Avaliação de Ferramentas de Automação para Engenheiros de Testes

Alexsandro D. de Oliveira Dórea[1], Fernanda de S. Carvalho[1], Monique T. Santos[2], Manoel Carvalho Marques Neto[1], David Moises[1]

Faculdade Ruy Barbosa – (FRB)
Rio Vermelho, 422 – Salvador – BA – Brasil

{alexdod, fernandadsc, manoelnetom}@gmail.com,
davidmbs@uefs.br,moniquetsantos@hotmail.com

Resumo. A dificuldade na escolha de ferramentas de automação é um problema que acarreta um alto nível de complexidade e custos. Escolher e relacionar as ferramentas mais adequadas para determinados tipos de testes exige uma grande quantidade de tempo, experiência e conhecimento dos engenheiros de testes. Isso torna o seu trabalho mais difícil, já que não existe uma análise descritiva e comparativa das ferramentas disponíveis no mercado. A ausência de informações sobre essas ferramentas reduz a produtividade e aumenta as rotinas de testes. Este trabalho tem como objetivo avaliar ferramentas de automação para engenheiros de testes, auxiliando-os na escolha da melhor ou mais adequada ferramenta.

Abstract. *The difficulty to choose automation tools is a highly complex and costly problem. Choosing and presenting the most adequate tools for determined types of tests demand a great amount of time, experience and knowledge of test engineers. This work becomes more difficult, because there is no descriptive and comparative analysis of the available tools on the market. The absence of information on these tools reduces productivity and increases tests routines. This work's objective is to evaluate automation tools for test engineers, assisting them to choose the most adequate tool.*

1 INTRODUÇÃO

A constante busca em introduzir no mercado *softwares* com qualidade mostra que realizar testes é um processo importante, já que a descoberta de defeitos e falhas, de forma tardia, dificulta a manutenção do sistema. Dessa forma, o processo de testes deve ser considerado um estágio essencial na produção e desenvolvimento de um bom sistema. Para conseguir alcançar esse resultado, o *software* deve ser verificado, validado e testado, a fim de garantir uma melhor aceitação em um mercado competitivo (MOREIRA FILHO; RIOS, 2003).

Com isso, determinar a maneira como testar o *software*, bem como identificar os casos que requeiram automação deve ser uma preocupação da equipe responsável por realizar os testes, pois, a partir dessa análise, irão alcançar seu objetivo, que é eliminar esforços duplicados e o retrabalho (SOMMERVILLE, 2003). Dessa maneira, a automação de testes

surge como um processo facilitador do desenvolvimento de sistemas, pois minimiza o tempo de trabalho e de manutenção.

Diante disso, este trabalho tem como objetivo avaliar, de forma descritiva e comparativa, ferramentas de automação de testes, mostrando-se importante no processo de desenvolvimento e execução de testes de *software*.

O presente texto está organizado da seguinte forma: a segunda seção detalha os conceitos de processos, técnicas, estratégias e automação de testes, sendo apresentados como referencial teórico para o projeto proposto. A terceira seção apresenta as atividades essenciais para o desenvolvimento do projeto, definição dos critérios, elaboração, aplicação e avaliação das respostas do questionário, validação dos critérios, seleção das ferramentas de testes, escala de avaliação. Por fim, uma seção com a análise crítica dos resultados.

2 PROJETO DE TESTES

A busca por softwares de qualidade mostra que ter um projeto de teste como um processo independente agrega maior valor ao produto final, já que a atividade de teste será executada por uma equipe especializada apenas em testar sistemas, conseguindo sistemas com menos defeitos e apresentando melhor qualidade e baixo custo de manutenção. (RIOS)

2.1 Processo de Testes

O processo de testes é uma forma de garantir produção e atingir boa qualidade do *software*, uma vez que o seu objetivo é encontrar erros e tentar corrigi-los antes da sua entrega (McGREGOR, 2001). A maneira como o *software* será testado assegura um resultado final satisfatório, pois testar significa executar de maneira controlada um processo, com a finalidade de alcançar o comportamento desejado e atender às suas especificações (MOREIRA FILHO; RIOS, 2003).

