PUC-SP

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

Rafael Botossi

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA DE AUTOMAÇÃO DE TESTES DO TIPO REGISTRO/REPRODUÇÃO PARA AMBIENTES *DESKTOP WINDOWS* UTILIZANDO TÉCNICAS DE PADRÕES DE PROJETO.

São Paulo

2019

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

PUC-SP

Rafael Botossi

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA DE AUTOMAÇÃO DE TESTES DO TIPO REGISTRO/REPRODUÇÃO PARA AMBIENTES *DESKTOP WINDOWS* UTILIZANDO TÉCNICAS DE PADRÕES DE PROJETO.

Relatório apresentado à Pontifícia Universidade Católica de São Paulo, como parte dos requisitos exigidos na conclusão do curso de Pós-Graduação em Engenharia de Software.

São Paulo

2019

PUC-SP

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

Rafael Botossi

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA DE AUTOMAÇÃO DE TESTES DO TIPO REGISTRO/REPRODUÇÃO PARA AMBIENTES *DESKTOP WINDOWS* UTILIZANDO TÉCNICAS DE PADRÕES DE PROJETO.

Orientador

Professor Dr. Renato Manzan

São Paulo

2019

Agradecimentos

Gostaria de agradecer ao meu orientador por acreditar no sucesso do projeto desde a primeira menção que fiz sobre a ideia do trabalho em sala de aula. Sem seu apoio muito provavelmente teria perseguido outros caminhos e foi muito recompensador poder criar este protótipo e estudar este tema.

Também agradeço a mim mesmo por manter a ideia de criar um projeto deste tipo viva por mais de 15 anos, quando tive contato pela primeira vez com ferramentas de testes automatizados, através da ferramenta *TestParter* da *Compuware* e me apaixonei por suas funcionalidades quase mágicas de imitar as ações do usuário.

Outra pessoa que preciso agradecer é minha esposa Laura por seu constante apoio na busca por novos conhecimentos e seu amor quase genial pelos estudos, motivando-me a sempre buscar um nível acima.

Dedicatória

Dedico este trabalho à todas as pessoas que ainda desenvolvem em ambientes com sistemas legados, que não possuem muitas ferramentas para testar seus sistemas e que, como eu, buscam incessantemente aumentar a qualidade de seu trabalho independentemente do ambiente e ferramenta que estão desenvolvendo.

Glossário

- API - *Application Programming Interface*;
- MVP - *Minimum Viable Product*;
- IDE - Integrated development environment.

Resumo

Este trabalho visou a criação de um protótipo de sistema de automação de testes do tipo gravação e reprodução de ações de usuário no *Windows desktop* e foi concebido por causa da quantidade consideravelmente baixa de sistemas do tipo e por conta do custo elevado de compra das ferramentas existentes. Ferramentas deste tipo atualmente são voltadas para o mercado de desenvolvimento web e mobile, deixando assim uma gama de sistemas legados sem o apoio das ferramentas de automação de testes, motivando assim o trabalho.

Através do estudo teórico sobre o assunto e do estudo das ferramentas existentes, foi possível traçar uma linha-guia das funcionalidades mínimas para o funcionamento do protótipo. Além disso, através da correta implementação de padrões de projeto para melhorar a robustez, manutenibilidade e segurança do protótipo, foi possível desenvolver o sistema com os requisitos essenciais para seu funcionamento, como a correta gravação das ações do usuário através do teclado e mouse, a criação de um script com as ações gravadas e a reprodução destas ações. Também foi possível desacoplar o projeto de interface gráfica de uso do sistema do projeto que realiza as ações, permitindo assim que trabalhos futuros implementem novas interfaces gráficas e/ou atualizem as funcionalidades existentes e implementem novas funcionalidades.

Palavras-chave: automatização de testes, teste de software, script de teste.

Abstract

This work was designed to create a tests automation system prototype of the type recording/playback of the actions of Windows desktop user and was conceived because of the relatively small number of systems of this type and the high cost of those tools. Tools of this type are currently aimed at the web and mobile development markets, leaving a wide range of legacy systems without the support of tests automation tools, thus motivating this work.

Through the theoretical study on the subject and the study of existing tools, it was possible to draw a guide of the minimum features for the prototype operation. In addition, through the correct implementation of design patterns, to improve the robustness, maintainability and security of the prototype, it was possible to develop the system with the essential requirements for its operation, such as the correct recording of user actions through the keyboard and mouse, the creation of a script with the recorded actions and the reproduction of these actions. It was also possible to uncouple the graphical user interface project from the project that perform the actions of recording and playback actions, thus allowing future works to implement new graphical user interfaces and/or update existing features and implement new features.

Keywords: test automation, software testing, test script.

Lista de Figuras

Lista de Tabelas

Índice

1. Introdução

Neste capítulo descreve-se a motivação para o trabalho proposto, bem como os objetivos do mesmo, suas delimitações e as contribuições esperadas em sua conclusão. Também será descrito o método de trabalho, a organização do texto e o cronograma para a execução das atividades.

1.1. Motivação

Desde o início do uso dos computadores no mundo comercial (RIOS; MOREIRA, 2013), os softwares lançados no mercado se caracterizam por apresentarem falhas relacionadas às funcionalidades, desempenho, segurança e outros. Este quadro ainda é vigente, o que indica que o uso de testes para garantir a qualidade de software está longe de ser o ideal. Ainda é comum ver equipes de desenvolvimento realizando os testes das funcionalidades que criaram e não repassando esse trabalho para uma equipe especializada em testes. Esse problema está relacionado a um número de fatores, entre eles o custo de se manter uma equipe especializada em testes. Mesmo nos casos onde a própria equipe de desenvolvimento também é responsável pelo teste, verifica-se em muitos casos o problema da robustez do teste (BASTOS et al., 2012), diminuindo assim a qualidade do software.

Na indústria brasileira, a utilização de testes de software ainda é realizada em grande parte de forma manual, muitas vezes somente após a criação de um módulo ou de um sistema inteiro. Essa abordagem manual e em muitos casos ad hoc leva à ocorrência de muitos problemas, tais como erros de regressão, logo ela deveria ser evitada (CHEQUE; KON, 2008). A utilização de ferramentas para automação de testes possibilita que um sistema possa crescer ao mesmo tempo em que suas funcionalidades são testadas de forma automática, possibilitando a diminuição da quantidade de problemas encontrados (CHEQUE; KON, 2011). Por conta deste cenário, diversas ferramentas para automatizar testes foram criadas, inicialmente para ambiente *desktop* e depois para ambiente *web*. Essas ferramentas, do tipo registro/reprodução de ações do usuário via teclado e mouse

realizam esta gravação para posterior reprodução das ações gravadas, gerando assim um caso de uso de teste que pode ser reproduzido a cada alteração no sistema desenvolvido.

Ao analisar o mercado atual, de forma geral, as ferramentas de automação de teste de software gratuitas são utilizadas de forma massiva no apoio às aplicações web, por causa da grande quantidade de aplicações criadas para esse ambiente. Este cenário, aliado à informação que as ferramentas atuais de automatização de testes para *desktop* possuem custo elevado e por vezes proibitivos para pequenas empresas, fazem com que toda uma gama de sistemas criados para ambientes desktop fique sem o apoio expressivo de ferramentas de testes automatizados (10 ferramentas de automação de testes mais usadas,07-2018) (ANDERSON, 2017), (Top 20 Automation testing tools, 07-2018), potencialmente aumentando assim a quantidade de problemas de qualidade dos sistemas criados para Windows *desktop.*

1.2. Objetivos

Este trabalho tem como objetivo desenvolver um protótipo de um sistema de automação de testes funcionais com *interface* gráfica do tipo registro/reprodução para ambiente *desktop Windows*, utilizando técnicas de padrões de projeto para facilitar a criação e manutenibilidade do projeto ao longo do tempo.

O protótipo de sistema consistirá de uma área de gravação das ações do usuário realizadas por mouse e teclado, que serão então salvas em um *script* para que seja possível realizar a reprodução dessas ações em um momento posterior.

1.3. Delimitações

É importante ressaltar que o protótipo não irá realizar a implementação de interpretadores de linguagens de programação para que o *script* possa ser lido e alterado por um programador para realização de automação de testes diferentes dos testes que foram gravados.

1.4. Contribuições

A principal contribuição deste trabalho é a criação de um protótipo de sistema para automação de testes do tipo registro/reprodução para ambientes *desktop Windows*, utilizando padrões de projeto e técnicas de engenharia de software para a composição de um código de fácil manutenibilidade e de fácil entendimento, pois os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente (GAMMA et al., 2008). Este protótipo poderá servir como ferramenta de automação de testes para pequenas empresas que não possuem nenhum tipo de teste automatizado para sistemas *desktops*. Também servirá como base para projetos futuros de automação de testes que possam implementar a criação/manutenção de *scripts* para facilitar o uso da ferramenta.

1.5. Método de Trabalho

A pesquisa será realizada através das atividades descritas abaixo:

1.5.1. Estado da Arte

Nesta etapa serão levantados trabalhos com temática similar ao sistema proposto para análise das tecnologias existentes e viabilidade do projeto, bem como verificação de melhores práticas ao desenvolver sistemas deste tipo.

1.5.2. Definição dos processos existentes em um sistema do tipo proposto

Através das informações levantadas na etapa de pesquisa bibliográfica, serão descritos os principais processos existentes em sistemas de automação de testes do tipo registro/reprodução para análise da viabilidade das funcionalidades em etapas futuras.

1.5.3. Levantamento de requisitos e escopo do sistema

Após a definição dos processos existentes em sistemas similares ao proposto, serão definidos os requisitos funcionais que o sistema deverá atender,

bem como o escopo do projeto, para delimitar o que poderá ser criado e o que não fará parte deste projeto.

1.5.4. Levantamento de tecnologias envolvidas

Definidos os requisitos e escopo do projeto, serão analisadas as tecnologias que poderão ser utilizadas para a criação do sistema proposto.

1.5.5. Criação de um produto mínimo viável (MVP) de sistema para verificar viabilidade de criação de um sistema do tipo proposto

Para garantir a possibilidade de criação de um sistema deste tipo na tecnologia escolhida na etapa anterior, será criado um protótipo básico que atende os requisitos funcionais básicos levantados nas etapas anteriores.

1.5.6. Criação do protótipo

Após a avaliação dos requisitos, será criado o sistema proposto, de acordo com as especificações levantadas.

1.5.7. Criação e execução de testes para o protótipo desenvolvido

Em sequência à criação do protótipo, serão criados e executados cenários de testes para garantir a qualidade do sistema proposto.

1.5.8. Análise dos resultados e entrega do protótipo

Após a criação e testes do sistema proposto, serão analisados os resultados obtidos e será verificado se o projeto atendeu às expectativas descritas no capítulo 1, bem como serão descritas melhorias que poderão ser implementadas no futuro.

1.6. Organização do texto

No capítulo 2, "Estado da arte", serão apresentados trabalhos similares aos do sistema proposto, quando houver, para traçar um paralelo ao que está disponível no mercado e o que será realizado durante o trabalho.

No capítulo 3, "Requisitos funcionais para a criação do sistema proposto e análise da tecnologia envolvida", serão descritas as funcionalidades presentes em sistemas de automação de testes do tipo registro/reprodução das ações do usuário no *Windows*, serão descritos os requisitos funcionais que o sistema deverá atender, definindo assim o escopo do projeto e também será realizada a análise da tecnologia envolvida na criação do protótipo.

No capítulo 4, "Prototipação básica para verificação da viabilidade do projeto", será abordada a criação do protótipo básico para verificação da viabilidade do projeto.

No capítulo 5, "Criação do protótipo de sistema de automação de testes funcionais com *interface* gráfica do tipo registro/reprodução para ambiente desktop *Windows*", será descrito como o sistema foi criado, bem como suas funcionalidades e informações relevantes ao projeto.

No capítulo 6, "Criação e execução de testes", será abordada a forma que os testes foram criados, sua execução e o resultado obtido.

No capítulo 7, "Considerações finais", será analisado o resultado da criação do protótipo de sistema, bem como traçado um paralelo em relação ao que foi proposto e o que foi obtido. Também serão indicadas melhorias em futuras implementações no protótipo e sugestões para trabalhos futuros.

2. Estado da arte

Neste capítulo será apresentada a base teórica para a necessidade proposta neste trabalho. Além disso serão listados sistemas similares aos do sistema proposto, quando houver, para traçar um paralelo ao que está disponível no mercado e o que será realizado durante o projeto.

2.1. Introdução

Existem vários tipos de testes de software. Eles podem ser realizados por diferentes pessoas, em diversas fases do projeto e com objetivos distintos. É importante conhecê-los para entender a sua importância e a sua aplicação durante todo o ciclo de vida do projeto de software (GONÇALVES, et al., 2019). Também existem diversas formas de visualizar os diferentes tipos de testes que existem, sob diversas visões. Algumas das formas que podemos visualizá-los é em relação às suas modalidades, em relação às suas naturezas e também em relação às formas como são realizados. Serão descritos nos tópicos abaixo estes tipos diferentes de visões do teste. Além disso, ainda neste capítulo, serão descritos os diferentes tipos de padrões de projeto existentes que poderão ser utilizados no desenvolvimento do sistema proposto em sua fase de desenvolvimento.

2.2. Tipos de testes em relação à sua modalidade

Existem três modelos diferentes de testes, caixa-branca, caixa-cinza e caixa-preta.

2.2.1. Modelo Caixa-Preta

Neste modelo, o teste é realizado sem que se saiba o código do software, ou seja, normalmente a interação é somente com a interface do sistema. São passadas informações de entrada e verificam-se se as saídas estão de acordo com o esperado.

2.2.2. Modelo Caixa-Branca

Neste modelo, o teste é realizado com conhecimento do código-fonte do sistema, ou seja, o teste é realizado na estrutura e lógica do software.

2.2.3. Modelo Caixa-Cinza

Neste modelo, mescla dos modelos Caixa-Preta e Caixa-Branca, o teste é realizado com conhecimento do código-fonte, porém o teste é realizado somente através da interface do sistema.

2.3. Tipos de testes em relação à sua natureza

Serão descritos neste tópico os tipos de testes em relação à sua natureza. Os textos dos tipos de testes a seguir baseiam-se nos tipos de testes descritos em (GONÇALVES, et al. 2019).

2.3.1. Testes de aceitação do utilizador

Usuário utilizam casos de uso e cenários para verificar que o sistema está coerente com o necessário.

2.3.2. Teste de configuração ou instalação

São realizados testes para verificar o comportamento do software quando instalado em diferentes configurações de hardware e software.

2.3.3. Teste de componentes

Os componentes do software são testados de forma isolada.

2.3.4. Teste de integração

Combinam-se os componentes para verificar seu funcionamento.

2.3.5. Teste de desempenho

Os testes focam no desempenho do sistema, de acordo com os requisitos não-funcionais.

2.3.6. Teste de estresse

O sistema passa por testes para avaliar o ponto de ruptura e as características das falhas encontradas ao atingi-lo.

2.3.7. Teste funcional

São realizados testes em cima dos requisitos funcionais, as funcionalidades e casos de uso.

2.3.8. Teste de integridade de sistema

O sistema passa por testes para verificar sua resistência a falhas.

2.3.9. Teste de montagem

Os componentes de software passam por testes em conjunto.

2.3.10. Teste de produto

O sistema é testado em relação aos seus requisitos funcionais.

2.3.11. Teste de performance

Testes de capacidade de resposta, disponibilidade e confiabilidade diante de determinada carga de trabalho, por determinado tempo.

2.3.12. Teste de segurança

São verificados se os dados e funcionalidades do sistema são acessados somente por quem deve acessá-los.

2.3.13. Teste de unidade

É testado um componente de código de forma isolada. Quem realiza este tipo de teste é o desenvolvedor, principalmente por ser um teste do tipo caixa-branca.

2.3.14. Teste de usabilidade

O foco do teste é a experiência do usuário, realizando análise em relação à *interface*, leiaute, funcionalidades e facilidade de utilização.

2.3.15. Teste de volume

Verifica o comportamento do sistema funcionando com capacidade “normal” em relação aos dados e transações de banco de dados em um longo período.

2.3.16. Outros tipos de teste

2.3.16.1. Teste de regressão

É o teste aplicado a versões mais recentes do software, para garantir que não existem novos defeitos nos módulos já analisados.

2.4. Tipos de testes em relação ao modo de se testar

2.4.1. Teste Manual

Os testes manuais são aqueles realizados por seres humanos, através de método *ad hoc*, navegando pelo software e realizando verificações sem o uso de script, ou através de ferramentas para auxiliar o teste, como checklists, ou seja, listas de tarefas a serem testadas. Existem diversos tipos de testes onde ainda é necessário o teste manual, como testes exploratórios, além de testes em pequenos sistemas e que só precisam ser realizados uma ou duas vezes, além de sistemas complexos onde a automatização pode ser cara de implementar.

2.4.2. Teste Automatizado

Testes automatizados, em oposição aos testes manuais, é a prática de tornar os testes de software independentes da intervenção humana. A independência da intervenção humana permite o aproveitamento dos benefícios de um computador, como a velocidade de execução, reprodutibilidade exata de um conjunto de ações, possibilidade de execução paralela de testes, flexibilidade na quantidade e momento das execuções dos testes e a facilidade da criação de casos complexos de testes (CHEQUE; KON, 2011).

Segundo Fester e Graham (1999), a automação de testes pode habilitar algumas tarefas de testes para serem feitas mais eficientemente do que poderiam ser feitas manualmente. Existem outros benefícios, como os listados abaixo:

1. Rodar testes de regressão existentes em uma nova versão de programas. Este é a tarefa mais óbvia, principalmente pelos motivos dos softwares estarem em constante manutenção.
2. Rodar mais testes mais frequentemente. Por conta da automação, os testes rodam mais rapidamente, e por conta disso é mais fácil rodar mais testes.
3. Realizar testes que seriam difíceis ou impossíveis de serem feitos manualmente. Tentar simular um ambiente de produção com, digamos 200 usuários, pode ser impossível, mas simular o input de 200 usuários pode ser simulado usando testes automatizados.
4. Melhor uso de recursos. Automatizar tarefas repetitivas pode melhorar o moral do time, e liberar testadores com experiência para outras tarefas mais difíceis.
5. Consistência e repetibilidade de testes. Os testes serão repetidos da mesma forma, melhorando a consistência dos testes.
6. Reuso de testes. Os testes automatizados podem ser reutilizáveis.

Dustin, et al (1999) também indicam na tabela abaixo o ganho em horas ao se realizar testes automatizados, comparando-os com o desempenho de realizar testes manuais. É importante ressaltar que na tabela, além das atividades da realização de testes, também são indicados tempos para o planejamento, o desenvolvimento do plano de testes, a análise do resultado e também os relatórios que indicam estes resultados.

Table 2.3 Manual Versus Automated Testing

Test Steps	Manual Testing (hours)	Automated Testing (hours)	Percentage Improvement with Tools
Test plan development	32	40	–25%
Test procedure development	262	117	55%
Test execution	466	23	95%
Test result analysis	117	58	50%
Error status/correction monitoring	117	23	80%
Report creation	96	16	83%
Total duration	1090	277	75%

Figura 1 - Testes manuais versus testes automatizados

2.5. Tipos de testes em relação ao sistema proposto

2.5.1. O protótipo de sistema proposto e os tipos de testes

O protótipo do sistema proposto neste trabalho irá atuar em partes específicas dos diversos tipos de testes.

Ele irá atuar na modalidade de caixa-preta, pois será um protótipo que irá gravar as ações do usuário via teclado e mouse, ou seja, irá atuar nas *interfaces* do software a ser testado. Em relação à natureza, poderá ser utilizado em mais de um tipo de teste, como os testes funcionais, de segurança, de performance, de desempenho, de produto, regressão, etc. E por último será do modo automatizado, ou seja, após a gravação do teste, será possível executá-lo sem a intervenção humana.

2.6. Algumas ferramentas de automação de testes do tipo gravação/reprodução das ações do usuário para ambiente *desktop Windows* e Web disponíveis atualmente

Serão descritos nos subcapítulos abaixo algumas das ferramentas mais utilizadas atualmente para realização de testes do tipo gravação/reprodução das ações do usuário. É importante ressaltar que em sua maioria, elas são ferramentas pagas e quando não são, suportam em sua maioria aplicações criadas para web.

2.6.1. Selenium (Aplicações *Web*)

Selenium é uma das ferramentas de automatização de testes para *browser open-source* mais conhecidas e utilizadas no mercado. Ele é dividido entre os produtos Selenium Remote Control (RC) e Selenium IDE, porém somente é possível realizar testes para aplicações criadas para a *web*.

