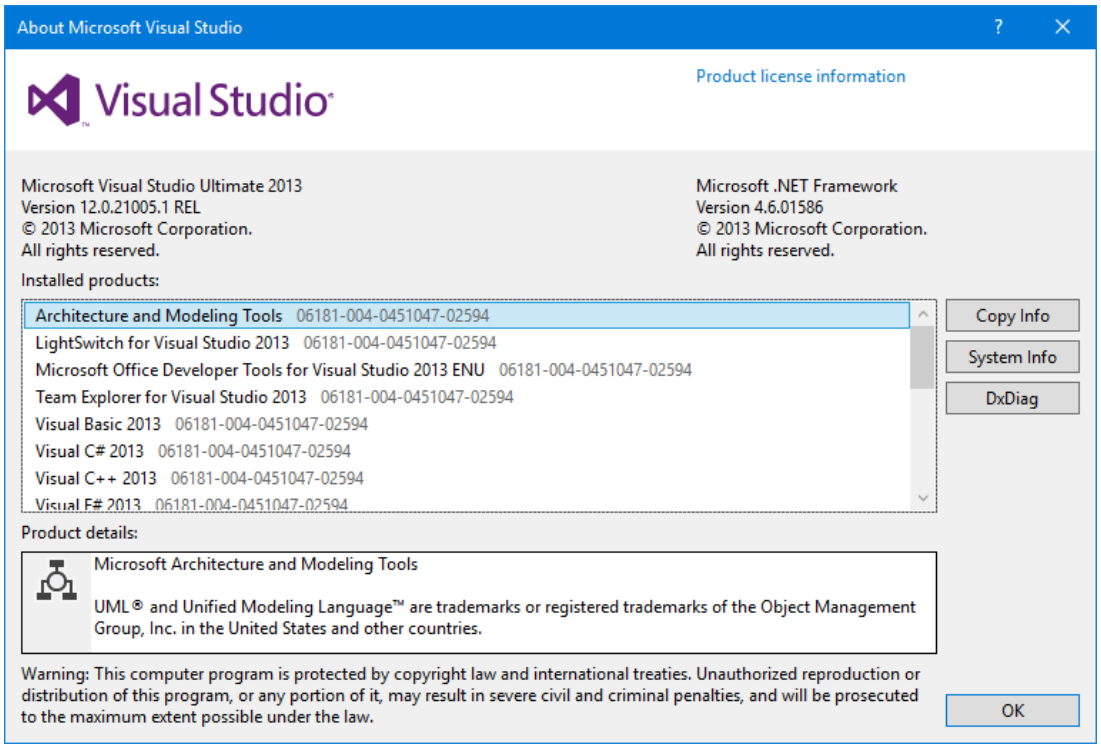


Figura 2. Visual Studio .Net.

3 METODOLOGIA

Nesse item é abordado o tipo de pesquisa e a metodologia de trabalho a ser utilizado para coletar dados e responder a questão de pesquisa proposta.

Trata-se de uma pesquisa qualitativa, pois procurará verificar a qualidade do software a ser testado, além de quantitativa (número de situações em que foram detectados erros no software testado). A mesma versará sobre a ferramenta a ser utilizada para a realização dos testes automatizados.

Diante disso, é utilizada a ferramenta Visual Studio 2013 Ultimate para desenvolvimento do software de automação a ser aplicado nos testes de um sistema especifico. A linguagem de programação a ser utilizada nesse desenvolvimento será o C#.

A escolha desta ferramenta em sua versão Ultimate deve-se pois, esta possui suporte à implementação de testes de *interface* de usuário codificados, esta funcionalidade não esta presente

em todas as versões da IDE Visual Studio, além da versão Ultimate, também há suporte a este recurso na versões Premium desta IDE.

A coleta de dados foi realizada por meio de uma observação não-participante, desse modo, realizou-se uma coleta de erros encontrados no software ao realizar a execução da automação no mesmo.

O público alvo desta pesquisa são as empresas de Tecnologia da Informação (TI) que possuem células de teste de software, porém, não utilizam a automação de testes para validação de sistemas. Desse modo, no decorrer deste trabalho será demonstrado a importância de sua utilização para complementar o processo de testes de softwares.

4 ANÁLISE E ESPECIFICAÇÃO DO SISTEMA

Este item tem como objetivo apresentar os requisitos para o sistema, os *stakeholders*[1], assim como, os modelos de domínio das entidades e as *interfaces* desenvolvidas para o sistema.

4.1 Introdução

Para realizar os testes de automação de software, foi utilizado um programa de controle de estoque, o qual possui funcionalidades como cadastro de categorias, fornecedores, produtos, usuários e movimentações de entrada e saída de estoque, além das telas de consulta das funcionalidades citadas anteriormente que o sistema possui.

Estas funcionalidades e requisitos do sistema utilizados para realizar a automação de testes utilizando a ferramenta de desenvolvimento de software Visual Studio e do programa para controle de estoque serão descritas nos itens a seguir. Será utilizado o diagrama de caso de uso, diagrama de classe, além de um organograma para descrevê-los.

4.2 Modelagem Conceitual

A estrutura do sistema de automação de testes é constituída por duas posições principais, dentre elas tem-se o cadastro e a pesquisa. Essas posições são representadas no organograma por retângulos, distribuídos de forma vertical e interligados por linhas que representam a hierarquia entre as interfaces nas quais foram implementadas a automação de testes.

As *interfaces* nas quais foram implementados são subdivididas em Cadastro (categoria, fornecedor, produto, movimentação, usuário) e Pesquisa (categoria, fornecedor, produto, movimentação). No organograma da Figura 3 é possível visualizar a hierarquia completa do sistema de automação do programa de controle de estoque.

[1] *Stakeholder* é uma pessoa ou grupo que possui participação, investimento ou ações e que possui interesse em uma determinada empresa ou negócio (BEZERRA, 2014).