Para alcançar esse objetivo, dois tipos de testes devem ser feitos: o de verificação e o de validação. A verificação é responsável por avaliar se o sistema está de acordo com a sua especificação, ou seja, verifica se o *software* está fazendo o que deveria fazer. O teste de validação, por sua vez, confirma se o sistema está sendo executado de forma apropriada (HUTCHESON, 2003). Dessa maneira, o processo de testes realiza e executa as principais atividades de testes e suas subetapas a fim de construir um *software* de confiança, funcional e que tenha um bom desempenho (MOREIRA FILHO; RIOS, 2003).

2.2 Técnicas de Testes

A necessidade de sistematizar os casos de testes, ao gerar o código-fonte de um programa, para que o maior número de erros seja encontrado é uma forma de projetar e alcançar metas (PRESSMAN, 2005). Funcionalidade e estrutura são dois critérios que devem ser observados e levados em conta no início do processo de evolução dos testes, pois essas características serão importantes para direcionar o rumo do projeto.

De forma convencional, o software poderá ser testado sob duas técnicas diferentes (PRESSMAN, 2005). A primeira refere-se ao paradigma funcional, também conhecido como teste caixa-preta, no qual os testes exercitam a interface do *software*. A segunda

analisa a estrutura lógica interna que será realizada sob o paradigma estrutural, conhecido como teste caixa-branca (PRESSMAN, 2005).

Portanto, é necessária a combinação dos resultados encontrados nos testes caixa-branca e caixa-preta para garantir que a interface e o funcionamento interno do *software* estejam corretos. (PRESSMAN, 2005). Por isso, o sistema deve ser testado sob os dois paradigmas, uma vez que um erro caixa-branca não implica em um erro caixa-preta.

2.3 Estratégias de Testes

Uma estratégia de testes é um processo que descreve como os casos de testes devem ser planejados e executados para a construção bem-sucedida de um *software* (PRESSMAN, 2005). Uma boa estratégia é responsável por verificar e validar o sistema, já que evita desperdício de tempo e dinheiro, esforço desnecessário e que alguns erros sejam camuflados sem ser descobertos (PRESSMAN, 2005).

A figura 1 mostra um processo de estratégia de testes. O teste de unidade marca o início, concentrando-se no código-fonte. O processo progride com o teste de integração, que foca o projeto. Em seguida, os requisitos são estabelecidos no teste de validação. Por fim, o sistema é testado como um todo, formando o teste de sistema (PRESSMAN, 2005).