- **Selenium IDE**: Produto para os *browsers Google Chrome* e *Firefox* para realizar simples gravações e reproduções de interações com o *browser*. A imagem abaixo mostra a interface do produto.

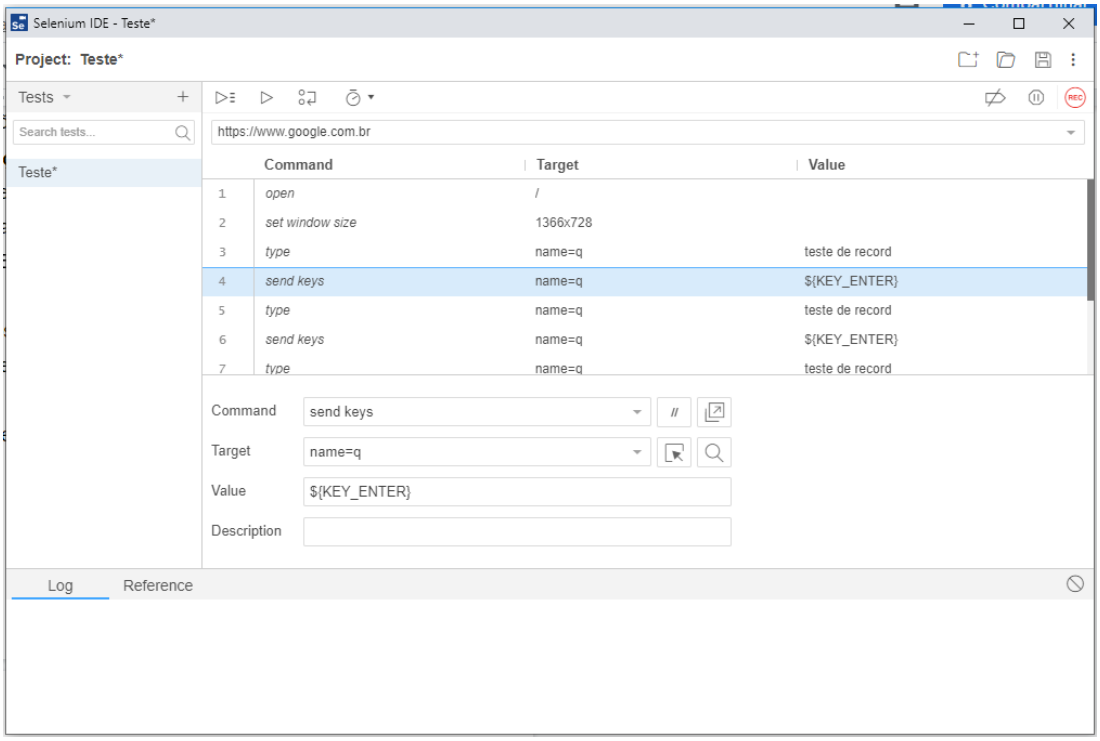


Figura 2 - Interface Selenium IDE

- **Selenium Remote Control (RC)**: Ferramenta específica para realizar testes com diversos *browsers* diferentes usando linguagens de mercado para criação de scripts. É necessário instalar um software chamado Selenium WebDriver em um local e o pacote específico da linguagem de programação que você deseja utilizar para a criação dos scripts. A figura abaixo demonstra a arquitetura deste sistema.

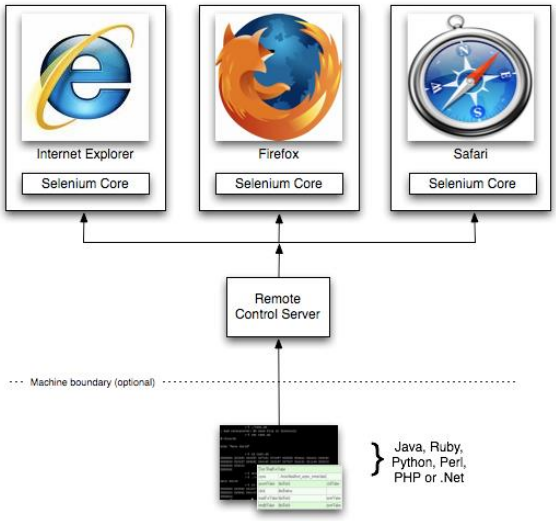


Figura 3 - Arquitetura Selenium RC

2.6.2. SmartBear TestComplete (Aplicações *desktop e web*)

A aplicação TestComplete é uma ferramenta criada para se realizar testes do tipo gravação e reprodução das ações do usuário tanto para aplicações desktop, quanto para aplicações web e mobile. Esta ferramenta não é open-source, sendo, portanto, paga. A imagem abaixo demonstra a interface do sistema.

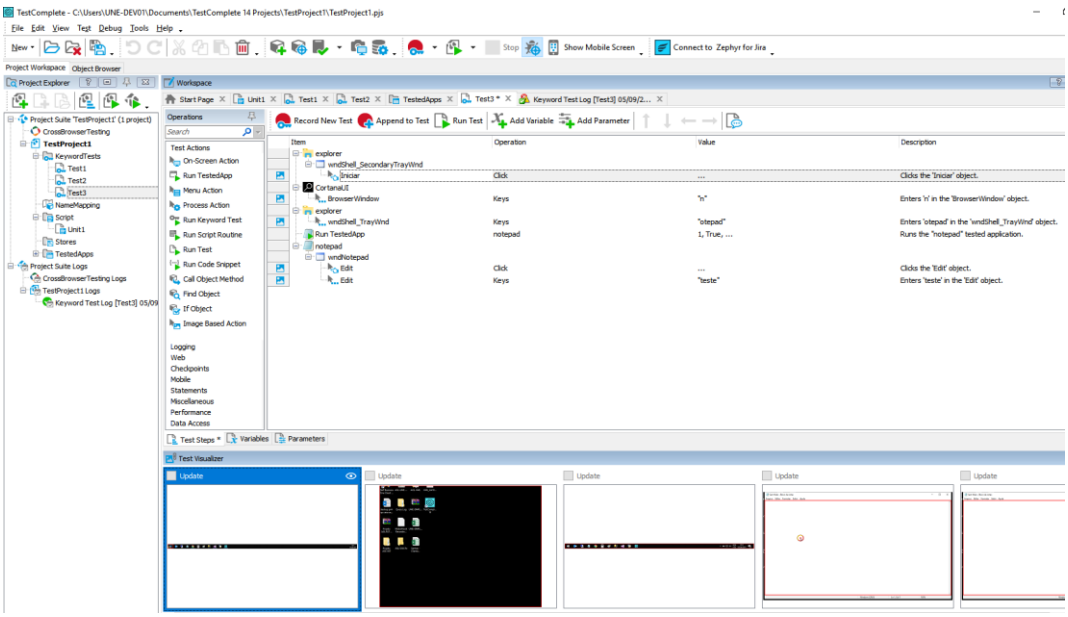


Figura 4 - Interface SmartBear TesteComplete

2.6.3. Telerik Test Studio

Muito parecida com a ferramenta TestComplete, a ferramenta Test Studio também realiza testes do tipo gravação e reprodução das ações do usuário para aplicações *desktop*, *web* e *mobile*, também sendo uma ferramenta paga. A imagem abaixo demonstra a interface do sistema.

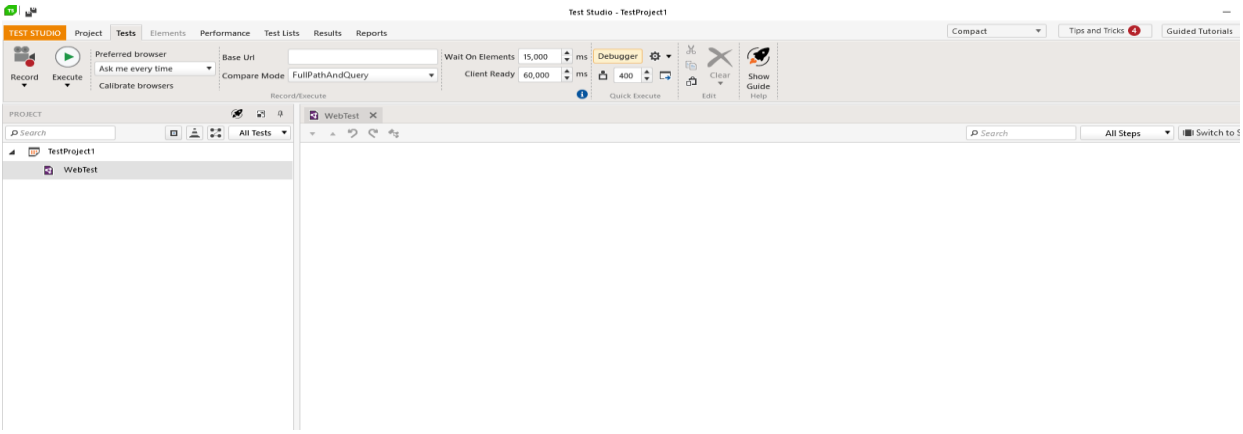


Figura 5 - Interface Telerik Test Studio

2.6.4. Micro Focus Unified Functional Testing (UFT One)

Anteriormente conhecido como QuickTest Professional da HP (Hewlett-Packard), esta ferramenta, assim como a TestComplete e a Telerik Test Studio, realiza testes em aplicativos criados para desktop, web e mobile. Esta ferramenta também é uma ferramenta paga. A imagem abaixo demonstra a interface do sistema.

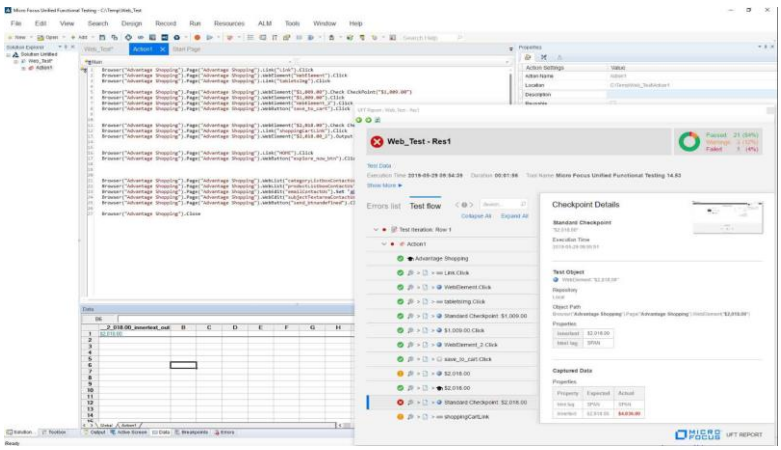


Figura 6 - Interface Micro Focus Unified Functional Testing

2.6.5. LogiGear TestArchitect

Seguindo a mesma lógica de produtos como TestComplete, Telerik Test Studio e HPE Unified Functional Testing, TestArchitect também realiza testes em aplicativos criados para desktop, web e mobile, sendo também pago. A imagem abaixo demonstra a interface do sistema.

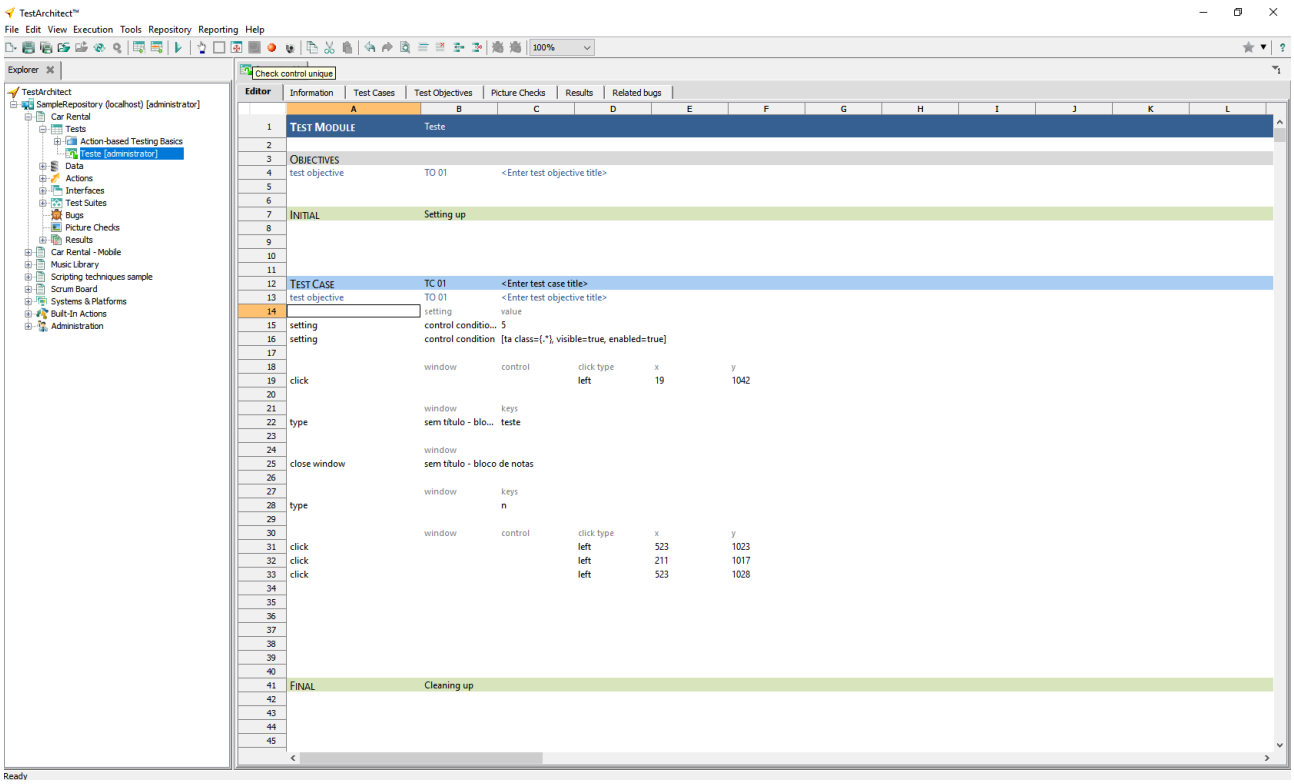


Figura 7 - Interface LogiGear TestArchitect

2.6.6. Ranorex Studio

Outra ferramenta parecida com as últimas ferramentas apresentadas, também para testes em aplicativos criados para desktop, web e mobile, também paga. A imagem abaixo demonstra a interface do sistema.

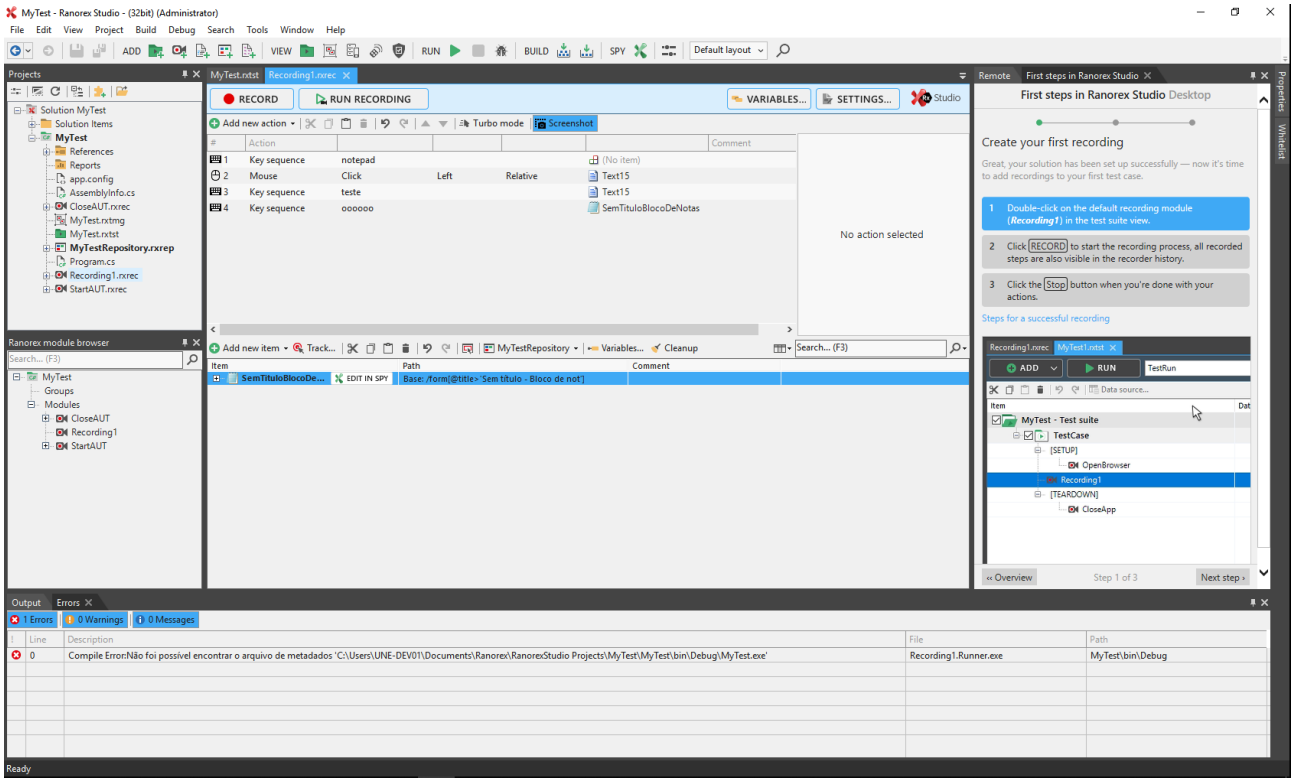


Figura 8 - Interface Ranorex Studio

A importância de uma ferramenta de automação de testes do tipo gravação/reprodução das ações do usuário para ambiente *desktop Windows* atualmente

Como indicado no tópico motivação do capítulo 1 e também através das ferramentas indicadas acima, ferramentas de automação do tipo gravação e reprodução das ações do usuário em ambiente *desktop* Windows que não são pagas são escassas. As ferramentas que são open-source são majoritariamente voltadas ao mercado de aplicativos *web*, deixando assim uma gama de aplicativos desktop, legados ou novos, sem o apoio de ferramentas que automatizam testes.

2.7. Padrões de Projeto

2.7.1. Introdução

Projetar software orientado a objeto não é fácil, mas projetar de forma a ser reutilizável é ainda mais difícil. O projeto deve ser específico para resolver o problema proposto, mas também genérico o suficiente para atender novos

problemas e requisitos futuros. Por conta desta necessidade, foram criados padrões de projetos, que nomeiam, abstraem e identificam os aspectos-chave de uma estrutura de projeto comum para torna-la útil para a criação de um projeto orientado a objetos reutilizável (GAMMA et al., 2008).

Cada padrão criado tem como tema um problema particular de projeto orientado a objeto, descrevendo a situação em que pode ser aplicado, as consequências, custos e benefícios de sua utilização (GAMMA et al., 2008).

Para este trabalho, foi definido que o uso de padrões de projeto poderiam beneficiar muito o projeto, pois além de facilitar a criação do sistema proposto, a ideia central do trabalho visa a criação de um protótipo de software que poderá ser utilizado em diversos projetos futuros, portanto seu código-fonte precisa estar de acordo com as ideias descritas nos tópicos acima.

2.7.2. Descritivo de alguns dos tipos de padrões de projeto

Neste tópico serão descritos brevemente alguns dos padrões de projetos mais utilizados, com o intuito de utilizá-los, quando possível, na criação do protótipo proposto neste trabalho. Todas as descrições utilizadas nos tópicos seguintes foram retiradas do livro ‘Padrões de Projeto’ (GAMMA et al., 2008).

2.7.2.1. Padrões de Criação

2.7.2.1.1. Padrão *Abstract Factory*

A intenção deste padrão é fornecer uma *interface* para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

É possível utilizar o padrão *Abstract Factory* quando:

- Um sistema deve ser independente de como seus produtos são criados, compostos ou representados;
- Um sistema deve ser configurado como um produto de uma família de múltiplos produtos;
- Uma família de objetos-produto for projetada para ser usada em conjunto, e é necessário garantir esta restrição;

- É necessário fornecer uma biblioteca de classes de produtos e é necessário revelar somente suas *interfaces*, não suas implementações.

2.7.2.1.2. Padrão *Builder*

A intenção deste padrão é separar a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações.

É possível utilizar o padrão *Builder* quando:

- O algoritmo para criação de um objeto complexo deve ser independente das partes que compõem o objeto e de como elas são montadas;
- O processo de construção deve permitir diferentes representações para o objeto que é construído.