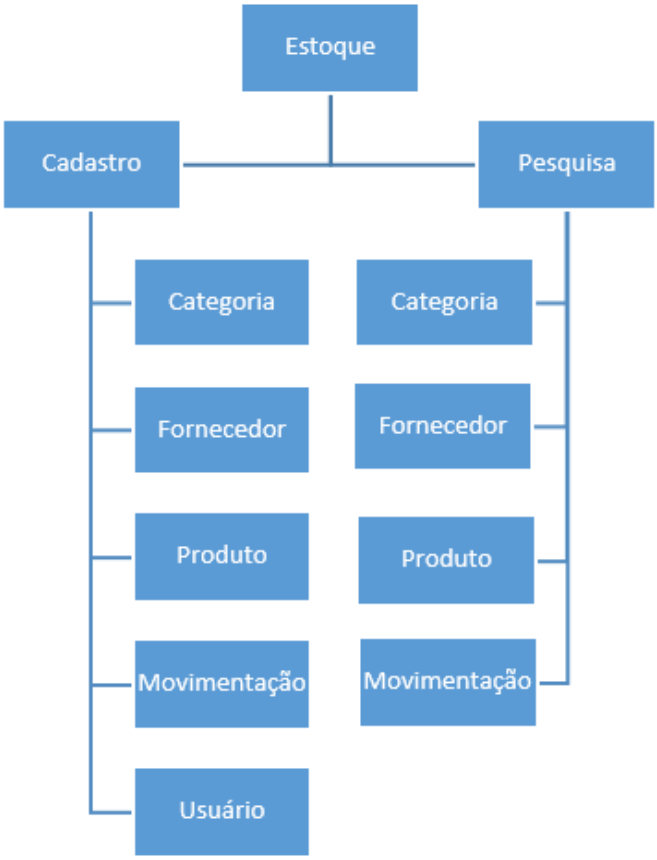


Figura 3. Organograma - Controle de estoque.

4.3 Arquitetura e Implementação

Para demonstrar o comportamento do sistema do ponto de vista dos usuários, criou-se o diagrama de caso de uso representado na Figura 4. Neste diagrama pode-se observar as funcionalidades do sistema representadas pelos casos de uso, atores e a comunicação entre os atores e os casos de uso. É possível identificar ainda que o ator “Cliente” possui acesso somente ao caso de uso “Entrada e Saída de Produtos”, já o ator “Funcionário” possui acesso a todas as funcionalidades do sistema.

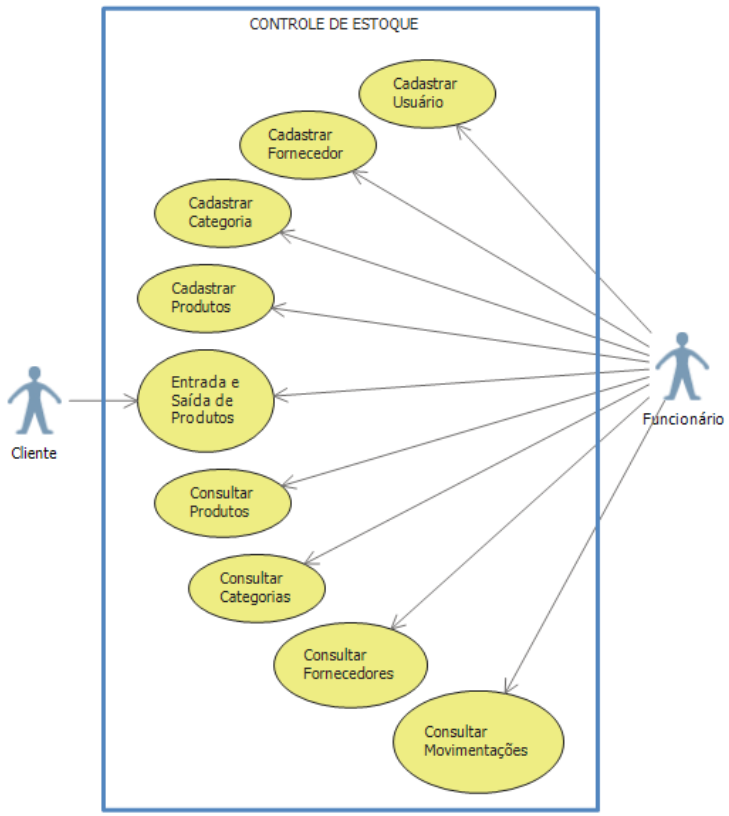


Figura 4. Diagrama de Caso de Uso do sistema de Controle de estoque.

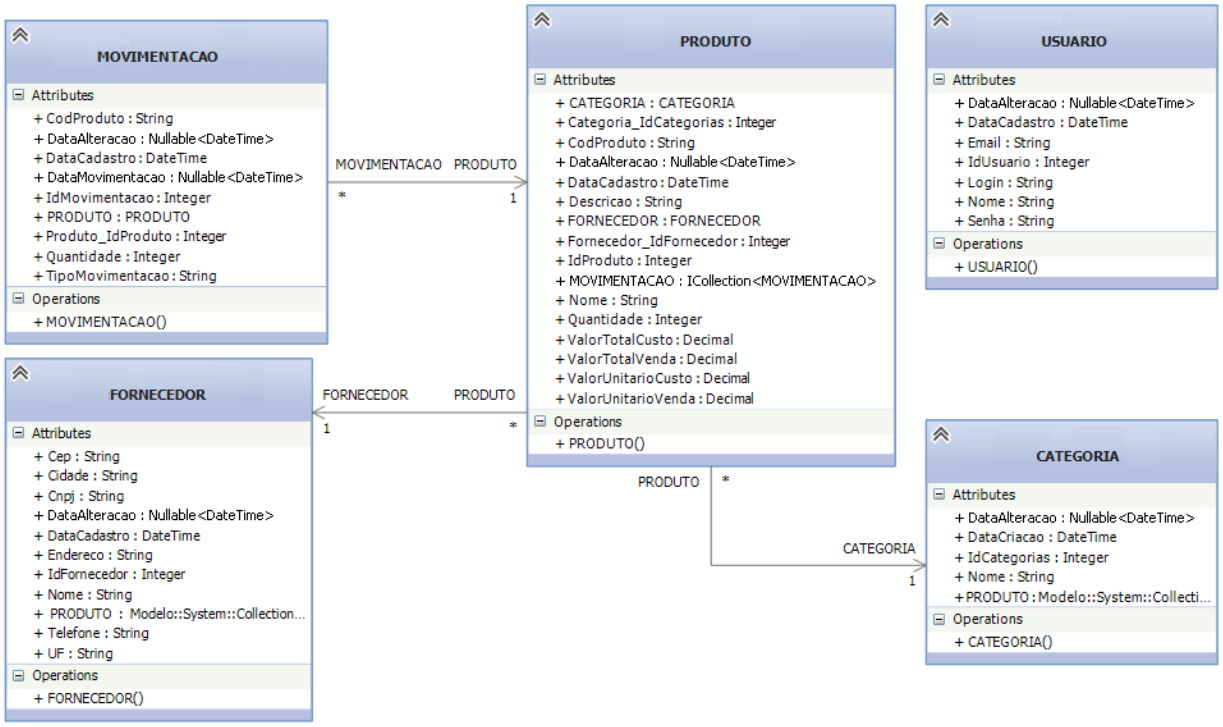


Figura 5. Diagrama de classe - Controle de estoque.