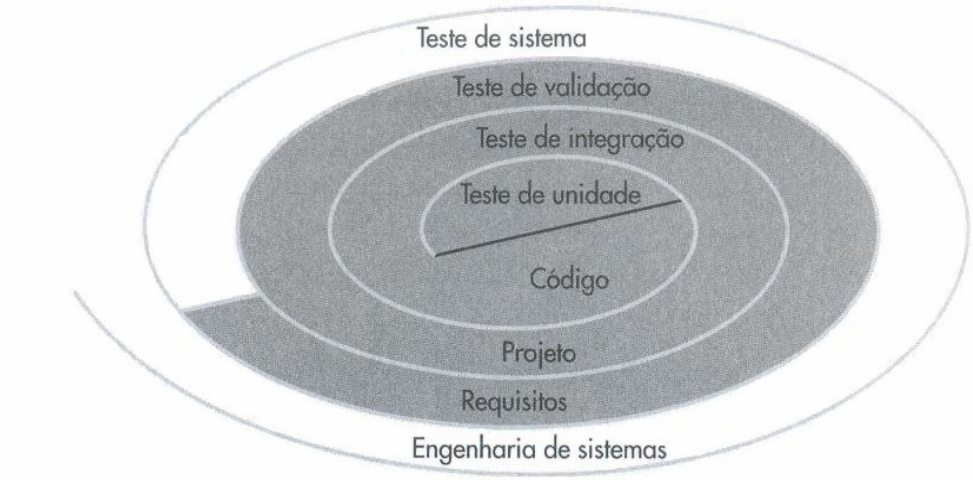

Figura 1. Estratégia de teste

2.4 Automação de Testes

Durante muito tempo, os dados e suas simulações eram gerados manualmente e poucas ferramentas estavam disponíveis no mercado. Esses problemas acarretaram o aumento de tempo e custo do processo de testes, reduzindo a qualidade dos *softwares*. Em virtude dessa situação, começaram a surgir novas ferramentas com a função de automatizar e dar suporte aos testes, bem como reduzir alguns dos problemas existentes (MOREIRA FILHO; RIOS, 2003).

A automação de testes é responsável por facilitar o processo de testes. Porém, antes de iniciar o projeto de automação, o plano de testes deve ser bem definido e estruturado. Para isso, algumas considerações devem ser avaliadas, como: definição de estratégias e técnicas

de testes, escopo, prazo de preparação, melhoria do processo de testes, seleção da ferramenta, metodologia, infra-estrutura e custo-benefício (MOREIRA FILHO; RIOS, 2003).

Um processo de automação é definido a partir do escopo do projeto. A equipe de testes e a empresa fazem um estudo dos casos de testes e avaliam a real necessidade de automatizá-los, levando em consideração o custo-benefício. Isso contribui para a melhoria do processo, uma vez que a finalidade é verificar se a automação de testes é adequada (MOREIRA FILHO; RIOS, 2003).

A seleção da ferramenta é outro fator importante, pois apóia o processo de automação de testes. Porém, essa é uma tarefa difícil, já que, antes de fazer a aquisição da ferramenta, uma série de decisões deve ser realizada por parte da equipe responsável, como fazer a avaliação das métricas utilizadas, selecionar e integrar a ferramenta com a metodologia de desenvolvimento de teste, planejar as atividades, instalar o *software* e, por fim, documentar todo o processo de seleção (MOREIRA FILHO; RIOS, 2003).

Portanto, automatizar consiste em planejar e analisar os casos de testes com a finalidade de facilitar e adequar o processo às necessidades da empresa.

3 ETAPAS DO PROJETO

Este projeto foi dividido em oito etapas distintas: definição dos critérios, elaboração, aplicação e avaliação das respostas do questionário, validação dos critérios, seleção, testes e validação das ferramentas, e conclusão.

3.1 Definição dos Critérios

Inicialmente, foram considerados os critérios específicos como, por exemplo: portabilidade, eficiência, *performance*, flexibilidade, entre outros. De acordo com Sebesta (2003), a portabilidade é responsável por facilitar a mudança dos programas de uma implementação para outra. Segundo Pressman (2005), a eficiência é um critério na qual os recursos computacionais devem ser avaliados de acordo com a realização funcional. O mesmo define a *performance* de uma determinada ferramenta a partir da velocidade do seu processamento e do seu tempo de resposta. Já a flexibilidade é representada pelo esforço utilizado para alterar um programa operacional. Esses conceitos demonstram a importância da utilização desses critérios para analisar as ferramentas automáticas de testes.

Com isso, foi criado um questionário com o propósito de determinar um padrão para análise de ferramentas automáticas. Dessa forma, esse processo mostrou-se importante para a avaliação das mesmas, transformando-se em um dos principais assuntos abordados neste projeto.

3.2 Elaboração, Aplicação e Avaliação das Respostas do Questionário

A atividade de levantamento dos critérios despertou um questionamento da real aplicabilidade dos mesmos no mercado. Por esse motivo, foi elaborado um questionário com a intenção de validar os critérios pré-definidos. O objetivo do questionário foi coletar e validar quais os critérios utilizados para a escolha da ferramenta de testes, a partir de um

responsável ou participante do projeto de teste de uma empresa. Outro ponto relevante focado na pesquisa foi saber o grau de maturidade das empresas quanto à utilização de ferramentas automáticas no processo de testes de *software*.