2.7.2.1.3. Padrão *Factory Method*

Definir uma *interface* para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O *Factory Method* permite adiar a instanciação para subclasses.

É possível utilizar o padrão *Factory Method* quando:

- Uma classe não pode antecipar a classe de objetos que deve criar;
- Uma classe quer que suas subclasses especifiquem os objetos que criam;
- Classes delegam responsabilidade para uma dentre várias subclasses auxiliares, e é necessário localizar o conhecimento de qual subclasse auxiliar que é a delegada.

2.7.2.1.4. Padrão *Prototype*

A intenção deste padrão é especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia desse protótipo.

É possível utilizar o padrão *Prototype* quando um sistema tiver que ser independente de como os seus produtos são criados, compostos e representados; e

- Quando as classes a instanciar forem especificadas em tempo de execução, por exemplo, por carga dinâmica; ou
- Para evitar a construção de uma hierarquia de classes de fábricas paralela à hierarquia de classes de produto; ou
- Quando as instâncias de uma classe puderem ter uma dentre poucas combinações diferentes de estados. Pode ser mais conveniente instalar um número correspondente de protótipos e cloná-los, ao invés de instanciar a classe manualmente, cada vez com um estado apropriado.

2.7.2.1.5. Padrão *Singleton*

A intenção deste padrão é garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.

É possível utilizar o padrão *Singleton* quando:

- For preciso haver apenas uma instância de uma classe, e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido;
- A única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

2.7.2.2. Padrões Estruturais

2.7.2.2.1. Padrão *Adapter*

A intenção deste padrão é converter a *interface* de uma classe em outra *interface*, esperada pelos clientes. O *Adapter* permite que classes com *interfaces* incompatíveis trabalhem em conjunto – o que, de outra forma, seria impossível.

É possível utilizar o padrão *Adapter* quando:

- For necessário usar uma classe existente, mas sua *interface* não corresponder à *interface* de que necessita;
- For necessário criar uma classe reutilizável que coopere com classes não-relacionadas ou não-previstas, ou seja, classes que não necessariamente tenham *interfaces* compatíveis;

- (Somente para adaptadores de objetos) For necessário usar várias subclasses existentes, porém, for impraticável adaptar essas *interfaces* criando subclasses para cada uma. Um adaptador de objeto pode adaptar a *interface* da sua classe-mãe.

2.7.2.2.2. Padrão *Bridge*

A intenção deste padrão é desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente.

É possível utilizar o padrão *Bridge* quando:

- For necessário evitar um vínculo permanente entre uma abstração e sua implementação. Isso pode ocorrer, por exemplo, quando a implementação deve ser selecionada ou alterada em tempo de execução.
- Tanto as abstrações como suas implementações tiverem de ser extensíveis por meio de subclasses. Neste caso, o padrão *Bridge* permite combinar as diferentes abstrações e implementações e estendê-las independentemente;
- Mudanças na implementação de uma abstração não puderem ter impacto sobre os clientes; ou seja, quando o código dos mesmos não puder ser recompilado.
- Houver uma proliferação de classes;
- For necessário compartilhar uma implementação entre múltiplos objetos (talvez usando a contagem de referências) e este fato deve estar oculto do cliente.

2.7.2.2.3. Padrão *Composite*

A intenção deste padrão é compor objetos em estruturas de árvore para representarem hierarquias partes-todo. *Composite* permite aos clientes tratarem de maneira uniforme objetos individuais e composições de objetos.

É possível utilizar o padrão *Composite* quando:

- For necessário representar hierarquias partes-todo de objetos;

- For necessário que os clientes sejam capazes de ignorar a diferença entre composições de objetos e objetos individuais. Os clientes tratarão todos os objetos na estrutura composta de maneira uniforme.

2.7.2.2.4. Padrão *Decorator*

A intenção deste padrão é, dinamicamente, agregar responsabilidades adicionais a um objeto. Os *Decorators* fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades.

É possível utilizar o padrão *Decorator*:

- Para acrescentar responsabilidades a objetos individuais de forma dinâmica e transparente, ou seja, sem afetar outros objetos;
- Para responsabilidades que podem ser removidas;
- Quando a extensão através do uso de subclasses não é prática. Às vezes, um grande número de extensões independentes é possível e isso poderia produzir uma explosão de subclasses para suportar cada combinação. Ou a definição de uma classe pode estar oculta ou não estar disponível para a utilização de subclasse.

2.7.2.2.5. Padrão *Facade*

A intenção deste padrão é fornecer uma *interface* unificada para um conjunto de *interfaces* em um subsistema. *Facade* define uma *interface* de nível mais alto que torna o subsistema mais fácil de ser usado.

É possível utilizar o padrão *Facade* quando:

- For necessário fornecer uma *interface* simples para um subsistema complexo. Os subsistemas se tornam mais complexos à medida que evoluem. A maioria dos padrões, quando aplicados, resulta em mais e menores classes. Isso torna o subsistema mais reutilizável e mais fácil de customizar, porém, também se torna mais difícil de usar para os clientes que não precisam customizá-lo. Uma fachada pode fornecer, por comportamento padrão, uma visão simples do sistema, que é boa o suficiente para a maioria

dos clientes. Somente os clientes que demandarem maior customização necessitarão olhar além da fachada;

- Existirem muitas dependências entre clientes e classes de implementação de uma abstração. Ao introduzir uma fachada para desacoplar o subsistema dos clientes e de outros subsistemas, estar-se-á promovendo a independência e portabilidade dos subsistemas.

2.7.2.3. Padrões Comportamentais

Os padrões comportamentais se preocupam com algoritmos e a atribuição de responsabilidades entre objetos. Os padrões comportamentais não descrevem apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles.

2.7.2.3.1. Padrão *Chain of Responsability*

A intenção deste padrão é evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar a solicitação. Encadear os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate.

É possível utilizar o padrão *Chain of Responsibility* quando:

- Mais de um objeto pode tratar uma solicitação e o objeto que a tratará não conhecido a priori. O objeto que trata a solicitação deve ser escolhido automaticamente;
- For necessário emitir uma solicitação para um dentre vários objetos, sem especificar explicitamente o receptor;
- O conjunto de objetos que pode tratar uma solicitação deveria ser especificado dinamicamente.

2.7.2.3.2. Padrão *Interpreter*

A intenção deste padrão é, dada uma linguagem, definir uma representação para a sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças dessa linguagem.

É possível utilizar o padrão *Interpreter* quando houver uma linguagem para interpretar e você puder representar sentenças da linguagem como árvores sintáticas abstratas. O padrão *Interpreter* funciona melhor quando:

- A gramática é simples. Para gramáticas complexas, a hierarquia de classes para a gramática se torna grande e incontrolável. Em tais casos, ferramentas tais como geradores de analisadores são uma alternativa melhor. Elas podem interpretar expressões sem a construção de árvores sintáticas abstratas, o que pode economizar espaço e, possivelmente, tempo;
- A eficiência não é uma preocupação crítica. Os interpretadores mais eficientes normalmente não são implementados pela interpretação direta de árvores de análise sintática, mas pela sua tradução para uma outra forma. Por exemplo, expressões regulares são freqüentemente transformadas em máquinas de estado. Porém, mesmo assim, o tradutor pode ser implementado pelo padrão *Interpreter*, sendo o padrão, portanto, ainda aplicável.

2.7.2.3.3. Padrão *Iterator*

A intenção deste padrão é fornecer um meio de acessar, seqüencialmente, os elementos de um objeto agregado sem expor a sua representação subjacente.

É possível utilizar o padrão *Iterator*:

- Para acessar os conteúdos de um objeto agregado sem expor a sua representação interna;
- Para suportar múltiplos percursos de objetos agregados;
- Para fornecer uma *interface* uniforme que percorra diferentes estruturas agregadas (ou seja, para suportar a iteração polimórfica).

2.7.2.3.4. Padrão *Observer*

A itenção deste padrão é definir uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.

É possível utilizar o padrão *Observer* em qualquer uma das seguintes situações:

- Quando uma abstração tem dois aspectos, um dependente do outro. Encapsulando esses aspectos em objetos separados, permite-se variá-los e reutilizá-los independentemente;
- Quando uma mudança em um objeto exige mudanças em outros, e você não sabe quantos objetos necessitam ser mudados;
- Quando um objeto deveria ser capaz de notificar outros objetos sem fazer hipóteses, ou usar informações, sobre quem são esses objetos. Em outras palavras, você não quer que esses objetos sejam fortemente acoplados.

2.7.2.3.5. Padrão *State*

A intenção deste padrão é permitir a um objeto alterar seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado sua classe.

É possível utilizar o padrão *State* em um dos dois casos seguintes:

- O comportamento de um objeto depende do seu estado e ele pode mudar seu comportamento em tempo de execução, dependendo desse estado;
- Operações têm comandos condicionais grandes, de várias alternativas, que dependem do estado do objeto. Esse estado é normalmente representado por uma ou mais constantes enumeradas. Freqüentemente, várias operações conterão essa mesma estrutura condicional. O padrão *State* coloca cada ramo do comando adicional em uma classe separada. Isto lhe permite tratar o estado do objeto como um objeto propriamente dito, que pode variar independentemente de outros objetos.

2.7.2.3.6. Padrão *Strategy*

A intenção deste padrão é definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. *Strategy* permite que o algoritmo varie independentemente dos clientes que o utilizam.

É possível utilizar o padrão *Strategy* quando:

- Muitas classes relacionadas diferem somente no seu comportamento. As estratégias fornecem uma maneira de configurar uma classe com um dentre muitos comportamentos;
- É necessário variantes de um algoritmo. Por exemplo, pode definir algoritmos que refletem diferentes soluções de compromisso entre espaço/ tempo. As estratégias podem ser usadas quando essas variantes são implementadas como uma hierarquia de classes de algoritmo;
- Um algoritmo usa dados dos quais os clientes não deveriam ter conhecimento. Use o padrão *Strategy* para evitar a exposição das estruturas de dados complexas, específicas do algoritmo;
- Uma classe define muitos comportamentos, e estes aparecem em suas operações como múltiplos comandos condicionais da linguagem. Em vez de usar muitos comandos condicionais, mova os ramos condicionais relacionados para a sua própria classe *Strategy*.

2.7.2.3.7. Padrão *Template Method*

A intenção deste padrão é definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. *Template Method* permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo.

É possível utilizar o padrão *Template Method*:

- Para implementar as partes invariantes de um algoritmo uma só vez e deixar para as subclasses a implementação do comportamento que pode variar;
- Quando o comportamento comum entre subclasses deve ser fatorado e concentrado numa classe comum para evitar a duplicação de código. Primeiramente, você identifica as diferenças no código existente e então separa as diferenças em novas operações. Por fim, você substitui o código que apresentava as diferenças por um método-template que chama uma dessas novas operações;

- Para controlar extensões de subclasses. Você pode definir um método template que chama operações “gancho” em pontos específicos, desta forma permitindo extensões somente nesses pontos.

3. Requisitos Funcionais para a criação do sistema proposto e análise da tecnologia envolvida

Neste capítulo serão descritas as funcionalidades presentes em sistemas de automação de testes do tipo registro/reprodução das ações do usuário no *Windows*, serão descritos os requisitos funcionais que o sistema deverá atender, definindo assim o escopo do projeto e também será realizada a análise da tecnologia envolvida na criação do protótipo.

3.1. Funcionalidades presentes em sistemas de automação de testes do tipo registro/reprodução de ações do usuário

As funcionalidades abaixo foram encontradas em sistemas do tipo gravação/reprodução das ações do usuário. É importante ressaltar que nem todas foram encontradas em todos os sistemas, sendo, portanto, um resumo das diferentes funcionalidades presentes.

- Gravação das ações do usuário via teclado e mouse;
- Reprodução das ações do usuário via teclado e mouse;
- Geração de script para posterior reprodução das ações do usuário;
- Integração de interpretador de linguagens de programação com o script gerado;
- Ações auxiliares para a integração de interpretador de linguagens de programação com o script gerado, como:
    - Ferramenta para verificar posição do mouse;
    - Ferramenta para verificar nome de janelas do *Windows* para posterior chamada via programação;

- Visualização das etapas do teste através de captura da tela (Imagem) ao gravar o teste;

3.2. Requisitos funcionais para a criação do sistema proposto

A escolha de quais requisitos funcionais devem ser implementados foi definida de acordo com as principais funcionalidades presentes em sistemas de automação de testes do tipo registro/reprodução de ações do usuário.

Na tabela abaixo é realizada a descrição resumida de cada requisito e nos próximos subcapítulos, é realizada a descrição detalhada dos requisitos que devem ser implementados neste trabalho, sendo eles os requisitos dos tipos “Essencial” e “Importante”. Os requisitos do tipo “Desejável” serão apenas registrados para que sejam implementados em trabalhos futuros.

Código	Tipo de Requisito Funcional	Módulo	Título
RF001	Essencial	Gravação	Gravação das ações de usuário
RF002	Essencial	Reprodução	Reprodução das ações de usuário
RF003	Importante	Gravação	Gravação de script das ações do usuário
RF004	Importante	Reprodução	Reprodução de script das ações dos usuários
RF005	Desejável	Gravação e Reprodução	Geração mais amigável de script de ações do usuário

			para o desenvolvedor
RF006	Desejável	Gravação e Reprodução	Integração de interpretador de linguagens de programação com o script gerado
RF007	Desejável	Script	Verificação do resultado de ação realizada para analisar se o teste funcionou ou não. Um exemplo seria verificar se está escrito algo específico em um arquivo de texto.
RF008	Desejável	Gravação	Gravação de capturas de tela para a visualização das ações que estão sendo gravadas, para posterior visualização após a gravação.

Tabela 1 – Requisitos Funcionais

3.2.1. Detalhamento dos requisitos funcionais dos tipos “Essencial” e “Importante”

Neste tópico serão detalhados os requisitos funcionais do tipo ‘Essencial” e “Importante”. Os requisitos do tipo ‘Desejável’ não serão detalhados.

3.2.1.1. RF001 - Gravação das ações do usuário

Neste tópico será detalhado o requisito e seu funcionamento através de uma tabela informando seu funcionamento e um fluxograma de interações entre o usuário e o sistema.

Código	RF001
Título	Gravação das ações do usuário
Módulo	Gravação
Versão	01
Prioridade	Essencial

Descrição	**Caminho Principal:**
	Como usuário devo poder gravar as ações que realizar com o mouse e com o teclado até que queira finalizar a gravação. As ações que devem ser gravadas são as seguintes:
	- **Mouse**
	  - Movimento do Mouse;
	  - Clique com o botão esquerdo do Mouse;
	  - Clique com o botão direito do mouse.
	- **Teclado**
	  - Todas as teclas pressionadas.

	Caminho Alternativo 01: Se em qualquer momento for apertado o botão para finalizar a gravação, a gravação deverá ser interrompida. **Caminho Alternativo 02:** Se em qualquer momento for pressionada a tecla 'Pause' do teclado, a gravação deverá ser interrompida.

Tabela 2 – RF001 – Gravação das ações do usuário

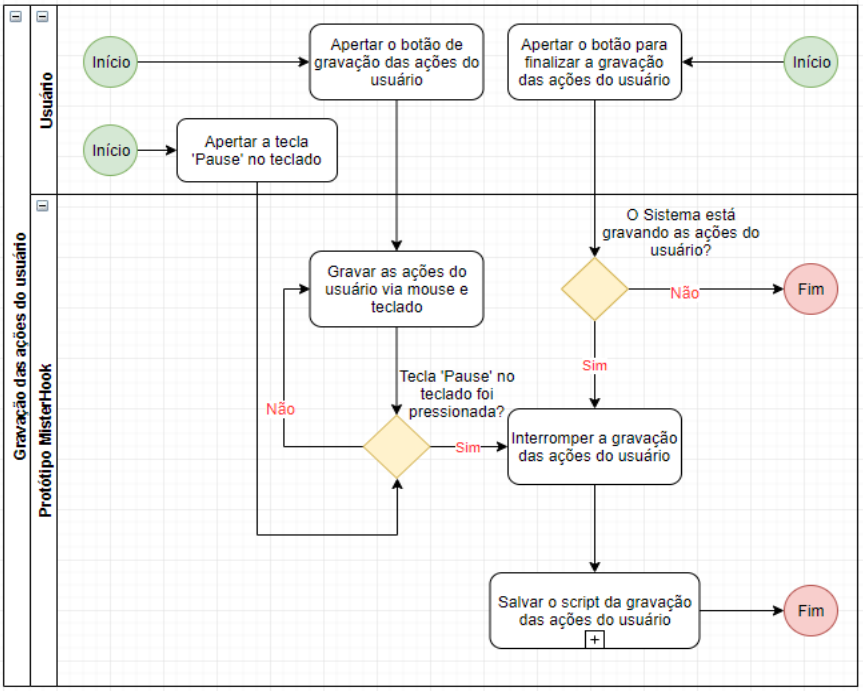


Figura 9 - Fluxograma RF001 - Gravação das ações do usuário

3.2.1.2. RF002 - Reprodução das ações do usuário

Neste tópico será detalhado o requisito e seu funcionamento através de uma tabela informando seu funcionamento e um fluxograma de interações entre o usuário e o sistema.

Código	RF002
Título	Reprodução das ações do usuário
Módulo	Reprodução
Versão	01
Prioridade	Essencial
Descrição	**Caminho principal:** Como usuário devo poder reproduzir as ações gravadas anteriormente pelo protótipo. **Caminho alternativo 01:** Se em qualquer momento for pressionada a tecla ‘Pause’ do teclado, a gravação deverá ser interrompida.

Tabela 3 – RF002 – Reprodução das ações do usuário

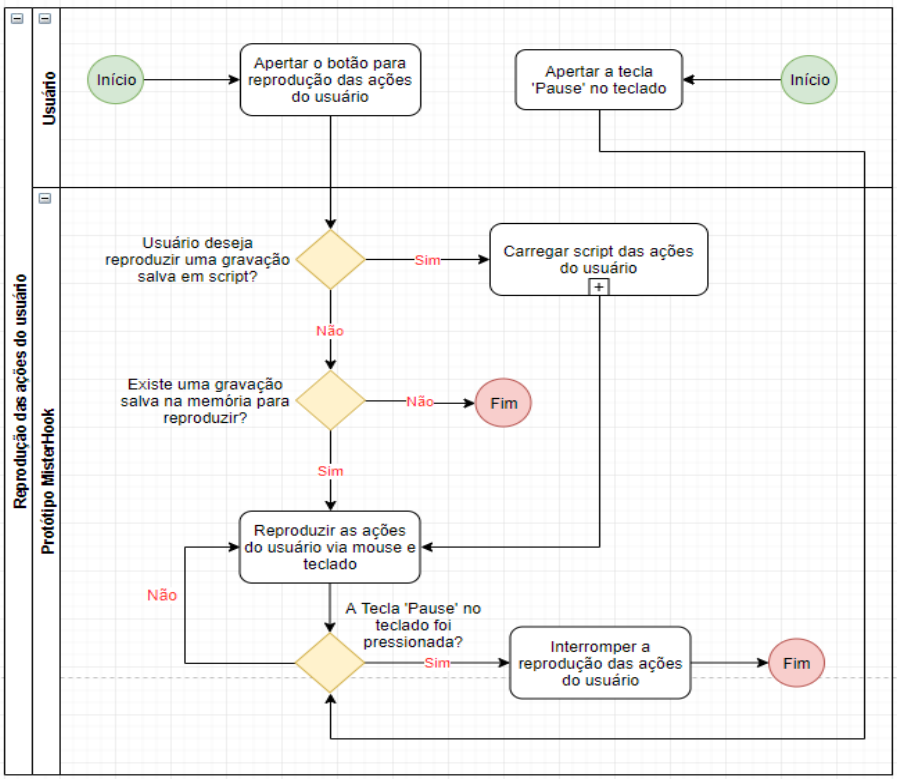


Figura 10 - Fluxograma RF002 - Reprodução das ações do usuário

3.2.1.3. RF003 - Gravação de script das ações do usuário

Neste tópico será detalhado o requisito e seu funcionamento através de uma tabela informando seu funcionamento e um fluxograma de interações entre o usuário e o sistema.

Código	RF003
Título	Gravação de script das ações do usuário
Módulo	Gravação
Versão	01
Prioridade	Importante

Descrição	**Caminho principal:** Como usuário, devo poder finalizar a gravação das ações que realizei e após a finalização o sistema deve gerar um script em arquivo de texto de todas as ações realizadas para posterior reprodução. **Caminho alternativo 01:** Se o usuário não escolher uma pasta, o sistema não deverá salvar a gravação.

