

Introdução a Testes de Software



Melissa Barbosa Pontes

melissa.pontes@cesar.org.br

Aluna de Mestrado Profissional em Engenharia de Software do Cesar.EDU, onde realiza pesquisas sobre predição de defeitos em software. Certificada em testes de software pelo ISTBQ (International Software Testing Qualification Board), atualmente trabalha como engenheira de testes no CESAR (Centro de Estudos e Sistemas Avançados do Recife), onde desempenha atividades de definição de processos, liderança e consultoria. Possui artigos e trabalhos publicados em eventos internacionais. Participa da organização de EBTS - Encontro Brasileiro de Testes de Software, que já se encontra na quarta edição.

A qualidade de um software pode estar fortemente relacionada à existência de defeitos inseridos durante o desenvolvimento ou manutenção de um produto. Uma das maneiras de identificar os defeitos de uma aplicação de forma que eles possam ser corrigidos é através das atividades de teste de software. Devido à sua importância e complexidade, teste de software pode ser responsável por uma parcela considerável dos custos de um projeto, por isso, este assunto merece atenção por parte das organizações que desejam sucesso com a condução das atividades do processo de testes.

O que é Teste de Software

Falar de testes de software é muito mais do que falar de execução de testes. Testar um software e relatar impressões e não conformidades é fornecer um diagnóstico do estado da aplicação, e é muito importante que este diagnóstico seja o mais completo e preciso possível, porque provavelmente ele vai servir de base para tomadas de decisões em relação ao projeto que está sendo analisado.

Neste artigo veremos

Neste artigo é abordado o assunto testes de software sem o relacionar diretamente com o processo de testes ou ferramentas de testes. Daremos ênfase ao que o testador faz no seu dia-a-dia, que dificuldades ele pode encontrar e de que maneira ele pode vencer essas dificuldades. Além disso, mostraremos como o desenvolvimento de habilidades como comunicação, pensamento crítico, pro atividade e criatividade pode ajudar o testador a desempenhar melhor suas atividades.

Qual a finalidade

Testes de software são atividades que contribuem com a qualidade do software a ser testado. Um bom planejamento dessas atividades pode significar economia para um projeto, visto que a identificação de defeitos no início do ciclo de desenvolvimento do produto pode reduzir os custos da sua correção, além de sua confiabilidade.

Quais situações utilizam esses recursos?

Além do conhecimento técnico necessário para a realização das atividades de testes de software, o testador pode buscar o desenvolvimento de habilidades pessoais para desempenhar melhor seu trabalho. Buscar aprimoramento além das dimensões técnicas ajuda a tornar um engenheiro de testes um profissional diferenciado na área, e faz com que algumas dificuldades que podem surgir no seu dia-a-dia sejam superadas com maior facilidade.

Testar uma aplicação é questioná-la, através de casos de teste e principalmente de observações, para analisar as respostas obtidas, pois estas podem revelar defeitos. Podemos entender por defeito tudo o que ameaça a qualidade do produto, levando em consideração que qualidade é o que o cliente quer. E o que o cliente quer é relativo, porque vai depender da finalidade da aplicação a ser desenvolvida. Por exemplo, uma aplicação pode estar em conformidade com a documentação de requisitos e ainda assim o cliente achar que ela não tem qualidade, pois apresenta demora em realizar determinada operação. Dependendo de onde a aplicação será utilizada, restrições de tempo podem ser cruciais.

O que não é Teste de Software

Para um melhor entendimento do que é teste, podemos analisar o que não é ou pelo menos não deveria ser teste de software.

Teste de software não é:

- Um processo burocrático: Apesar de ter um processo bem definido para a realização das atividades de testes de software, com atividades e papéis bem explicados, este processo não deve ser burocrático, nem inibir a criatividade do testador, que deve sentir a vontade para explorar a aplicação em busca de defeitos, mesmo que exista um roteiro de testes a seguir. Isso quer dizer que o testador não precisa deter-se a documentações, pois nem sempre estas estão atualizadas e livres de erros. O testador pode seguir sua intuição, desde que utilizando experiência e bom senso. Neste caso, a documentação do projeto pode servir como guia, mas não deve ser considerada uma verdade absoluta e inquestionável;
- Atividade limitada a seguir um roteiro: Se os testadores se limitam a fazer somente o que pede o roteiro com os casos de teste, provavelmente deixarão de perceber situações estranhas merecedoras de investigação na aplicação. A preocupação em ler o passo a passo do roteiro que é designado ao testador pode tirar seu poder de pensamento crítico em relação ao que está sendo avaliado. O testador no exercício da execução de testes deve ter plena liberdade de experimentar situações novas, mesmo que estas lhe venham na cabeça naquele momento de sua atividade. Ele deve também ter liberdade de discutir situações duvidosas que possa encontrar, mesmo que estas venham de uma observação feita por ele, que não estava previamente planejada pela pessoa que escreveu os casos de teste;
- Uma atividade que pode ser bem desempenhada por qualquer pessoa: Com isso, quer-se dizer que as atividades de testes não serão bem desempenhadas se quem as tiver realizando não seja um profissional preparado para tal. Para se testar eficientemente uma aplicação, é preciso possuir algumas habilidades, um conjunto de conhecimentos específicos de técnicas e alguma experiência no assunto;
- Uma atividade de garantia de qualidade: Não devemos confundir informar sobre a qualidade do produto com garantir a qualidade do produto gerado. As atividades de teste podem contribuir para a qualidade do produto final, mas a garantia dessa qualidade é de responsabilidade de toda a equipe do projeto. Dizer que testes são atividades de garantia de qualidade pode dar a impressão de que um software que não tem qualidade é um produto mal testado;

- Uma atividade que pode ser deixada para o final do projeto: Quando se trata testes de software como apenas uma fase do ciclo de desenvolvimento de um projeto, corre-se o risco de ter o tempo dedicado a essas atividades reduzido, seja por atrasos no cronograma de desenvolvimento ou alocação tardia de testadores no projeto;

- Uma atividade destrutiva: O livro *Lessons Learned in Software Testing* traz uma visão interessante sobre testes como sendo uma atividade tão construtiva quanto o desenvolvimento do software que será testado. No livro, os autores afirmam que teste não quebra um software, mas nos tira a ilusão de que ele funciona corretamente.

A Importância de Testes de Software

Quanto mais cedo um defeito for encontrado no ciclo de vida de um software, mais barato é o custo da sua correção. De acordo com Myers, autor do livro *The Art of Software Testing*, corrigir um defeito que se propagou até o ambiente de produção pode chegar a ser cem vezes mais caro do que corrigir este mesmo defeito em fases iniciais do projeto. Muitos dos defeitos que são encontrados no código da aplicação são originados na fase de levantamento de requisitos. Alguns estudos publicados no *Chaos Report*, um relatório elaborado pelo grupo de pesquisa Standish Group, em 2004, revelam que aproximadamente 45% dos recursos de um projeto de software nos Estados Unidos e Europa chegam a ser dedicados a atividades de testes e simulações. Assim, testes de software devem ser encarados como investimento em qualidade, e até mesmo economia, já que é uma atividade que identifica defeitos o quanto antes em um projeto.

Então, como podemos testar um software?

Existe mais de uma maneira de se conduzir as atividades de testes de software em um projeto. Bret Pettichord, co-fundador da Watir-Craft (<http://www.watircraft.com/>), uma empresa de consultoria em automação de testes, separa testes em escolas, de acordo com algumas características, tais como afinidade intelectual, objetivos definidos, hierarquia de valores, vocabulário comum, dentre outras. Essas escolas, através de suas diferentes visões, fornecem cinco maneiras distintas de entender e realizar testes de software. As escolas são: Escola Analítica, Padronizada, de Qualidade, Dirigida a Contexto e Escola Ágil. Segundo Pettichord, a Escola Analítica deu origem à Escola Padronizada, e esta, originou as demais [Fonte: *Schools of Software Testing*, Bret Pettichord].

A **Escola Analítica** vê teste como uma atividade altamente técnica e rigorosa, uma visão comum no meio acadêmico. Os casos de teste nessa escola possuem somente uma resposta certa. Um alto nível de documentação é requerido e os testadores devem verificar se o software está de acordo com esta documentação.