Conforme o diagrama de classe representado na Figura 5, observa-se o detalhamento das entidades utilizadas no sistema de controle de estoque com seus atributos, como, por exemplo, a entidade Categoria que possui atributos como “Nome”, “IdCategoria”, entre outros, além disso, também está definido os relacionamentos entre as entidades do sistema.

4.4 Especificação das *Interfaces* do Sistema

Conforme o diagrama de caso de uso representado na Figura 4, as *interfaces* a serem utilizadas no sistema de controle de estoque para realização da automação de testes serão apresentadas nos itens a seguir.

4.4.1 Cadastro de Usuário

Esta *interface* é responsável pelo cadastro de usuários que terão acesso ao sistema, possui apenas dois campos para preenchimento, dentre eles são: usuário e senha para acesso, conforme Figura 6.

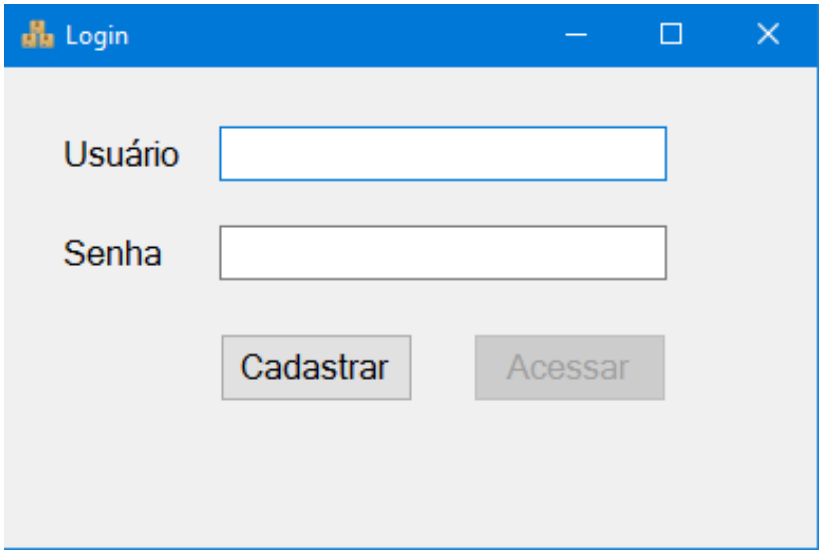


Figura 6. *Interface* de cadastro de usuário e login.

4.4.2 Cadastro de fornecedor

O formulário de cadastro de fornecedores possui alguns dados básicos sobre fornecedor como: Cnpj, telefone, dados do endereço e nome. Na Figura 7 tem-se a ilustração da tela de cadastro de fornecedores do sistema.

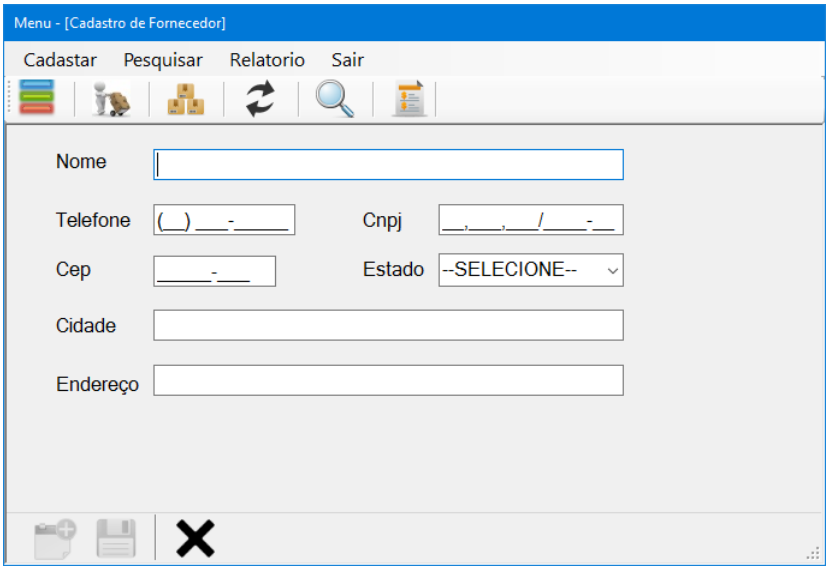


Figura 7. *Interface* de cadastro de fornecedor.

4.4.3 Cadastro de categoria

A tela de cadastro de categorias é responsável pelo cadastro das categorias de produtos, em que cada produto deve estar associado a uma categoria. Na Figura 8 pode ser observado a *interface* responsável pelo cadastro destas categorias.

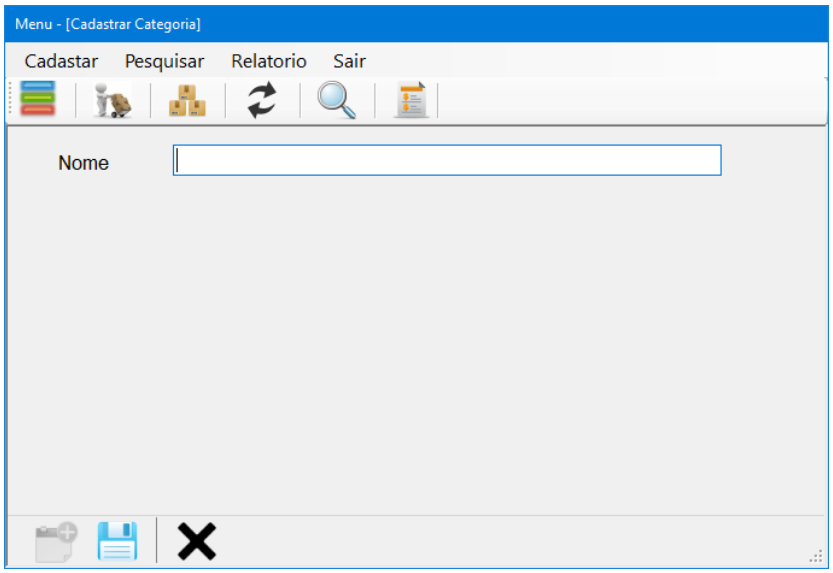


Figura 8. *Interface* de cadastro de categoria.