Tabela 4 – RF003 – Gravação de script das ações do usuário

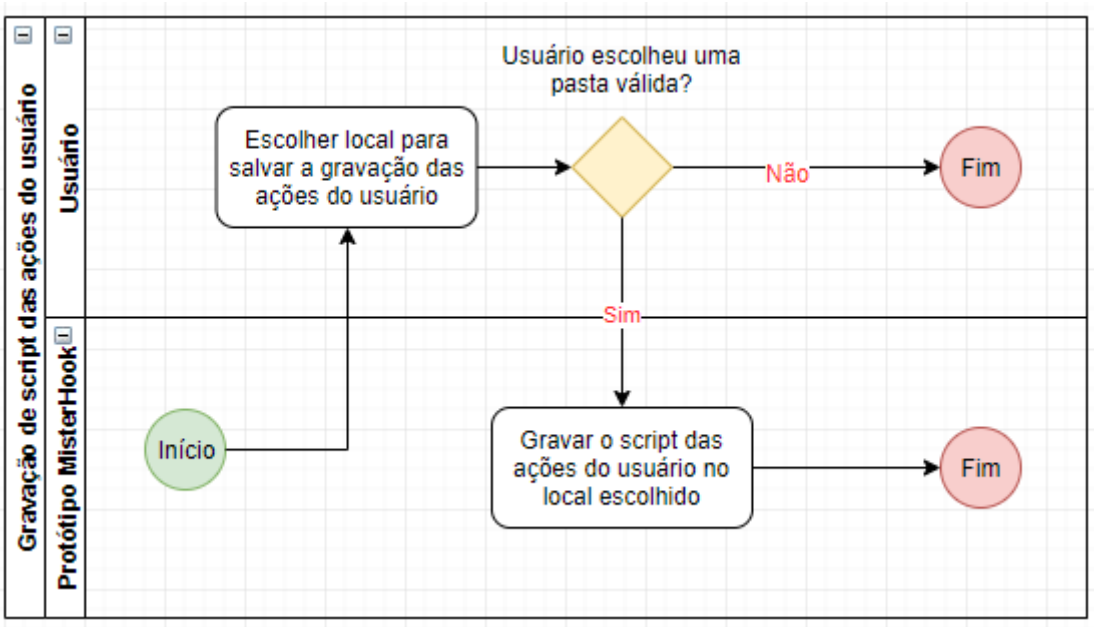


Figura 11 - Fluxograma RF003 - Gravação de script das ações do usuário

3.2.1.4. RF004 - Reprodução de script das ações dos usuários

Neste tópico será detalhado o requisito e seu funcionamento através de uma tabela informando seu funcionamento e um fluxograma de interações entre o usuário e o sistema.

Código	RF004
Título	Reprodução de script das ações dos usuários
Módulo	Reprodução
Versão	01
Prioridade	Importante
Descrição	**Caminho principal:** Como usuário, devo poder realizar a reprodução das ações gravadas anteriormente em script pelo sistema, escolhendo um arquivo de script. **Caminho alternativo 01:** Se o usuário não escolher um arquivo, o sistema não deverá reproduzir a gravação. **Caminho alternativo 02:** Se o arquivo escolhido não for válido, o sistema deverá emitir uma mensagem avisando a ocorrência.

Tabela 5 - RF004 - Reprodução do script das ações do usuário

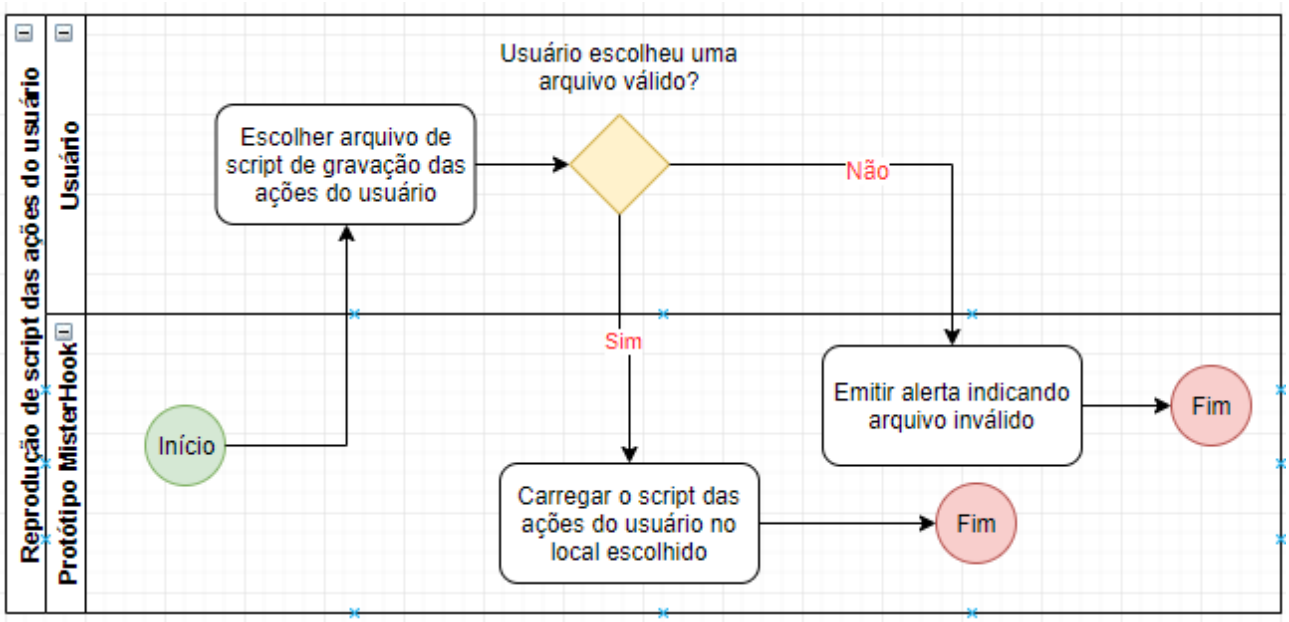


Figura 12 - RF004 - Reprodução de script das ações do usuário

3.3. Análise da tecnologia envolvida

3.3.1. Linguagem de programação utilizada

A escolha da linguagem de programação utilizada foi baseada na linguagem que o autor do trabalho possui maior conhecimento e facilidade de trabalhar. Neste caso, a linguagem VB.NET. Esta linguagem possui muita semelhança com a linguagem de programação C#.NET, existindo diversos conversores entre as linguagens, caso posteriormente alguém ache necessário realizar a conversão.

3.3.2. Padrões de projeto escolhidos para facilitar a manutenibilidade do código, simplificar o entendimento e garantir que o protótipo possa evoluir de forma controlada no futuro

Foram utilizados no protótipo três padrões de projeto e um quarto foi identificado para trabalhos futuros. No capítulo 5, onde é descrito o protótipo em detalhes, são informadas mais detalhadamente suas implementações e no capítulo 7 é descrita a possível implementação do quarto padrão de projeto.

- **Padrão de projeto Singleton** – Foi utilizado nos objetos que auxiliam a reprodução e gravação das ações do usuário, possibilitando assim acesso único e controlado aos objetos;
- **Padrão de projeto Facade** – Como o protótipo utiliza uma biblioteca de funcionalidades, foi criado um objeto fachada para os objetos de gravação de ações do usuário, garantindo assim o uso apenas da classe de reprodução correta;
- **Padrão de projeto Proxy** – Para a realização de registro e futura implementação de licenças de uso da biblioteca de funcionalidades criada, foi implementada uma camada proxy para as classes de gravação de ações do usuário, não poluindo assim as classes-base;
- **Padrão de projeto Interpreter** – Este padrão não foi utilizado no protótipo, porém é de extrema utilidade em trabalhos futuros, visto a possibilidade de interpretar comandos digitados pelo usuário, estendendo assim as funcionalidades do script gerado. Este ponto será descrito em detalhes no capítulo 7.

3.3.3. Tipo de gravação de dados e reprodução dos dados capturados pelo protótipo

Foi escolhido o formato arquivo de texto para o *script* das ações do usuário, devido a facilidade de escrita e leitura destes arquivos. Como os scripts gravados são de ações de automação de testes, não houve a necessidade de utilização de segurança para a criação destes arquivos.

3.3.4. Controle de versionamento do protótipo

Para o controle de versionamento do protótipo foi escolhido o *GitHub* por conta de seu massivo uso pela comunidade de desenvolvimento de software, além da facilidade de uso.

3.3.5. Disponibilização do código-fonte

Para a disponibilização do código-fonte, foi escolhido o GitHub, por ser uma ferramenta online de controle de versionamento com o recurso de disponibilização online do código-fonte do protótipo básico e do protótipo.

3.3.6. Licença de uso

O protótipo e todo código-fonte criado para este trabalho de conclusão de curso é de uso livre, sem quaisquer restrições em relação a licenças.

4. Prototipação básica para verificação da viabilidade do projeto

Neste capítulo será abordada a criação do protótipo básico (*Minimum Viable Product* - MVP) para verificação da viabilidade do projeto. É importante ressaltar que programas do tipo proposto neste trabalho não possuem codificação de nível trivial, portanto a criação de um protótipo básico com as funcionalidades essenciais, e se possível as funcionalidades marcadas como importantes, é necessária antes da criação do protótipo em si.

4.1. Atendimento dos requisitos funcionais essenciais

O protótipo básico atendeu aos requisitos funcionais ‘RF001 - Gravação de dados do usuário’, ‘RF002 - Reprodução de dados do usuário’, ‘RF003 - Gravação de script das ações do usuário’ e ‘RF004 - Reprodução de script das ações do usuário’.

Os requisitos ‘RF005 - Geração mais amigável de script de ações do usuário para o desenvolvedor’ e ‘RF006 - Integração de interpretador de linguagens de programação com o *script* gerado‘ não foram atendidos por serem do tipo ‘Desejável’ e por este ser ainda o MVP do projeto.

4.2. Atendimento da tecnologia envolvida

O protótipo básico implementou a maioria das especificações indicadas no capítulo 3. Apenas o tópico de utilização de padrões de projeto que não foi implementado neste protótipo básico, por conta de sua natureza de MVP.

- Linguagem de programação VB.NET;
- Gravação de scripts via arquivo de texto;
- Controle de versionamento GitHub;
- Disponibilização do código-fonte no GitHub;
- Nenhum padrão de projeto foi utilizado para a prototipação básica.

4.3. Desenvolvimento e Dificuldades enfrentadas

Para a realização deste MVP, foi necessário o conhecimento do funcionamento de *Application Programing Interfaces* (APIs) do *Windows*, pois além de ser a principal interface do *Windows* para o tipo de funcionalidade desejada, de gravação e reprodução de ações do usuário via mouse e teclado, é também a forma mais segura de se realizar este tipo de ação, sendo nativa do próprio *Windows*.

A API do *Windows* para realizar este tipo de comportamento é a API '*SetWindowsHookEx'*. Segundo o site da própria Microsoft, esta API instala um procedimento 'Gancho' (*Hook*[1]) dentro de uma corrente de ganchos. Você pode instalar um procedimento de gancho para monitorar o sistema em relação a certos tipos de eventos. Esses eventos estão associados tanto quanto a *thread*[2] específica, quanto todas as threads no mesmo desktop da *thread* que está sendo executada (SetWindowsHookEx Function, 06-2019).

Um dos parâmetros para a instalação deste gancho é o tipo de evento que deseja monitorar, sendo uma delas as ações de Teclado (WH_KEYBOARD_LL_ e outra as ações de Mouse (WH_MOUSE_LL), usadas neste protótipo básico através das chamadas abaixo, apenas para conhecimento:

```
Private Const WH_KEYBOARD_LL As Integer = 13

SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardHookStructDelegate, _hinstance, 0)

Private Const WH_MOUSE_LL As Integer = 14

SetWindowsHookEx(WH_MOUSE_LL, MouseHookProcDelegate, _hinstance, 0)
```

[1] Hook - Procedimento Gancho usado para monitorar eventos do sistema *Windows*;

[2] Thread - Thread é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se auto dividir em duas ou mais tarefas. É o termo em inglês para Linha ou Encadeamento de Execução;

A maior dificuldade para realizar a gravação e reprodução das ações dos usuários é que a maioria do material online consultado indica o uso de outros tipos de eventos para monitoramento e reprodução das ações do usuário via teclado e mouse, o evento WH_JOURNALRECORD e o evento WH_JOURNALPLAYBACK. Alguns dos materiais consultados estavam fazendo uso destes tipos de eventos e disponibilizaram o código-fonte, porém não foi possível realizar o monitoramento desta forma por problemas que não foram possíveis de solucionar.

Por não ser possível utilizar a API 'SetWindowsHookEx' com o tipo de evento WH_JOURNALPLAYBACK para realizar a reprodução das ações do usuário, foram usadas outras APIS também disponibilizadas pelo *Windows* para realizar estas ações, sendo elas:

- Teclado
  - keybd_event (Reprodução de Teclas).
- Mouse
  - SetCursorPos (Reprodução de movimento do mouse);
  - mouse_event (Reprodução de clique do mouse).

4.4. Interface

A *interface* do protótipo básico é minimalista e possui somente três botões, necessários para o atendimento completo de seus requisitos, conforme imagem abaixo.

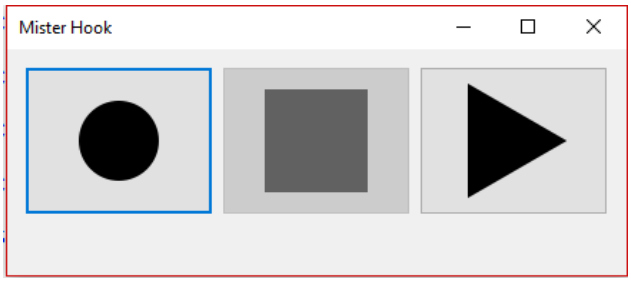


Figura 13 - Interface do protótipo básico

Um botão para gravação das ações do usuário, um botão para parar a gravação e um botão para reproduzir a ação gravada.

Além destes botões é importante ressaltar que enquanto está ocorrendo uma gravação ou reprodução, se o usuário apertar a tecla ‘Pause’ do teclado, a ação sendo executada será interrompida.

Outras ações realizadas são:

- Escolha de pasta ao parar a gravação, para salvar o script gravado;
- Escolha de arquivo de script, se o usuário desejar, ao apertar o botão de reprodução das ações do usuário.

4.5. Resultados obtidos

Após o desenvolvimento do protótipo básico foi possível atender os requisitos essenciais e importantes descritos no tópico 4.1. Este resultado possibilitou que o trabalho pudesse ser realizado, pois por não ser trivial, se estes requisitos não fossem atendidos, não seria possível dar continuidade ao trabalho.

Também foi definido um nome para a ferramenta criada, sendo escolhido o nome Mister Hook, explicitando assim o uso de *hooks* para a gravação das ações do usuário.

4.6. Disponibilização do resultado

O projeto do protótipo básico está disponível no repositório do GitHub a seguir:

https://github.com/RafaBotossi/MisterHookBasicPrototype

Além disso, o código-fonte também estará disponível na área de anexos deste trabalho, em sua totalidade, para fácil acesso.

5. Criação do protótipo de sistema de automação de testes funcionais com *interface* gráfica do tipo registro/reprodução para ambiente *desktop Windows*

Neste capítulo será descrito como o sistema foi criado, bem como suas funcionalidades e informações relevantes ao projeto.

5.1. Descrição do protótipo

Após a criação do MVP descrito no capítulo anterior, foi criado o protótipo final, utilizando-se das funcionalidades presentes no MVP e estendendo as mesmas para desenvolver dois projetos que trabalham em conjunto, o projeto MisterHookLibrary e o projeto MisterHookPrototype.

5.2. Atendimento da tecnologia envolvida

O protótipo (MisterHookPrototype) e a biblioteca de classes (MisterHookLibrary) implementaram todas as especificações de tecnologia indicadas no capítulo 3.

1. Linguagem de programação VB.NET;
2. Gravação de scripts via arquivo de texto;
3. Controle de versionamento GitHub;
4. Disponibilização do código-fonte no GitHub;
5. Utilização de padrões de projeto (*Singleton, Facade, Proxy*).

5.1. Atendimento dos requisitos funcionais essenciais

O protótipo atendeu aos requisitos funcionais ‘RF001 - Gravação de dados do usuário’, ‘RF002 - Reprodução de dados do usuário’, ‘RF003 - Gravação de script das ações do usuário’ e ‘RF004 - Reprodução de script das ações do usuário’.

Os requisitos ‘RF005 - Geração mais amigável de script de ações do usuário para o desenvolvedor’ e ‘RF006 - Integração de interpretador de linguagens de programação com o *script* gerado‘ não foram atendidos por sua complexidade de desenvolvimento no espaço de tempo disponível e por conta de novos itens que surgiram e que possuíram maior importância no decorrer do projeto, descritos no tópico abaixo.

5.2. Dificuldades enfrentadas durante o desenvolvimento

No começo do projeto, ao serem levantados os requisitos essenciais e desejáveis, não foram identificados dois pontos importantes: A experiência do usuário ao utilizar o sistema e a aplicabilidade do projeto em projetos futuros desenvolvidos por outras pessoas. Por conta desta falha no levantamento de requisitos, foi necessário desenvolver estes dois pontos, considerados essenciais para a criação do protótipo, fazendo com que os requisitos ‘RF005 - Geração amigável do script de ações do usuário’ e ‘RF006 - Integração de interpretador de linguagens de programação com o *script* gerado’ fossem adiados e consequentemente descartados para a entrega deste trabalho.

Os dois pontos descritos acima foram implementados da seguinte forma:

1 – Experiência do usuário ao utilizar o sistema - Atualização da interface gráfica, possibilitando um uso mais simples do sistema e também facilitando a criação de scripts e sua reprodução;

2 – Aplicabilidade do projeto em projetos futuros desenvolvidos por outras pessoas - Criação de uma biblioteca de classes para desacoplar a interface gráfica da implementação da gravação e reprodução das ações do usuário, possibilitando assim o uso por outros desenvolvedores que podem desejam criar a sua própria interface gráfica e/ou usar a biblioteca de forma automatizada, sem interface gráfica.

O projeto de biblioteca de classes foi chamado de *MisterHookLibrary* e o projeto do protótipo foi chamado de *MisterHookPrototype*.

5.3. Funcionalidades do protótipo

5.3.1. Gravação de ações do usuário via mouse e teclado

O protótipo implementou a gravação de ações do usuário via mouse e teclado, ou seja, ao realizar uma gravação, todas as teclas pressionadas e movimentos e cliques do mouse serão gravados, assim como o tempo em que ele demorou para fazer todas estas ações, refletindo assim exatamente a ação realizada.

Ao clicar no botão de gravação, o sistema irá salvar em memória as ações realizadas e ao parar a gravação, estas ações serão gravadas na caixa de texto abaixo.

Figura 14 - Gravação de ações do usuário

Figura 15 - Parar a gravação

5.3.2. Criação de script das ações gravadas

Após realizar a gravação das ações do usuário e parar de gravar, serão informadas na caixa de texto todas as ações que o usuário realizou, no formato de script. Será possível então salvar o script através do menu ‘Arquivo -> Salvar Script (CTRL+S)’, ou através do atalho de teclado CTRL+S (Tecla Control).

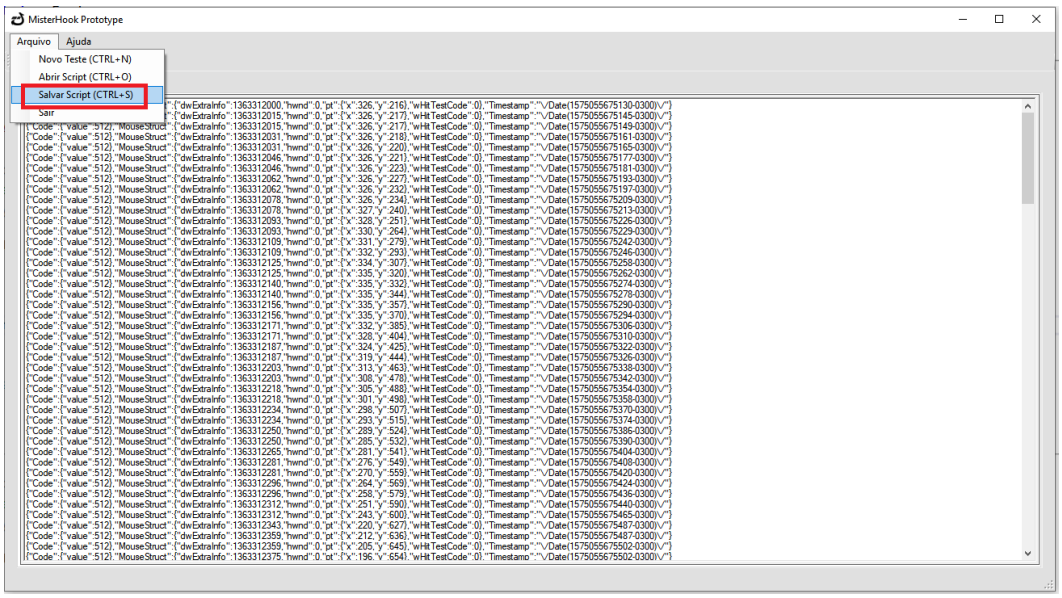


Figura 16 - Salvar Script

5.3.3. Abertura de arquivos de script para alteração/visualização

É possível abrir um script já salvo através do menu ‘Arquivo-> Abrir Script (CTRL+O)’, ou então do atalho de teclado CTRL+O (Tecla Control).

