Conhecimento e Teste Exploratório: um modelo de captação e execução

Karen Braga, **Eduardo Pretz**

Centro Universitário Feevale – Instituto de Ciências Exatas e Tecnológicas - Novo Hamburgo – RS – Brasil

karencbraga@gmail.com.br, epretz@feevale.br

Abstract*. Since the system lacked much documentation and the test was applied on a tight schedule, it became necessary to look for a kind of assay, among the available ones, that was both dynamic and effective – inherent characteristics of the exploratory test. This paper presents a study about exploratory testing, knowledge management and a model for storing the link between the tester's knowledge and the test itself, also included are requisites that could be identified as the exploration were running along. This report was important to check the viability of planning the exploratory test before its execution, and also to create a repository of know-how about test strategies and execution history.*

Resumo. *A falta de documentação dos sistemas aliada ao pouco tempo despendido ao teste, provocou a necessidade de buscar, dentre os testes existentes, uma forma de teste que fosse dinâmico e eficaz - características inerentes ao teste exploratório. Neste artigo é apresentado um estudo sobre teste exploratório, conhecimento e um modelo que armazena o elo entre conhecimento do testador e o teste, bem como os requisitos identificados no decorrer da exploração. Esse registro foi importante tanto para verificar a viabilidade de planejar o teste exploratório antes da execução, como também para gerar um repositório de conhecimento de estratégias de teste e histórico da execução.*

1. Introdução

Um dos grandes problemas enfrentados por empresas de desenvolvimento de *software* está relacionado à falta de documentação de sistemas legados. Esta falta impacta não somente no desenvolvimento e manutenção de sistemas, mas também no teste, impossibilitando seu planejamento e dificultando sua execução, que muitas vezes é realizada de maneira *ad-hoc*. Para suprir esse problema, o teste exploratório pode ser uma solução imediata.

O objetivo deste trabalho é propor um modelo de registro do teste exploratório com a finalidade de manter seu histórico, bem como a captura do conhecimento do testador durante a sua execução, visando formar uma base de conhecimento de estratégias de teste realizadas pelo testador. Atualmente, o teste exploratório está associado à realização de testes de maneira informal, ou seja, sem o estabelecendo de objetivos e estratégia bem definidos para a sua execução.

Para a realização deste trabalho, foi necessária uma revisão bibliográfica sobre teste exploratório, conhecimento e pesquisa sobre ferramentas para registro e captura do teste. A seção 2 aborda o conceito sobre teste exploratório, enfatizando aspectos relacionados ao seu gerenciamento e apresentando a abordagem SBTM (*Session-Based Test Management*) proposta por BACH (2009c). O teste exploratório se baseia na intuição e depende do

conhecimento, habilidade e experiência do testador. Tal conhecimento não é registrado a fim de servir de conhecimento a outro testador. Aspectos como: “O que é o conhecimento?”, “Como ele é adquirido?”, “Como ele é transformado?” serão tratados na seção 3. A partir da identificação dos requisitos durante os testes, estes serão associados aos casos de teste visando a rastreabilidade. Na seção 4 é apresentado de maneira sucinta para que serve a rastreabilidade e onde aplicá-la. Esta seção apresenta também os itens que compõem os casos de teste, destacando o papel principal da rastreabilidade. O modelo que propõe o registro dos requisitos e do conhecimento do testador durante o teste é apresentado na seção 5. Na seção 6 é realizado um experimento com o modelo sobre um módulo de aplicação educacional, apresentando os resultados do teste. E, por último, as conclusões são apresentadas na seção 7.

2. Teste exploratório

O termo teste exploratório, preconizado por Cem Kaner no livro *Testing Computer Software*, refere-se a uma abordagem de teste muito diferente do teste orientado, ou teste com roteiro. Ao invés de realizar uma análise seqüencial de necessidades, seguida pela concepção e documentação e execução de casos de teste, o teste exploratório foi definido, por James Bach (2009a), como um processo em que o testador aprende, elabora o design do teste e executa simultaneamente enquanto explora o produto.

No teste exploratório, o testador projeta casos de testes na medida em que os testes são realizados, e não dias, semanas ou meses antes do início dos testes. Além disso, as informações coletadas pelo testador na execução de um conjunto de testes servem de guia para projetar e executar o próximo conjunto de testes (Copeland 2004).

Muitas vezes esse processo é confundido com teste *ad-hoc* que, por definição, refere-se a um processo improvisado de teste, com a função de buscar erros.

Para Bach (2009c) o teste exploratório deve ser utilizado nas seguintes situações:

- Necessidade em fornecer feedback rápido sobre um novo produto ou serviço.
- Necessidade de aprender o produto rapidamente.
- Diversificar os testes e melhorá-los.
- Encontrar o erro mais importante no menor espaço de tempo.
- Investigar e isolar um defeito específico.
- Investigar a situação de risco em especial, a fim de avaliar a necessidade de testes com script nessa área.

O resultado de uma sessão de teste exploratório é um conjunto de notas sobre o produto, falhas encontradas, e o registro de como o produto foi testado. Os elementos que compõem o teste exploratório são (Bach 2009b):

- Exploração do produto: descobrir e registrar o objetivo e as funções do produto, tipos de dados processados e áreas de potencial instabilidade. Esta fase depende do entendimento da tecnologia utilizada, informações sobre o produto e usuários, e a quantidade de tempo disponível para fazer o trabalho.

- Projeto de teste: determinar as estratégias de operação, observação e avaliação do produto.
- Execução do teste: operar o produto, observar o comportamento e utilizar informações para formar hipóteses de como o produto funciona.
- Heurísticas: são guias ou regras que ajudam o testador a decidir o que fazer, o que deverá ser testado e como testá-lo.
- Resultados: é o resultado do teste. Este é finalizado uma vez que o testador produziu resultados que atendam os requisitos especificados.