A aplicação do questionário contou com a participação de 13 empresas, entre elas fábricas de *softwares* e empresas desenvolvedoras de sistemas, de Salvador, sendo que, somente 7 dessas empresas realizam testes com ferramentas automáticas. A pesquisa foi dividida em 8 partes: ferramenta adotada, técnicas de teste, estratégias de teste, métricas, escopo do sistema, tecnologia, preço da ferramenta e critérios para escolha da ferramenta.

Os resultados obtidos mostram o comportamento das empresas diante do processo de desenvolvimento de testes. Assim, constatou-se que o *JUnit* é a ferramenta automática de teste mais adotada dentro das organizações. Com relação às técnicas de teste, verificou-se que as empresas, ao testar, sempre utilizam o teste caixa preta, ou a combinação desse com o teste caixa branca. Os testes de unidade e de sistema foram as duas estratégias mais utilizadas no momento de testar e validar os sistemas, o que comprova o resultado da ferramenta de automação mais utilizada pelas empresas, *JUnit*, uma ferramenta para testes de unidade.

Portanto, além das estratégias, técnicas e tecnologia utilizadas para a realização dos testes, o tamanho do sistema, o preço, a *performance* e eficiência também são alguns dos critérios que interessam e determinam a escolha de uma ferramenta de teste por uma empresa.

3.3 Validação dos Critérios

A etapa de validação começou com a análise das respostas do questionário, responsável por descobrir qual a interferência e importância dos critérios na escolha da ferramenta automática de testes. Em virtude da ausência de um padrão de critérios para análise das ferramentas, esta etapa foi essencial no processo de desenvolvimento deste projeto.

Em seguida, detectou-se a necessidade de avaliar as ferramentas a partir de critérios técnicos, como: *performance*, legibilidade, flexibilidade, portabilidade, eficiência e interoperabilidade, com a intenção de determinar um padrão para análise das ferramentas. Para concretizar esse objetivo, disponibilizou-se ainda, a possibilidade das empresas adicionarem algum outro critério que fosse utilizado por elas no momento de analisar e escolher a ferramenta automática de testes.

Ao final da análise das respostas do questionário, concluiu-se que as organizações procuram ferramentas que apresentem alto grau de *performance* e eficiência, visto que os dois critérios atingiram as maiores porcentagens, 31% e 26%, respectivamente. Além disso, foi possível observar também que 71% das empresas apontam o preço e 57% apontam o tamanho do sistema como fatores determinantes para escolha da ferramenta de teste.

3.4 Seleção das Ferramentas de Testes

Ao final da validação dos critérios foi inicializado o processo de seleção das ferramentas de testes. Essa etapa caracterizou-se por catalogar as ferramentas automáticas de testes, descrevendo-as com base nos critérios gerais e específicos. Conforme apresentado no

questionário, esse projeto limitou o universo do processo de seleção, focando apenas em ferramentas automáticas gratuitas que utilizam a tecnologia Java para aplicações *desktop*.

Tabela 1. Descrição das Ferramentas

Nome	Versão	Tipo de Teste
FINDBugs	1.2	Cobertura de código
Emma	2.1	Cobertura de código para aplicações e para teste de unidade
TestNG	5.5	Unidade
AgitarOne	4.0	Unidade
JUnit	4.3.1	Unidade
TeamCity	2.0	Integração
EasyAccept	1.0.2	Validação

A tabela 1 apresenta as ferramentas automáticas de teste, enfatizando critérios importantes, como: nome, versão, tipo de teste.

3.5 Escala de Avaliação

Nesta etapa será descrita a escala utilizada para tornar a avaliação das ferramentas mais consistente. A mesma foi elaborada com base nos critérios definidos no referencial teórico e na análise dos resultados do questionário aplicado.