4.4.4 Cadastro de produtos

Na tela de cadastro de produtos, o funcionário irá cadastrá-lo informando dados básicos sobre o produto e irá vinculá-lo a um fornecedor e a categoria cadastrada no sistema. Na Figura 9 pode-se visualizar a *interface* do cadastro de produtos.

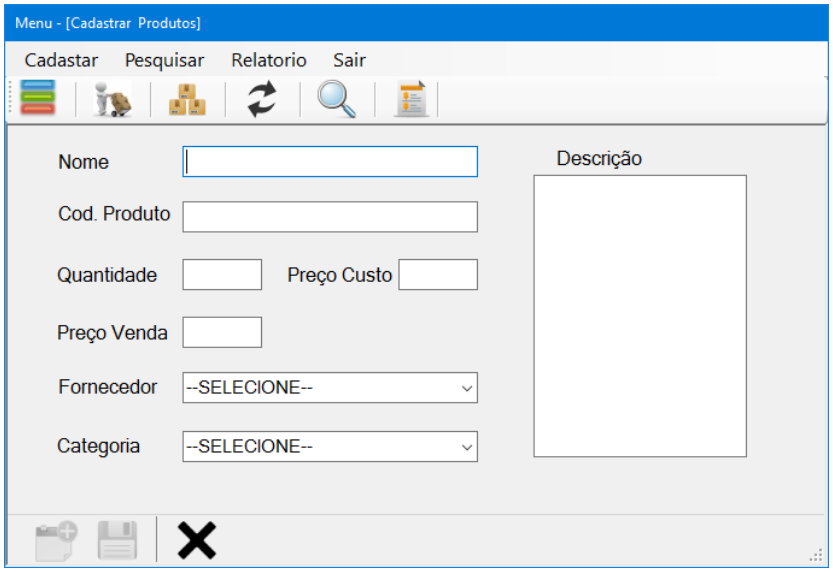


Figura 9. *Interface* de cadastro de produto.

4.4.5 Entrada e saída de produtos

Na tela de cadastro de movimentações de produtos, conforme ilustração disponível na Figura 10, pode-se realizar movimentações de entrada e saída de estoque. Caso for uma movimentação de entrada de estoque, o funcionário irá informar o produto, quantidade de entrada em estoque e a data em que a movimentação está ocorrendo para efetuá-la. Se for uma movimentação de saída de estoque, terá a interação com o cliente, o qual irá efetuar a compra de produto(s) através do atendimento por um funcionário para realizar a movimentação.

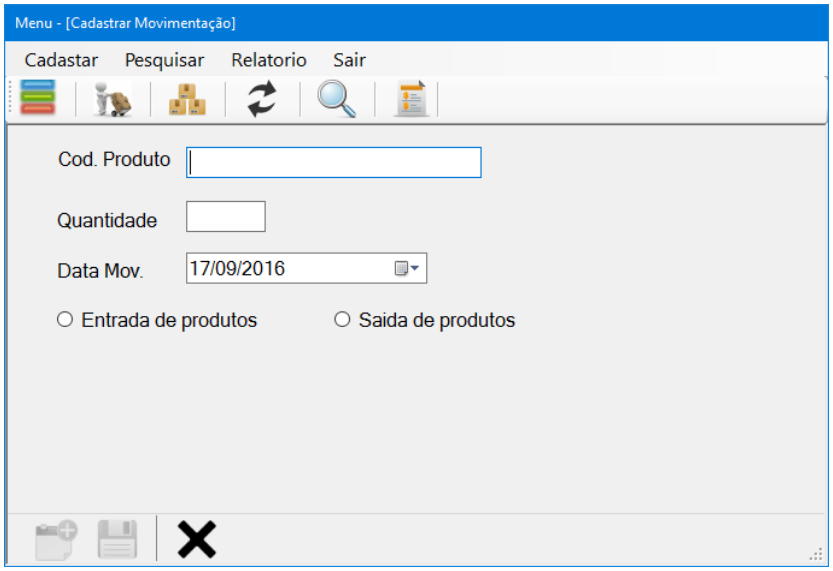


Figura 10. *Interface* de cadastro de movimentações.

4.4.6 Consulta de produtos, categorias, fornecedores e movimentações

Através da tela de consulta representada pela Figura 11**,** o funcionário terá acesso a todas as informações cadastradas no sistema de controle de estoque como: categorias de produtos existentes; fornecedores responsáveis pelo fornecimento de produtos; produtos disponíveis para venda; movimentações de entrada de estoque obtidas através da compra de produtos por fornecedores e saída de estoque realizadas através da venda de produtos para clientes.

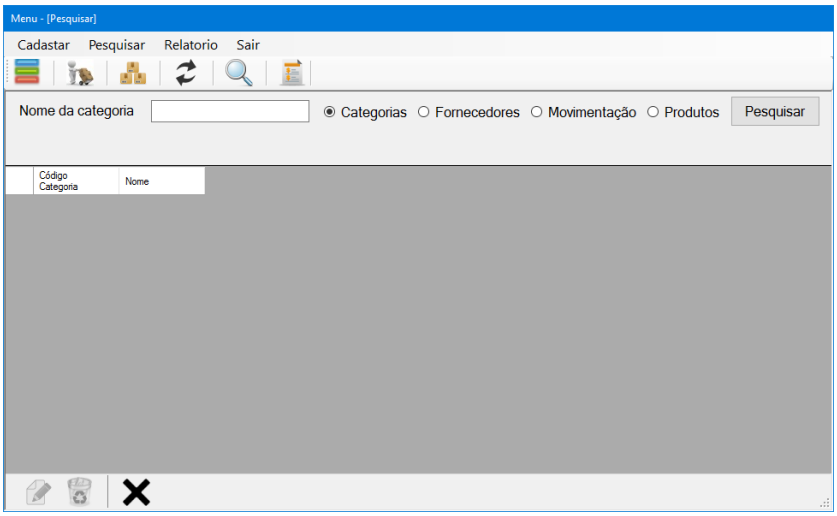


Figura 11. *Interface* de pesquisas em geral.

