Tratativa de falhas e automação de testes funcionais: O caso do WMS GTI Plug

Ricardo Terra[1,2]

[1]Cabtec Tecnologia e Sistemas
[2]Universidade Federal de Lavras

terra@ufla.br

Resumo.
A satisfação do usuário é um dos fatores para o sucesso de um sistema de informação. Este artigo descreve uma consultoria técnica para aumentar a satisfação dos usuários de um sistema de gestão de armazenagem (GTI Plug) por meio de soluções elegantes para tratativa de falhas e automação de testes funcionais.

1. Introdução

A satisfação do usuário final é um dos fatores para o sucesso de um sistema de informação [4, 1, 3]. A Cabtec Tecnologia e Sistemas – uma empresa especialista em *outsourcing* de mobilidade, identificação e rastreabilidade de dados – provê, dentre suas diversas soluções, o WMS[1] GTI Plug[2] que é um sistema completo de gestão de armazenagem. Este artigo descreve uma consultoria técnica do autor deste artigo frente à demanda de aumentar a satisfação dos usuários do GTI Plug por meio de soluções elegantes para tratativa de falhas e automação de testes funcionais. Tal consultoria retrata um comportamento proativo da empresa em antecipar potenciais problemas que poderiam vir a ocorrer frente a implementação de diversas novas funcionalidades, rotatividade de pessoal, etc.

2. Motivação (a.k.a., problema)

Um sistema de WMS é responsável por todo recebimento e expedição que ocorrem em uma empresa. Por exemplo, uma operação que não pode ser concluída pode atrasar inúmeros processos dentro de uma empresa. Como um outro exemplo, uma falha na operação pode não atualizar o estoque e impactar em tomadas de decisões.

O GTI Plug é hoje usado na gestão de armazenagem de 14 organizações (doravante denominados *clientes*). Como trata-se de um sistema *multitenant* [2] em que um *único* sistema atende a *vários* clientes, o sistema está em constante evolução. O problema é que sistemas com um fluxo alto de modificações, novas funcionalidades e melhorias tendem a ser mais susceptíveis a falhar. Ainda mais preocupante, uma falha em uma funcionalidade desenvolvida inicialmente para um único cliente pode impactar todos.

3. Soluções para a Indústria

Embora diversas soluções técnicas foram implementadas para aumentar a satisfação do cliente do GTI Plug, esta seção descreve as duas soluções – tratativa de falhas (Seção 3.1) e automação de testes funcionais (Seção 3.2) – que foram mais eficazes em aumentar a

[1]*Warehouse Management System*
[2]http://cabtec.com.br/gti-plug

satisfação dos clientes sob a perspectiva do gerente de projetos responsável pelo GTI Plug. É importante salientar que tal perspectiva foi construída com base na análise dos *feedbacks* dos clientes logo após as soluções terem sido implantadas em produção.

3.1. Tratativa de falhas

Como era: A tratativa de falhas previamente existente consistia basicamente em: (i) não deixar a falha "explodir" no cliente, i.e., apenas apresentar a clássica mensagem "Ocorreu um erro desconhecido" e (ii) prover um botão para relatar o problema ao suporte. Embora essa tratativa seja aceitável em diversos sistemas, ela pode não ser eficaz em aumentar a satisfação do usuário final.

Isso se justifica pois o usuário final pode não saber exatamente o que motivou a falha ou pode não relatar o ocorrido. Por exemplo, foram encontradas evidências de que o usuário só relatava a falha após novas tentativas. O problema era que, em caso de êxito na nova tentativa de realizar a operação, o usuário não relatava a falha e ainda passava a sensação de que "*o sistema às vezes falha, mas uma hora funciona*".

Como está: A tratativa de erros é atualmente gerenciada. Quando ocorre uma falha, persiste-se todas as informações da falha (e.g., cliente, usuário, funcionalidade e *stack-trace*), informa-se ao usuário o número de identificação daquela falha e envia automaticamente um e-mail ao gerente de projetos.