Este teste caracteriza-se pela dinamicidade. Para isso, Bach (2009c) busca uma maneira de planejar os testes e documentar sua execução, a fim de saber o que foi testado, o que foi encontrado e quais são as prioridades para os próximos testes, sem obstruir a flexibilidade de testar o produto de forma exploratória. Destes estudos, surgiu a técnica SBTM (Bach 2009c)(Crispin 2009).

2.1 Session-Based Test Management

Durante a jornada de estudos sobre o teste exploratório, Bach (2009c) percebeu que testadores faziam muitas coisas durante o dia além de testar. Para monitorar os testes, seria preciso encontrar uma forma de distinguir o teste de qualquer outra coisa. Foi quando surgiu a idéia das sessões.

Uma sessão não é um caso de teste ou registro de um *bug*, mas uma unidade básica de teste. Um bloco ininterrupto[1], revisável[2] e objetivo[3] de um esforço de teste.

Cada sessão dura em torno de 90 minutos. O tempo não é marcado rigorosamente, pois o autor (Bach 2009c) considera que um bom teste é mais importante do que o tempo que foi gasto para se testar. Se uma sessão durar aproximadamente 45 minutos, ela é chamada de sessão curta. Se ela demorar mais que duas horas, é chamada de sessão longa. Por causa de reuniões, e-mails e outras atividades importantes, espera-se que cada testador realize não mais que três sessões por dia.

O que acontece em cada sessão depende do testador e da missão da sessão. Um testador pode ser alocado para analisar uma função, ou procurar um problema particular, ou verificar um conjunto de *bugs* consertados. Os riscos de negócio podem ser considerados para identificar missões da sessão, caso seja de conhecimento da equipe de teste. Uma missão serve para que o testador mantenha o foco e ajuda-o a testar a aplicação da melhor forma possível, eliminando distrações e tangentes que poderia encontrar. A missão não determina como ele deve realizar a tarefa. As ações específicas necessárias para satisfazê-la são deixadas a critério do testador, que as determina no momento em que ele começa a testar a aplicação.

---

[1] Significa ausência de e-mails, reuniões, conversas ou ligações telefônicas.
[2] Significa que um relatório é produzido e pode ser examinado por uma terceira parte, como o gerente de testes, e provê informações sobre o que aconteceu.
[3] Significa que cada sessão é associada a uma missão – o que é testado ou qual problema está sendo procurado.

Após o término de uma sessão, reuniões são realizadas com o líder de teste a fim de entender e validar o relatório da sessão e, também, fornecer *feedback* e treinamento ao testador. Estas reuniões visam identificar o quanto pode ser feito em uma sessão de teste e, através do rastreamento de quantas sessões são realmente realizadas em um determinado período de tempo, é adquirida a habilidade de estimar a quantidade de trabalho envolvido em um ciclo de teste permitindo prever quanto tempo o teste vai durar mesmo sem o trabalho ter sido planejado em detalhes.

Para saber quais tarefas foram realizadas durante o teste, é necessário que os testadores as reportem de forma genérica. O testador precisa registrar a atividade executada, reações do sistema, dados utilizados, condições, diagnósticos ou idéias. (Lyndsay 2009)

As sessões de teste são divididas em três tipos de tarefas (Bach 2009c)(Crispin 2009): *Design* e Execução[4], Investigação e *Report* de *bugs*[5] e *Setup* da sessão[6], que são chamadas de TBS (*Task Breakdown Structure*). Após, é necessário que os testadores estimem o tempo gasto para cada tipo de tarefa. Isso inclui, também, os tempos gastos em missões associadas às sessões e o tempo de teste gasto em oportunidades, ou seja, o tempo despendido a qualquer teste que não esteja descrito na missão da sessão.

Lyndsay (2009) introduz um conceito importante que contribuiu ao método de Bach (2009b), que são os pontos de teste (*test points*) com o objetivo de controlar o escopo do teste. No projeto no qual Lyndsay (2009) participou não existia uma lista de teste nem tampouco requisitos do software e histórico de testes realizados. Para que ele pudesse controlar o que estava sendo testado, foi necessário levantar os aspectos que deveriam ser explorados e identificar as fontes de apoio (*oracles/sources*). Com base nessas informações, partiu-se para elaboração dos *test points*[7] que teriam como características:

- Um *test point* levaria entre 20 minutos e 4 horas. Esta estimativa de tempo é feita no momento em que o ponto de teste é definido. Pode ser redefinido durante os testes.
- Cada *test point* possui uma escala de risco. Esta avaliação também é feita como parte do processo de definição do ponto de teste.
- Cada trabalho de teste está relacionado a *test point*. Pode conter um ou vários pontos que deverão ser investigados durante o tempo da sessão.

O que difere um ponto de teste de uma sessão é que o primeiro está relacionado com a unidade de trabalho, enquanto que o segundo com a unidade de tempo (Lyndsay 2009). Um ponto de teste pode ser repetido e uma sessão de testes está prevista, acontece e é registrada.

A lista de *test points* é dinâmica, ou seja, novos pontos são adicionados com base em erros encontrados e correções. Além disso, uma nova compreensão da funcionalidade do software pelo testador é adquirida durante a exploração.

---

[4] Significa navegar pelo produto e procurar problemas.

[5] É o que acontece quando o testador se depara com um comportamento que parece ser incorreto.

[6] É qualquer outra coisa que os testadores fazem para que tarefas de "Design e Execução" e "Investigação e *Report* de *bugs*" sejam realizadas. Isso inclui tarefas como: configuração de equipamento, localização de materiais, leitura de manuais, ou escrita de relatório da sessão.