A **Escola Padronizada** vê teste como uma maneira de medir progresso com ênfase em custo, se utilizando de padrões e repetições. O processo de testes deve ser previsível, planejado e reproduzível. As certificações de testes existentes hoje no mercado seguem nessa linha.

A **Escola de Qualidade** enfatiza o processo e requer disciplina.

Algumas vezes se posiciona como um gargalo, pois em alguns casos, os testadores têm que proteger o usuário de um software ruim, e os engenheiros de qualidade que decidem se o software pode ser liberado.

A **Escola Dirigida a Contexto** tem o foco nas pessoas. Os testadores se colocam no lugar do cliente ou usuário na hora de procurar por defeitos. Admite que teste é uma atividade que requer experiência, habilidades, aprendizado rápido, atividade mental e multidisciplinaridade. Preocupa-se em focar no que agrega valor no momento. Os testes geralmente são projetados no mesmo momento da sua execução.

A **Escola Ágil** enfatiza automação de testes. Vê testes como um complemento do desenvolvimento, uma atividade que indica que o desenvolvimento está finalizado. Geralmente usa desenvolvimento dirigido a testes.

Na prática, visando o mercado, a escola dirigida a contexto traz muitos benefícios, pois seus adeptos procuram desenvolver diversas habilidades que lhes permite superar algumas dificuldades nos projetos, como por exemplo, documentação ruim ou inexistente, e até mesmo testar sob pressões de prazo, com custo direcionado a testes reduzido. Adeptos dessa escola assumem que a aplicação deve ser explorada além das fronteiras da documentação existente, e fazem isso testando além dos roteiros, quando estes existem. Daqui em diante, seguiremos a linha de raciocínio dos adeptos da Escola Dirigida a Contexto, e vamos então, entender porque testar bem uma aplicação requer experiência, habilidades, aprendizado rápido, atividade mental e multidisciplinaridade.

Como Testar sem o Documento de Requisitos?

Não é uma situação rara a equipe de projeto de software receber uma documentação de requisitos confusa, cheia de ambigüidades, e com situações que não são testáveis. Muitas vezes uma documentação pobre distrai testadores inexperientes, que podem até utilizar requisitos incorretos como fonte para casos de teste. Dependendo da maneira como a documentação está escrita, pode ser que o testador perca o foco do que a aplicação de fato deve fazer, e preste atenção em detalhes que não são os mais importantes do ponto de vista do cliente. Sem falar que uma documentação pode dar a idéia de cem por cento de cobertura dos testes em relação aos requisitos, quando podem existir muitas funcionalidades que ainda não foram testadas na aplicação. Pode também dar margem para que o testador derive casos de teste incompletos, já que o seu foco está voltado para cobrir somente a documentação. Vários problemas podem ocorrer quando temos casos de testes incompletos. Estes testes podem passar, mesmo existindo defeitos no software.

O que fazer então se o testador estiver numa situação parecida com essa? Testar! Como? Utilizar-se de heurísticas é uma boa saída. De acordo com Michael Bolton, consultor da empresa Developsense (www.developsense.com), heurística é um método falível de se resolver um problema. Sendo falível, não é perigoso utilizar esse método para testes? Absolutamente! Ainda segundo Michael, heurística quase sempre funciona,

e somente às vezes falha. A utilização de heurísticas esta diretamente relacionada com uma abordagem exploratória, uma maneira de ver a aplicação com a visão do usuário. O testador então assume certas condições como verdade, e parte delas para analisar a aplicação. Qualquer não conformidade com o que era esperado, não deve ser imediatamente considerada um problema. Pelo fato de se estar usando um método falível, essa situação deve ser mais bem analisada, muitas vezes discutida com o restante da equipe, para somente então tomarmos uma atitude de registrar um defeito. Uma maneira de entender bem como se utilizar de heurísticas é utilizar um exemplo do nosso dia-a-dia. Um bom exemplo é: Em épocas de fim de ano, muitas lojas de Shopping Centers contratam trabalhadores temporários. Existe a possibilidade de esse fato ocorrer todo ano? Sim, existe! Mas não dá para ter certeza de que isso sempre irá ocorrer. Porém, a chance de que ocorra pode direcionar nossas ações. Podemos tomar a decisão de elaborar bem nosso currículo, para que fique atrativo para os empregadores. Para isso, podemos investir em algum treinamento, com a intenção de mostrar que estamos capacitados para tais vagas. Assim, essa heurística termina nos servindo de guia, assim como nas atividades de teste.

