APLICABILIDADE DE TESTES DE SOFTWARE COM FERRAMENTA IDE

ARAÚJO, Aline Vilela de; FREIRE, Gabrielli Ferreira Figueiredo; ROCHA, Lucas de Tharso; GARCIA, Ranieri Alves(1); REIS, José Cláudio de Sousa(2).
(1) Acadêmicos do oitavo período do Curso de Ciência da Computação, UNIFENAS, Alfenas.
(2) Professor do Curso de Ciência da Computação, UNIFENAS, Alfenas.

RESUMO:

O teste de software surge de uma grande necessidade no ambiente de desenvolvimento de software - reduzir custos e evitar a disseminação de erros no processo de desenvolvimento. Este trabalho tem como objetivo aplicar os seguintes testes: Teste de unidade, teste de navegação, teste de desempenho, teste de fumaça e teste de estresse através do Visual Studio 2010. Foram utilizadas para o desenvolvimento do sistema as tecnologias: ASP.NET, C#, SQL Server 2005 e o Visual Studio 2010. Também foram utilizadas no desenvolvimento do sistema a Arquitetura Orientada a Objetos, Arquitetura em Camadas, Modelo Relacional de Banco de Dados, UML e UP. O trabalho foi realizado desenvolvendo-se um software para a realização dos testes e, posteriormente, foram realizados os testes neste software desenvolvido. Foram aplicados os testes e os resultados foram avaliados. Pode-se concluir que a aplicação dos testes de software permite que se avalie a aplicação a partir de dados concretos. Os testes de software fornecem diversas informações ao testador e à equipe de desenvolvimento. Com essas informações, correções e melhorias podem ser realizadas não só na aplicação, mas também no servidor de hospedagem.

PALAVRAS CHAVE: testes, software, visual studio

ABSTRACT:

Software testing comes a great need in the software development environment - to reduce costs and prevent the spread of errors in the development process. This paper aims to apply the following test: Unit test, navigation test, performance test, smoke test and stress test using Visual Studio 2010. It was used for system development these technologies: ASP.NET, C # , SQL Server 2005 and Visual Studio 2010. It also was used in the development system-oriented architecture objects, Layered Architecture, Relational Model Database, UML and UP. The work was carried out developing a software for the tests and were later performed the tests developed this software. The tests were applied and the results were evaluated. It can be concluded that the software testing of the application allows assessing the application based on concrete data. Software testing provides various information to the tester and the development team. With this information, corrections and improvements can be made not only in implementation but also in the hosting server.

KEYWORDS: tests, software, visual studio

1. INTRODUÇÃO

1.1 Contexto

O teste de software surge de uma grande necessidade no ambiente de desenvolvimento de software - reduzir custos e evitar a disseminação de erros no processo de desenvolvimento.

Investir e dedicar tempo ao projeto e execução de testes significa redução de gastos na construção de um software. Quanto mais cedo um erro na codificação do software é identificado, menor é o custo para sua correção.

Além da redução nos custos de desenvolvimento de software, ao evitar-se a propagação dos erros, contribui-se para a obtenção de um atributo muito procurado por quase todas as empresas que desenvolvem software: qualidade.

Um cliente não quer descobrir defeitos enquanto usa o produto final. Produto que muitas das vezes possuem um alto custo. E por outro lado, a empresa que desenvolveu o software, também não quer que o cliente encontre defeitos em seu produto final. Isto abala a imagem e a credibilidade da empresa.

Os motivos para se testar um software são muitos e o teste está se tornando indispensável para um desenvolvimento com qualidade. Surge então a necessidade de se testar o software.

1.2 Objetivo

Os objetivos específicos é aplicar os seguintes testes: Teste de unidade, teste de navegação, teste de desempenho, teste de fumaça e teste de estresse através do Visual Studio 2010.

1.3 Hipótese

O uso de ferramentas IDE para teste de software, aumenta a produtividade, padroniza e documenta os testes.

2. REFERENCIAL TEÓRICO

2.1 Qualidade de software

Segundo Garvin (1984), qualidade é um conceito complexo e multifacetado que pode ser descrito sob cinco pontos de vista diferentes: visão transcendental, visão do usuário, visão do fabricante, visão do produto e visão baseada em valor. A visão transcendental sugere que a qualidade é algo que se reconhece imediatamente, mas não consegue se definir. A visão do usuário visa a qualidade tendo como objetivo atender a um usuário final. A visão do fabricante vê a qualidade como o grau de atendimento às especificações do produto. A visão do produto está ligada às características do produto. E a visão baseada em valor vê a qualidade como um valor que o cliente estaria disposto a pagar por um produto.

Qualidade de software pode ser definida no sentido mais geral como uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que produzem e para aqueles que o utilizam (BESSIN, 2004).

Segundo Pressman (2011), uma gestão de qualidade efetiva define a infraestrutura que dá suporte a qualquer tentativa de construir um produto de software de alta qualidade. Um produto útil fornece as funções e os recursos que um usuário final deseja. E, um software de alta qualidade, gera benefícios tanto para a empresa de software quanto para os usuários finais.

2.2 Garantia de qualidade de software

A garantia de qualidade de software engloba uma série de atividades que podem ser resumidas em: padrões, revisões e auditorias, testes, coleta e análise de erros, gerenciamento de mudanças, educação, gerência de fornecedores, administração de segurança, proteção e administração de riscos (Horch, 2003).

Pressman (2011), diz que a garantia da qualidade de software é composta por uma série de tarefas associadas a dois elementos distintos: os engenheiros de software que realizam o trabalho técnico e um grupo de garantia

de qualidade de software que é responsável pelo planejamento, supervisão, manutenção de registros, análise e relatórios referentes à garantia de qualidade.