[7] Exemplo de *test point* - Examinar o controle do usuário utilizando o tipo de usuário xxx e grupo yyy

Tinkham (2009) em seu estudo comenta que dois testadores com a mesma missão de teste podem tomar dois caminhos totalmente diferentes para atingir o objetivo. Nenhum dos caminhos é necessariamente melhor ou pior que o outro (embora um possa encontrar erros que o outro não). Cada testador desenvolve um estilo próprio e está baseado em diversos fatores, incluindo as experiências passadas, habilidades específicas, conhecimento aprofundado sobre o negócio, e aspectos de sua personalidade, bem como o estilo de aprendizagem (a forma como o testador aprende a informação).

3. Conhecimento

No item anterior, foi mencionado que o teste exploratório tem como uma de suas características a aquisição do conhecimento enquanto explora a aplicação. Mas o que é conhecimento? Como ele é adquirido? Como ele é transformado?

Conhecimento não é dado e nem informação, apesar de estarem relacionados, nem tão pouco são considerados sinônimos. Para Marçula (2009), a informação é criada quando uma pessoa que possua os dados consegue formatá-los, filtrá-los ou resumi-los, de maneira que tenha algum sentido ou alguma utilidade na obtenção de algum resultado (por exemplo, a realização de uma determinada tarefa de teste). A informação, quando interpretada ou usada para tomar uma ação ou decisão, gera um resultado, que é um novo conhecimento. Esse novo conhecimento é armazenado junto com outros já obtidos formando o conhecimento acumulado.

A informação só se torna conhecimento quando alguém a aprende e potencialmente cria novas informações e idéias (Marçula 2009). Portanto, o conhecimento está ligado à utilização da informação. Ele pode ser adquirido por diversos meios, incluindo reuniões, conferências, experiências de projetos anteriores, pesquisas, publicações e entre as relações interpessoais.

O conhecimento pode ser classificado em (Stewart 1998):

- Conhecimento Tácito – conhecimento que você não sabe que tem.
- Conhecimento Explícito – conhecimento que você sabe que tem.
- Lacunas Conhecidas – Conhecimento que você sabe que não tem.
- Lacunas Desconhecidas – Conhecimento que você não sabe que não tem.

No entanto, o conhecimento tácito é o tipo de conhecimento mais difícil de ser expresso, por ser altamente pessoal (Nonaka e Takeuchi 1997). É difícil de formalizá-lo, comunicá-lo e está profundamente enraizado na ação e comprometimento do indivíduo. Este conhecimento consiste em habilidades técnicas, crenças, perspectivas e percepções.

O conhecimento tácito e o conhecimento explícito não são entidades isoladas, mas complementares, pois interagem um com o outro e realizam trocas nas atividades criativas dos seres humanos. Nonaka e Takeuchi (2008) apresentam quatro modos de conversão do conhecimento:

- Socialização – conversão de conhecimento tácito em conhecimento tácito.
  Compartilhar e criar conhecimento tácito através da experiência.

- Externalização – conversão de conhecimento tácito em conhecimento explícito. Articular o conhecimento tácito através do diálogo e da reflexão.
- Combinação – conversão de conhecimento explícito em conhecimento explícito. Sistematizar e aplicar o conhecimento explícito e a informação.
- Internalização – conversão de conhecimento explícito em conhecimento tácito. Aprender e adquirir novo conhecimento tácito na prática.

O teste exploratório tem como meta utilizar os recursos disponíveis, habilidades e conhecimento da melhor maneira possível, a fim de encontrar o maior número de erros críticos dentro do tempo disponível. É baseado na habilidade e conhecimento do testador sobre o processo e técnicas de teste. Tal conhecimento, uma vez externalizados de maneira formal, servirá de base para novos conhecimentos a outros testadores.

4. Suíte, Casos de teste, Requisitos e Rastreabilidade

Quando se possui a tarefa de testar um software, a primeira atividade a ser realizada é uma análise do sistema em questão e definir quais itens serão testados. Feito isso, para cada item devem ser elaborados casos de teste. Casos de teste, ou *test cases*, servem para definir um teste e o que será testado com ele, como um roteiro a ser seguido pelo testador (Hailpern 2002). Molinari (2008) acrescenta que casos de teste são um conjunto de entradas, condições de execução e resultados esperados para um objeto em particular. Uma suíte de teste representa uma coleção de casos de teste. Ela contém instruções e objetivos para cada coleção e pré-requisitos para a sua execução, caso seja necessário (Petroski 2009).

O objetivo destes casos é indicar ao testador os passos que este deve executar para alcançar a parte da aplicação que poderia tornar-se vulnerável, caso não fosse implementada cuidadosamente. Sendo assim, é preciso adicionar os requisitos propostos a algum modelo que descreva o comportamento do software, tais como caso de uso. De posse desta representação, o próximo passo é interpretar o modelo e gerar os casos de teste.

A documentação de um caso de teste deve compreender uma série de dados. São eles (IEEE 1998)(Rios 2007): um identificador do caso de teste, um estado inicial, uma seqüência de operações de teste, dados de entrada, ambiente necessário para execução do teste, procedimentos especiais, dependências – requisitos e/ou outros casos de teste e os resultados esperados.

Rios (2007) define que os casos de teste estabelecem quais informações serão empregadas durante os testes e quais os resultados esperados. No entanto, é necessário determinar a massa de dados a ser utilizada no teste, a fim de validar todos os requisitos do software.

Uma vez descritos os artefatos (requisitos e casos de teste), é necessário relacioná-los com os componentes que os implementaram (Sayão 2009). Este relacionamento denomina-se rastreabilidade.

A rastreabilidade de requisitos pode ser vista como a habilidade de acompanhar e descrever a vida de um requisito, em ambas as direções. Ela é utilizada a fim de prover o relacionamento entre requisito e outros artefatos do sistema. Esses relacionamentos permitem aos projetistas mostrar que o projeto atende aos requisitos. A rastreabilidade também apóia a detecção precoce de requisitos não atendidos pelo software e permite que

mudanças em qualquer artefato - requisitos, especificação e implementação - sejam rastreadas através do sistema (Sayão 2009).