Tabela 2. Escala de Avaliação

Critérios	Escala			
	Características	Ótima	Boa	Regular
Performance	• Completude com sucesso dos scripts de Teste sem falhas e dentro do tempo esperado • Requerido para múltiplos usuários • Requerido (por transação) para um usuário único	Atende as três características	Atende pelo menos duas características	Apenas uma
Legibilidade	• Estruturas que identifiquem o início e o final do código	Atende as três características	Atende pelo menos duas características	Apenas uma

	• Palavras reservadas • Facilidade na leitura do código			
Flexibilidade	• Código aberto • Permite extensões	Atende as duas características	Atende pelo menos uma característica	Não atende
Portabilidade	• Plataformas diferentes	Três ou mais	Pelo menos duas	Apenas uma
Interoperabilidade	• Padrão aberto a fim de oferecer uma comunicação transparente entre os sistemas.	Atende	_____	Não atende
Eficiência	• Atender os critérios de flexibilidade, performance, legibilidade e interoperabilidade.	Atende aos critérios descritos nas características	Atende à pelo menos dois critérios descritos nas características	Atende à apenas um critério descrito nas características

3.6 Avaliação das Ferramentas

Nesta seção foram avaliadas ferramentas para teste de unidade, integração, validação e sistema. Para cada tipo de teste foram utilizados casos de testes específicos, que mostram os testes para sucesso e falha, bem como os resultados esperados.

Para o teste de unidade foram utilizados dois casos de teste. O primeiro, denominado Pessoa Física, possuía os métodos *insert*, *update*, *delete*, *getAll*, *getPrimaryKey*, e apresentava como resultado esperado, um número inteiro 1 (um) para o teste de sucesso e o número inteiro 0 (zero) pra o teste de falha. No segundo caso de teste, denominado Operações, existia os métodos soma, subtração, divisão, multiplicação e cálculo do fatorial, e neste, os testes de sucesso esperavam como retorno um resultado correto, e os testes de falha aguardavam resultados incorretos.

Para o teste de integração não foi necessário o uso de casos de teste, já que a ferramenta avaliada para esse tipo de teste utiliza os *frameworks JUnit*, *NUnit* e *TestNG* para realização dos testes de unidade.

Para o teste de validação foi utilizado o caso de teste *MonipolyGame*, no qual foi necessário criar para execução dos testes, uma classe "*MonopolyFacade.java*" e um arquivo com vários *scripts* (escritos abrangendo situações de sucesso e falha) para testes das regras de negócio.

3.7 Análise dos Resultados

Esta seção descreve os resultados da avaliação das ferramentas de unidade, integração, validação e sistema, com base nos critérios validados e na escala de avaliação.

3.7.1 Teste de Unidade

As ferramentas analisadas para o teste de unidade foram: *JUnit*, *AgitarOne*, *TestNG*, *EMMA*, *FINDBugs*.

3.7.1.1 *JUnit*

É um *framework* que facilita a criação automática de testes de unidade com apresentação dos resultados, já que pode verificar se cada método de uma classe funciona da forma esperada, exibindo uma possível falha. É utilizado tanto para a execução de baterias de testes como para extensão (DIASPAR, 2002).

Apresentou ótima *performance* e ótima legibilidade, pois atendeu as características da escala de avaliação. Possui uma sintaxe própria, onde consegue determinar o início da execução da classe de teste através do método *setUp* e o final com o método *tearDown*, e exige que os métodos de teste possuam um padrão utilizando o prefixo *test*.

Tem ótima flexibilidade, já que seu código fonte é aberto e permite extensões, para melhorias e criações de novas funcionalidades. A portabilidade foi classificada como ótima por ser desenvolvida na linguagem Java, esta é interpretada, ou seja, o compilador gera um código independente de máquina chamado *bytecode*. Os *bytecodes* são a linguagem da máquina virtual Java – um programa que simula a operação de um computador e roda sua própria linguagem de máquina.

Tem ótima interoperabilidade e é bastante eficiente para seu domínio, pois atendeu a característica definida na escala de avaliação. Além disso, permite o acoplamento às ferramentas de desenvolvimento, como Eclipse.