Um sistema de garantia de qualidade pode ser definido como a estrutura organizacional com responsabilidades, procedimentos, processos e recursos para implementação da gestão da qualidade. Os sistemas de garantia de qualidade são criados para garantir que as organizações satisfaçam às expectativas do cliente (ANSI/ASQ, 1987).

O princípio de Pareto diz que oitenta por cento dos defeitos podem ser associados a vinte por cento de todas as possíveis causas. Fato que leva a isolar os vinte por cento de causas vitais na estatística da garantia da qualidade (PRESSMAN, 2011).

As técnicas de estatísticas da garantia da qualidade para software demonstraram fornecer um aperfeiçoamento substancial da qualidade. Em alguns casos, as organizações atingiram uma redução de cinquenta por cento por ano nos defeitos após a aplicação destas técnicas (ARTHUR, 1997).

2.3 Teste de software

Testabilidade de software é a facilidade com que um programa de computador pode ser testado. Um software testável apresenta as seguintes características: operabilidade, observabilidade, controlabilidade, simplicidade, estabilidade e compreensibilidade.

Todo software em processo de criação possui uma etapa de fundamental importância em seu desenvolvimento, que define se o programa funcionará, de fato, como foi solicitado. Essa etapa é a de testes. O principal objetivo do teste é localizar o maior número de erros com a menor quantidade de esforço e tempo.

Existem conceitos importantes – defeito, erro e falha – relacionados aos testes de software convencionais que devem ser compreendidos e diferenciados de forma bem clara pelo desenvolvedor para que o objetivo principal em questão seja alcançado. Um defeito de software é um ato inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema

ou utilizar um método ou uma ferramenta. Um erro é uma manifestação concreta de um defeito, ou seja, um resultado inesperado na execução. Falhas são comportamentos inesperados do software pelo usuário, que podem ter sido oriundas de diversos erros. Da mesma forma, alguns erros podem nunca causar falha alguma.

Um conceito acarreta outro: defeitos causam erros e erros causam falhas. Se o defeito for encontrado e corrigido na fase de desenvolvimento do software – mais especificamente na fase de testes – a chance de haver erros e falhas no programa é quase nula.

2.4 Estratégia geral de teste de software

Segundo Pressman, (2011), a estratégia de teste de software fornece um roteiro que descreve os passos a serem executados como parte do teste, define quando esses passos são planejados e então executados, e quanto trabalho, tempo e recursos serão necessários. Portanto, qualquer estratégia de teste deve incorporar.

- Planejamento dos testes
- Projeto de casos de teste
- Execução dos testes
- Coleta e avaliação dos dados resultantes

3. MATERIAL E MÉTODOS

As seguintes tecnologias foram utilizadas para o desenvolvimento do sistema proposto neste projeto:

- *ASP. NET;*
- *C#;*
- *SQL Server 2005;*
- *VS 2010 (Visual Studio):* Ferramenta IDE (*Integrated Development Environment*) que possui suporte total à plataforma. NET e suporta várias linguagens de programação atualmente atualizadas no desenvolvimento de sistemas, possui ferramentas

para todo o ciclo de desenvolvimento, dando suporte para Arquitetos, gerentes, desenvolvedores e testadores do sistema;

Os seguintes métodos foram utilizados para o desenvolvimento do sistema proposto neste projeto.

- Arquitetura Orientada a Objetos;
- Arquitetura em Camadas;
- Modelo Relacional de Banco de Dados;
- UML;
- UP;

O trabalho será desenvolvido em quatro etapas. Na primeira etapa será realizado um levantamento bibliográfico para obter informações sobre os conceitos de qualidade e testes de software.

Na segunda etapa, será realizada a seleção da tecnologia a ser utilizada.

Na terceira etapa, será desenvolvido um software que será utilizado para a realização de testes.

Na quarta e última etapa, serão realizados os testes. Após o sistema ser testado serão avaliados os resultados.

- Teste de Unidade;
- Teste de Navegação;
- Teste de Desempenho;
- Teste de Fumaça;
- Teste de Estresse;

4 DESENVOLVIMENTO

4.1 Sistema Teste Vocacional

O sistema Teste Vocacional realiza testes vocacionais online. Esta aplicação web foi desenvolvida visando identificar o perfil psicológico dos vestibulandos através de algumas perguntas práticas e elaborar uma lista com as profissões mais adequadas a serem seguidas baseadas nas respostas do usuário.

4.2 Testes de Software com a ferramenta IDE Visual Studio 2010

O IDE Visual Studio 2010 é visto pela maioria de seus usuários como apenas uma ferramenta para desenvolvimento de aplicações, ou seja, apenas para escrita de códigos e compilação dos mesmos. Na verdade, o IDE em questão vai além. Podemos utilizar o Visual Studio 2010 para testar o que foi desenvolvido também. É uma ferramenta para testadores, arquitetos e, como a maioria já sabe, programadores. Por padrão, ela já dá suporte a TDD (*Test Driven Developement*) e possibilita a utilização de BDD (*Behavior Driven Develoment*) através do complemento *Specflow*.