5. **O Modelo**

Nas seções anteriores foi mencionado que o teste exploratório precisa ser planejado para que seja eficiente. Visto que não há documentação para planejar os testes, este modelo parte das informações coletadas, seja pelo especialista, código fonte, ou manuais, se houver.

A primeira atividade a ser realizada é identificar junto ao especialista o propósito do software, bem como as funcionalidades principais e secundárias (contribuintes). Para Bach (2009b), uma função é considerada primária se ela estiver associada e for essencial ao propósito do produto. Uma função é classificada como contribuinte quando ela não for primária e contribuir para a utilidade do produto. Uma vez definidas as funcionalidades e classificadas, o próximo passo é identificar a missão das sessões, os *test points* e os riscos associados.

Algumas ferramentas serão utilizadas para auxiliar no registro e captura dos testes, que são: *Testlink*, *Mantis* e *Wink*.

Para que fosse possível registrar a bateria de testes realizados e o conhecimento do testador, foram necessárias algumas adaptações nos itens que comporiam a suíte e os casos de teste disponíveis na ferramenta de gestão de teste.

A suíte de teste conterá os seguintes campos, conforme pode ser visto na Figura 1:

- Missão do teste/Test Points/Riscos
- Tempo da missão

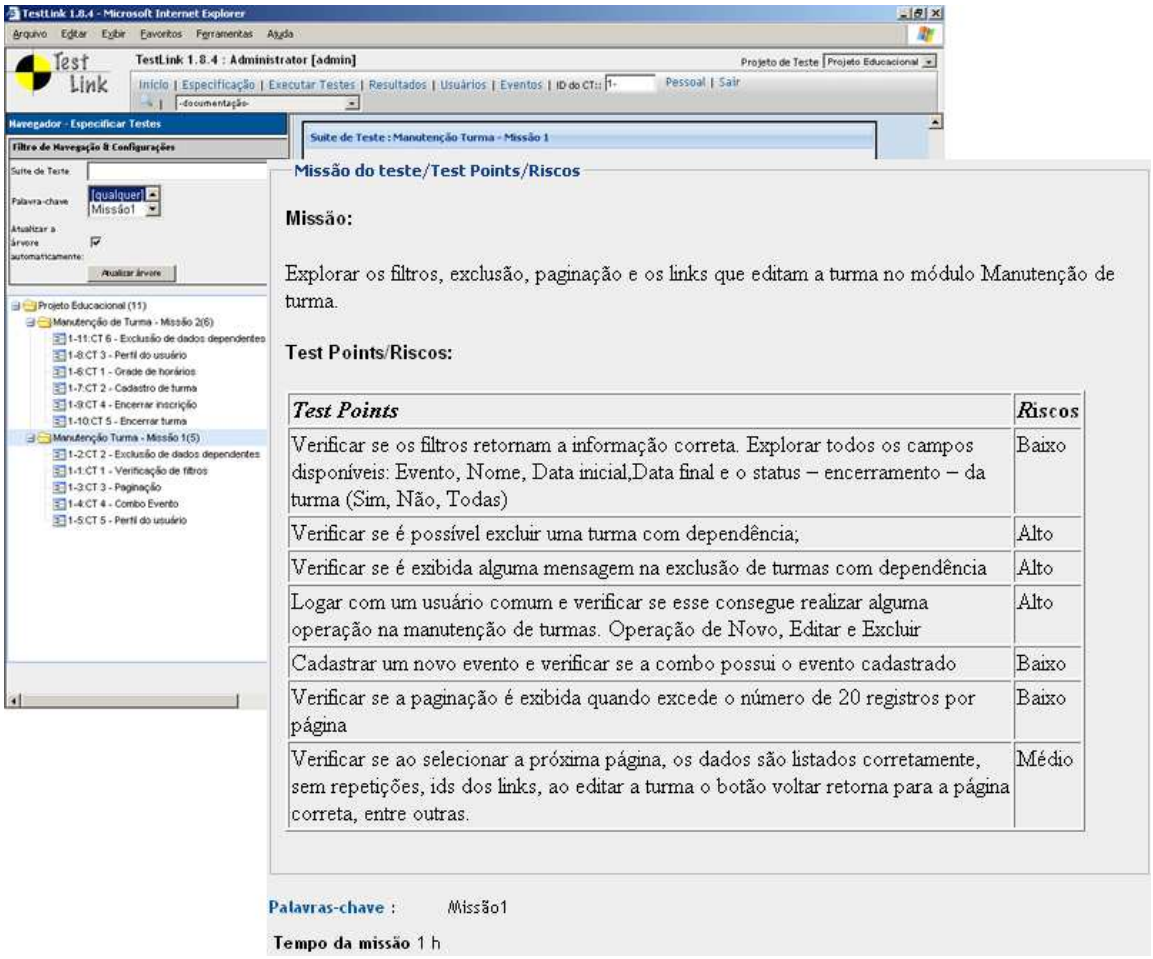

Figura 1. Missão 1 cadastrada na Ferramenta *Testlink*
Fonte: Da autora

Já os casos de teste conterão os seguintes campos, conforme pode ser visto na Figura 2:

- Título do Caso de Teste
- Sumário: breve descrição do que foi realizado na exploração, ambiente necessário para execução do teste e procedimentos especiais
- Passos / Resultados Esperados / Dúvidas durante o teste: seqüência de operações de teste (dados de entrada), resultados esperados e dúvidas encontradas no decorrer da execução com as respectivas respostas
- Conhecimento tácito do testador sobre o teste: Esta informação será fornecida pelo testador durante a execução da sessão de teste, pois visa registrar o conhecimento tático do testador. Quando o testador recebe uma missão para testar certa funcionalidade, este já possui em mente quais estratégias utilizará para atingir o objetivo no teste exploratório (Lyndsay 2009). Será nesse campo que ele irá descrever a percepção do porque realizou o teste. A missão poderá ser descrita, bem como os pontos de teste que orientaram a tomada de decisão.