3.7.1.2 *AgitarOne*

Esta ferramenta por ser uma extensão do *JUnit*, possui todas as características do mesmo, com uma nova funcionalidade, a geração automatizada das classes de teste. Para que os testes pudessem ser concluídos, foi utilizado o Caso de Teste Operações. Nessa situação, o *AgitarOne* foi muito eficiente, porque conseguiu criar a classe de teste contemplando todas as situações de sucesso e falha.

De acordo com a escala de avaliação descrita anteriormente, o *AgitarOne* apresentou ótima *performance*, portabilidade e interoperabilidade. A eficiência, legibilidade, flexibilidade foi classificada como boa.

3.7.1.3 *TestNG*

Esta ferramenta possui características semelhantes ao *JUnit*, como a interface usual, a maneira de testar o sucesso e a falha, a *performance* e o fato de ser acoplado às ferramentas de desenvolvimento. Porém, novas funcionalidades foram inseridas, tornando-a mais fácil de usar.

De acordo com a escala de avaliação descrita anteriormente, a *performance*, a portabilidade e a interoperabilidade foram avaliadas como ótima. Já a eficiência foi avaliada como boa.

Apresentou ótima flexibilidade, pois permitiu a criação de dependências entre os métodos de teste. Quando comparada com o *JUnit* mostrou uma maior flexibilidade, já que o *JUnit* realizou os testes de unidade de forma isolada, método a método, não permitindo a relação de dependência entre os mesmos

3.7.1.4 *EMMA*

É uma ferramenta Java de código aberto que analisa a cobertura de código tanto de aplicações como de testes de unidade. Para avaliar esta ferramenta foi utilizada a *EclEmma*, que é um *EMMA* para Eclipse.

Com base nos critérios específicos e na escala de avaliação descritos anteriormente, o *EclEmma* apresentou ótima *performance*, pois conseguiu analisar tanto o código da aplicação quanto os de testes de unidade em um ótimo tempo, aproximadamente 15 segundos para o término da análise para o caso de teste Pessoa Física e Operações.

Apresentou também boa legibilidade, pois exibiu como resultado da análise o percentual do código atingido, indicando as instruções que foram atingidas após a execução dos casos de teste. Tem ótima flexibilidade, portabilidade e interoperabilidade, e boa eficiência.

3.7.1.5 *FINDBugs*

É uma ferramenta de código aberto para a linguagem Java, cujo objetivo é encontrar erros no código da aplicação, a partir da inspeção dos *bytecodes*.

De acordo com a escala de avaliação, o *FINDBugs* apresentou boa *performance* e legibilidade. Tem ótima flexibilidade, portabilidade e interoperabilidade.

Possui boa eficiência, pois analisou a totalidade dos códigos, apesar de não testar a vulnerabilidade da aplicação.

3.7.1.6 Gráfico Comparativo

Nesta seção será apresentado o gráfico comparativo das ferramentas avaliadas de teste de unidade.

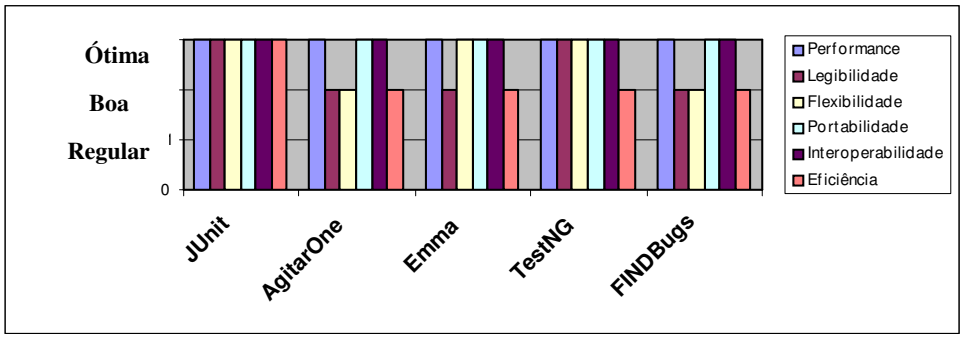

Figura 2 - Gráfico comparativo

Dessa forma, conclui-se que a ferramenta *JUnit* para teste de unidade foi a que atingiu os melhores resultados, confirmando a sua maior aceitação no ambiente de testes dentro das empresas.

