Experiência em Automação do Processo de Testes em Ambiente Ágil com SCRUM e ferramentas OpenSource

Eliane F. Collins[1], Luana M. de A. Lobão[1, 2]

[1]Instituto Nokia de Tecnologia (INdT)
Caixa Postal 7200 – 69048-660 – Manaus – AM – Brasil

[2]Escola Superior de Tecnologia (EST) – Universidade do Estado do Amazonas (UEA)
Caixa Postal 1200 – 69065-020 – Manaus – AM – Brasil

{eliane.collins, luana.lobao}@indt.org.br

Abstract. *This paper describes the practical experience to automate testing activities using a development process based on the agile method Scrum. When adopting a testing process, there is improvement, stability and maturity of the software. The experiment results show the viability and benefits of adapting and automate testing activities with open source tools ranging from the gain in time execution of the tests and defects detection, to the reliability of the customer for final product, without an increase in costs.*

Resumo. *Este artigo descreve a experiência prática de se automatizar atividades de teste utilizando um processo de desenvolvimento baseado no método ágil Scrum. Quando se adota um processo de teste, nota-se a melhora, estabilidade e o amadurecimento do software. Os resultados da experiência mostraram a viabilidade e as vantagens de se adequar e automatizar atividades de teste com ferramentas abertas que vão desde o ganho em tempo de execução dos testes e detecção de defeitos, até a confiabilidade do cliente pelo produto final, sem o aumento de custos.*

1. Introdução

As metodologias ágeis de desenvolvimento de software se destacam dos processos de desenvolvimento tradicionais devido, principalmente, ao fato de darem prioridade ao desenvolvimento de funcionalidades através de código executável ao invés da produção de extensa documentação escrita, e ainda, respostas rápidas às mudanças e colaboração com o cliente ao invés de seguir planos rígidos e negociações contratuais [Leal 2009].

Parte importante do processo de desenvolvimento de software, o processo de qualidade envolvido requer planejamento e atenção, visto que garantir a qualidade do software desenvolvido é tão relevante quanto a sua codificação. Assim como a qualidade do produto deve ser alcançada em um meio de desenvolvimento clássico, ela também precisa ser levada em consideração nos ambientes ágeis.

Salvo algumas exceções, a maioria das empresas opta por garantir um nível de qualidade através de testes manuais, após o término de módulos específicos ou até mesmo do sistema inteiro [Bernardo e Kon 2008]. Isso é uma escolha que, dependendo do contexto do projeto, pode acarretar sérios problemas, comprometendo o andamento dos processos de desenvolvimento e teste.

O artigo tem como objetivo compartilhar uma experiência prática no uso de automação de testes funcionais, em um ambiente de desenvolvimento ágil com base no método *Scrum*, mostrando que pode ser possível garantir o mínimo de qualidade em um projeto de software mesmo com poucos recursos dedicados a teste e sem custos a mais com ferramentas proprietárias para o projeto, viabilizando assim a importância de um processo de teste.

Na seção 2 abordaremos a automação de testes, suas vantagens e técnicas para aplicá-la em um projeto. Na seção 3 é contextualizado o ambiente em que se deu a experiência, metodologia de desenvolvimento usada e o projeto piloto. Na seção 4 é abordado como a execução do processo de teste foi feita e os resultados obtidos. Por último na seção 5 temos a conclusão e lições aprendidas com o processo.

2. Automação de Testes

Há diversas medidas que podem ser tomadas para se obter testes melhores, mais ágeis, efetivos e baratos. Dentre estas podemos citar: sistematizar as atividades de testes; definir os papéis e responsabilidades vinculadas ao teste; antecipar a preparação dos testes já durante as fases de construção; conhecer as técnicas relacionadas ao tema; testar características diferentes do software nas diferentes fases de testes; estimar adequadamente os esforços para os testes; ter um controle efetivo das falhas encontradas; e automatizar os processos de testes, tanto planejamento quanto execução [Costa 2006].

Automatizar testes significa fazer uso de softwares que controlem a execução dos casos de teste. O uso desta prática pode reduzir o esforço necessário para os testes em um projeto de software, ou seja, executar maiores quantidades de testes em tempo reduzido. Testes manuais que levariam horas para serem totalmente executados poderiam levar minutos caso fossem automatizados [Tuschling 2008].

Apesar dos testes manuais também encontrarem erros em uma aplicação, é um trabalho desgastante que consome muito tempo. Automatizar seus testes significa executar mais testes, com mais critérios, em menos tempo e sem muito esforço na execução [Costa 2006]. Sem contar as inúmeras possibilidades que um script automático traz, como: cobertura de requisito e profundidade nos testes (**extensibilidade**), além de ser efetivo quanto à quantidade de dados de entrada que podem ser processadas (**confiabilidade**). Outras vantagens, que podem ser destacadas ao transformar suas rotinas de testes em scripts automáticos, são: **segurança**, **reusabilidade** e **manutenibilidade** [Bernardo e Kon 2008], [Costa 2006].