O recurso TDD, possibilita o desenvolvimento de casos de teste antes mesmo da escrita dos códigos. Uma vez que se tem os casos de teste, pode-se desenvolver as funcionalidades apropriadas para que se siga com os passos seguintes. É um recurso que faz o caminho reverso do modelo de codificação padrão, em que o código é escrito primeiro e depois testado. Este método traz diversos benefícios como:

- A capacidade de combater pequenas falhas do sistema inicialmente, e evoluir o mesmo conforme os requisitos propõem
- Os testes passam a compor a documentação do sistema
- Os testes servem como um conjunto de testes de regressão, garantindo que as mudanças de código futuras não atrapalhem as funcionalidades existentes

O recurso BDD, mantém o fluxo padrão de desenvolvimento de sistemas. Codificação, testes, compilação, mais testes e disponibilização. Todavia, é dado

uma ênfase na especificação de comportamentos que são incompreensíveis para os usuários, somente entendem os envolvidos no desenvolvimento. Ele aborda questões como: "*O quanto devo especificar?"* E *"Como devo organizar e nomear minhas especificações para que elas sejam mais úteis?"*

Para fazer isso, o BDD diz que se deve elevar a mente para um nível de abstração comportamental além da implementação do código. Então, não se pensa em "*Constructor_Test*" ou mesmo "*Constructor_NullParam _ThrowsArgumentNullException*", mas em vez disso: "A operação não pode ser realizada sem um endereço de destinatário", por exemplo. A mudança de ênfase e terminologia nos leva a escrever especificações mais úteis.

O Visual Studio oferece uma gama de testes que são realizados de forma automática ou manual, a escolha do testador.

Abaixo, os tipos de teste oferecidos pela ferramenta e que foram abordados no projeto:

- Testes de Unidade (Manual e automatizado)
- Testes de Integração (Manual e automatizado)
- Testes de Interface Gráfica de Usuário (Manual e automatizado)
- Testes Funcionais (Manual e automatizado)
- Testes de Fumaça (Manual e automatizado)
- Testes de Desempenho (Somente para aplicações web)
- Testes de Carga (Somente para aplicações web)
- Testes de Stress (Somente para aplicações web)
- Testes de Recuperação (Somente para aplicações web)
- Spike Test (Somente para aplicações web)

4.3 Aplicação dos testes de unidade

Para a realização dos testes de unidade, foi criado um projeto de testes chamado de Testes. Neste Projeto foi criada a classe de testes UsuarioDALTest.cs que realiza os testes na classe UsuarioDAL.cs.

A classe UsuarioDAL.cs testada possui os seguintes métodos:

```
public UsuarioDAL() { }
public bool CadastrarU(String Nome, String Email, String Senha)
{ ... }
public bool AlterarU(int Codigo, String Nome, String Email,
String Senha){ ... }
public String ConsultarU(int Codigo) { ...}
public bool ExcluirU(int Codigo) { ... }
public List<String> PreencheDropU() { ... }
public bool AlterarSenhaUsuario(String Email, String Senha)
{ ...}
public bool ConsultarLoginUser(String user) { ...}
```

Foram criados na classe UsuarioDALTest.cs do projeto de testes os métodos de teste de unidade e todos os métodos da classe UsuarioDAL foram testados.

A Figura 1 mostra um exemplo de execução de um teste de unidade com seus respectivos resultados:

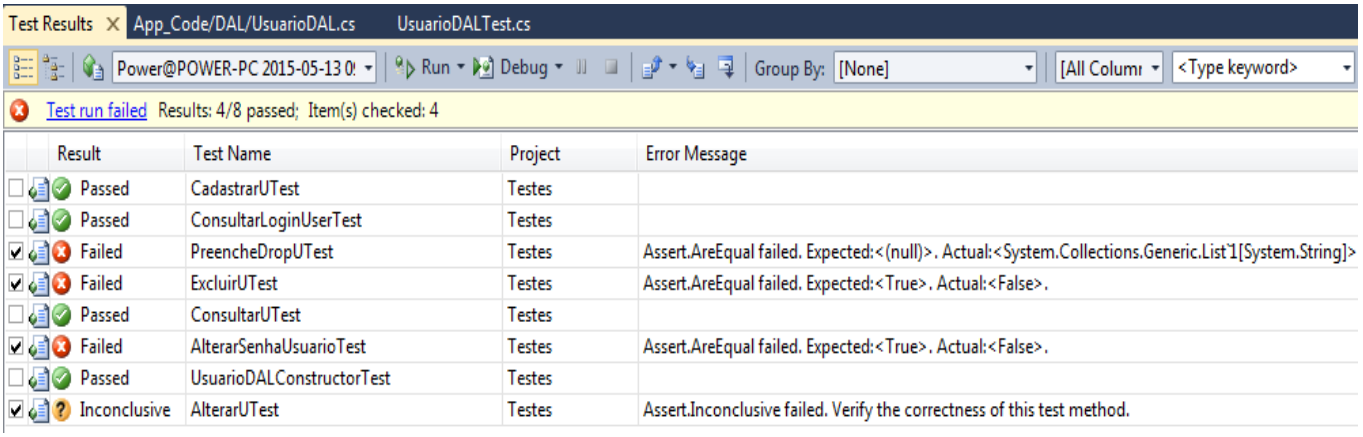


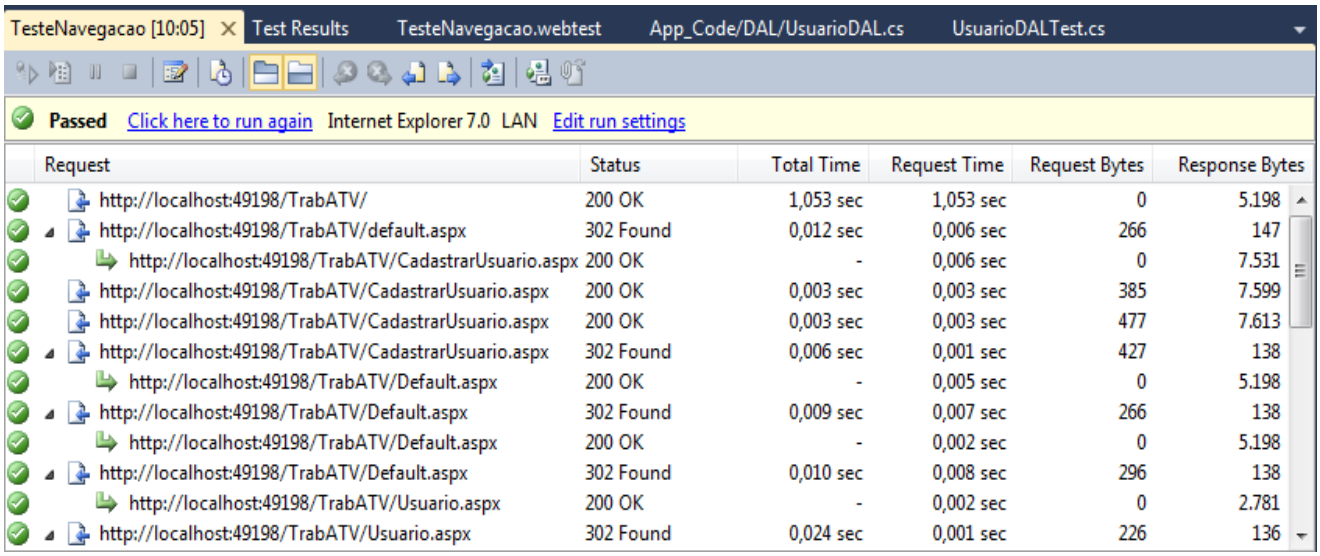
FIGURA 1 – Exemplo de resultado da execução dos testes de unidade

4.4 Aplicação dos testes de navegação e performance

Os testes de navegação e performance consistem em uma série de solicitações HTTP. O teste de navegação verifica se os links das páginas estão funcionando corretamente e se as páginas solicitadas são obtidas. O teste de performance verifica o tempo de resposta da aplicação.

Ao final do teste, é gerado um relatório fornecendo diversas informações ao testador. Informações estas que serão descritas neste item.

Na FIG. 2 está ilustrado o resultado do teste de navegação e performance na aplicação Teste Vocacional que obteve Passed como resultado e onde pode-se observar: a URL requisitada, o Status (se a página web foi obtida ou não), o Total Time (tempo total gasto para requisição e obtenção de uma resposta), o Request Time (tempo gasto na solicitação da página) o Request Bytes (número de Bytes contidos na requisição) e o Response Bytes (número de Bytes contidos na resposta.



Request	Status	Total Time	Request Time	Request Bytes	Response Bytes
http://localhost:49198/TrabATV/	200 OK	1,053 sec	1,053 sec	0	5.198
http://localhost:49198/TrabATV/default.aspx	302 Found	0,012 sec	0,006 sec	266	147
http://localhost:49198/TrabATV/CadastrarUsuario.aspx	200 OK	-	0,006 sec	0	7.531
http://localhost:49198/TrabATV/CadastrarUsuario.aspx	200 OK	0,003 sec	0,003 sec	385	7.599
http://localhost:49198/TrabATV/CadastrarUsuario.aspx	200 OK	0,003 sec	0,003 sec	477	7.613
http://localhost:49198/TrabATV/CadastrarUsuario.aspx	302 Found	0,006 sec	0,001 sec	427	138
http://localhost:49198/TrabATV/Default.aspx	200 OK	-	0,005 sec	0	5.198
http://localhost:49198/TrabATV/Default.aspx	302 Found	0,009 sec	0,007 sec	266	138
http://localhost:49198/TrabATV/Default.aspx	200 OK	-	0,002 sec	0	5.198
http://localhost:49198/TrabATV/Default.aspx	302 Found	0,010 sec	0,008 sec	296	138
http://localhost:49198/TrabATV/Usuario.aspx	200 OK	-	0,002 sec	0	2.781
http://localhost:49198/TrabATV/Usuario.aspx	302 Found	0,024 sec	0,001 sec	226	136

FIGURA 2 – Resultado dos testes de navegação e performance na aplicação Teste Vocacional

Pode-se observar dentre as várias informações contidas neste teste que a página default.aspx foi obtida com sucesso, teve um tempo total de requisição e resposta de 0,012 segundos, teve o tempo gasto com a requisição de 0,006 segundos, teve 266 Bytes na requisição da página e teve 147 Bytes como resposta.

4.5 Aplicação do teste Fumaça

O teste Fumaça testa o aplicativo sob carregamentos leves e curtas durações. Para realizar o teste fumaça na aplicação Teste Vocacional foram definidos dez usuários virtuais simulando o acesso a aplicação durante cinco minutos.

A execução do teste fumaça gera diversos gráficos e tabelas com os resultados do teste. O teste obteve Passed (passou) para todos os dez usuários virtuais como mostra a FIG. 3. Cada linha em azul representa a atividade de um usuário virtual.