Para esta funcionalidade não foi implementado um validador de script, para ter certeza de que é um script criado pelo sistema MisterHookPrototype, ou seja, será possível abrir qualquer arquivo do tipo texto (*.txt).

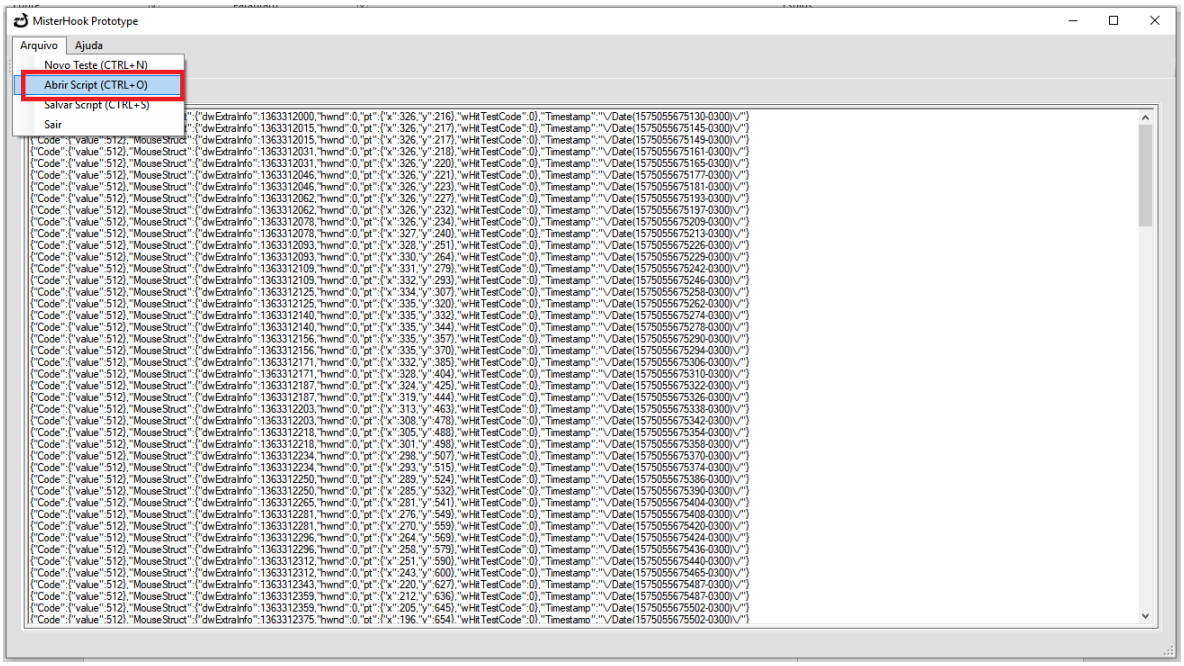


Figura 17 - Abrir Script

5.3.4. Reprodução das ações gravadas via script

Após a gravação de um script, ou ao abrir um script, será possível reproduzir as ações que o usuário realizou neste script apertando o botão ‘Reproduzir’.

É possível parar a reprodução a qualquer momento apertando a tecla pause no teclado.

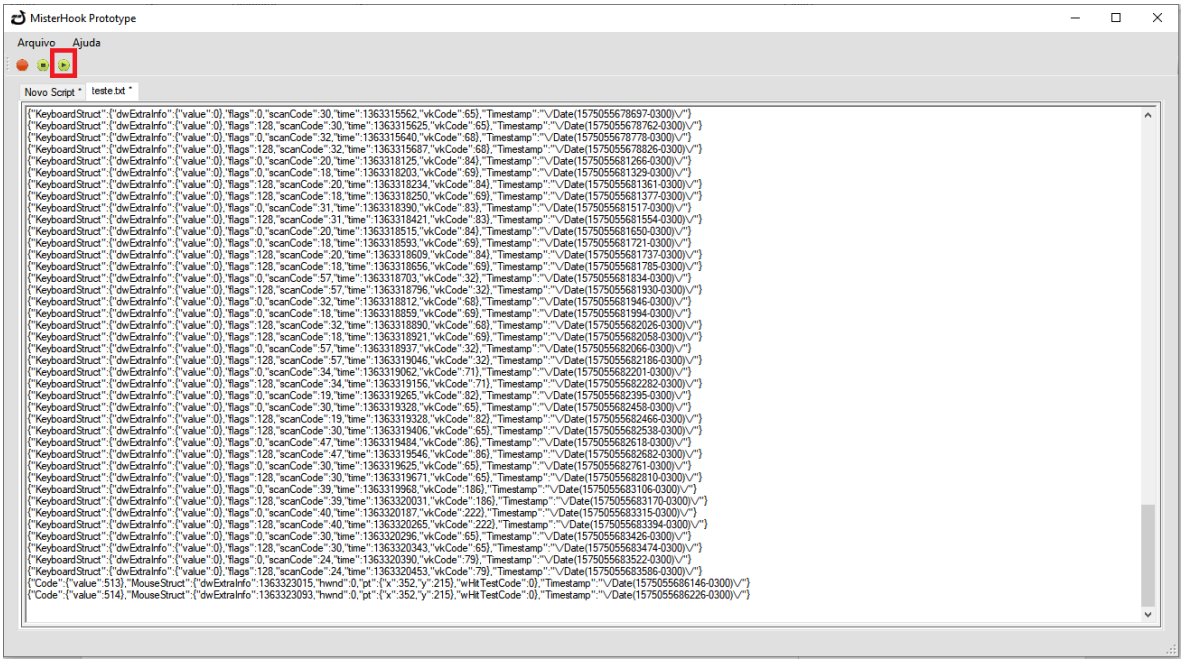


Figura 18 - Reprodução das ações do usuário

5.3.5. Outras funcionalidades

- Sair do sistema
  - Ao clicar no menu ‘Arquivo -> Sair’, o sistema será finalizado.
- Sobre o sistema
  - Ao clicar no menu ‘Ajuda -> Sobre’, será apresentada uma tela indicando o motivo da criação do sistema, bem como o nome de seu desenvolvedor.

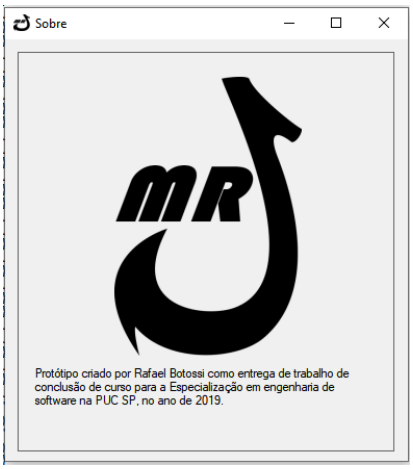


Figura 19 - Tela sobre

5.4. Componentes

5.4.1. Biblioteca MisterHookLibrary

Esta biblioteca de classes foi criada para facilitar o uso dos módulos de gravação e reprodução de ações do usuário por outros desenvolvedores e também diminuir o acoplamento com o programa executável, que implementa interface gráfica para execução das ações da biblioteca.

A biblioteca é composta por quatro pastas principais, Interfaces, Models, Proxies e Structs.

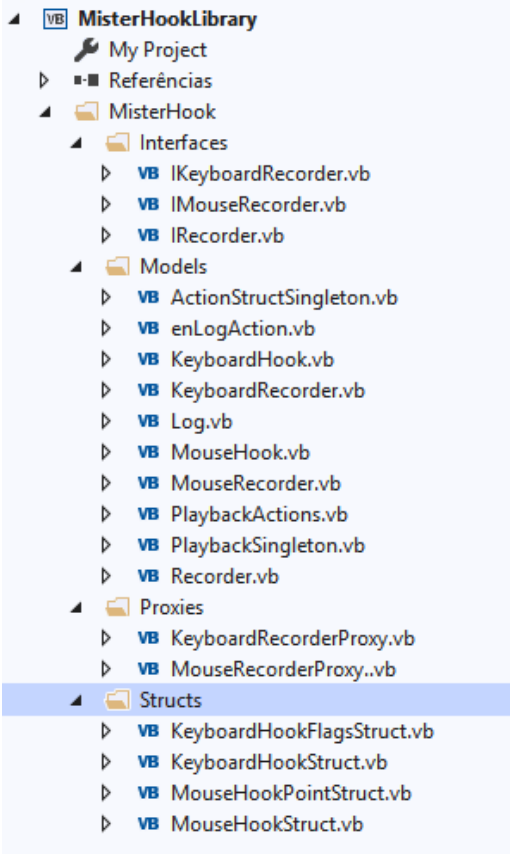


Figura 20 - Menu MisterHookLibrary

- **Interfaces –** Local onde são armazenadas todas as interfaces usadas pela biblioteca. *IRecorder*, *IKeyboardRecorder*, e *IMouseRecorder*, sendo essas duas últimas herdadas da classe *IRecorder*. O Maior intuito do uso dessas interfaces é seu uso no padrão de projeto **Proxy**;

- **Models –** Local onde são armazenadas todas as classes usadas na biblioteca. A classe *Recorder* implementa o padrão de projeto **Facade** para as classes *KeyboardRecorder* e *MouseRecorder*. Também nessa pasta ficam os modelos das classes que implementam o padrão de projeto **Singleton**, *PlaybackSingleton* e *ActionStructSingleton*;
- **Proxies –** Local onde são armazenadas as classes que utilizam o padrão de projeto **Proxy**, *KeyboardRecorderProxy* e *MouseRecorderProxy*. Também nestas classes é realizado o uso de registro (*Log*) das ações gravadas;
- **Structs –** Local onde são armazenadas todas as estruturas usadas no projeto. Essas estruturas são utilizadas principalmente para implementar corretamente os Hook's de ações de teclado e mouse realizadas pelo usuário.

A maioria das classes da biblioteca possuem a marcação *Friend*, que permite que as classes fiquem visíveis somente internamente à biblioteca, facilitando assim seu uso no protótipo, deixando visíveis apenas as classes que podem ser realmente utilizadas. Este tipo de marcação foi fundamental para a correta implementação do padrão de projeto **Facade**.

```
Namespace MisterHook.Models

    ''' <summary>
    ''' Classe criada para o record de Keyboard
    ''' Marcada como Friend para visibilidade somente interna, para uso do padrão de projeto Facade
    ''' </summary>
    7 referências
    Friend Class KeyboardRecorder
        Implements Interfaces.IKeyboardRecorder
```

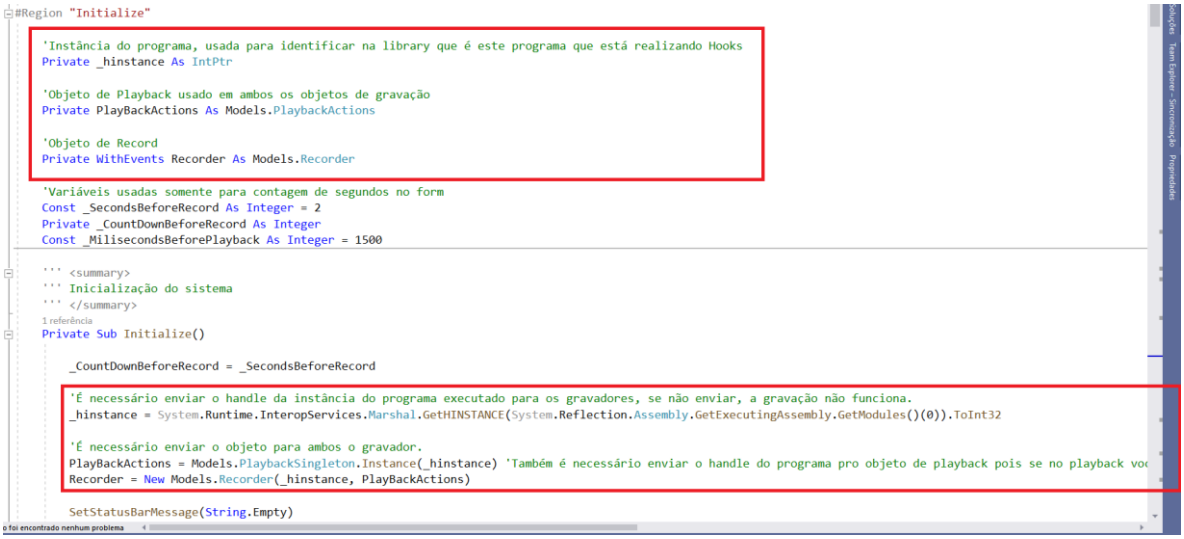
Figura 21 - Classe Friend (Padrão de Projeto Facade)

5.4.2. Programa executável MisterHookPrototype

O programa executável utiliza a biblioteca MisterHookLibrary para implementar as gravações e reproduções de ações do usuário, portanto serve apenas como interface gráfica para as funcionalidades presentes na biblioteca.

Os únicos pontos onde a biblioteca é utilizada são:

- Início do formulário FrmGeral, iniciando as variáveis necessárias para a posterior gravação e reprodução das ações do usuário.



```
#Region "Initialize"

    'Instância do programa, usada para identificar na library que é este programa que está realizando Hooks
    Private _hinstance As IntPtr

    'Objeto de Playback usado em ambos os objetos de gravação
    Private PlayBackActions As Models.PlaybackActions

    'Objeto de Record
    Private WithEvents Recorder As Models.Recorder

    'Variáveis usadas somente para contagem de segundos no form
    Const _SecondsBeforeRecord As Integer = 2
    Private _CountDownBeforeRecord As Integer
    Const _MilisecondsBeforePlayback As Integer = 1500

    ''' <summary>
    ''' Inicialização do sistema
    ''' </summary>
    1 referência
    Private Sub Initialize()

        _CountDownBeforeRecord = _SecondsBeforeRecord

        'É necessário enviar o handle da instância do programa executado para os gravadores, se não enviar, a gravação não funciona.
        _hinstance = System.Runtime.InteropServices.Marshal.GetHINSTANCE(System.Reflection.Assembly.GetExecutingAssembly.GetModules()(0)).ToInt32

        'É necessário enviar o objeto para ambos o gravador.
        PlayBackActions = Models.PlaybackSingleton.Instance(_hinstance) 'Também é necessário enviar o handle do programa pro objeto de playback pois se no playback voc
        Recorder = New Models.Recorder(_hinstance, PlayBackActions)

        SetStatusBarMessage(String.Empty)
```

Figura 22 - Inicialização da biblioteca MisterHookLibrary

- Iniciar gravação e parar gravação.

```
488         ''' Evento de início de Gravação.
489         ''' </summary>
            1 referência
490         Public Sub StartRecording()
491
492             Try
493
494                 Recorder.StartRecording()
495
496             Catch ex As Exception
497                 MessageBox.Show(ex.Message)
498             End Try
499
500         End Sub
501
502         ''' <summary>
503         ''' Evento de Stop de gravação
504         ''' </summary>
            1 referência
505         Public Sub StopRecording()
506
507             Try
508
509                 If Recorder IsNot Nothing Then
510                     Recorder.StopRecording()
511                 End If
512
513             Catch ex As Exception
514                 MessageBox.Show(ex.Message)
515             End Try
516
517         End Sub
```

Figura 23 - Iniciar e finalizar gravação

5.5. O uso dos Padrões de Projeto

5.5.1. Implementação do padrão de projeto Singleton

Através da implementação deste padrão de projeto, foi possível garantir que o acesso a um determinado objeto fosse único, não criando assim novas instâncias da mesma classe.

Foram escolhidos dois pontos da biblioteca para implementar este padrão, a classe *PlaybackActions* e o objeto *ActionStructs*. A classe *PlaybackAction* é utilizada para realizar a reprodução das ações do usuário, por isso sua escolha, que deve sempre garantir que apenas uma reprodução esteja em andamento em todos os momentos e a lista de objetos ActionStructs, que representa todas as ações que estão sendo gravadas, sendo também necessário uma instância única do objeto.

```
Namespace MisterHook.Models

    ''' <summary>
    ''' Classe usada para retornar apenas um objeto ActionStructs, utilizando o padrão de projeto Singleton
    ''' </summary>
    11 referências
    Friend Class ActionStructSingleton

        Private Shared _ActionStructs As List(Of Object)

        ''' <summary>
        ''' Construtor vazio pois o uso da função será Shared
        ''' </summary>
        0 referências
        Protected Sub New()
        End Sub

        ''' <summary>
        ''' Retorna objeto Singleton
        ''' </summary>
        ''' <returns>Objeto Singleton</returns>
        10 referências
        Public Shared Function ActionStructs() As List(Of Object)

            If _ActionStructs Is Nothing Then
                _ActionStructs = New List(Of Object)
            End If

            Return _ActionStructs

        End Function

    End Class

End Namespace
```

Figura 24 - Classe Singleton para acesso à lista de objetos ActionStructs

```
If json IsNot Nothing Then
    If json.Contains("MouseStruct") Then
        Dim MouseHookModel As New MisterHook.Models.MouseHook
        Dim MS As New IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(json))
        Dim ser As New Runtime.Serialization.Json.DataContractJsonSerializer(MouseHookModel.GetType())
        MouseHookModel = ser.ReadObject(MS)
        MS.Close()
        ActionStructSingleton.ActionStructs.Add(MouseHookModel)
    Else
        Dim KeyboardHookModel As New MisterHook.Models.KeyboardHook
        Dim MS As New IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(json))
        Dim ser As New Runtime.Serialization.Json.DataContractJsonSerializer(KeyboardHookModel.GetType())
        KeyboardHookModel = ser.ReadObject(MS)
        MS.Close()
        ActionStructSingleton.ActionStructs.Add(KeyboardHookModel)
    End If
End If
```

Figura 25 - Uso do Singleton de ActionStructs

```
1 referência
Public Class PlaybackSingleton

    Private Shared _instance As PlaybackActions

    ''' <summary>
    ''' Construtor vazio pois o uso da função será Shared
    ''' </summary>
    0 referências
    Protected Sub New()
    End Sub

    ''' <summary>
    ''' Retorna objeto Singleton
    ''' </summary>
    ''' <param name="hinstance">Instância do sistema que chamou a library</param>
    ''' <param name="PathToSave">Caminho para salvar o Playback</param>
    ''' <param name="FileName">Nome do arquivo para salvar o Playback</param>
    ''' <returns>Objeto Singleton</returns>
    0 referências
    Public Shared Function Instance(hinstance As IntPtr, Optional PathToSave As String = "", Optional FileName As String = "")

        If _instance Is Nothing Then
            _instance = New PlaybackActions(hinstance, PathToSave, FileName)
        End If

        Return _instance

    End Function

End Class
```

Figura 26 - Classe Singleton para acesso á Classe PlaybackActions

```
'É necessário enviar o objeto para ambos o gravador.
PlayBackActions = Models.PlaybackSingleton.Instance(_hinstance)
Recorder = New Models.Recorder(_hinstance, PlayBackActions)
```

Figura 27 – Uso do Singleton de PlaybackActions

5.5.2. Implementação do padrão de projeto Proxy

Através da implementação deste padrão de projeto, foi possível criar o log do sistema desacoplado das classes que implementam a gravação das ações do

usuário, além de possibilitar que seja implementado em trabalhos futuros uma camada de segurança e autenticação às classes de gravação de ações do usuário.