5 ESTUDO DE CASO

A fim de alcançar os objetivos propostos neste trabalho e solucionar o problema de pesquisa foi criado uma ferramenta para efetuar a automação de testes do sistema de controle de estoque. Foi implementado a automação de testes das *interfaces* de cadastro de usuário, produtos, fornecedores, categorias e consultas em geral.

5.1 Projeto

O desenvolvimento da ferramenta de automação de testes do sistema de controle de estoque necessita da versão Ultimate ou Premium do Visual Studio, pois essas versões são as únicas que oferecem suporte à criação de testes de *interface* de usuário codificados (Coded UI Test).

5.1.1 Criando um projeto Coded Ui Test

Os testes de *interface* de usuário devem estar contidos em um projeto Coded Ui Test, para isso, no Visual Studio deve-se ir em *Add*, *New Project* e selecionar *Visual C#*, em seguida, escolher a opção presente na lista *Test*, desse modo será listado as opções de projetos para criação de testes automatizados, deve ser escolhido a opção Coded UI Test Project, a qual irá se utilizar para criar a ferramenta de automação, conforme Figura 12.

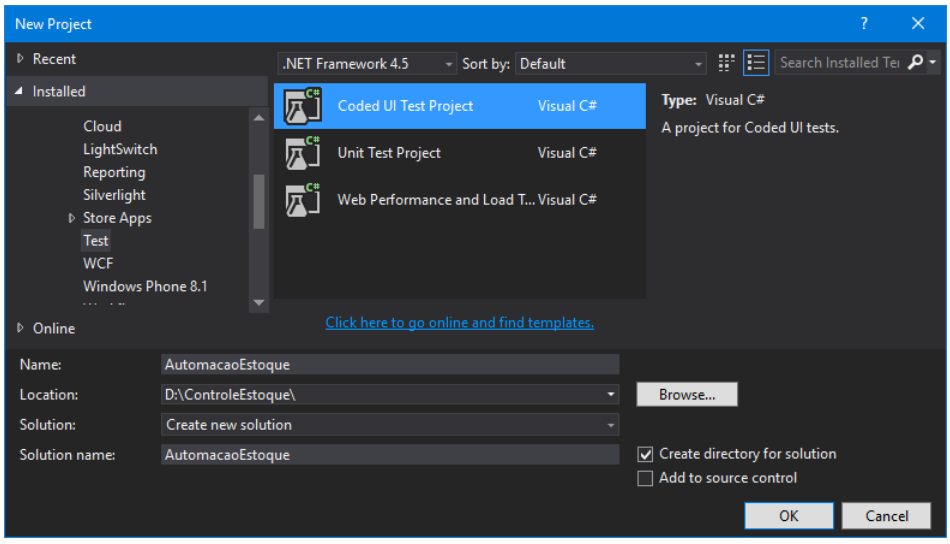


Figura 12. Criação de projeto Coded UI Test.

5.1.2 Implementando a automação da interface de cadastro de produtos

Com o projeto criado, deve-se adicionar um novo arquivo do tipo Coded UI Test ao projeto, para isso, no projeto deve-se abrir o menu de atalhos e adicionar um novo item do tipo Coded UI Test, conforme Figura 13.

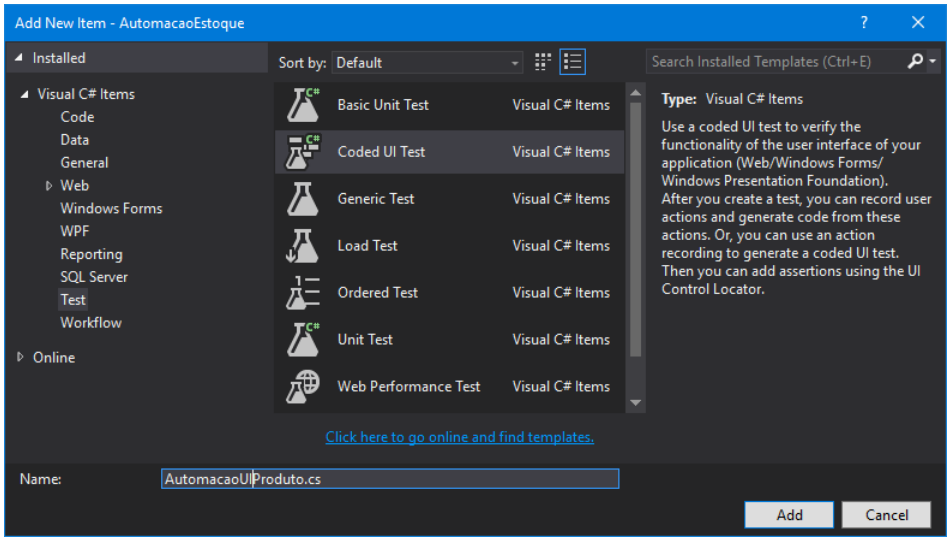


Figura 13. Criação de arquivo Coded UI Test.

Esta classe terá inicialmente a seguinte estrutura conforme Listagem 1, nela pode-se visualizar que a classe possui a assinatura [*CodedUITest*], a qual identifica que a mesma é uma classe teste de *interface* do usuário codificado. Para identificar que um determinado método será de testes deve ser adicionado o atributo [*TestMethod*] acima do mesmo, por padrão, inicialmente já é criado na estrutura da classe um método definido como sendo um método de teste.

Listagem 1. Código fonte arquivo ProdutoTest.

```
1. (...)
2. [CodedUITest]
3. public class ProdutoTest
4. {
5. private TestContext testContextInstance;
6. public TestContext TestContext
7. {
8. get { return testContextInstance; }
9. set { testContextInstance = value; }
10. }
11. [TestMethod]
12. public void CadastrarProduto()
13. {
14. }
15. }
16. (...)
```

Após criar o arquivo, será exibida uma caixa de diálogo solicitando para escolher a opção em que deseja-se realizar a criação dos testes de *interface* de usuário codificado, onde deverá ser realizada a escolha da opção de gravar ações, editar o mapa de *interface* do usuário ou adicionar asserções.