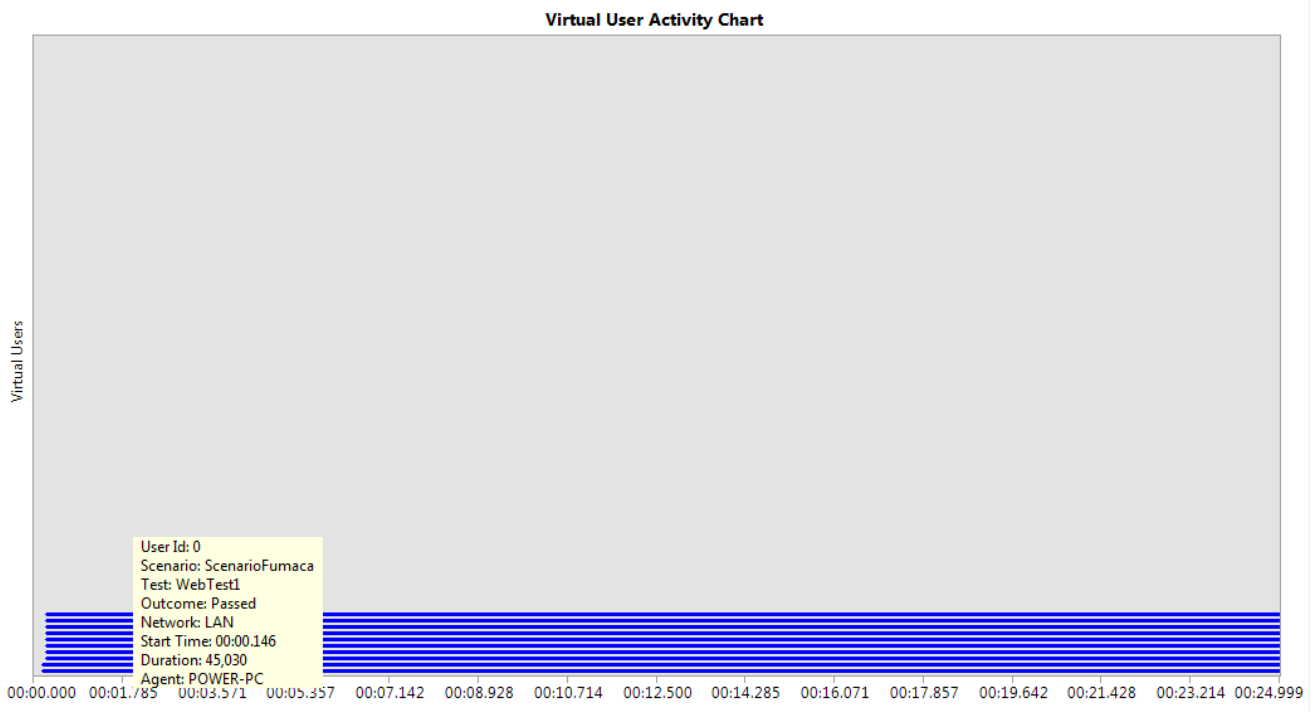


FIGURA 3 – Gráfico de atividade dos usuários virtuais

Os indicadores chave do teste fumaça podem ser verificados no gráfico da FIG. 4 e na tabela da FIG. 5.

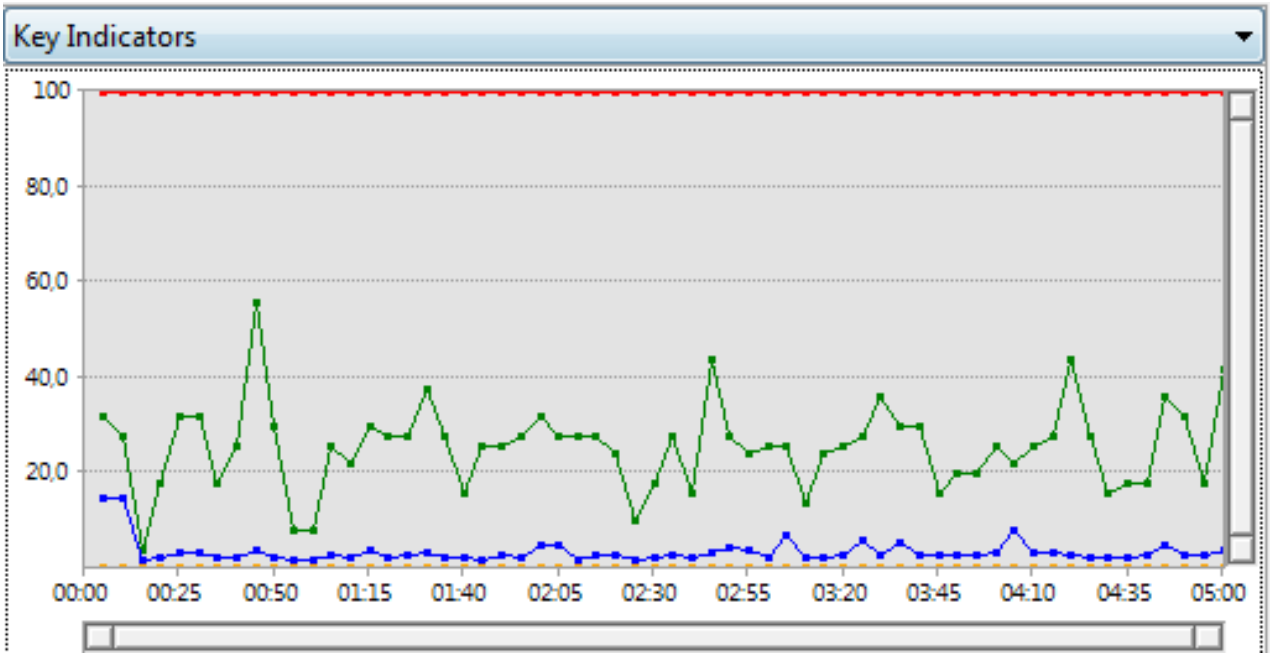


FIGURA 4 – Gráfico com os indicadores chave do teste fumaça

Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
User Load	_Total	LoadTest:Scenario	POWER-PC		10	10	10	10
Pages/Sec	_Total	LoadTest:Page	POWER-PC		10	0,40	5,60	2,58
Avg. Page Time	_Total	LoadTest:Page	POWER-PC		10	0,17	1,50	0,37
Errors/Sec	_Total	LoadTest:Errors	POWER-PC		0	0	0	0
Threshold Violations/Sec	_Total	LoadTest:Errors	POWER-PC		0	0	0	0

FIGURA 5 – Tabela com os indicadores chave do teste fumaça

O tempo de resposta das páginas pode ser verificado no gráfico da FIG. 6 e na tabela da FIG. 7.