Foram escolhidas duas classes para esta implementação, *KeyboardRecorder* e *MouseRecorder*, pois são as duas classes principais de gravação das ações do usuário. Para isso, foram criadas as interfaces *IRecorder, IKeyboardRecorder e IMouseRecorder*, a fim de se obter um contrato fixo e específico, que todas as classes conhecem. Após isso, foram criadas as classes *KeyboardRecorderProxy* e *MouseRecorderProxy*, ambas implementando a interfaces correspondentes. Essas classes são o único ponto de contato com a classe *Recorder*, e são responsáveis por instanciar as classes (*KeyboardRecorder e MouseRecorder*) e chamar seus métodos, além de realizar o log das funcionalidades antes e depois das chamadas.

```
Friend Class MouseRecorderProxy
    Implements Interfaces.IMouseRecorder

    'Eventos usados para Log
    Public Shared Event MouseLDown()
    Public Shared Event MouseLUp()
    Public Shared Event MouseRDown()
    Public Shared Event MouseRUp()
    Private Event IMouseRecorder_MouseLDown() Implements Interfaces.IMouseRecorder.MouseLDown
    Private Event IMouseRecorder_MouseLUp() Implements Interfaces.IMouseRecorder.MouseLUp
    Private Event IMouseRecorder_MouseRUp() Implements Interfaces.IMouseRecorder.MouseRUp
    Private Event IMouseRecorder_MouseRDown() Implements Interfaces.IMouseRecorder.MouseRDown

    Private WithEvents _MouseRecorder As Models.MouseRecorder

    1 referência
    Public Sub New(hinstance As IntPtr, Optional ByRef oPlayback As Models.PlaybackActions = Nothing)

        _MouseRecorder = New Models.MouseRecorder(hinstance, oPlayback)

    End Sub

    ''' <summary>
    ''' Realiza a gravação de teclas
    ''' </summary>
    15 referências
    Public Sub StartRecording() Implements Interfaces.IMouseRecorder.StartRecording

        LogAction(Models.enLogAction.MouseStartRecording, "init")
        _MouseRecorder.StartRecording()
        LogAction(Models.enLogAction.MouseStartRecording, "end")

    End Sub
```

Log das ações antes e depois das chamadas à classe-base

Figura 28 - Padrão Proxy MouseRecorder

5.5.3. Implementação do padrão de projeto Facade

Para que fosse possível acessar apenas uma classe de gravação única, e não uma classe de gravação de mouse e outra de teclado, foi criada a classe *Recorder*, implementando assim o padrão de projeto Facade. Este padrão de projeto foi implementado criando-se a classe Recorder e também fazendo com que as classes *KeyboardRecorderProxy* e *MouseRecorderProxy* fossem visíveis apenas internamente à biblioteca (Uso da marcação *Friend* na classe).

```
1 referência
Public Class Recorder
    Implements Interfaces.IRecorder

    'Objetos de Gravação de Mouse e Keyboard com Eventos
    Private WithEvents KeyboardRecorder As Proxies.KeyboardRecorderProxy
    Private WithEvents MouseRecorder As Proxies.MouseRecorderProxy
```

Figura 29 - Classe Recorder (Padrão Facade)

```
''' <summary>
''' Realiza a gravação de teclas e ações do Mouse
''' </summary>
15 referências
Public Sub StartRecording() Implements Interfaces.IRecorder.StartRecording

    KeyboardRecorder.StartRecording()
    MouseRecorder.StartRecording()

End Sub

''' <summary>
''' Pára a gravação
''' </summary>
16 referências
Public Sub StopRecording() Implements Interfaces.IRecorder.StopRecording

    If KeyboardRecorder IsNot Nothing Then
        KeyboardRecorder.StopRecording()
    End If

    If MouseRecorder IsNot Nothing Then
        MouseRecorder.StopRecording()
    End If

    If _oPlayback IsNot Nothing Then
        RaiseEvent GetRecordedActions(_oPlayback.GetRecordedActions())
    End If

End Sub
```

Figura 30 - Chamadas de método único da classe Recorder

5.6. Disponibilização do resultado

Os projetos do protótipo e da biblioteca estarão disponíveis no repositório do GitHub a seguir:

https://github.com/RafaBotossi/MisterHookLibrary

https://github.com/RafaBotossi/MisterHookPrototype

Além disso, o código-fonte também estará disponível em formato compactado (.Zip) junto ao presente trabalho.

6. Criação e execução de testes

Neste capítulo será abordada a forma que os testes foram criados, sua execução e o resultado obtido.

6.1. Tipos de testes realizados

Após a definição de como seria realizado o código do protótipo, foi realizada a análise dos tipos de testes passíveis de serem implementados no projeto.

Em um primeiro momento foi pensado nos testes unitários, testes de componente e nos testes de integração, porém após a análise foi verificado que os testes unitários seriam possíveis apenas com grande complexidade de desenvolvimento, por conta do grau de complexidade do projeto, que envolve o *input* de dados por parte do usuário para os testes de gravação de ações via teclado e mouse, inviabilizando um código de fácil implementação, pois simular ações de usuário via teclado e mouse não são de desenvolvimento trivial. Portanto, os testes escolhidos a serem realizados foram os testes de componente e integração, realizados de forma manual, com o apoio de um script de testes.

Além disso, a categoria dos testes realizados se restringiu apenas aos testes essenciais para garantir a funcionalidade do protótipo, porém é importante ressaltar que testes de segurança, de escalabilidade, de performance e muitos outros tipos não foram realizados por escolha do autor do trabalho, apesar de sua importância para o projeto.

6.2. Script de testes de componente e integração

Após o desenvolvimento das funcionalidades do protótipo, foi criado um script para teste e validação das mesmas.

Módulo	**Código do Teste/Funcionalidade**	**Descrição do Teste**	**Resultado**

Gravação	TST001 - Gravar Ação do usuário via Mouse e Teclado	Como usuário do sistema eu devo poder abrir a tela do sistema e clicar no botão para gravar as ações que desejo realizar. Para realizar este teste serão realizadas as seguintes ações após clicar no botão gravar, localizado no canto superior esquerdo do sistema, e esperar os segundos indicados para início da gravação: - Clicar no ícone do Windows no canto inferior esquerdo; - Digitar “Notepad” no teclado e apertar a tecla enter; - Após abrir o bloco de notas (Notepad), digitar “Teste da funcionalidade de gravação” e fechar o bloco de notas clicando no botão X no canto superior direito e escolhendo a opção “Não Salvar”; - Clicar no botão de parar a gravação (STOP), localizado no canto superior esquerdo do sistema. O sistema deverá apresentar o script gravado. **Cenário Alternativo 01:** Realizar outros tipos de gravação para validar o funcionamento do sistema. (Item aberto por conta da subjetividade do teste de cada indivíduo).	Passou
Script	TST002 - Salvar script das ações	Como usuário do sistema eu devo poder salvar o script gravado anteriormente e	Passou

	gravadas em arquivo de texto	apresentado em tela, no formato arquivo de texto (.txt) e em diretório de minha escolha. **Para realizar este teste faça as ações a seguir:** - Abrir o sistema; - Digitar a informação “Teste de gravação de script em arquivo texto” na caixa de texto da aba apresentada; - Clicar no menu “Arquivos->Salvar Script” - Salvar o arquivo no Desktop e verificar seu conteúdo para identificar se está igual ao digitado. **Cenário Alternativo 01:** - Realizar o teste acima, porém em vez de clicar no menu indicado, apertar o atalho de teclado CTRL+S (Control + S). **Cenário Alternativo 02:** Na tela de escolha de nome de arquivo para salvar o script, cancelar a ação para verificar comportamento do sistema.	
Script	TST003 - Abrir script das ações gravadas em arquivo de texto	Como usuário do sistema eu devo poder abrir um script salvo anteriormente. **Para realizar este teste faça as ações a seguir:** - Criar um arquivo de texto no bloco de notas, digitar o texto “Teste de abertura de script” e gravar o mesmo no desktop; - Abrir o sistema; - Clicar no menu “Arquivo->Abrir Script” e escolher o arquivo criado anteriormente;	Passou (Parcial – Cenário Alternativo 3 não implementado)

		- Verificar se o script foi aberto e está apresentando as mesmas informações salvas no arquivo. **Cenário Alternativo 01:** - Realizar o teste acima, porém em vez de clicar no menu indicado, apertar o atalho de teclado CTRL+O (Control + O). **Cenário Alternativo 02:** Na tela de escolha de arquivo para abrir, cancelar a ação para verificar comportamento do sistema. **Cenário Alternativo 03:** Tentar abrir um arquivo de script inválido. (Este teste foi indicado, porém não foi implementado no código a verificação de script inválido).	
Script	TST004 - Novo Teste	Como usuário do sistema eu devo poder criar um novo teste, sem afetar os testes que estão abertos. **Para realizar este teste faça as ações a seguir:** - Abrir o sistema; - Clicar no menu “Arquivo->Novo Script” e verificar se foi criada uma nova aba para a execução do novo script. **Cenário Alternativo 01:** - Realizar o teste acima, porém em vez de clicar no menu indicado, apertar o atalho de teclado CTRL+N (Control + N).	Passou

Reprodução	TST005 - Reproduzir ações gravadas anteriormente	Como usuário do sistema eu devo poder reproduzir as ações gravadas anteriormente em script. **Para realizar este teste faça as ações a seguir:** - Abrir o sistema; - Abrir o script gravado anteriormente através do teste TST003 ou realizar a gravação e um novo teste de acordo com o teste TST001; - Apertar o botão reproduzir (Play) e verificar se as ações realizadas condizem com o teste gravado. **Cenário Alternativo 01:** - Após o início da reprodução, apertar a tecla 'Pause' do teclado para identificar se o sistema interrompe a reprodução das ações do usuário via teclado e mouse.	Passou
Outros	TST006 – Abrir a tela Sobre	Como usuário do sistema eu devo poder abrir a tela sobre. Para realizar este teste, faça as ações a seguir: - Abrir o sistema; - Clicar no menu "Ajuda->Sobre". A tela do sistema deverá abrir	Passou
Outros	TST007 – Fechar o sistema	Como usuário do sistema eu devo poder finalizar o sistema. **Para realizar este teste, faça as ações a seguir:** - Abrir o sistema; - Clicar no menu "Arquivo->Sair".	Passou

		Cenário Alternativo 01: Caso exista algum script aberto e não salvo, será exibida uma mensagem de confirmação de finalização do sistema, pois os dados não salvos serão perdidos.	

6.3. Resultados obtidos

A realização dos testes foi muito importante para a validação das funcionalidades criadas e constituem um conjunto mínimo de testes para validar o sistema e garantir a qualidade do protótipo.

7. Considerações finais

Neste capítulo será analisado o resultado da criação do protótipo de sistema, bem como traçado um paralelo em relação ao que foi proposto e o que foi obtido. Também serão indicadas melhorias em futuras implementações no protótipo e sugestões para trabalhos futuros.

7.1. Sobre o projeto MisterHook

A criação de sistemas não é uma tarefa trivial. É necessário levantar requisitos, pensar nos diversos cenários de testes, realizar a criação de MVP's e protótipos. Essas ações são de grande valia para que no momento de se desenvolver o sistema, a quantidade de retrabalho seja mínima. Mesmo realizando todas estas etapas e pensando muito sobre o assunto antes de começar, ao se desenvolver o sistema, conceitos são melhores entendidos, novas idéias surgem e o trabalho desenhado sofre mudanças.

Este foi o cenário que ocorreu ao desenvolver o protótipo do sistema MisterHook. Ao se desenhar todas as funcionalidades, não foi pensado no uso do sistema pelo usuário e/ou desenvolvedor, trazendo assim uma gama de novas necessidades que ganharam prioridade sobre outras necessidades. Uma delas sendo a interface gráfica usando como modelo as interfaces gráficas dos softwares apresentados no capítulo 2 e a outra sendo a necessidade de se criar uma biblioteca de classes para uso por outros desenvolvedores que podem não desejam uma interface gráfica para uso das funcionalidades criadas.

Por conta dessas novas necessidades, não foi possível implementar duas das funcionalidades que poderiam deixar o trabalho mais completo, a funcionalidade de um script mais amigável ao usuário, explicitado através do requisito 'RF005 - Geração mais amigável de script de ações do usuário para o desenvolvedor' e a funcionalidade de interpretação de comandos no script, usando o padrão de projeto Interpreter, explicitado através do requisito 'RF006 - Integração de interpretador de linguagens de programação com o script gerado'.

O estudo do código necessário para implementar este projeto foi particularmente difícil para o autor do trabalho, por conta do assunto de se gravar as ações do usuário e reproduzi-las posteriormente não ser um assunto de grande extensão na internet, havendo assim poucos tópicos para consulta.

Apesar da implementação dos dois requisitos mencionados anteriormente não terem sido realizadas e também da dificuldade de encontrar material sobre o assunto na internet, o protótipo pode ser considerado como implementado com êxito, pois as funcionalidades essenciais foram criadas e funcionam de acordo com o que foi desenhado inicialmente, traçando assim uma trilha para quem quiser desenvolver um sistema parecido com o sistema proposto neste trabalho, ou estudar mais a fundo o funcionamento de um sistema do tipo.

7.2. Sobre o trabalho acadêmico

O estudo sobre as ferramentas de testes de gravação e reprodução das ações do usuário existentes proporcionou ao autor do trabalho diversas ideias de implementação para a criação do protótipo, principalmente na área de interface gráfica. Além disso, o estudo sobre os diversos tipos diferentes de padrões de projeto proporcionou um entendimento mais profundo sobre o tema, possibilitando a criação de códigos mais robustos e menor refatoração de código ao se deparar com novas necessidades.

Também, através deste trabalho acadêmico, será possível entender mais sobre os diferentes tipos de testes e onde o teste automatizado se encaixa, além de propiciar aos desenvolvedores de sistemas parecidos com o trabalho proposto um ponto de partida para a criação de novos sistemas.

7.3. Melhorias a serem realizadas em trabalhos futuros

Existem diversas melhorias a serem implementadas antes do protótipo poder ser entregue como um sistema completo ao mercado. Abaixo serão listadas algumas destas melhorias, a fim de se criar uma linha-guia para possíveis trabalhos futuros, do autor ou não.

- **Implementação do requisito ‘RF005 - Geração mais amigável de script de ações do usuário para o desenvolvedor’ –** Este é um dos pontos mais fáceis de serem implementados. Se, ao executar os métodos de gravação de Mouse e Teclado, o sistema salvar um comando mais simples indicando o que ocorreu, ficará fácil ao desenvolvedor entender o script. Um exemplo disso é a linha de script abaixo, que poderia ser escrita de forma mais clara, mostrada também no exemplo abaixo:
  - Linha de script atual para a gravação da ação de movimento do mouse:
    - {"Code":{"value":512},"MouseStruct":{"dwExtraInfo":1363312000,"hwnd":0,"pt":{"x":326,"y":216},"wHitTestCode":0},"Timestamp":"\/Date(1575055675130-0300)\/"};
  - Linha de script após a alteração, para a gravação da ação de movimento do mouse:
    - MoveMouse(326,216,0.1), onde os primeiros parâmetros poderiam ser coordenadas X e Y e o parâmetro final um indicador de tempo de espera antes ou após a realização da ação (para correta interpretação do tempo que o usuário realizou a ação);
- **Implementação do requisito ‘RF006 - Integração de interpretador de linguagens de programação com o script gerado’ –** Através do uso do padrão de projeto Interpreter, é possível aumentar a capacidade de testes do protótipo. O projeto atualmente conta com a gravação e reprodução das ações do usuário, porém após a implementação deste padrão de projeto, seria possível realizar alterações no script de modo a torná-lo reutilizável em diferentes casos de testes. Um exemplo seria realizar a gravação das ações do usuário, onde ele digitaria um valor em uma caixa de texto de um programa qualquer. Com a versão atual, apenas este valor será digitado, porém com a implementação deste requisito seria possível

repetir várias vezes o mesmo bloco de script para que sejam digitadas informações diversas, por exemplo vindo de um banco de dados, e não de um input manual de teclado, conforme o exemplo abaixo indica;

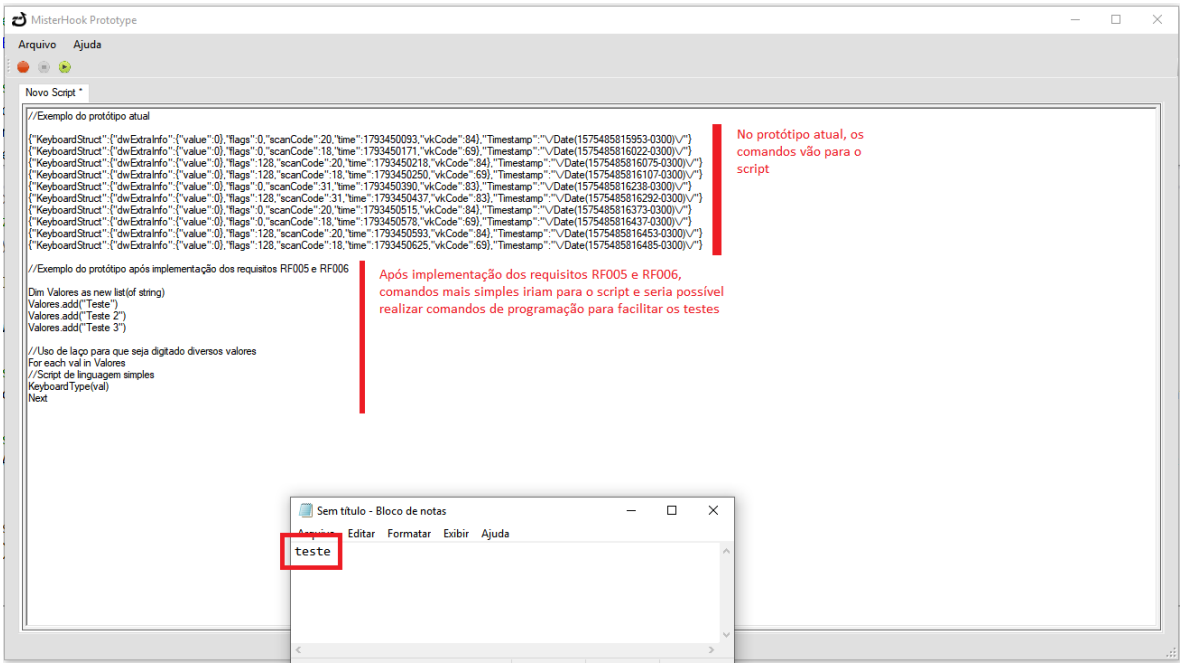


Figura 31 - Possível implementação requisitos RF005 e RF006

- **Implementação do requisito 'RF007 - Verificação do resultado de ação realizada para analisar se o teste funcionou ou não' –** As ferramentas de testes possuem funcionalidades para indicar se um determinado teste foi validado ou falhou, geralmente chamado de *'Assert'*. Este requisito indica a criação de uma funcionalidade no sistema para realizar este tipo de ação. Esta funcionalidade deverá ser criada no projeto da biblioteca de classes e utilizada na interface gráfica. Um exemplo de como poderia ser a parte visual desta implementação é indicada a seguir;

Figura 32 - Possível implementação do requisito RF007 (Com apoio do requisito RF005)

- **Implementação do requisito 'RF008 - Gravação de capturas de tela para a visualização das ações que estão sendo gravadas, para posterior visualização após a gravação' –** Outro requisito interessante para ser implementado é o requisito RF008, que indica a gravação da tela a cada clique do mouse ou em momentos específicos definidos via script, para posterior visualização pelo desenvolvedor do script. Esta funcionalidade é de grande auxílio para visualizar o que foi feito em um determinado teste, no momento de atualização do script em questão. Para implementar este requisito seria necessário realizar a gravação do script em um banco de dados e criar um mecanismo para entender em qual momento foi realizado o print para que seja visualizado posteriormente.