Desse modo, será exibido o construtor de testes de *interface* de usuário codificado (Coded UI Test), que possui as seguintes funcionalidades numeradas na Figura 14:

1- Iniciar e pausar a gravação do mapeamento do teste.

2- Editar etapas mapeadas no teste

3- Adicionar afirmações

4- Gerar o código do teste mapeado

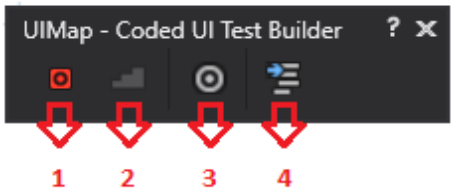


Figura 14. Construtor de testes de *interface* de usuário codificado

Para iniciar a gravação dos testes da *interface* de cadastro de produto do sistema de controle de estoque deve-se escolher a funcionalidade “1” referente a Figura 14, a qual iniciará a gravação das ações efetuadas na *interface* de cadastro de produtos. Com isso, na *interface* de cadastro de produtos, realiza-se o cadastro de um produto para que seja efetuado o mapeamento das ações realizadas na tela, após concluir todas as ações julgadas necessárias para efetuar os testes da *interface* de cadastro de produto, deve-se clicar no botão referente a funcionalidade “1” para pausar a gravação das ações.

Caso tenha sido realizada alguma ação incorreta durante o mapeamento do teste, após realizar a pausa da gravação, no botão referente à funcionalidade “2” (Editar etapas mapeadas no teste) pode-se realizar a edição das ações efetuadas durante a gravação.

Utilizando a funcionalidade “3” (Adicionar Afirmações) é possível comparar o valor de um campo da *interface* com outro valor, como no caso de cálculos, assim, caso seu valor estiver incorreto é possível exibir uma mensagem de erro.

Ao finalizar a gravação e edição dos testes da tela de cadastro de produto, para gerar o código mapeado deve-se escolher o botão referente à funcionalidade “4” (Gerar o código do teste mapeado) e informar o nome e uma descrição para o método de teste, desse modo o Visual Studio irá realizar a geração do código fonte de todo o teste mapeado.

Na classe *ProdutoTest* do tipo *Coded UI Test*, ao gerar o código foi adicionado no método CadastroProduto uma chamada para o método de teste gerado na classe *UIMap* pelo construtor de testes conforme representado pela Listagem 2.

Listagem 2. Código fonte classe ProdutoTest.

```
1. (...)
2. [TestMethod]
3. public void CadastrarProduto()
4. {
5. this.UIMap.CadastrarProduto();
6. }
7. (...)
```

Ao efetuar a geração do código fonte através do construtor, também é criado um arquivo chamado UIMap.uitest se ainda não existir no projeto, caso contrário apenas será acrescentado no código fonte as informações referentes ao teste mapeado. Na Listagem 3 é possível visualizar a parte principal do código fonte no qual foi gerado ao realizar o mapeamento da *interface* de cadastro de produtos.

Listagem 3. Código fonte classe UIMap.

```
1. (...)
2. public void CadastrarProduto()
3. {
4. #region Variable Declarations
5. WinTitleBar uIMenuTitleBar = this.UIMenuWindow.UIMenuTitleBar;
6. WinMenuItem uIProdutosMenuItem =
   this.UIMenuWindow.UIMsMenuMenuBar.UICadastarMenuItem.UIProdutosMenuItem;
7. WinEdit uITxtNomeEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtNomeWindow.UITxtNomeEdit;
8. WinEdit uITxtCodProdutoEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtCodProdutoWindow.UITxtCodProd
   utoEdit;
9. WinEdit uITxtQuantidadeEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtQuantidadeWindow.UITxtQuantid
   adeEdit;
10. WinEdit uITxtPrecoCustoEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtPrecoCustoWindow.UITxtPrecoCu
   stoEdit;
11. WinEdit uITxtPrecoVendaEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtPrecoVendaWindow.UITxtPrecoVe
   ndaEdit;
12. WinComboBox uICbFornecedorComboBox =
   this.UIMenuWindow.UICadastrarProdutosWindow.UICbFornecedorWindow.UICbFornecedo
   rComboBox;
13. WinComboBox uICbCategoriasComboBox =
   this.UIMenuWindow.UICadastrarProdutosWindow.UICbCategoriasWindow.UICbCategoria
   sComboBox;
14. WinEdit uITxtDescricaoEdit =
   this.UIMenuWindow.UICadastrarProdutosWindow.UITxtDescricaoWindow.UITxtDescrica
   oEdit;
15. WinButton uISalvarButton =
   this.UIMenuWindow.UICadastrarProdutosWindow.UIStatusStrip1StatusBar.UISalvarBu
   tton;
16. WinButton uIOKButton = this.UIMensagemWindow.UIOKWindow.UIOKButton;
17. WinButton uIFecharButton =
   this.UIMenuWindow.UICadastrarProdutosWindow.UIStatusStrip1StatusBar.UIFecharBu
   tton;
18. #endregion
19. Mouse.Click(uIMenuTitleBar, new Point(52, 2));
20. Mouse.Click(uIProdutosMenuItem, new Point(80, 15));
21. uITxtNomeEdit.Text = this.CadastrarProdutoParams.UITxtNomeEditText;
```

```
22. Keyboard.SendKeys(uITxtNomeEdit,
    this.CadastrarProdutoParams.UITxtNomeEditSendKeys, ModifierKeys.None);
23. uITxtCodProdutoEdit.Text =
    this.CadastrarProdutoParams.UITxtCodProdutoEditText;
24. Keyboard.SendKeys(uITxtCodProdutoEdit,
    this.CadastrarProdutoParams.UITxtCodProdutoEditSendKeys, ModifierKeys.None);
25. uITxtQuantidadeEdit.Text =
    this.CadastrarProdutoParams.UITxtQuantidadeEditText;
26. Keyboard.SendKeys(uITxtQuantidadeEdit,
    this.CadastrarProdutoParams.UITxtQuantidadeEditSendKeys, ModifierKeys.None);
27. uITxtPrecoCustoEdit.Text =
    this.CadastrarProdutoParams.UITxtPrecoCustoEditText;
28. Keyboard.SendKeys(uITxtPrecoCustoEdit,
    this.CadastrarProdutoParams.UITxtPrecoCustoEditSendKeys, ModifierKeys.None);
29. uITxtPrecoVendaEdit.Text =
    this.CadastrarProdutoParams.UITxtPrecoVendaEditText;
30. Keyboard.SendKeys(uITxtPrecoVendaEdit,
    this.CadastrarProdutoParams.UITxtPrecoVendaEditSendKeys, ModifierKeys.None);
31. uICbFornecedorComboBox.SelectedItem =
    this.CadastrarProdutoParams.UICbFornecedorComboBoxSelectedItem;
32. uICbCategoriasComboBox.SelectedItem =
    this.CadastrarProdutoParams.UICbCategoriasComboBoxSelectedItem;
33. uITxtDescricaoEdit.Text = this.CadastrarProdutoParams.UITxtDescricaoEditText;
34. Mouse.Click(uISalvarButton, new Point(18, 22));
35. Mouse.Click(uIOKButton, new Point(43, 16));
36. Mouse.Click(uIFecharButton, new Point(14, 10));
37. }
38. (...)
```