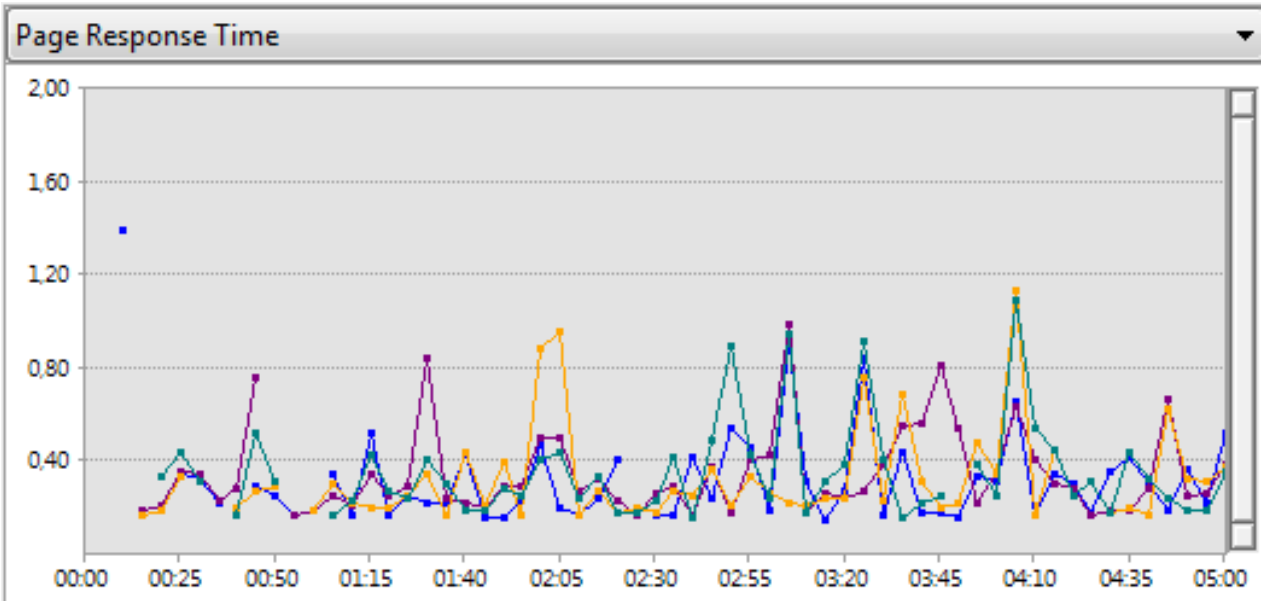


FIGURA 6 – Gráfico com o tempo de resposta das páginas

Page Response Time							
Avg. Page Time	TrabATV{GET}	LoadTest:Page	POWER-PC	-	-	-	-
Avg. Page Time	default-aspx{POST}	LoadTest:Page	POWER-PC	-	-	-	-
Avg. Page Time	CadastrarUsuario-aspx{POST}	LoadTest:Page	POWER-PC	2	0,15	1,40	0,40
Avg. Page Time	Default-aspx{POST}	LoadTest:Page	POWER-PC	2	0,17	0,99	0,35
Avg. Page Time	Usuario-aspx{POST}	LoadTest:Page	POWER-PC	2	0,17	1,14	0,34
Avg. Page Time	Teste-aspx{POST}	LoadTest:Page	POWER-PC	2	0,16	1,09	0,36

FIGURA 7 – Tabela com o tempo de resposta das páginas

Também é mostrado no teste fumaça, uma tabela com os resultados globais do teste ilustrada na FIG.8.

Overall Results

Max User Load	10
Tests/Sec	0,22
Tests Failed	0
Avg. Test Time (sec)	42,8
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	2,58
Avg. Page Time (sec)	0,37
Requests/Sec	13,1
Requests Failed	0
Requests Cached Percentage	38,1
Avg. Response Time (sec)	0,087
Avg. Content Length (bytes)	1.683

FIGURA 8 - Tabela com os resultados globais do teste fumaça

4.6 Aplicação do teste de estresse

O teste de estresse testa a aplicação por um período maior de duração e com acesso de um maior número de usuários. Para realização do teste de estresse no sistema Teste Vocacional foram definidos duzentos usuários virtuais por um período de sessenta minutos.

Os resultados do teste de estresse dependem não somente da qualidade da aplicação testada mas também do hardware da máquina onde está hospedada a aplicação. O hardware utilizado para realização do teste possuía 4GB de memória RAM e um processador dual core.

Os resultados do teste serão mostrados e discutidos. Apesar do tempo de resposta das páginas ser bem maior no teste de estresse se comparado ao tempo de resposta das páginas do teste fumaça, o gráfico de atividades dos usuários virtuais obteve Passed (passou) para todos os duzentos usuários virtuais como pode ser observado na FIG.9.