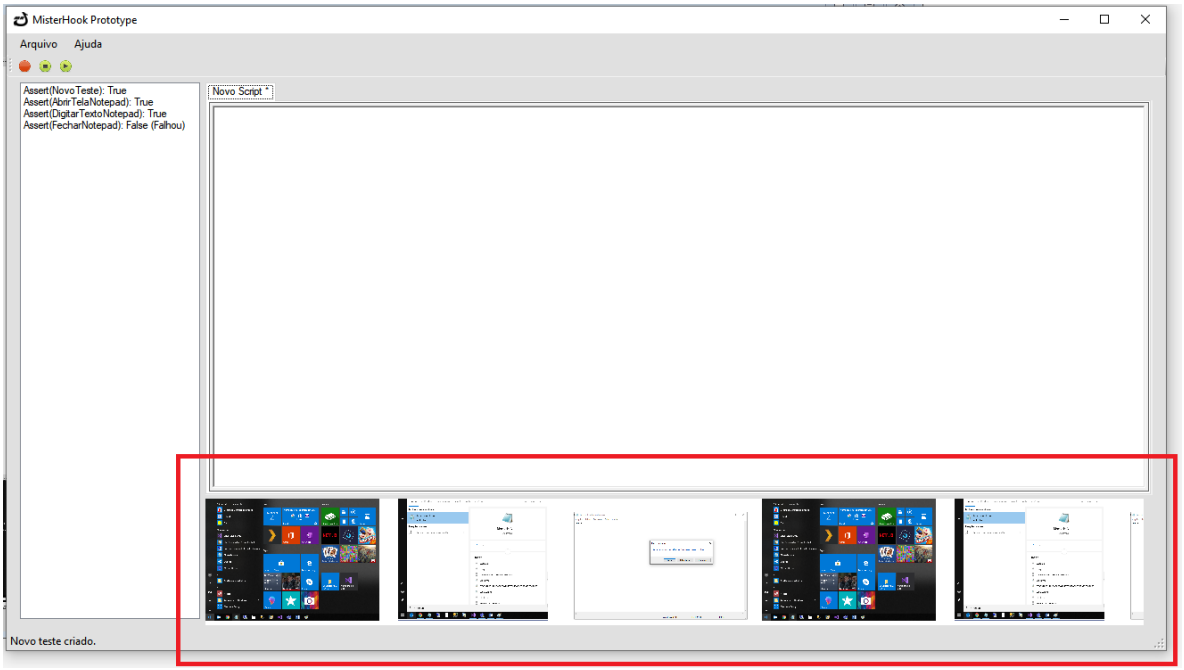


Figura 33 - Possível implementação do requisito RF008

- **Outras melhorias –** Além das melhorias apresentadas acima, é possível indicar diversas outras, como a disponibilização de ferramentas para descobrir a posição do mouse (para os casos onde o desenvolvedor quer criar o script sem gravá-lo primeiro); comandos de script para facilitar a criação do mesmo, como abertura de telas, fechamento de telas, foco em telas, abrir e fechar programas com apenas uma linha de código e afins; repositório de templates de script, para a comunidade poder criar e compartilhar templates e ferramenta de salvamento de scripts na nuvem.

7.4. Conclusão

Após a finalização deste trabalho, foi possível criar um protótipo de ferramenta de testes automatizados do tipo gravação e reprodução de ações do usuário, específico para ambientes *Windows Desktop e a*pesar de ainda não ser possível o uso da ferramenta em ambientes corporativos por conta da falta de algumas funcionalidades, através da implementação completa dos requisitos indicados no capítulo 7.3 será possível ter uma ferramenta mais robusta e que

possa ser utilizada pelo mercado, sendo assim uma opção atrativa considerando o custo zero de implantação da ferramenta.

Além disso, uma utilização que o protótipo do sistema já permite é a utilização na automação de processos simples e repetitivos, realizados por pessoas que não necessitam de conhecimentos avançados em programação, permitidos através da interface gráfica do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

10 ferramentas de automação de testes mais usadas. Prime Control, 2018, Disponível em: <http://www.primecontrol.com.br/10-ferramentas-de-automacao-de-testes-mais-usadas/>. Acesso em: Julho de 2018.

ANDERSON, Brian. Best Automation Testing Tools, Medium, 2017. Disponível em: <https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2>. Acesso em: Julho de 2018.

BASTOS, Anderson; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. Base de conhecimento em teste de software. São Paulo, 1ª edição, Editora Martins fontes, 2012.

CHEQUE, Paulo; KON, Fabio. A importância dos testes automatizados: Controle ágil, rápido e confiável de qualidade. Engenharia de Software Magazine, p. 1:3:54-57, 2008.

CHEQUE, Paulo; KON, Fabio. Padrões de testes automatizados. Dissertação de mestrado, Instituto de Matemática e estatística Universidade de São Paulo, p. 9-10, 2011.

DUSTIN, Elfriede; RASHKA, Jeff; et al. Automated Software Testing: introduction, management and performance. p. 50, 1999.

FEWSTER, Mark; GRAHAM, Dorothy. Software Test Automation: Effective use of teste execution tools. Addison-Wesley, p. 9-10, 1999.

Função SetWindowsHookEx, Microsoft, 2018. Disponível em: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa>. Acesso em: Junho de 2019.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Padrões de projeto - Soluções reutilizáveis de software orientado a objetos. p. 17-20. Editora Bookman, 2008.

GONÇALVES, Priscila de F., et al, Testes de software e gerência de configuração, p. 14-15:25, 2019.

RIOS, Emerson; Moreira, Trayahú. Teste de Software. São Paulo, 3º edição revisada e atualizada, Editora Alta Books, 2013.

Top 20 Automation testing tools, Software Testing Help, 2019. Disponível em: <https://www.softwaretestinghelp.com/top-20-automation-testing-tools>. Acesso em Julho de 2018.

ANEXO I - CÓDIGO-FONTE DO PROTÓTIPO MISTERHOOK

O código-fonte do protótipo é inviável de ser adicionado neste documento por conta do seu tamanho, portanto será disponibilizado na web nos endereços indicados no fim do capítulo 5.

ANEXO III - CÓDIGO-FONTE DO PROTÓTIPO BÁSICO PARA VERIFICAÇÃO DA VIABILIDADE DO PROJETO

O código-fonte estará sempre anexo a este trabalho de forma integral, assim trabalhos futuros poderão utilizar este conhecimento para exploração de novas funcionalidades. Cada item em negrito equivale a um arquivo/classe.

MODELS

KEYBOARDHOOKMODEL

```
NAMESPACE MISTERHOOK.MODELS

  'MODELO CRIADO PARA TAMBÉM TER O TIMESTAMP DE QUANDO FOI PRESSIONADA A TECLA
  PUBLIC CLASS KEYBOARDHOOKMODEL
    PUBLIC TIMESTAMP AS DATE 'TIMESTAMP USADO PARA FAZER O TIMING ENTRE AS AÇÕES
    PUBLIC KEYBOARDSTRUCT AS STRUCTS.KEYBOARDHOOKSTRUCT 'ESTRUTURA DE HOOK DO TECLADO
  END CLASS

END NAMESPACE
```

MOUSEHOOKMODEL

```
NAMESPACE MISTERHOOK.MODELS

  PUBLIC CLASS MOUSEHOOKMODEL
    PUBLIC CODE AS INTPTR
    PUBLIC TIMESTAMP AS DATE
    PUBLIC MOUSESTRUCT AS STRUCTS.MOUSEHOOKSTRUCT
  END CLASS

END NAMESPACE
```

PLAYBACK

```
IMPORTS MISTERHOOK.MISTERHOOK
IMPORTS MISTERHOOK.MISTERHOOK.MODELS
IMPORTS MISTERHOOK.MISTERHOOK.STRUCTS
IMPORTS SYSTEM.WEB.SCRIPT.SERIALIZATION

NAMESPACE PLAYBACK

  ''' <SUMMARY>
  ''' CLASSE USADA PARA REALIZAR PLAYBACK DA GRAVAÇÃO REALIZADA
  ''' </SUMMARY>
  PUBLIC CLASS PLAYBACKACTIONS

    PRIVATE _ACTIONSTRUCTS AS LIST(OF OBJECT) 'AÇÕES DE TECLADO OU MOUSE EM ORDEM CRONOLÓGICA
    PRIVATE _STOPPLAYBACK AS BOOLEAN 'SE É NECESSÁRIO DAR STOP NO PLAYBACK
```

```
    Private WithEvents KeyboardRecorder As Record.KeyboardRecorder 'Usamos uma gravação de teclas para verificar se a pessoa clicou na tecla Pause para parar a gravação

    Private _HInstance As IntPtr = IntPtr.Zero 'É necessário passar o handle da aplicação que chamou a gravação/playback, se não o hook não funciona
    Private _PathToSave As String, _Filename As String 'Locais para salvar o script se indicado

    ''' <summary>
    ''' Construtor da classe de Playback
    ''' </summary>
    ''' <param name="hInstance">É necessário passar o handle da aplicação que chamou a gravação/playback, se não o hook não funciona</param>
    ''' <param name="PathToSave">Qual o caminho do script para rodar o playback</param>
    ''' <param name="FileName">Qual o nome do arquivo do script para rodar o playback</param>
    Public Sub New(hInstance As IntPtr, Optional PathToSave As String = "", Optional FileName As String = "")

        _HInstance = hInstance
        _PathToSave = PathToSave
        _Filename = FileName

    End Sub

    ''' <summary>
    ''' Realiza o Playback gravado anteriormente
    ''' </summary>
    Public Sub Playback(Optional PathAndFilename As String = "")

        If Not String.IsNullOrEmpty(PathAndFilename) Then
            FillActionStructs(PathAndFilename)
        End If

        _StopPlayback = False

        If _ActionStructs IsNot Nothing AndAlso _ActionStructs.Count > 0 Then

            'Cria uma gravação de teclado para ver se a pessoa vai apertar a tecla Pause para parar o playback
            KeyboardRecorder = New Record.KeyboardRecorder(_HInstance)
            KeyboardRecorder.StartRecording()

            Dim lastTimestamp As Date = "1899-01-01"
            For Each oStruct As Object In _ActionStructs

                If _StopPlayback Then Exit Sub

                'Verifica se faz playback de teclado ou mouse
                Sleep(oStruct, lastTimestamp)
                If oStruct.GetType() Is GetType(KeyboardHookModel) Then
                    KeyboardPlayback(oStruct)
                ElseIf oStruct.GetType() Is GetType(MouseHookModel) Then
                    MousePlayback(oStruct)
                End If

            Next

            'Finaliza verificação de teclas do playback
```

```
        If KeyboardRecorder IsNot Nothing Then
            KeyboardRecorder.StopRecording()
            KeyboardRecorder = Nothing
        End If

    End If

End Sub

Private Sub FillActionStructs(PathAndFilename As String)

    If System.IO.File.Exists(PathAndFilename) Then

        Dim reader As IO.StreamReader = My.Computer.FileSystem.OpenTextFileReader(PathAndFilename)

        EraseData()

        If _ActionStructs Is Nothing Then
            _ActionStructs = New List(Of Object)
        End If

        Dim json As String = String.Empty
        Do

            json = reader.ReadLine

            If json IsNot Nothing Then
                If json.Contains("MouseStruct") Then
                    Dim MouseHookModel As New Models.MouseHookModel
                    Dim MS As New IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(json))
                    Dim ser As New Runtime.Serialization.Json.DataContractJsonSerializer(MouseHookModel.GetType())
                    MouseHookModel = ser.ReadObject(MS)
                    MS.Close()
                    _ActionStructs.Add(MouseHookModel)
                Else
                    Dim KeyboardHookModel As New Models.KeyboardHookModel
                    Dim MS As New IO.MemoryStream(System.Text.Encoding.UTF8.GetBytes(json))
                    Dim ser As New Runtime.Serialization.Json.DataContractJsonSerializer(KeyboardHookModel.GetType())
                    KeyboardHookModel = ser.ReadObject(MS)
                    MS.Close()
                    _ActionStructs.Add(KeyboardHookModel)
                End If
            End If

        Loop Until json Is Nothing

        reader.Close()

    End If

End Sub

''' <summary>
''' Pára o playback
''' </summary>
```

```
Public Sub StopPlayback()

    _StopPlayback = True

End Sub

''' <summary>
''' Entre as ações é necessário esperar o tempo que o usuário esperou para realizar a ação. Utiliza os timestamps para isso
''' </summary>
''' <param name="oStruct">Estrutura</param>
''' <param name="lastTimestamp">Último Timestamp para cálculo do tempo entre ações</param>
Private Sub Sleep(oStruct As Object, ByRef lastTimestamp As Date)

    Dim timestamp As Date = oStruct.timestamp
    If lastTimestamp = "1899-01-01" Then
        lastTimestamp = timestamp
    End If
    Dim span As TimeSpan = timestamp - lastTimestamp
    Dim ms As Integer = span.TotalMilliseconds
    lastTimestamp = timestamp
    Threading.Thread.Sleep(ms) 'Espera ms de tempo antes da próxima ação

End Sub

''' <summary>
''' Realiza o Log da ação nos objetos de gravação de Teclado e Mouse
''' </summary>
''' <param name="KeyStruct">Pode ser KeyboardHookModel ou MouseHookModel</param>
Public Sub LogAction(KeyStruct As Object)

    If _ActionStructs Is Nothing Then
        _ActionStructs = New List(Of Object)
    End If
    _ActionStructs.Add(KeyStruct)

End Sub

Public Sub EraseData()

    _ActionStructs = New List(Of Object)

End Sub

''' <summary>
''' Salva o que foi gravado em arquivo
''' </summary>
''' <param name="PathToSave">Caminho para salvar o arquivo de script</param>
''' <param name="Filename">Nome do arquivo de script para salvar</param>
Public Sub SaveRecordToFile(PathToSave As String, Filename As String)

    If _ActionStructs IsNot Nothing AndAlso _ActionStructs.Count > 0 Then

        If Not String.IsNullOrEmpty(PathToSave) Then

            If String.IsNullOrEmpty(Filename) Then
```

```
            Filename = String.Format("MisterHook Recording {0}.txt", Now.ToString("yyyy-MM-dd hhmmss"))
        End If

        If System.IO.Directory.Exists(PathToSave) Then

            Dim PathAndFilename As String = PathToSave & "\" & Filename
            Dim objWriter As New System.IO.StreamWriter(PathAndFilename)

            Dim x As Integer = 0
            For Each oStruct As Object In _actionStructs

                x += 1

                If oStruct.GetType() Is GetType(KeyboardHookModel) Then

                    Dim KeyboardHookModel As KeyboardHookModel = oStruct

                    Dim ms As New System.IO.MemoryStream()
                    Dim serializer As New System.Runtime.Serialization.Json.DataContractJsonSerializer(KeyboardHookModel.GetType())
                    serializer.WriteObject(ms, KeyboardHookModel)
                    Dim json() As Byte = ms.ToArray()
                    ms.Close()
                    Dim str As String = System.Text.Encoding.UTF8.GetString(json, 0, json.Length)
                    objWriter.WriteLine(str)

                ElseIf oStruct.GetType() Is GetType(MouseHookModel) Then

                    Dim MouseHookModel As MouseHookModel = oStruct

                    Dim ms As New System.IO.MemoryStream()
                    Dim serializer As New System.Runtime.Serialization.Json.DataContractJsonSerializer(MouseHookModel.GetType())
                    serializer.WriteObject(ms, MouseHookModel)
                    Dim json() As Byte = ms.ToArray()
                    ms.Close()
                    Dim str As String = System.Text.Encoding.UTF8.GetString(json, 0, json.Length)
                    objWriter.WriteLine(str)

                End If

            Next

            objWriter.Close()

        End If

    End If

End If

End Sub

''' <summary>
''' Checa se o usuário apertou a tecla Pause para parar o Playback
''' </summary>
''' <param name="Key">Tecla Pressionada</param>
```

```
    Private Sub KeyboardRecorder_KeyDown(Key As Keys) Handles KeyboardRecorder.KeyDown

        If Key = Keys.Pause Then
            KeyboardRecorder.StopRecording()
            KeyboardRecorder = Nothing
            StopPlayback()
        End If

    End Sub

#Region "Keyboard"

    Private Const KEYEVENTF_KEYUP = &H2 'Constante de tecla solta

    Private Declare Sub keybd_event Lib "user32.dll" (ByVal bVk As Byte, ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long) 'Usado para realizar o 'send keys' para o playback

    ''' <summary>
    ''' Realiza o Playback do Teclado
    ''' </summary>
    ''' <param name="oStruct">Estrutura do tipo KeyboardHookModel</param>
    Private Sub KeyboardPlayback(ByRef oStruct As Object)

        Dim KeyboardHookModel As KeyboardHookModel = oStruct

        'Na ação de keyup tivemos que forçar usando a const KEYEVENTF_KEYUP, pois enviar como a versão abaixo não funcionou
        If KeyboardHookModel.KeyboardStruct.flags = KeyboardHookFlagsStruct.LLKHF_UP Then
            keybd_event(KeyboardHookModel.KeyboardStruct.vkCode, 0, KEYEVENTF_KEYUP, 0)
        Else
            keybd_event(KeyboardHookModel.KeyboardStruct.vkCode, KeyboardHookModel.KeyboardStruct.scanCode, KeyboardHookModel.KeyboardStruct.flags, KeyboardHookModel.KeyboardStruct.dwExtraInfo)
        End If

    End Sub

#End Region

#Region "Mouse"

    'Constantes usadas no playback. Essas de cima são diferentes das realizadas na gravação, por isso a necessidade de um De/Para
    Private Const MOUSEEVENTF_LEFTDOWN As UInt32 = &H2
    Private Const MOUSEEVENTF_LEFTUP As UInt32 = &H4
    Private Const MOUSEEVENTF_RIGHTDOWN As UInt32 = &H8
    Private Const MOUSEEVENTF_RIGHTUP As UInt32 = &H10
    Private Const MOUSEEVENTF_MIDDLEDOWN As UInt32 = &H20
    Private Const MOUSEEVENTF_MIDDLEUP As UInt32 = &H40
    Private Const MOUSEEVENTF_XDOWN As UInt32 = &H80
    Private Const MOUSEEVENTF_XUP As UInt32 = &H100
    Private Const MOUSEEVENTF_WHEEL As UInt32 = &H800
    Private Const MOUSEEVENTF_VIRTUALDESK As UInt32 = &H4000
    Private Const MOUSEEVENTF_ABSOLUTE As UInt32 = &H8000
    Private Const INPUT_MOUSE As UInt32 = 0

    Private Const WM_MOUSEMOVE As Integer = &H200
    Private Const WM_LBUTTONDOWN As Integer = &H201
```

```
    Private Const WM_LBUTTONUP As Integer = &H202
    Private Const WM_LBUTTONDBLCLK As Integer = &H203
    Private Const WM_RBUTTONDOWN As Integer = &H204
    Private Const WM_RBUTTONUP As Integer = &H205
    Private Const WM_RBUTTONDBLCLK As Integer = &H206
    Private Const WM_MBUTTONDOWN As Integer = &H207
    Private Const WM_MBUTTONUP As Integer = &H208
    Private Const WM_MBUTTONDBLCLK As Integer = &H209
    Private Const WM_MOUSEWHEEL As Integer = &H20A
    Private Const WM_MOUSEHWHEEL As Integer = &H20E

    Public Declare Auto Function SetCursorPos Lib "user32.dll" (ByVal X As Integer, ByVal Y As Integer) As Long 'Posiciona o Mouse para o playback
    Private Declare Sub mouse_event Lib "user32.dll" (dwFlags As UInteger, dx As UInteger, dy As UInteger, dwData As UInteger, dwExtraInfo As Integer) 'Realiza o Click do Mouse para o playback

    ''' <summary>
    ''' Realiza o Playback do Mouse
    ''' </summary>
    ''' <param name="oStruct">Estrutura do tipo MouseHookModel</param>
    Private Sub MousePlayback(ByRef oStruct As Object)

        Dim MouseHookModel As MouseHookModel = oStruct

        'Se for movimento de mouse, usa Cursor.Position(), se for click, então usa mouse_event
        If MouseHookModel.Code = WM_MOUSEMOVE Then
            Cursor.Position() = New Point(MouseHookModel.MouseStruct.pt.x, MouseHookModel.MouseStruct.pt.y)
        Else
            mouse_event(GetMouseEventCode(MouseHookModel.Code), 0, 0, 0, 0)
        End If

    End Sub

    'Função De/Para dos códigos de gravação para os de playback
    Private Function GetMouseEventCode(Code As IntPtr)

        Select Case Code
            Case WM_LBUTTONDOWN
                Return MOUSEEVENTF_LEFTDOWN
            Case WM_LBUTTONUP
                Return MOUSEEVENTF_LEFTUP
            Case WM_RBUTTONDOWN
                Return MOUSEEVENTF_RIGHTDOWN
            Case WM_RBUTTONUP
                Return MOUSEEVENTF_RIGHTUP
            Case Else
                Return Nothing
        End Select

    End Function

#End Region

End Class
```

```
End Namespace
```