No código presente na Listagem 3, pode-se visualizar que cada componente no qual foi executado alguma ação durante a gravação do teste de *interface* tornou-se uma variável no código gerado, para cada tipo de componente é criado uma variável com tipagem específica como por exemplo o componente de texto é criado com tipo *WinEdit*, a caixa de seleção é criada com tipo *WinComboBox*, botões de ação com tipagem *WinButton*. Também há alguns eventos como *Mouse.Click*, o qual recebe a posição na tela onde irá ocorrer o *click* e *Keyboard.SendKeys*, o qual irá escrever no componente da *interface*, como um campo de texto por exemplo a informação que estiver recebendo por parâmetro.

Essas informações referente aos parâmetros estão em uma classe criada através da geração automática do código fonte. Na Listagem 4 observa-se os parâmetros gerados com base nas informações cadastradas durante a geração do teste de automação da interface de cadastro de produtos.

Listagem 4. Código fonte classe de parâmetros.

```
1. (...)
2. public class CadastrarProdutoParams
```

```
3.  {
4.  #region Fields
5.  ///<summary>
6.  /// Type 'Notebook Asus i7' in 'txtNome' text box
7.  ///</summary>
8.  public string UITxtNomeEditText = "Notebook Asus i7";
9.  ///<summary>
10. /// Type '{Tab}' in 'txtNome' text box
11. ///</summary>
12. public string UITxtNomeEditSendKeys = "{Tab}";
13. ///<summary>
14. /// Type '106610' in 'txtCodProduto' text box
15. ///</summary>
16. public string UITxtCodProdutoEditText = "106610";
17. ///<summary>
18. /// Type '{Tab}' in 'txtCodProduto' text box
19. ///</summary>
20. public string UITxtCodProdutoEditSendKeys = "{Tab}";
21. ///<summary>
22. /// Type '120' in 'txtQuantidade' text box
23. ///</summary>
24. public string UITxtQuantidadeEditText = "120";
25. ///<summary>
26. /// Type '{Tab}' in 'txtQuantidade' text box
27. ///</summary>
28. public string UITxtQuantidadeEditSendKeys = "{Tab}";
29. ///<summary>
30. /// Type '4000' in 'txtPrecoCusto' text box
31. ///</summary>
32. public string UITxtPrecoCustoEditText = "4000";
33. ///<summary>
34. /// Type '{Tab}' in 'txtPrecoCusto' text box
35. ///</summary>
36. public string UITxtPrecoCustoEditSendKeys = "{Tab}";
37. ///<summary>
38. /// Type '4500' in 'txtPrecoVenda' text box
39. ///</summary>
40. public string UITxtPrecoVendaEditText = "4500";
41. ///<summary>
42. /// Type '{Tab}' in 'txtPrecoVenda' text box
43. ///</summary>
44. public string UITxtPrecoVendaEditSendKeys = "{Tab}";
45. ///<summary>
46. /// Select 'Fornecedor Informatica LTDA' in 'cbFornecedor' combo box
47. ///</summary>
48. public string UICbFornecCboSelectedItem = "Fornecedor Informatica LTDA";
49. ///<summary>
50. /// Select 'Informatica' in 'cbCategorias' combo box
51. ///</summary>
52. public string UICbCategoriasComboBoxSelectedItem = "Informatica";
53. ///<summary>
54. /// Type 'notebook' in 'txtDescricao' text box
55. ///</summary>
56. public string UITxtDescricaoEditText = "notebook";
```

```
57. #endregion
58. }
59. (...)
```

Para concluir a implementação do teste de automação da *interface* de cadastro de produto, devem ser realizados dois ajustes no código fonte da automação.

Dentre eles, o primeiro ajuste a ser realizado é a criação de um método que irá efetuar a inicialização da aplicação de controle de estoque, conforme Listagem 5, onde primeiramente é validado se o programa já está em execução para assim efetuar seu encerramento, após isso, utilizando o método *Launch* da classe *ApplicationUnderTest* é efetuado a inicialização da aplicação de controle de estoque para iniciar a execução dos testes de automação selecionados.

Listagem 5. Método para inicialização do programa.

```
1.  public ApplicationUnderTest InicializaAplicacao(ApplicationUnderTest appTest)
2.  {
3.  Framework.TaskKillByImage("ControleEstoque.exe");
4.  try
5.  {
6.        appTest= ApplicationUnderTest.Launch(@"D:\ControleEstoque.exe");
7.  }
8.  catch
9.  {
10. Assert.Fail("O teste falhou! A aplicação não iniciou corretamente!");
11. }
12. return appTest;
13. }
```

O ajuste seguinte necessário de ser realizado é a validação do efetuamento do *login* primeiramente antes de efetuar qualquer outro teste de automação no sistema. Desse modo, conforme Listagem 6, no método “IniciarTestes”, é efetuado uma validação para identificar se a *interface* de *login* está ativa no momento. Caso estiver, será realizado o *login* no sistema para prosseguir com a execução da automação do programa.