- Bugs: integração com o Mantis
- Palavra-chave: para auxiliar na busca
- Risco
- Dependências: requisitos (rastreabilidade)
- Anexos: Vídeo gerado pela ferramenta Wink

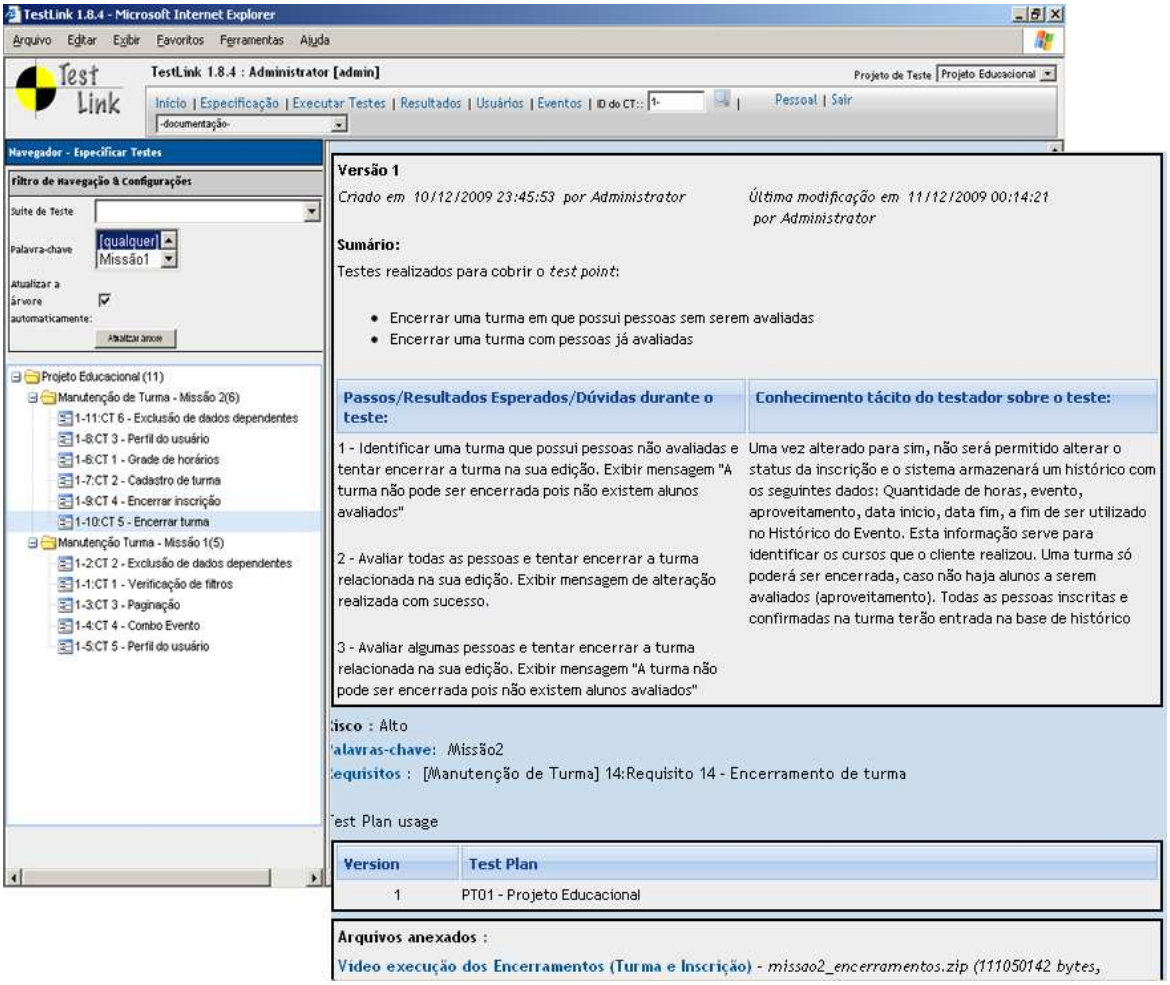

Figura 2. Caso de teste Encerramento de turma.
Fonte: Da autora

A fim de adequar a nomenclatura do teste exploratório com a ferramenta de gestão de teste, equivaleu-se uma missão a uma suíte de teste, em que a missão conterá os diversos casos de testes identificados durante a exploração. A missão do teste e os *test points* estarão descritos na suíte de teste e um caso de teste deverá explorar um ou mais *test points* estabelecidos para a sessão.

Todos os materiais gerados durante a sessão de teste serão anexados à suíte correspondente, exceto os filmes que estarão associados a cada caso de teste.

Cada missão de teste deverá conter as seguintes informações:

- O que testar
- Porque testar
- Como testar

E, como resultado da execução, a aprovação ou reprovação da funcionalidade. Para isso, a ferramenta de gestão de teste foi customizada para aceitar tais informações, conforme mencionado anteriormente.

A Figura 3 mostra o fluxo do funcionamento do registro do teste desde a execução.