Testes melhores e mais elaborados contribuem para um aumento na qualidade do produto final. Porém, construir uma suíte de scripts de teste automáticos requer uma padronização de atividades e conhecimento em codificação e análise por parte do testador. Para fazer testes automáticos eficientes, caso o teste seja de unidade, é necessário entender a arquitetura do software e a base do código. Caso o teste seja de sistema, é necessário conhecer as regras de negócio, as funcionalidades, e os valores e situações permitidas como resultado.

Em todos os casos de automação, considerando testes de unidade ou testes de sistema, o esforço inicial requerido é maior. Isso ocorre porque é necessário fazer um planejamento para saber O QUÊ, COMO e POR QUE será automatizado. Depois de completado o planejamento, então deve-se passar para a confecção dos scripts automáticos.

2.1. Técnicas de Automação

Existem várias abordagens para se automatizar casos de teste, dentre elas se destaca a utilização de Ferramentas baseadas em Interface Gráfica que possuem capacidade de gravar e executar casos de teste. São as ferramentas conhecidas como *rec-and-play* (*Record and Playback*). Nessa abordagem a ferramenta interage diretamente com a aplicação, simulando um usuário real. Na medida em que a aplicação está sendo executada manualmente, a ferramenta oferece um suporte de gravação: ela captura as ações e as transforma em scripts. Estes podem ser executados posteriormente [Fantinato et al., 2009].

É bom frisar que para testes utilizando esta abordagem, o software que será testado não precisa sofrer alteração alguma. Não há necessidade de modificação, no sentido de padronizar o código, para que a aplicação se torne fácil de testar (testabilidade). Os testes nessa abordagem são baseados na mesma interface gráfica que o usuário irá utilizar. O trabalho aqui é encontrar uma ferramenta que ofereça o recurso necessário para testar sua aplicação específica.

Existem ferramentas *rec-and-play* para aplicações Desktop (Java Swing, interface gráfica como o KDE) e Web. A maior desvantagem de se utilizar esta técnica de automação é que o script se torna "escravo" da interface da aplicação. Para que scripts, utilizando esta abordagem, sejam criados, as ferramentas fazem uso dos nomes, posições e das propriedades dos componentes e objetos que estão dispostos na interface da aplicação. Se alguma mudança de posição, nome, tipo do componente, ocorrer, o script "quebra", ou seja, não funciona mais [Fantinato et al., 2009].

Outra maneira, muito utilizada, de automatizar casos de teste é com o uso de Lógica de Negócio (*Script Programming*). Neste caso são observadas as funcionalidades da aplicação, sem interagir com a interface gráfica. Com esta abordagem serão testadas das maiores até as menores porções do código: funções, métodos, classes, componentes, entre outros [Fantinato et al., 2009].

Geralmente é pedido que se faça uma alteração no código para que o trabalho de automação fique mais simples e produtivo. Muitas vezes o código é melhorado (*Refactoring*) e padronizado para que se consiga testar mais facilmente e sem muitos problemas. São necessários profissionais com conhecimento em código e programação para criar os scripts automatizados. O uso deste método traz muitos benefícios, por exemplo, testes que necessitam de centenas de repetições, cálculos complexos e até mesmo integração entre sistemas são feitos facilmente utilizando esta abordagem.

Existem bibliotecas, ferramentas e frameworks que suportam esta abordagem. Bons exemplos são: *JUnit*, para testar unidade de código Java; *FitNesse*, que é um framework para testes de aceitação; *MbUnit*, para testar unidade em códigos .NET; *Nester*, para fazer testes de mutação em códigos C#; *httpUnit*, para testar aplicações WEB; *Cactus*, para testar EJB; *jfcUnit* e *Abbot*, para testar aplicações baseadas em interfaces gráficas [OpenSource Testing Tools 2009].

Entre as maneiras de se automatizar, o presente artigo tem como base a abordagem *rec-and-play*. É com ela que compartilharemos as experiências, destacando suas vantagens em um projeto Web que foi desenvolvido utilizando SCRUM.

3. Contexto do Ambiente de Teste

O Instituto Nokia de Tecnologia (INdT) é uma instituição independente e sem fins lucrativos comprometida com a realização de pesquisa e desenvolvimento de soluções

tecnológicas através do desenvolvimento de aplicações, novas tecnologias e conceitos. As principais áreas do INdT são Software Livre e Interfaces de Usuário, Tecnologias de Produto e Manufatura, Experiências em Serviços e Tecnologias de Rede.