Listagem 6. Método para validação tela login**.**

```
1.  public void IniciarTestes(UIMap UIMap)
2.  {
3.  if (UIMap.UILoginWindow.WaitForControlReady())
4.  {
5.      UIMap.Login();
6.  }
7.  }
```

5.1.3 Executando os testes de automação no sistema

Com a implementação completa do Coded UI Test, para executar os testes automatizados criados deve-se abrir o gerenciador de testes. Este se encontra no menu de atalho *Test*, item Windows onde, deve-se selecionar a opção *Test Explorer* conforme Figura 15.

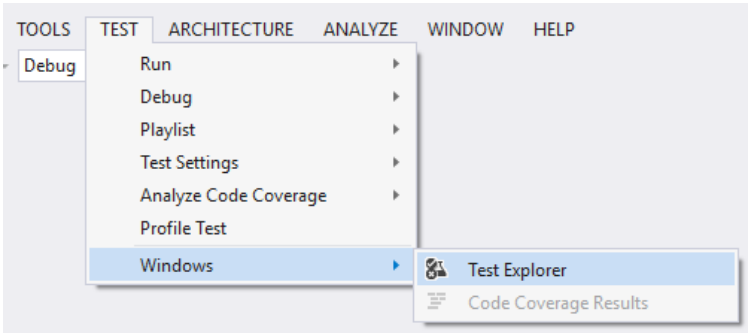


Figura 15. Menu gerenciador de testes.

Após abrir o gerenciador de testes, ilustrado na Figura 16, tem-se os métodos de testes listados onde podem ser executados utilizando a opção *Run All*, esta opção irá executar todos os testes presentes na lista, ou também pode-se optar por executar apenas o(s) teste(s) desejado(s) clicando sobre o mesmo e selecionando a opção *Run Selected Tests*.

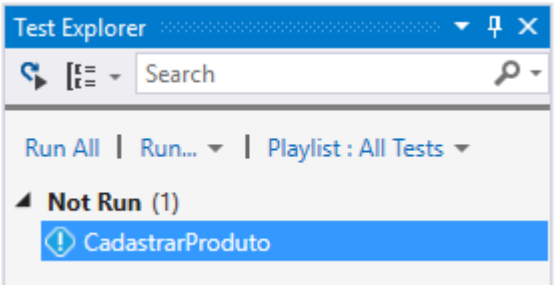


Figura 16. Janela de gerenciamento de testes.

Ao realizar a execução dos testes automatizados, assim que forem finalizados, os mesmos serão enquadrados em duas categorias. A primeira que representa os testes que obtiveram sucesso na execução e a segunda que representa os testes que não foram aprovados em sua execução por apresentarem alguma divergência.

Após a execução dos nove casos de teste da automação do sistema de controle de estoque, foi possível identificar que apenas cinco deles obtiveram sucesso em sua execução, nos demais

casos de testes foram encontrados alguns erros como: validação de campos incorreta, limite de caracteres maior que tamanho configurado no banco de dados, formatação de dados inadequados.

Desse modo, é possível compreender a necessidade de utilização dos testes automatizados para complementarem os demais testes de um sistema, pois assim é possível evitar com que erros como os encontrados anteriormente na execução da automação e outros como cálculos de todos os tipos cheguem até o cliente e acabem se tornando um problema ainda maior.

6 CONSIDERAÇÕES FINAIS

Conforme apresentado no presente trabalho, a automação de testes é uma boa opção para complementar os demais testes de software e, assim, obter melhor qualidade do produto final a ser entregue para o cliente ou usuário. O diferencial da automação de testes, a qual foi implementada para o sistema de controle de estoque, é a remoção da necessidade de pessoas para realizar os testes gerais de todo o sistema após fazer alguma alteração no mesmo, assim, sendo necessário apenas executar a automação do sistema para validar suas regras e comportamentos de interface.

O trabalho desenvolvido propôs a criação de uma ferramenta para realizar a automação de teste do sistema de controle de estoque e assim, demonstrar as vantagens em utilizá-la. Este, partiu da ideia de automatizar o sistema proposto a fim de, através dos testes automatizados, validá-lo e obter o resultado de sua execução, utilizando a ferramenta de desenvolvimento de software Visual Studio para realizar sua implementação.

Após a realização do desenvolvimento do projeto, notou-se que a execução da ferramenta de automação de testes no sistema de controle de estoque obteve êxito e foi possível identificar erros no programa de controle de estoque após sua execução.

Desse modo, conclui-se que a automação de testes é essencial para complementar os demais testes de software, pois utilizando-a é possível detectar erros que podem ter passados despercebidos nas demais etapas de testes, e assim evitando com que estes sejam encontrados por cliente ou usuários do sistema.

Como trabalhos futuros pode-se apontar a realização de um estudo comparativo entre o *framework* de automação de testes utilizado neste trabalho com outros dos principais *frameworks* utilizados para este mesmo fim, demonstrando assim, pontos positivos e negativos de cada um deles.

7 REFERÊNCIAS BILIOGRÁFICAS

BEZERRA, Filipe. **O que é um Stakeholders?,** jul. 2014. Disponível em: < https://goo.gl/N4i6kO>. Acessado em 10 nov. 2016.

DEITEL, H. M et al. **C#: Como programar**. São Paulo: Person Makron Books, 2003. 1152 p.

IZABEL, Leonardo R. P. **Testes automatizados no processo de desenvolvimento de softwares**. Rio de Janeiro: Universidade Federal do Rio De Janeiro, 2014.

MSDN. **Introdução à linguagem C # e .NET Framework**, jul. 2016. Disponível em: < https://goo.gl/H9X5C5>. Acessado em 15 nov. 2016.

MSDN. **Usar automação de interface do usuário para testar código**, jul. 2016. Disponível em: <https://goo.gl/pqrPT3>. Acessado em 13 ago. 2016.

MULLER, Nícolas. **Framework, o que é e para que serve?**, nov. 2008. Disponível em: <https://goo.gl/OUWSDp>. Acessado em 20 jul. 2016.

NETO, Arilo C. D. **Introdução a Teste de Software**. [Editorial]. Engenharia de Software, v. 1, p. 54-59, jun., 2010.

RIOS Emerson, MOREIRA Trayahú R. **Teste de Software**. 3ª Ed. Rio de Janeiro: Alta Books, 2013. 304 p.

VICCARI, Leonardo Davi. **Automação de teste de software através de linhas de produtos e testes baseados em modelos**. Porto Alegre: Pontifícia Universidade Católica do Rio Grande do Sul, 2009.