Na prática então, a utilização de heurísticas nas atividades de testes de software pode dar ao testador noções de como a aplicação deveria se comportar sob determinadas circunstâncias. Através de seu uso, o testador vai analisando a aplicação e construindo um modelo mental. Por exemplo, ele pode começar a testar o produto em relação à consistência com suas próprias funcionalidades. Qualquer fluxo que esteja fora do padrão do resto da aplicação é passível de investigação. Também pode analisar se a aplicação está consistente com seu padrão visual, com o que ela se propõe a fazer, com a maneira que um usuário comum esperaria utilizá-la, e muitas outras conformidades podem ser testadas, sem a necessidade de uma vasta documentação que explique cada item acima. Testar dessa maneira pode fazer com que um testador envie seu relatório de casos de teste executados, todos passando, e ainda assim escrever uma seção com vários defeitos encontrados, provenientes dessa exploração. Não testar dessa maneira pode fazer com que um testador envie seu relatório de casos de teste executados, todos passando, e ainda assim a equipe entregue uma aplicação cheia de defeitos para o cliente.

Sinal Verde, Perigo!

Se existe algo em toda a documentação gerada pelo processo de testes de software que não recebe muita atenção, é a seção verde do relatório de defeitos. Os testes que estão falhando geralmente ficam marcados de vermelho, para que recebam atenção especial, pois esses testes revelaram defeitos que vão precisar ser analisados. Então, por que deveríamos nos preocupar com os testes que estão passando? Porque, dependendo da maturidade do time que executou os testes, ou da maneira como a organização encara a execução de testes, pode ser que muita coisa importante tenha sido deixada de lado. Algum comportamento estranho na aplicação pode não

ter sido percebido, porque em muitos casos, os testadores ficam mais preocupados em seguir o roteiro de casos de teste, tentando fazer exatamente o que o passo a passo manda fazer, do que ver a aplicação tentando imaginar como realmente ela deveria funcionar em certas circunstâncias. Nesses casos, não é raro casos de testes incompletos darem a falsa idéia de estabilidade do sistema, enquanto defeitos continuam escondidos, a espera de alguém experiente para revelá-los.

Qual a saída? Existem alternativas para contornar este problema. Uma delas é utilizar o teste como base para a execução, e não se limitar ao que ele manda fazer. Entender qual o objetivo do teste e tentar alcançá-lo não somente através do passo a passo que foi projetado, mas imaginar outras maneiras de chegar ao mesmo objetivo, por outros caminhos na aplicação. Esse tipo de atividade requer atenção aos detalhes, diversas vistas de um só ponto (funcionalidade em questão), facilidade de reproduzir com clareza situações vividas, boa memória, clareza de escrita, pensamento crítico, etc.

O que pode ajudar muito a tornar a equipe de testes proativa e experiente para que realizem as atividades dessa maneira é buscar o desenvolvimento de cada um. Buscar um time multidisciplinar pode trazer muitos benefícios para uma organização. Assim como em qualquer profissão, o desenvolvimento de habilidades pessoais pode fazer a diferença para um time.

Habilidades do Testador

Diversas são as habilidades que podem ser desenvolvidas por cada integrante de um time de testes. Muitas habilidades estão diretamente relacionadas com algumas atividades de testes, e podem dar um diferencial ao profissional que busca desenvolvê-las, tornando suas atividades mais completas. A seguir, mencionaremos algumas habilidades e como o desenvolvimento delas pode ajudar nas atividades de teste.

Comunicação

Uma das principais atividades do engenheiro de testes é comunicar. Seja comunicação escrita ou oral, esta deve ser feita com muita clareza e precisão. Um testador deve relatar suas observações de maneira que outros integrantes do projeto possam entender rapidamente, pois o desenvolvedor, por exemplo, terá que examinar as informações que os testadores o enviam para entender o que tem de errado na aplicação. O testador deve munir a equipe do projeto de contexto, acerca do que foi encontrado. Deve relatar em que condições estava a aplicação no momento que ocorreu a falha, qual o fluxo percorrido, listar tudo o que foi encontrando pelo caminho, mesmo que não esteja descrito no caso de testes.