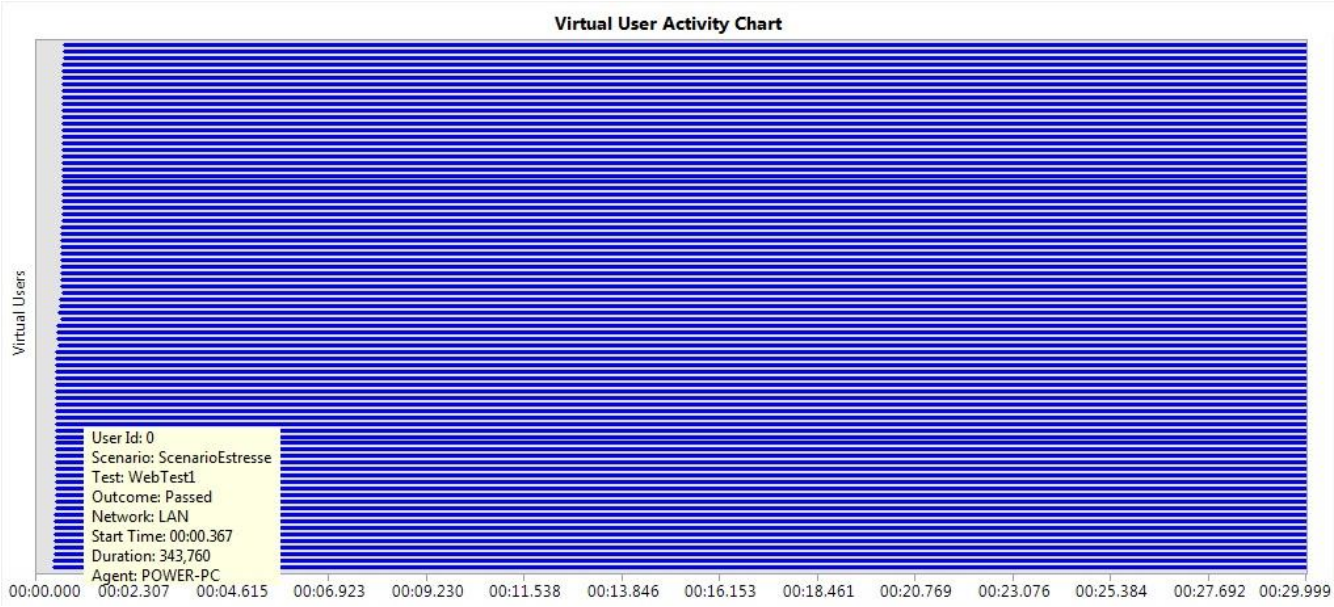


FIGURA 9 - Gráfico de atividade dos usuários virtuais no teste de estresse

Os indicadores chave do teste de estresse podem ser observados no gráfico da FIG. 10 e na tabela da FIG. 11.

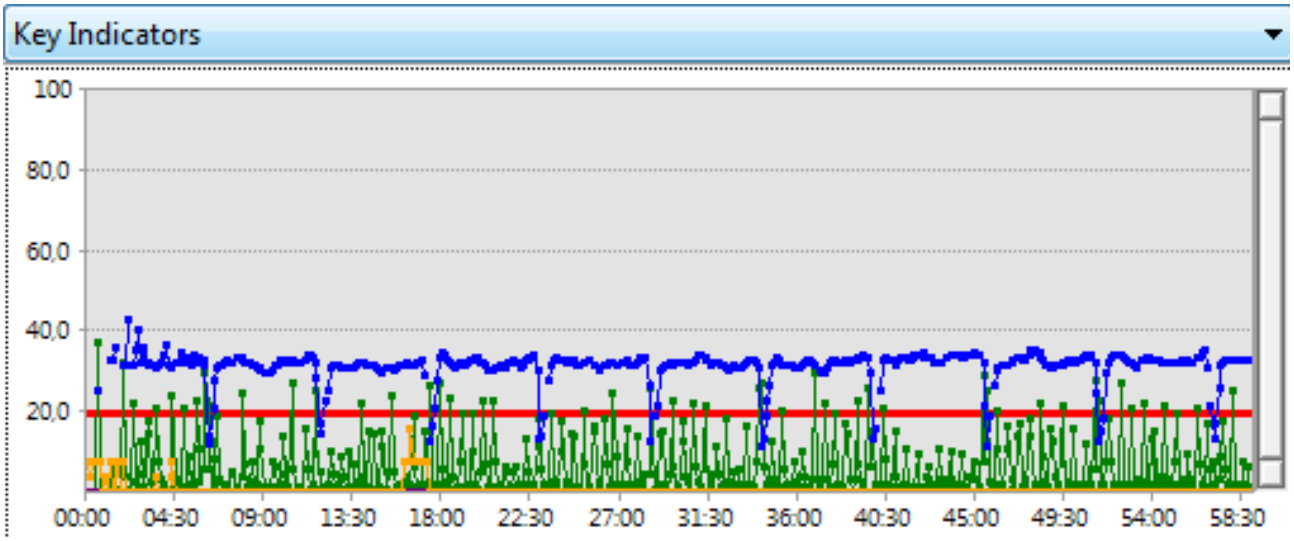


FIGURA 10 – Gráfico com os indicadores chave do teste de estresse

Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
User Load	_Total	LoadTest:Scenar...	POWER-PC		1.000	200	200	200
Pages/Sec	_Total	LoadTest:Page	POWER-PC		100	0	37,8	6,44
Avg. Page Time	_Total	LoadTest:Page	POWER-PC		100	11,9	43,1	30,7
Errors/Sec	_Total	LoadTest:Errors	POWER-PC		0	0	0	0
Threshold Violations/Sec	_Total	LoadTest:Errors	POWER-PC		10	0	1,60	0,040

FIGURA 11 – Tabela com os indicadores chave do teste de estresse

O tempo de resposta das páginas pode ser verificado no gráfico da FIG. 12 e na tabela da FIG. 13.