3.7.2 Teste de Integração

Esta seção descreve os resultados da avaliação da ferramenta *TeamCity* com base nos critérios validados e na escala de avaliação.

3.7.2.1 *TeamCity*

É uma ferramenta que funciona de forma integrada com ferramentas de gerenciamento de configuração, como CVS (*Concurrent Version System*) e *Visual Source Safe*.

A principal finalidade é a execução de testes no momento que o desenvolvedor atualiza o objeto modificado no repositório, neste momento a ferramenta testa o objeto modificado de forma integrada com o restante da aplicação, na versão que se encontra no repositório. Caso possua erro no objeto modificado, ou em algum objeto dependente deste, a ferramenta exibe de forma imediata um relatório com todas as notificações.

Com base nos critérios específicos descritos anteriormente, esta ferramenta apresentou boa *performance* e legibilidade. Tem flexibilidade regular, pois seu código fonte não é aberto, não permitindo extensões. Possui portabilidade regular, já que para atender a todas plataformas foram desenvolvidas várias versões. Tem interoperabilidade e eficiência regulares.

3.7.3 Teste de Validação

Esta seção descreve os resultados da avaliação da ferramenta *EasyAccept* com base nos critérios validados e na escala de avaliação.

3.7.3.1 *EasyAccept*

É uma ferramenta que auxilia a criação, execução e automatização de testes de validação. Para execução dos testes foi necessário que os clientes desenvolvessem uma classe *Façade*, exposição lógica da aplicação para a realização dos testes, e criem os *scripts* de teste, que pode chamar a lógica de negócio e produzir uma saída, como pode ser observado na figura 3

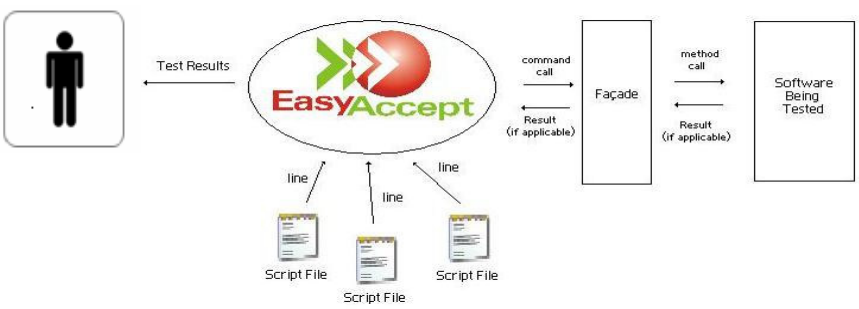

Figura 3 - Modelo de execução do *EasyAccept*

Para a avaliação da ferramenta foi utilizado o caso de teste *MonopolyGame.*

Com base nos critérios específicos descritos anteriormente, esta ferramenta apresentou boa *performance* e legibilidade, por utilizar a linguagem de script para sua execução.

Tem ótima flexibilidade, pois permitiu através do *Façade* a implementação de algumas validações, podendo até integrar com outras ferramentas de teste, como *JUnit*. Possui ótima portabilidade e interoperabilidade, e tem boa eficiência.

4 CONCLUSÃO

A constante busca por *softwares* de qualidade tem mostrado que o processo de testes é uma fase importante na produção e desenvolvimento de um sistema. Ao optar por realizar testes utilizando ferramentas automáticas, os engenheiros de *software* conseguem detectar falhas e defeitos, evitando o retrabalho e esforços duplicados.

Durante a elaboração deste trabalho, observou-se a ausência de uma análise criteriosa das ferramentas automáticas de testes. Essa foi uma das dificuldades encontradas para realizar o processo de avaliação, pois não existia um padrão de critérios, tarefa que foi suprida com a aplicação do questionário, que descobriu a realidade do ambiente de testes dentro das empresas.