Se um defeito não é bem descrito, pode não receber a atenção merecida no momento em que a equipe decide priorizar defeitos para correção. Também, um desenvolvedor pode não conseguir reproduzir o problema encontrado, por falta de informações relevantes no relatório de defeitos. A comunicação escrita merece um cuidado especial, pois os relatórios de defeitos devem ser escritos de maneira imparcial. Não se deve criticar o desenvolvedor por um defeito encontrado, pois todos,

desenvolvedores e testadores, estão no projeto com o mesmo objetivo, que é entregar um produto de qualidade ao cliente. Então, é necessário muito cuidado com os termos que serão utilizados, principalmente se testadores e desenvolvedores são de países diferentes, se possuem culturas diferentes. Nesses casos, a utilização de certos termos pode causar mal entendidos difíceis de serem resolvidos. A comunicação oral também tem sua importância. Testadores devem comunicar o estado da aplicação com o intuito de ajudar, de mostrar o caminho percorrido em busca do defeito.

Um relatório de defeitos deve ser escrito para seu público-alvo. O que isso quer dizer? Um relatório feito para os desenvolvedores deve ser escrito em nível técnico. Já um relatório feito para a coordenação do projeto, talvez precise ser escrito de maneira gerencial. Em um relatório gerencial, pode ser interessante descrever como um problema em uma funcionalidade pode aborrecer o cliente, ou porque isso pode atrasar a saída do produto para o mercado, por exemplo.

Pensamento Crítico

Pensamento crítico é o que pode fazer com que um testador, ao final da execução de um ciclo de testes, perceba que todos os casos de teste da suíte estão passando, e ainda assim, ter um relatório de defeitos sem casos de testes associados a eles. Esse olhar crítico pode impedir que um produto seja lançado no mercado mesmo que a maioria de seus casos de teste esteja passando.

Também é entender que não adianta sair executando todos os testes que existem na ferramenta de gerenciamento de testes, se estes não fizerem sentido para o momento que o time está passando. Testes devem ser selecionados com critério para compor um ciclo de execução. Ao invés de colocar todos os casos de testes existentes no ciclo, muitas vezes é mais prudente analisar as funcionalidades que estão sendo desenvolvidas pela equipe. Devem-se analisar quais funcionalidades são críticas ou quais funcionalidades estão prestes a ser liberadas para o mercado. Áreas de risco devem ter prioridade, e caso o tempo seja curto para a execução dos testes, uma boa estratégia é essencial. Deve ser analisado o que será testado primeiro, o que pode ser deixado para depois, e o que pode ficar sem ser testado. Tomar esse tipo de decisão não é fácil e exige preparo e conhecimento do projeto.

Proatividade

Uma pergunta muito comum em eventos de testes de software no Brasil e no mundo é: De que maneira você mede a produtividade ou eficiência dos seus testadores? É muito comum testadores serem medidos pela quantidade de casos de testes que conseguem executar em um período de tempo. Que tipos de problemas isso pode acarretar se essa for a única maneira de avaliar a equipe?

Testes mal executados, visto que os testadores podem realizar suas atividades em busca de números, e não de qualidade. Perda do foco, porque os testadores vão ficar preocupados em executar a quantidade de casos de testes que lhes é atribuída por dia/semana. Podem deixar de perceber se algo estranho acontecer e não estiver descrito no caso de teste, pois eles podem considerar

que não tem tempo a perder analisando coisas extras. Além de que um testador proativo pode ter seu trabalho considerado igual ao de um testador que não tem proatividade alguma, porque ambos terão executado seus casos de testes no tempo acordado.

Mas o trabalho do testador proativo está por trás disso tudo. Ele deve encontrar mais defeitos na aplicação, relatar em detalhes tudo o que observou e encontrou durante seus testes, fornecer os dados de testes em arquivos para facilitar a reprodução do defeito, estar disponível para que o desenvolvedor esclareça alguma coisa que tenha sido mal entendida, etc. Por isso, somente contar quantos casos de testes foram executados pode esconder todas essas atividades que também merecem muita importância. Existem gerentes de testes, principalmente em multinacionais, que analisam as impressões do cliente, depois que o produto é entregue, como forma de analisar o desempenho da sua equipe. E, apesar de saber que é impraticável testar todas as combinações de todos os fluxos da aplicação, de maneira que não podemos garantir que uma aplicação não tenha nenhum defeito, os defeitos que são encontrados pelo cliente também servem de indicativo de como a equipe está se saindo na realização das suas atividades.