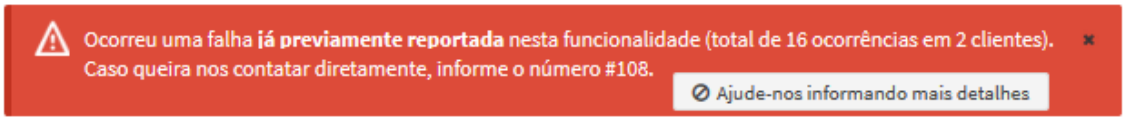

Figura 1. Reporte de falha ao usuário final

Como ilustrado na Figura 1, existe contadores do número de ocorrências da falha e de clientes afetados. Portanto, quando a mesma falha ocorrer ao usuário realizar uma nova tentativa, o mesmo identificador é apresentado e ainda relatando que já ocorreu previamente. Isso faz com que usuário saiba que a Cabtec está ciente da falha.

Essa solução promove priorização na correção. Falhas que ocorreram em diversos clientes ou que ocorreram muitas vezes devem ser priorizadas. A Cabtec – visando satisfação e robustez – incorporou no seu processo que falhas devem ser replicadas com testes funcionais automatizados e serem prontamente corrigidas.

Tecnicamente, como se trata de um sistema desenvolvido majoritariamente com JavaServer Faces[3], (a) implementou-se e registrou-se um interceptador para *actions* (`@Interceptor`) com um método que intercepta (`@AroundInvoke`) as requisições e (b) também customizou-se uma fábrica de manipulação de exceções (`exception-handler-factory`) que especifica a implementação de uma subclasse de `ExceptionHandlerWrapper` que intercepta demais problemas, como renderização de páginas, por exemplo.

[3]https://www.oracle.com/java/technologies/javaserverfaces.html

3.2. Automação de testes funcionais

Como era: O sistema possuía testes funcionais automatizados com o uso do *framework* Selenium[4]. Os testes eram criados usando o Selenium IDE que grava a interação do usuário para se reproduzir quando desejar (*record-and-playback*). De forma sucinta, essa estratégia é vantajosa no sentido da facilidade na criação de testes automatizados.

Como principal desvantagem, entretanto, está a manutenibilidade pois o uso do recurso *record-and-playback* gera código com alta verbosidade e baixa legibilidade. Como evidência prática, desenvolvedores ao realizarem uma modificação que impactava um teste funcional, optavam por remover o teste ao invés de ajustá-lo.

Como está: Os testes funcionais são atualmente automatizados com o uso do *framework* Cabelenium, o qual herda todas as funcionalidades do Selenium com adição de consistência em banco de dados (BD) e orientação à funcionalidade (termo cunhado neste artigo). Para ilustrar a mecânica do *framework*, suponha que exista uma entidade EXPEDIÇÃO:

1. Criação da classe `ExpedicaoTestCase` que deve possuir (a) os campos da tela como atributos, (b) o atributo que identifica uma expedição de forma única anotado com `@Id`, (c) o método `novo()` que cria uma nova expedição preenchendo os campos da tela com os valores dos atributos e verificando se foi corretamente persistido; e (d) o método `edita(id)` que busca o registro `id` e deixa em modo de edição.
2. Criação do `expedicao-clean.sql` que é um *script* SQL que exclui tudo no sistema relacionado àquela expedição. Por definição, os testes devem ser autocontidos e, portanto, o *clean* é sempre executado antes e depois de cada teste. Antes para limpar vestígios de uma execução anterior com falha e depois para garantir que o teste não gere efeitos colaterais em outros testes.
3. Criação de um método que facilite a criação de expedições na classe base de teste.
4. Criação dos testes funcionais usando – sempre que possível – a chamada de métodos na classe base de teste ou de outras classes (fomentar reúso).