Record

KeyboardRecorder

```
Imports MisterHook.MisterHook.Models
Imports MisterHook.MisterHook.Structs
Imports System.Windows.Forms
Imports System.Globalization
Imports System.Runtime.InteropServices

Namespace MisterHook.Record

    Public Class KeyboardRecorder

#Region "DLL"

        'Realiza o Hook
        <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
        Private Overloads Shared Function SetWindowsHookEx(ByVal idHook As Integer, ByVal HookProc As KeyboardHookProcedure, ByVal hInstance As IntPtr, ByVal wParam As Integer) As Integer
        End Function

        'Chama próxima Ação
        <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
        Private Overloads Shared Function CallNextHookEx(ByVal idHook As Integer, ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer
        End Function

        'Remove o Hook
        <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
        Private Overloads Shared Function UnhookWindowsHookEx(ByVal idHook As Integer) As Boolean
        End Function

#End Region

        'Eventos usados para Log
        Public Shared Event KeyDown(ByVal Key As Keys)
        Public Shared Event KeyUp(ByVal Key As Keys)

        Private Const HC_ACTION As Integer = 0

        'Constantes de Keyboard
        Private Const WH_KEYBOARD_LL As Integer = 13
        Private Const WM_KEYDOWN = &H100
        Private Const WM_KEYUP = &H101
        Private Const WM_SYSKEYDOWN = &H104
        Private Const WM_SYSKEYUP = &H105
        Private Const WM_ESCAPE = &H100

        'Função que vai realizar a gravação de teclas
        Private Delegate Function KeyboardHookProcedure(ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer
```

```
Private KeyboardHookStructDelegate As KeyboardHookProcedure = New KeyboardHookProcedure(AddressOf HookKey)
Private HHookID As IntPtr = IntPtr.Zero

Private _oPlayback As Playback.PlaybackActions
Private _hinstance As IntPtr = IntPtr.Zero

''' <summary>
''' Construtor da Classe de gravação de Teclado
''' </summary>
''' <param name="hinstance">É necessário passar o handle do programa que está realizando a gravação, senão não funciona.</param>
''' <param name="oPlayback">Se quiser realizar o playback depois, é necessário passar este objeto</param>
Public Sub New(hinstance As IntPtr, Optional ByRef oPlayback As Playback.PlaybackActions = Nothing)

  _oPlayback = oPlayback
  _hinstance = hinstance

End Sub

''' <summary>
''' Realiza a gravação de teclas
''' </summary>
Public Sub StartRecording()

  HHookID = SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardHookStructDelegate, _hinstance, 0)
  If HHookID = IntPtr.Zero Then
    Throw New Exception("Não foi possível realizar o Hook de Teclado")
  End If

End Sub

''' <summary>
''' Pára a gravação
''' </summary>
Public Sub StopRecording()

  Finalize()

End Sub

''' <summary>
''' Pára a gravação
''' </summary>
Protected Overrides Sub Finalize()

  If Not HHookID = IntPtr.Zero Then
    UnhookWindowsHookEx(HHookID)
  End If
  MyBase.Finalize()

End Sub

''' <summary>
''' Função que grava a tecla
''' </summary>
```

```
    ''' <param name="nCode">Ação</param>
    ''' <param name="wParam">Código da Tecla</param>
    ''' <param name="lParam">Estrutura de tecla pressionada</param>
    ''' <returns></returns>
    Private Function HookKey(ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer

        Dim HookReturn As Integer = CallNextHookEx(IntPtr.Zero, nCode, wParam, lParam) 'Faz a chamada antes de tudo, para evitar problemas
        If (nCode = HC_ACTION) Then

            Select Case wParam
                Case WM_KEYDOWN, WM_SYSKEYDOWN, WM_KEYUP, WM_SYSKEYUP 'Somente grava estes tipos

                    Dim KeyboardStruct As Structs.KeyboardHookStruct = CType(Marshal.PtrToStructure(lParam, KeyboardStruct.GetType()), Structs.KeyboardHookStruct) 'Transforma lparam na estrutura KeyboardHookStruct
                    Dim KeyboardHookModel As New Models.KeyboardHookModel
                    With KeyboardHookModel
                        .Timestamp = DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss.fff", CultureInfo.InvariantCulture) 'Salva timestamp para uso no playback
                        .KeyboardStruct = KeyboardStruct
                    End With

                    LogAction(KeyboardHookModel) 'Realiza log da tecla pressionada
                    RaiseEvents(wParam, KeyboardStruct) 'Chama Eventos para log

            End Select
        End If
        Return HookReturn

    End Function

    ''' <summary>
    ''' Chama eventos para log
    ''' </summary>
    ''' <param name="wParam">Tecla Pressionada</param>
    ''' <param name="KeyboardStruct">Estrutura de Tecla para passar pros eventos</param>
    Private Sub RaiseEvents(wParam As IntPtr, KeyboardStruct As Structs.KeyboardHookStruct)

        Dim Key As Keys = CType(KeyboardStruct.vkCode, Keys)

        If wParam = WM_KEYDOWN OrElse wParam = WM_SYSKEYDOWN Then
            RaiseEvent KeyDown(Key)
        ElseIf wParam = WM_KEYUP OrElse wParam = WM_SYSKEYUP Then
            RaiseEvent KeyUp(Key)
        End If

    End Sub

    ''' <summary>
    ''' Realiza Log da ação para o Playback
    ''' </summary>
    ''' <param name="KeyboardHookModel">Classe da ação realizada</param>
    Private Sub LogAction(KeyboardHookModel As Models.KeyboardHookModel)

        If _oPlayback IsNot Nothing Then
            _oPlayback.LogAction(KeyboardHookModel)
```

```
        End If

    End Sub

End Class

End Namespace
```

MouseRecorder

```
Imports MisterHook.Models
Imports MisterHook.Structs
Imports System.Globalization
Imports System.Runtime.InteropServices

Namespace MisterHook.Record

    Public Class MouseRecorder

#Region "DLL"

    'Realiza o Hook
    <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
    Private Overloads Shared Function SetWindowsHookEx(ByVal idHook As Integer, ByVal HookProc As MouseHookProc, ByVal hInstance As IntPtr, ByVal wParam As Integer) As Integer
    End Function

    'Chama próxima Ação
    <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
    Private Overloads Shared Function CallNextHookEx(ByVal idHook As Integer, ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer
    End Function

    'Remove o Hook
    <DllImport("user32.dll", CharSet:=CharSet.Auto, CallingConvention:=CallingConvention.StdCall)>
    Private Overloads Shared Function UnhookWindowsHookEx(ByVal idHook As Integer) As Boolean
    End Function

#End Region

    'Eventos usados para Log
    Public Shared Event MouseLDown()
    Public Shared Event MouseLUp()
    Public Shared Event MouseRDown()
    Public Shared Event MouseRUp()

    Private Const HC_ACTION As Integer = 0

    'Constantes de Mouse
    Private Const WH_MOUSE As Integer = 7
    Private Const WH_MOUSE_LL As Integer = 14
    Private Const WM_MOUSEMOVE As Integer = &H200
    Private Const WM_LBUTTONDOWN As Integer = &H201
    Private Const WM_LBUTTONUP As Integer = &H202
    Private Const WM_LBUTTONDBLCLK As Integer = &H203
```

```
Private Const WM_RBUTTONDOWN As Integer = &H204
Private Const WM_RBUTTONUP As Integer = &H205
Private Const WM_RBUTTONDBLCLK As Integer = &H206
Private Const WM_MBUTTONDOWN As Integer = &H207
Private Const WM_MBUTTONUP As Integer = &H208
Private Const WM_MBUTTONDBLCLK As Integer = &H209
Private Const WM_MOUSEWHEEL As Integer = &H20A
Private Const WM_MOUSEHWHEEL As Integer = &H20E

'Função que vai realizar a gravação de mouse
Private Delegate Function MouseHookProc(ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer

Private MouseHookProcDelegate As MouseHookProc = New MouseHookProc(AddressOf HookMouse)
Private HHookID As IntPtr = IntPtr.Zero

Private _oPlayback As Playback.PlaybackActions
Private _hinstance As IntPtr = IntPtr.Zero

''' <summary>
''' Construtor da Classe de gravação de Teclado
''' </summary>
''' <param name="hinstance">É necessário passar o handle do programa que está realizando a gravação, senão não funciona.</param>
''' <param name="oPlayback">Se quiser realizar o playback depois, é necessário passar este objeto</param>
Public Sub New(hinstance As IntPtr, Optional ByRef oPlayback As Playback.PlaybackActions = Nothing)

    _oPlayback = oPlayback
    _hinstance = hinstance

End Sub

''' <summary>
''' Realiza a Gravação de Mouse
''' </summary>
Public Sub StartRecording()

    HHookID = SetWindowsHookEx(WH_MOUSE_LL, MouseHookProcDelegate, _hinstance, 0)
    If HHookID = IntPtr.Zero Then
        Throw New Exception("Não foi possível realizar o Hook de Mouse")
    End If

End Sub

''' <summary>
''' Pára a gravação
''' </summary>
Public Sub StopRecording()

    Finalize()

End Sub

''' <summary>
''' Pára a gravação
''' </summary>
Protected Overrides Sub Finalize()
```

```
        If Not HHookID = IntPtr.Zero Then
            UnhookWindowsHookEx(HHookID)
        End If
        MyBase.Finalize()

    End Sub

    ''' <summary>
    ''' Função que grava a ação do Mouse
    ''' </summary>
    ''' <param name="nCode">Ação</param>
    ''' <param name="wParam">Código da ação do mouse</param>
    ''' <param name="lParam">Estrutura da ação do mouse</param>
    ''' <returns></returns>
    Private Function HookMouse(ByVal nCode As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer

        Dim HookReturn As Integer = CallNextHookEx(IntPtr.Zero, nCode, wParam, lParam) 'Faz a chamada antes de tudo, para evitar problemas
        If (nCode = HC_ACTION) Then

            Select Case wParam
                Case WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP 'Somente grava estes tipos

                Dim MouseStruct As Structs.MouseHookStruct = CType(Marshal.PtrToStructure(lParam, MouseStruct.GetType()), Structs.MouseHookStruct) 'Transforma lparam na estrutura MouseHookStruct
                Dim MouseHookModel As New Models.MouseHookModel
                With MouseHookModel
                    .Code = wParam 'Salva código, pois não tem essa info no MouseHookStruct
                    .Timestamp = DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss.fff", CultureInfo.InvariantCulture) 'Salva timestamp para uso no playback
                    .MouseStruct = MouseStruct
                End With

                LogAction(MouseHookModel) 'Realiza log da tecla pressionada
                RaiseEvents(wParam) 'Chama Eventos para log

            End Select

        End If
        Return HookReturn

    End Function

    ''' <summary>
    ''' Chama eventos para Log
    ''' </summary>
    ''' <param name="wParam">Ação do Mouse</param>
    Private Sub RaiseEvents(wParam As IntPtr)

        If wParam = WM_LBUTTONDOWN Then
            RaiseEvent MouseLDown()
        ElseIf wParam = WM_LBUTTONUP Then
            RaiseEvent MouseLUp()
        ElseIf wParam = WM_RBUTTONDOWN Then
            RaiseEvent MouseRDown()
```

```
        ElseIf wParam = WM_RBUTTONUP Then
            RaiseEvent MouseRUp()
        End If

    End Sub

    ''' <summary>
    ''' Realiza Log da ação para o Playback
    ''' </summary>
    ''' <param name="MouseHookModel">Classe da ação realizada</param>
    Private Sub LogAction(MouseHookModel As models.MouseHookModel)

        If _oPlayback IsNot Nothing Then
            _oPlayback.LogAction(MouseHookModel)
        End If

    End Sub

  End Class

End Namespace
```

Structs

KeyboardHookFlagsStruct

```
Namespace MisterHook.Structs

  'Flags da estrutura de Keyboard
  <Flags()>
  Public Enum KeyboardHookFlagsStruct As UInt32
    LLKHF_EXTENDED = &H1
    LLKHF_INJECTED = &H10
    LLKHF_ALTDOWN = &H20
    LLKHF_UP = &H80
  End Enum

End Namespace
```

KeyboardHookStruct

```
Imports System.Runtime.InteropServices

Namespace MisterHook.Structs

  'Estrutura do hook de keyboard
  <StructLayout(LayoutKind.Sequential)>
  Public Structure KeyboardHookStruct
    Public vkCode As UInt32 'Código da tecla pressionada
    Public scanCode As UInt32
    Public flags As KeyboardHookFlagsStruct 'Flags da tecla pressionada
    Public time As UInt32
```

```
    Public dwExtraInfo As UIntPtr
  End Structure

End Namespace
```

MouseHookPointStruct

```
Imports System.Runtime.InteropServices

Namespace MisterHook.Structs

  'Estrutura de ponto para uso na estrutura de hook de mouse
  <StructLayout(LayoutKind.Sequential)> Public Structure MouseHookPointStruct
    Public x As Integer
    Public y As Integer
  End Structure

End Namespace
```

MouseHookPointStruct

```
Imports System.Runtime.InteropServices

Namespace MisterHook.Structs

  'Estrutura do hook de mouse
  <StructLayout(LayoutKind.Sequential)> Public Structure MouseHookStruct
    Public pt As MouseHookPointStruct 'Ponto do mouse na tela
    Public hwnd As Integer 'Handle da Janela (É possível pegar o nome da janela usando o hwnd)
    Public wHitTestCode As Integer
    Public dwExtraInfo As Integer
  End Structure

End Namespace
```

FORMULÁRIO

```
Imports MisterHook.MisterHook

Public Class FrmGeral

  Private hinstance As IntPtr

  'Objetos de Gravação de Mouse e Keyboard com Eventos
  Private WithEvents KeyboardRecorder As Record.KeyboardRecorder
  Private WithEvents MouseRecorder As Record.MouseRecorder

  'Objeto de Playback usado em ambos os objetos de gravação
  Private PlayBackActions As Playback.PlaybackActions

  'Variáveis usadas somente para contagem de segundos no form
  Const _SecondsBeforeRecord As Integer = 2
  Private _CountDownBeforeRecord As Integer
```

```
    Const _MilisecondsBeforePlayback As Integer = 1500

    Private _PathToSave As String, _Filename As String

#Region "Eventos do Form"

    'Load do Form, apenas seta a quantidade de segundos para contagem de gravação
    Private Sub frmGeral_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        _CountDownBeforeRecord = _SecondsBeforeRecord

        'É necessário enviar o handle da instância do programa executado para os gravadores, se não enviar, a gravação não funciona.
        Dim hInstance As IntPtr = System.Runtime.InteropServices.Marshal.GetHINSTANCE(System.Reflection.Assembly.GetExecutingAssembly.GetModules()(0)).ToInt32

        'Se for haver playback, é necessário enviar o objeto para ambos os gravadores.
        PlayBackActions = New Playback.PlaybackActions(hInstance) 'Também é necessário enviar o handle do programa pro objeto de playback pois se no playback você quiser apertar a tecla 'Pause' para parar o playback, é possível, ou seja, existe um gravador de teclado também sendo executado no playback

    End Sub

    'Evento do botão de gravação. Altera os estados dos botões para não deixar clicar duas vezes e também inicia o Timer para gravação.
    Private Sub btnRecord_Click(sender As Object, e As EventArgs) Handles btnRecord.Click

        btnRecord.Enabled = False
        btnStop.Enabled = True
        btnPlayback.Enabled = True

        lblRecord.Text = String.Format("A sua gravação começará em {0} segundos.", _CountDownBeforeRecord)
        lblRecord.Visible = True

        tmrRecord.Enabled = True

    End Sub

    'Evento do botão de parar gravação. Altera os estados dos botões para não deixar clicar duas vezes e também pára a gravação.
    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles btnStop.Click

        btnRecord.Enabled = True
        btnStop.Enabled = False
        btnPlayback.Enabled = True

        tmrRecord.Enabled = False
        lblRecord.Visible = False

        StopRecording()

    End Sub

    'Evento do botão para realizar playback do que foi gravado. Altera os estados dos botões para não deixar clicar duas vezes, inicia playback e ao terminar reativa botões.
    Private Sub btnPlayback_Click(sender As Object, e As EventArgs) Handles btnPlayback.Click

        btnRecord.Enabled = False
        btnStop.Enabled = False
```

```
        BtnPlayback.Enabled = False

        PlayBack()

        BtnRecord.Enabled = True
        BtnStop.Enabled = False
        BtnPlayback.Enabled = True

    End Sub

    'Timer para início da gravação.
    Private Sub TmrRecord_Tick(sender As Object, e As EventArgs) Handles TmrRecord.Tick

        _CountDownBeforeRecord -= 1

        lblRecord.Text = String.Format("A sua gravação começará em {0} segundos.", _CountDownBeforeRecord)

        If _CountDownBeforeRecord = 0 Then
            TmrRecord.Enabled = False
            _CountDownBeforeRecord = _SecondsBeforeRecord
            lblRecord.Visible = False
            StartRecording()
        End If

    End Sub

#End Region

#Region "Eventos de Gravação e Playback"

    'Evento de início de Gravação.
    Public Sub StartRecording()

        Try

            PlayBackActions.EraseData()

            KeyboardRecorder = New Record.KeyboardRecorder(hInstance, PlayBackActions)
            KeyboardRecorder.StartRecording()
            MouseRecorder = New Record.MouseRecorder(hInstance, PlayBackActions)
            MouseRecorder.StartRecording()

        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try

    End Sub

    'Evento de Stop de gravação
    Public Sub StopRecording()

        Try

            If KeyboardRecorder IsNot Nothing Then
                KeyboardRecorder.StopRecording()
```

```
        End If

        If MouseRecorder IsNot Nothing Then
          MouseRecorder.StopRecording()
        End If

        If PlayBackActions IsNot Nothing Then
          If (Fbd1.ShowDialog() = DialogResult.OK) Then
            _PathToSave = Fbd1.SelectedPath
          End If
          _Filename = String.Format("MisterHook Recording {0}.txt", Now.ToString("yyyy-MM-dd hhmmss"))
          PlayBackActions.SaveRecordToFile(_PathToSave, _Filename)
        End If

      Catch ex As Exception
        MessageBox.Show(ex.Message)
      End Try

    End Sub

    'Evento de Playback do que foi gravado.
    Public Sub PlayBack()
      Try

        Dim PathAndFilename As String = String.Empty
        If MessageBox.Show("Deseja carregar arquivo para playback? Se não, irá rodar playback da última gravação.", "Escolha de tipo de
Playback", MessageBoxButtons.OKCancel) = DialogResult.OK Then

          Dim fd As OpenFileDialog = New OpenFileDialog()

          fd.Title = "Open File Dialog"
          fd.InitialDirectory = "C:\"
          fd.Filter = "All files (*.*)|*.*|All files (*.*)|*.*"
          fd.FilterIndex = 2
          fd.RestoreDirectory = True

          If fd.ShowDialog() = DialogResult.OK Then
            PathAndFilename = fd.FileName
          End If

        End If

        Threading.Thread.Sleep(_MilisecondsBeforePlayback)
        If PlayBackActions IsNot Nothing Then
          PlayBackActions.Playback(PathAndFilename)
        End If

      Catch ex As Exception
        MessageBox.Show(ex.Message)
      End Try

    End Sub

#End Region
```

```
#Region "Eventos da DLL"

    'Evento passado do objeto de gravação de Keyboard para o Form. Aqui está sendo usado para fazer o Break da gravação de forma que seja possível alterar a tecla responsável por isso.
    Private Sub KeyboardRecorder_KeyDown(Key As Keys) Handles KeyboardRecorder.KeyDown

        If Key = Keys.Pause Then 'Estou usando a tecla 'Pause' do teclado neste caso

            btnRecord.Enabled = True
            btnStop.Enabled = False
            btnPlayback.Enabled = True

            StopRecording()

        End If

        Debug.Print(String.Format("{0} - Tecla {1} Pressionada", Now, Key.ToString())) 'Também possível fazer Log da ação

    End Sub

    'Função comentada por questões de visualização mais clean no Debug
    'Private Sub KeyboardRecorder_KeyUp(Key As Keys) Handles KeyboardRecorder.KeyUp
    'Debug.Print(String.Format("{0} - Tecla {1} Solta", Now, Key.ToString()))
    'End Sub

    'Log da ação do usuário ao apertar o botão esquerdo do mouse
    Private Sub MouseRecorder_MouseLDown() Handles MouseRecorder.MouseLDown
        Debug.Print(String.Format("{0} - Botão Esquerdo do Mouse Pressionado", Now))
    End Sub

    'Log da ação do usuário ao soltar o botão esquerdo do mouse
    Private Sub MouseRecorder_MouseLUp() Handles MouseRecorder.MouseLUp
        Debug.Print(String.Format("{0} - Botão Esquerdo do Mouse Solto", Now))
    End Sub

    'Log da ação do usuário ao apertar o botão direito do mouse
    Private Sub MouseRecorder_MouseRDown() Handles MouseRecorder.MouseRDown
        Debug.Print(String.Format("{0} - Botão Direito do Mouse Pressionado", Now))
    End Sub

    'Log da ação do usuário ao soltar o botão direito do mouse
    Private Sub MouseRecorder_MouseRUp() Handles MouseRecorder.MouseRUp
        Debug.Print(String.Format("{0} - Botão Direito do Mouse Solto", Now))
    End Sub

#End Region

End Class
```