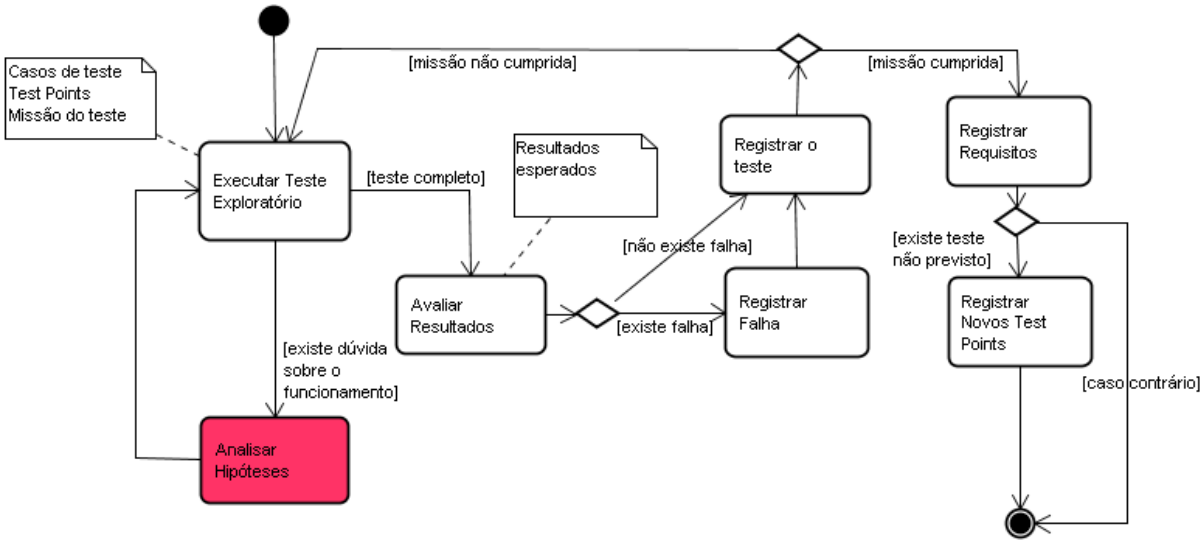

Figura 3. Fluxo de funcionamento do modelo de teste
Fonte: Da autora

No diagrama de atividades, apresentado na Figura 3, a imagem simbolizando uma nota e a linha tracejada que são ligados às atividades representam os insumos que abastecem o modelo. Para iniciar o fluxo são obrigatórios: os *test points*, a Missão do teste e o resultado esperado de cada *test point*.

O modelo de execução do teste exploratório inicia quando o testador recebe uma requisição de teste com uma missão e o que observar/explorar no decorrer da investigação (*test points*). Se houverem casos de teste anteriores, estes também servirão de insumo para o teste.

Durante a sessão de teste, que durará entre 20 minutos a 2 horas, todos os *test points* são explorados. Os *test points* identificados como de alto risco são os primeiros a serem executados, pois requerem mais tempo de teste. Se no decorrer do teste houver dúvidas em relação ao funcionamento de determinado controle, o testador conscientemente formula algumas hipóteses relacionadas ao resultado obtido e volta ao teste, a fim de confirmar ou não suas hipóteses. Quando estas forem sanadas, o testador irá confrontar o(s) resultado(s) obtido(s) com o resultado esperado, que já é de seu conhecimento. Um bom exemplo para o processo de utilização da formulação de hipótese vinculada ao teste é o comportamento do botão que, ao ser clicado, desencadeia uma ação inesperada. As hipóteses auxiliarão na investigação do motivo dessa ação ter ocorrido.

O testador deverá sempre buscar informações sobre o comportamento correto do software obtido durante a sessão. Em caso de dúvidas, o modelo SBTM (Bach 2009c) prevê uma área chamada de *Issues* onde o testador as descreve para depois serem tratadas durante a reunião diária entre os testadores e o líder de teste. Esta área estará disponível na ferramenta de gestão de teste para ser descrito junto ao caso de teste.

Após a avaliação do resultado, o teste é registrado independente de seu *status*. Em caso de erro, este é registrado na ferramenta de gestão de erros com o ID identificado dentro do caso de teste. O fluxo fica em *loop* até que todos os *test points* sejam cobertos, indicando o fim da missão. Quando a missão for finalizada, o testador registra os requisitos de sistema identificados no decorrer do teste e os novos *test points* originados das baterias de teste. Estes novos *test points* irão ocasionar novos casos de testes, caso não haja nenhum relacionado ao domínio do *test point* identificado.

Um ponto importante de se destacar são os registros dos pequenos testes realizados durante a sessão. Estes pequenos testes gerarão históricos de todos os passos realizados pelo testador durante a exploração dos *test points*, o que será útil na reprodução dos mesmos pelo testador, ou por outra pessoa, em uma nova bateria de execução, ou seja, não será necessário repensar os testes, pois estes já estarão registrados.

A ferramenta de gestão de testes disponibiliza a funcionalidade de impressão de relatórios, o que permite visualizar as informações lançadas durante a execução do teste. Com as mudanças realizadas para adaptar ao modelo, as informações presentes no relatório serão:

- Missão do teste, *Test Points* e Riscos
- Tempo da missão
- Identificador do caso de teste
- Autor
- Sumário
- Passos/Resultados Esperados/Dúvidas durante o teste
- Conhecimento tácito do testador sobre o teste
- Requisitos
- Palavras-chave
- Risco

Ao final da execução do teste exploratório, as seguintes informações sobre o módulo testado estarão registradas:

- Casos de teste
- Requisitos funcionais e não-funcionais (se houver)
- Registro dos *bugs*
- Vídeo da execução do teste

- Rastreabilidade

Todo o conhecimento do teste ficará centralizado na ferramenta de gestão de teste para consultas, reprodução do teste, novos conhecimentos, entendimento da estratégia aplicada pelo testador durante a investigação, entre outras aplicações. Uma limitação encontrada na ferramenta *Testlink* é que a mesma não dispõe de uma pesquisa genérica para que possa buscar os casos de teste cadastrados, independente de seu projeto. A ferramenta possui um mecanismo de busca, mas somente dentro de um projeto específico. O ideal, para gerenciar o conhecimento, seria dispor de uma estrutura de fácil interação para trazer os dados cadastrados, bem como a possibilidade de qualquer usuário contribuir na manutenção do conhecimento introduzido na ferramenta. Isso só seria possível, caso todos logassem como administradores. A incorporação do mecanismo de busca é uma importante sugestão para a próxima versão da ferramenta.