A Figura 2 ilustra o método `planejarExpedicao()` que cria um produto chamando o método `novo` de `ProdutoTestCase` (linha 9). Por meio de um JSON no formato atributo/valor, são inicializados os atributos da classe `ProdutoTestCase`, o que justifica o item 1a acima. Uma sugestão de *clean* (linha 7) já é dada assim que conclui o método `novo` com base no atributo com a anotação `@Id` (item 1b acima).

```
1   @Test
2   @Clean({"Expedicao:ARTIGO-EXP",
3           "Armazenar:ARTIGO-REC",
4           "IniciarRecebimento:ARTIGO-REC",
5           "Recebimento:ARTIGO-REC",
6           "ImprimirEtiqueta:ARTIGO-PRD",
7           "Produto:ARTIGO-PRD"})
8   public void planejarExpedicao() {
9       this.asYouDesire(ProdutoTestCase.class, "novo").single("produto-artigo.json").run();
10      this.novaImpressaoEtiquetas("ARTIGO-PRD", armazem, 15, null);
11      this.novoRecebimentoCompleto("ARTIGO-REC", "[1;ARTIGO-PRD;15]", null);
12      this.novoArmazenamentoCompleto("ARTIGO-REC");
13      this.assertResultInDBMS(new SQL("estoque.sql").addParam("idExterno", "ARTIGO-PRD"), 15, 0);
14      this.novaExpedicao("ARTIGO-REC", "[1;ARTIGO-PRD;14]", null);
15      this.assertResultInDBMS(new SQL("estoque.sql").addParam("idExterno", "ARTIGO-PRD"), 1, 14);
16  }
```

Figura 2. Teste funcional usando Cabelenium

[4]https://www.selenium.dev/

Métodos criados para fomentar o reúso (item 3) imprimem 15 etiquetas (linha 10), planejam e efetuam o recebimento de 15 unidades (linha 11), armazenam as 15 unidades (linha 12). Depois, confere-se no BD se o estoque disponível foi para 15 (linha 13, penúltimo argumento), planeja-se uma expedição de 14 itens (linha 14) e, por fim, confere-se no BD se os estoques disponível e reservado foram para, respectivamente, 1 e 14 (linha 15, os dois últimos argumentos).

Antes e após a execução, os *scripts* de *clean* são executados (linhas 2 a 7). A cada execução de um método `novo` em uma classe que possua algum atributo anotado com `@Id`, uma sugestão de *clean* é dada. O clean permite otimizar correções. Por exemplo, o teste falha e o desenvolvedor corrigiu um *bug* na tela de expedição. Ao invés de executar todo o teste para uma aferição temporária, ele pode executar apenas o *clean* de expedição e reexecutar somente as linhas 13 a 15.

As principais funcionalidades do GTI Plug já possuem testes funcionais automatizados. Na Cabtec, configurou-se o servidor de integração contínua para garantir que logo que seja realizado o *deploy* do sistema, os testes funcionais sejam executados para garantir que o cliente tenha a satisfação de utilizar o sistema sem "surpresas".

4. Considerações finais

A Cabtec oferece o WMS GTI Plug aos seus clientes por meio de contratos. Isso indica que os clientes devem estar *sempre* satisfeitos para que não cancelem seus contratos. Este artigo descreveu duas soluções técnicas que foram implementadas no GTI Plug para aumentar a satisfação dos usuários: tratativa de falhas e automação de testes funcionais. Por limitações de espaço, outras soluções não foram descritas. Enfim, as soluções propostas aumentam a satisfação do usuário por prover um sistema mais robusto (menos falhas) e com maior qualidade (mais correto).

Por último, mas não menos importante, este artigo objetiva instigar outras empresas e instituições de P&D a adotar uma estratégia similar para automação de testes funcionais que, embora mais custosa que o recurso *record-and-playback*, provê melhor legibilidade, modularidade e manutenibilidade.

Referências

[1] Muhammad Al-Khaldi and R. Wallace. The influence of attitudes on personal computer utilization among knowledge workers: The case of saudi arabia. *Information & Management*, 36:185–204, 10 1999.

[2] Chris Brook. Saas: Single tenant vs multi-tenant - what's the difference?, 2020. Acesso em: 01/07/2020, Disponível em: https://digitalguardian.com/blog/saas-single-tenant-vs-multi-tenant-whats-difference".

[3] Mohsen Dastgir and Ahmad S Mortezaie. Factors affecting the end-user computing satisfaction. *Business Intelligence Journal*, 5(2):292–298, 2012.

[4] Bernadette Szajna and Richard Scamell. The effects of information system user expectations on their performance and perceptions. *MIS Quarterly*, 17:493–516, 12 1993.