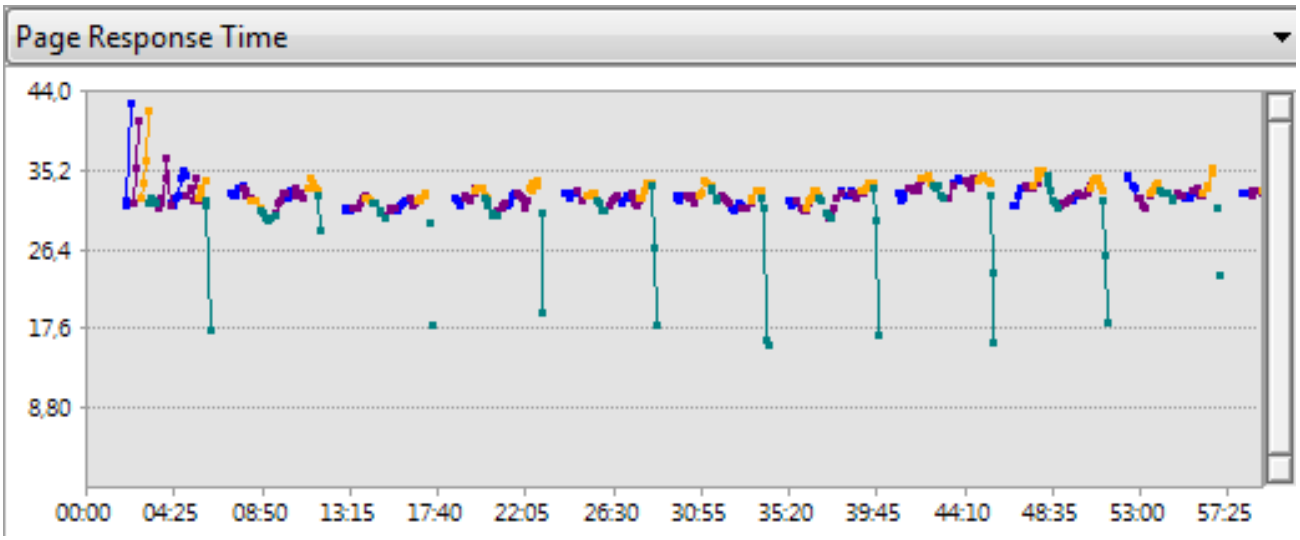


FIGURA 12 - Gráfico com o tempo de resposta das páginas

Page Response Time								
Avg. Page Time	TrabATV{GET}	LoadTest:Page	POWER-PC	-	-	-	-	-
Avg. Page Time	default-aspx{POST}	LoadTest:Page	POWER-PC	-	-	-	-	-
Avg. Page Time	CadastrarUsuario-aspx{POST}	LoadTest:Page	POWER-PC	44	30,8	43,1	32,5	
Avg. Page Time	Default-aspx{POST}	LoadTest:Page	POWER-PC	44	30,2	41,0	32,4	
Avg. Page Time	Usuario-aspx{POST}	LoadTest:Page	POWER-PC	44	31,1	42,2	33,3	
Avg. Page Time	Teste-aspx{POST}	LoadTest:Page	POWER-PC	44	15,8	34,8	31,5	

FIGURA 13 - Tabela com o tempo de resposta das páginas

A tabela com os resultados globais do teste de estresse é mostrada na FIG. 14.

Overall Results	
Max User Load	200
Tests/Sec	0,56
Tests Failed	0
Avg. Test Time (sec)	339
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	6,44
Avg. Page Time (sec)	30,7
Requests/Sec	32,9
Requests Failed	0
Requests Cached Percentage	38,2
Avg. Response Time (sec)	7,62
Avg. Content Length (bytes)	1.453

FIGURA 14 - Tabela com os resultados globais do teste de estresse

5. RESULTADOS E DISCUSSÃO

Pode-se observar na aplicação dos testes de software no sistema Teste Vocacional resultados específicos para cada teste. A aplicação do teste de unidade permitiu que se testassem os métodos da classe UsuarioDAL.cs verificando-se as saídas com diversos valores de entrada.

Os testes de navegação permitiram verificar se os links das páginas estavam funcionando adequadamente. De acordo com os testes, a aplicação Teste Vocacional obteve êxito no teste de navegação.

Com os testes de performance pode-se observar o tempo de resposta das páginas, o tempo gasto na requisição das páginas, número de bytes enviados e recebidos, entre outras informações.

O teste Fumaça testou o aplicativo sob carregamentos leves e curtas durações. Foi observado o comportamento de dez usuários virtuais durante cinco minutos. Os dez usuários virtuais obtiveram Passed (passou) no teste executado.

O teste de estresse testou a aplicação Teste Vocacional por um período de tempo maior e com um maior número de usuários acessando a aplicação. Apesar do fato dos duzentos usuários virtuais obterem Passed (passou) no teste executado, o tempo de resposta das páginas obtido não foi aceitável para uma aplicação web. Isto indica que para que a aplicação Teste Vocacional suporte um maior número de usuários, o hardware do servidor de hospedagem deve ser melhorado.

6. CONCLUSÃO

Pode-se concluir que a aplicação dos testes de software permite que se avalie a aplicação a partir de dados concretos.

Os testes de software fornecem diversas informações ao testador e à equipe de desenvolvimento. Com essas informações, correções e melhorias podem ser realizadas não só na aplicação, mas também no servidor de hospedagem.

REFERÊNCIAS

ANSI/ASQ. **Quality Systems Terminology.** 1987.

ARTHUR, L,J. **Quantum improvements in software system quality.** CACM, vol. 40, no.6 June 1997, p. 47-52.

BESSIN, J. **The Business Value of Quality.** IBM developerWorks, June 15, 2004, <www-128.ibm.com/developerworks/rational/library/4995.html>

GARVIN,D. **Whats Does "Product Quality Really Mean".** Sloan Management Review, Fall 1984, p. 24-45.

GARVIN, D. **Competing on the Eight dimensions of quality.** Harvard Business Review, November 1987, p. 101-109

HORCH, J. **Pratical Guide to Software quality management.** 2 ed., Artech House, 2003.

PRESSMAN, R.S. **Engenharia de software – uma abordagem profissional.** 7 Ed. AMGH Editora LTDA, 2011.