Na próxima seção, será apresentada a aplicação do modelo proposto.

6. **Estudo de Caso**

O software a ser testado pertence a uma empresa que desenvolve sistemas voltados à informatização de órgãos públicos, especialmente Prefeituras Municipais e Câmaras de Vereadores. A empresa possui três produtos: ERP, administração educacional e geoprocessamento. Para fim deste trabalho, o modelo será aplicado sobre a ferramenta educacional.

Este estudo de caso está dividido em quatro fases: conversa com especialista; preparação e execução do teste exploratório; validação dos requisitos e, por fim, apresentação dos erros encontrados ao especialista.

A ferramenta que foi sugerida para aplicar o modelo proposto neste artigo é uma aplicação legada que não possui documentação de seu funcionamento. Para este caso não foi possível elaborar a documentação do teste e, após, executar os casos de teste, como é o processo normal de teste. Foi necessário realizar o processo inverso, ou seja, a partir da execução do teste chegar aos requisitos e casos de teste.

A primeira etapa foi, em conjunto com o especialista, identificar as funcionalidades principais e secundárias do sistema e o propósito da ferramenta. Como funções principais podem ser citadas: Cadastro de usuários, Eventos, Manutenção de Turmas, Inscrições e Emissão de Certificado. Destas, somente o módulo de Manutenção de Turmas será utilizado para o teste.

A funcionalidade de Manutenção de Turmas tem como objetivo cadastrar/editar as turmas dos cursos oferecidos pela empresa. Uma vez aberta uma turma, o cliente pode realizar sua inscrição no curso através da área administrativa da ferramenta. Para isso, ele precisa ter um *login* e senha.

Durante as conversas com o especialista foram identificadas algumas características que podem ser consideradas requisitos de negócio e sistema. Com base nessas características foram elaborados os *test points* que irão compor a missão do teste. Dentre as características e informações coletadas destacam-se as seguintes:

1. O sistema deve apresentar possibilidade de encerrar uma turma. Uma vez alterado para “sim”, não será permitido alterar o *status* da inscrição e o sistema armazenará

um histórico. Uma turma só poderá ser encerrada caso não haja alunos a serem avaliados (aproveitamento).

2. O sistema deve exibir paginação ao completar 20 registros na tela.

O teste possuirá duas missões, sendo a primeira relacionada com a tela principal, onde o testador deverá explorar os filtros, exclusão, paginação e os *links* que editam a turma. Já a segunda missão estará relacionada ao cadastro e edição da turma, sendo necessário explorar todos os campos disponíveis e a grade de horários. No Quadro 1 é apresentado um exemplo do mapeamento entre as características, a missão do teste, os *test points* e os riscos associados.

Característica	*Test points*	Risco	Missão
1	Encerrar uma turma que possui pessoas sem serem avaliadas e verificar se o sistema informa que não é possível.	Alto	2
	Encerrar uma turma com pessoas já avaliadas. Deverá ser possível.	Alto	
	Encerrar uma turma com algumas pessoas avaliadas e outras não e verificar se o sistema informa que não é possível.	Alto	
2	Verificar se ao selecionar a próxima página, os dados são listados corretamente, sem repetições; com ids nos links; ao editar a turma o botão voltar retorna para a página correta.	Médio	1

Quadro 1. Mapeamento entre as características do software, *test point*, riscos e missão do teste.
Fonte: Da autora.

Para este estudo foram utilizadas as ferramentas:

- *Mantis*: para o registro dos *bugs* encontrados durante a execução do teste.
- *Testlink*: para o registro dos casos de teste e requisitos identificados durante a execução do teste, bem como a rastreabilidade entre os artefatos.
- *Wink*: para capturar a execução do teste

A ferramenta *Testlink* possui um módulo que permite o cadastramento dos requisitos com a finalidade de associá-lo ao caso de teste. A rastreabilidade entre requisitos/casos de teste terá como finalidade a identificação de baterias de testes essenciais de serem executadas na eventual manutenção de um requisito. Sabendo-se quais casos de teste que testarão o requisito alterado, estes serão primeiramente executados em um teste de regressão. A possibilidade de estabelecer a rastreabilidade entre os artefatos do sistema foi um dos motivos pelo qual se aderiu a essa ferramenta para planejar a gestão do teste.

As ferramentas *Mantis* e *Testlink* foram integradas para que fosse possível ligar os *bugs* encontrados aos casos de teste identificados e registrados durante a investigação. Esta ligação serviu também para realizar a rastreabilidade entre os *bugs* encontrados e os casos de teste. Cada teste registrado no caso de teste possui um número que era fornecido ao *Mantis* juntamente com a palavra TC, o nome do projeto, o identificador do caso de teste e o teste que ocasionou o erro, a fim de manter a rastreabilidade. A estrutura do registro dos

testes segue a sentença: Passos do teste / Resultado esperado. Por exemplo: TC_ManutençãoTurmas_CadastroDeTurma_4 que refere-se ao teste: 4 - Nos campos de data, informado data inicial > data final > salvar / Exibir mensagem (*bug*). Se o resultado obtido não for semelhante ao resultado esperado, este gerará uma entrada na ferramenta *Mantis* e, no caso de teste, um indicativo de (*bug*) conforme mostrado no exemplo.

Todos os dados foram lançados na ferramenta *Testlink*. Para isso, foi necessário criar um projeto e plano de teste, ambos genéricos, para seguir o fluxo da ferramenta. As missões foram cadastradas como suítes, conforme mencionado na seção anterior.