Criatividade

Uma pergunta importante. Quem usa a criatividade no seu time de testes? Quando estamos seguindo um processo de testes bem definido, cada um tem seu papel dentro do time. Tem o arquiteto, que organiza a estratégia de testes e os planeja. Depois o projetista de testes pega esse planejamento e escreve os casos de teste, e então os executores pegam esses casos de testes projetados e os executa. Se os executores só executam o que o projetista determinou que ele iria fazer, e se o projetista só projeta o que o arquiteto determinou que ele iria projetar, o uso da criatividade fica restrito a um terço do time. No máximo o projetista pode interferir em alguma coisa, mas os executores ficam somente seguindo um roteiro, algo que o mandaram fazer. Quais são as conseqüências desse tipo de processo? A lista é grande, mas aqui vão alguns itens que merecem atenção especial:

1. Com o tempo a execução de testes pode se tornar uma atividade cansativa, à medida que os testes sofrem poucas modificações, ou o testador já tenha executado aquela suíte algumas vezes;

2. É gerada uma forte dependência de roteiros de teste, que por conseqüência devem depender de uma documentação de requisitos;

3. Manutenção difícil da suíte de testes. Tendo em vista que requisitos mudam com freqüência, e isso impacta diretamente na suíte de testes gerada.

Como o desenvolvimento e uso da criatividade podem ajudar a todo o time? À medida que os testadores, nesse caso, os executores, se sentem livres para explorar a aplicação da maneira que acham mais conveniente, podem utilizar o roteiro de testes somente como uma garantia de cobertura do cenário do requisito correspondente, e depois, começar a investigar detalhes na aplicação. Para desempenhar essa atividade, é fundamental que os executores tenham um bom conhecimento do domínio da aplicação, habilidades técnicas, envolvimento com o setor de

marketing da empresa, para entender o que o mercado espera do produto, etc. Uma reunião de brainstorm com todo o time, principalmente com arquitetos e desenvolvedores, pode gerar muitas idéias de cenários que podem ser explorados. O testador usa a sua imaginação, mas pode ser ajudado pelo resto da equipe. Se colocar no lugar de grupos de pessoas que vão utilizar a aplicação também pode fazer surgir novas idéias. Entender como se comportam esses grupos vai dar uma noção do que eles vão buscar na aplicação, ou o que eles gostariam de ver.

Conclusões

Testes de Software estão muito mais relacionados com o que o testador pode fazer no seu dia-a-dia do que com documentações e realização de atividades em seqüência. Apesar da enorme importância de ter um processo como base, é imprescindível o desenvolvimento das habilidades das pessoas, para que elas saibam realizar bem suas atividades até mesmo em empresas que não têm nenhum processo bem definido. É importante a consciência de que testar um software exige experiência, e que isso faz a diferença no momento em que existem pressões de prazo, orçamento, ausência de ferramentas, enfim, limitações de recursos no projeto. Em suma, trata-se de uma atividade que requer um perfil próprio para o profissional, diferenciado do perfil do desenvolvedor, e que requer também muita especialização, tendo em vista a quantidade de áreas distintas que existem dentro da área de testes de software, por exemplo: Testes de Unidade, Testes de Funcionalidades, Testes de Desempenho e Testes de Segurança, dentre varias outras áreas. ●

Referências

Lessons Learned in Software Testing - Cem Kaner, James Bach, Bret Pettichord.

Black Box Software Testing - <http://www.testingeducation.org/BBST/>

Rapid Software Testing - http://www.satisfice.com/info_rst.shtml

Foundations of Software Testing: ISTQB Certification - Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black

The snake in the monkey's shadow. Pradeep Soundararajan - <http://testertested.blogspot.com>

Schools of Software Testing - Bret Pettichord

The Art of software Testing - Glenford J. Myers

Chaos Report 2004 - www.standishgroup.com

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto.

Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