Dessa forma, outro fator que pode ser avaliado com a análise das respostas obtidas com o questionário, é que as empresas não têm um padrão a seguir no momento da escolha de uma ferramenta automática. Sendo assim, fica evidente que essa é uma das dificuldades encontradas, visto que, o processo de automação de testes apesar de facilitar a execução dos testes, é um processo caro e de longo prazo. Portanto, a escolha de uma ferramenta que não atenda as necessidades da empresa, bem como os seus processos de testes, poderá aumentar os custos e reduzir a produtividade.

Com isso, a análise dos resultados obtidos com o questionário foi determinante para iniciar o planejamento do processo de avaliação das ferramentas. Esse processo definiu o escopo do projeto, limitando-o em ferramentas gratuitas que utilizam a tecnologia Java para aplicações *desktop*. Com o intuito de tornar a análise das ferramentas mais consistente, foi criada também uma escala de avaliação para padronizar o processo.

Neste projeto foram testadas ferramentas automáticas para os quatro tipos de testes: unidade, integração, validação e sistema. Sendo que, a maior quantidade de ferramentas analisadas foi de teste de unidade, já que há uma maior disponibilidade das mesmas no mercado, o que mostra também um dos motivos pelo qual as empresas automatizam mais os seus testes de unidade.

Como trabalho futuro aconselha-se a criação de um catálogo que disponibilize informações das ferramentas automáticas, com a intenção de facilitar e padronizar o processo de testes e a escolha das mesmas. Outra sugestão para um trabalho futuro seria o desenvolvimento de uma ferramenta de sistema gratuita que utilize a tecnologia Java para *Windows Form*. Uma última sugestão seria a criação de uma ferramenta gratuita para teste de unidade que consiga suprir as fragilidades encontradas nas ferramentas analisadas e comparadas.

REFERÊNCIAS BIBLIOGRÁFICAS

DIASPAR, Software Services; Junit: A Starter Guide, 2002. Capturado em 02 de outubro de 2006. On-line. Disponível na Internet http://www.diasparsoftware.com/articles/Junit/jUnitStarterGuid.html

GAO, Jerry. *Testing and Quality Assurance for Component-Based Software*. 1 ed. Boston: Artech House, 2003.

GRAHAM, Dorothy. *Software Test Automation*: effective use of test execution tools. 1. ed. Harlow: ACM Press, 1999.

HUTCHESON, Marnie L. *Software Testing Fundamentals-Methods and Metrics*. 1. ed. Indianapolis: Wiley Publishing Inc., 2003.

McGREGOR, John D. *A practical guide to testing object-oriented software*. 1. ed. United States of America: Addison Wesley, 2001.

MOREIRA FILHO, Trayahú R.; RIOS, Emerson. *Projeto & Engenharia de Software*: teste de software. 1. ed. Rio de Janeiro: Alta Books, 2003.

MYERS, Glenford J. *The Art of Software Testing*. 2. ed. Hoboken: John Wiley & Sons, Inc., 2004.

PFLEEGER, Shari L. *Engenharia de Software:* teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

PRESSMAM, Roger S. *Engenharia de Software*. 5. ed. São Paulo: McGraw-Hill, 2005.

RIOS, Gerência de Projeto de Testes Segundo o Modelo do PMI. Capturado em 10 de outubro de 2007. On-line. Disponível na Internet http://www.bfpug.com.br/islig-rio/Downloads/Gerencia_Projeto_Testes_PMI.pdf

ROCHA, André D.; SIMÃO, Adenilson S.; MALDONADO, José C.; MASIERO, Paulo C. Uma ferramenta baseada em aspectos para o teste funcional de programas Java In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, n.6, 2005, Uberlândia.

SEBESTA, Robert W. Conceitos de Linguagens de Programação. 5. ed. Porto Alegre: Bookman, 2003.

SOMMERVILLE, Ian. *Engenharia de Software*. 6. ed. São Paulo: Addison Wesley, 2003.

TIAN, Jeff. *Software Quality Engineering*: testing, quality assurance, and quantifiable improvement. 1. ed. Hoboken: John Wiley & Sons, Inc., 2005.