As seguintes informações sobre os resultados dos testes são importantes de serem citadas:

- O tempo de teste para investigar todos os *test points* foi de aproximadamente 3 horas, sendo que o tempo da missão 1 foi de 1 hora e a missão 2 levou 2 horas**;**
- O número de casos de testes identificados foram 11, sendo 5 para missão 1 e 6 para a missão 2. Para gerar os casos de teste foi realizado um agrupamento dos *test points* semelhantes que deveriam explorar determinada funcionalidade. Por exemplo, todos os *test points* relacionados à grade de horários do módulo de Manutenção de Turmas permaneceu no caso de teste Grade de horário. Pode-se aumentar o detalhamento dos casos de testes desmembrando os *test points* mas, para isso, o testador levaria mais tempo para registrá-los.
- O número de requisitos envolvidos neste módulo foram 18
- Número de novos Test points: 3
- Número total de Test points: 35

Planejar o teste exploratório não demanda tempo e pode ser utilizado sem perder a agilidade e dinamicidade do teste.

7. Conclusão

O teste exploratório é uma abordagem extremamente eficaz se aplicado com a finalidade de buscar erros e inconsistências. No entanto, é necessário um planejamento para que o teste não perca o seu propósito, que é encontrar erros de forma ordenada e não *ad-hoc*.

O processo de registro do teste exploratório não foi burocrático, exigindo do testador a síntese dos passos realizados durante o teste. Para dar mais fluidez ao teste, foi utilizada a ferramenta *Wink* a fim de auxiliar na identificação dos testes realizados e a descrição dos erros encontrados nas missões. Em razão de o teste exploratório ser livre, é comum realizar diversos testes focados em um determinado escopo. Com isso, a gravação ajudou na lembrança da bateria de testes realizados. O testador ficou concentrado no teste e não no registro.

Todos os pequenos testes que cobriam os *test points* foram registrados dentro de um caso de teste. Com isso, não foi possível utilizar o recurso de métricas disponível na ferramenta *Testlink*. Somente as métricas dos erros foram reais, pois os erros estavam registrados no *Mantis*.

Uma limitação encontrada na ferramenta *Testlink* é que a mesma não dispõe de uma pesquisa genérica para que possa buscar os casos de teste cadastrados, independente de seu projeto. A ferramenta possui uma ferramenta de busca, mas somente dentro de um projeto específico. O ideal, para gerenciar o conhecimento, seria dispor de um mecanismo de fácil interação para trazer os dados cadastrados. A incorporação deste mecanismo de busca é uma importante sugestão para a próxima versão da ferramenta.

Um problema encontrado durante a validação do modelo foi a dificuldade de externalizar o conhecimento tácito do testador, o que confirma a afirmação de Nonaka e Takeuchi (1997) citado na seção 3. De fato, é muito difícil descrever o pensamento, as idéias e as percepções.

Como trabalho futuro, é necessário identificar uma forma de estabelecer a cobertura do teste sem se ter formalmente os requisitos do sistema, com base apenas nos *test points,* e calcular o esforço do testador gasto durante o teste exploratório.

8. **Referências Bibliográficas**

BACH, James. (2009a) “Exploratory Testing Explained”. http:// http://www.satisfice.com/articles/et-article.pdf , Outubro.

BACH, James. (2009b) “General Functionality and Stability Test Procedure” http://www.satisfice.com/tools/procedure.pdf, Outubro.

BACH, James. (2009c) “Session-Based Test Management” http://www.satisfice.com/articles/sbtm.pdf, Outubro.

BASTOS, Anderson; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. (2007) “Base de conhecimento em teste de software**”**. [2. ed. rev.] São Paulo, SP: Martins Fontes.

COPELAND, Lee. (2004) “A practitioner's guide to software test design”. Norwood, EUA: Artech House.

CRISPIN, Lisa; GREGORY, Janet. (2009) “Agile Testing: A Practical Guide for Testers and Agile Teams”. Addison-Wesley.

HAILPERN, B. e SANTHANAM, P. (2002) “Software debugging, testing and verification”. In: IBM Systems Journal, vol. 41, no 1, páginas 4 - 12, Allen Press, Inc.

“IEEE Standard for Software Test Documentation” **(**1998). http://ranger.uta.edu/~odell/Senior_Design_Common_Materials/IEEE%20Std%20829-1998.pdf., Outubro.

LYNDSAY, James. (2009) “Adventures in Session-Based Testing” http://www.workroom-productions.com/papers/AiSBTv1.2.pdf, Novembro.

MARÇULA, Marcelo. (2009) “Metodologia para gestão do conhecimento apoiada pela tecnologia da informação”**.** In: XIX ENEGEP. Encontro Nacional de Engenharia de Produção. Rio de Janeiro. 1999**.** http:// www.abepro.org.br/biblioteca/ENEGEP1999_A0278.PDF, Novembro.

MOLINARI, Leonardo. (2008) “Testes Funcionais de Software”. Visual Books.

NONAKA, Ikujiro; TAKEUCHI, Hirotaka. (1997) “Criação do conhecimento na empresa”**.** Rio de Janeiro: Campus.

NONAKA, Ikujiro; TAKEUCHI, Hirotaka. (2008) “Gestão do conhecimento”**.** Porto Alegre: Bookman.

PETROSKI, Bruno Martins. (2009) “Geração automática de casos de teste automatizados no contexto de uma suíte de testes em telefones celulares”, http://www.labsoft.ufsc.br/publications/monografia_petroski.pdf, Dezembro.

SAYÃO, M.; LEITE, J. C. (2009) “Rastreabilidade de Requisitos”, http://www-di.inf.puc-rio.br/~julio/rastreabilidade5.pdf, Outubro.

STEWART, Thomas . (1998), “Capital intelectual”, Rio de janeiro: Editora Campus.

TINKHAM, Andy; KANER, Cem. (2009) “Learning Styles and Exploratory Testing”, www.testingeducation.org/a/lset.pdf, Novembro.