A área onde a experiência foi desenvolvida foi Tecnologias de Produto e Manufatura, onde os projetos de desenvolvimento de software não possuíam um processo de teste estruturado. Para o projeto piloto contamos com 6 colaboradores envolvidos na implementação do software, onde a metodologia de desenvolvimento adotada foi o *Scrum*.

Scrum é uma metodologia ágil e flexível, centrada no trabalho em equipe, utilizada para o desenvolvimento incremental e iterativo de qualquer produto, no caso do nosso artigo, desenvolvimento de software. Seu uso melhora a comunicação e aumenta a cooperação entre os envolvidos, com isso o ganho na produtividade aumenta [Bissi 2007].

O *Scrum* possui um processo bem definido com uma etapa de planejamento e de encerramento. Entre estas, existe uma fase chamada de *Sprint*, seu tempo de duração varia, o período mais comum é de 2 a 4 semanas. Esta iteração ocorre várias vezes durante o desenvolvimento do projeto. São elas que caracterizam as Metodologias Ágeis [Tavares 2008].

A metodologia controla as funcionalidades que deverão ser implementadas, através de uma lista chamada *Product BackLog*. Para cada *Sprint*, é feita uma reunião inicial de planejamento (*Sprint Planning Meeting)*, onde itens desta lista são priorizados pelo cliente (*Product Owner)*. A equipe então define as funcionalidades que poderão ser atendidas dentro da iteração. Essa lista planejada para a iteração é chamada de *Sprint Backlog* [Tavares 2008]. Durante o *Sprint* as atividades definidas são divididas em tarefas que são acompanhadas pela equipe através da reunião diária (*Daily meeting*), que dura no máximo quinze minutos, onde problemas e impedimentos para a execução são identificados e resolvidos.

As principais características do *Scrum* são: auto-organização da equipe de desenvolvimento, acompanhamento da evolução do produto, desenvolvimento incremental das funcionalidades do produto [Schwaber 2004] e gerenciamento dos requisitos através do documento chamado “backlog do produto”.

O cenário ao qual o projeto se encontrava era muito parecido com o que ocorria em outros projetos, contava apenas com um recurso qualificado e dedicado a tarefa de teste, pouco tempo foi estimado para as atividades de teste, sem ferramentas proprietárias para auxiliar na execução e sem disponibilidade de recursos financeiros para essa atividade.

Com isso, se tornava inviável executar um processo de teste tradicional, então se optou por executar as principais atividades de um processo de teste, porém minimizando a documentação. Os documentos utilizados foram apenas o Plano de Teste, a Especificação de Casos de Teste e Relatório de Execução de Casos de Teste. A comunicação da equipe tinha que ser maior e rápida, por isso o testador ficou alocado próximo a equipe de desenvolvimento, assim qualquer dúvida referente ao backlog e as funcionalidades eram rapidamente comunicadas. Os testes especificados deveriam ser em sua grande maioria, automatizados com ferramentas abertas para que fossem executados rapidamente, principalmente, para fazer testes de regressão e sem custos adicionais.

Como métrica, a execução de testes deveria cobrir 100% dos requisitos funcionais e não-funcionais do sistema que estavam descritos no backlog. O testador também utilizou a técnica de testes exploratórios, onde os cenários especificados eram explorados para encontrar falhas, executando a navegação da aplicação por diversos caminhos além do caminho ótimo.

3.1. Projeto Piloto

A aplicação OCSS (*On Line Customer Satisfaction Survey*) tinha por objetivo ser um sistema capaz de fornecer o resultado sobre Pesquisa de Satisfação do Cliente ao time de desenvolvimento do projeto relacionado. Com ele era possível o cadastro de projetos, líderes, clientes e dos times de desenvolvimento destes projetos e o envio de um formulário de satisfação para o cliente responder. Ao final, o time que fazia parte do projeto recebia um e-mail com as impressões do cliente sobre o projeto entregue.

A aplicação foi desenvolvida nas plataformas: Web, linguagem Ruby, framework Rails [Rails 2009], Aptana Studio como IDE (Ambiente Integrado de Desenvolvimento) e banco de dados MySQL.

4. Executando o Processo com Testes Automatizados

O processo de teste para o OCSS foi organizado de modo que pudesse ser executado em conjunto com a metodologia Scrum. Para isso, o *Product Backlog* contava não só com as estórias como também com os critérios de aceitação, usados como base para a construção dos casos de teste. A cada iteração ou "*Sprint*", que tinha duração média de duas semanas, eram escolhidas as estórias a serem implementadas, as atividades de desenvolvimento e planejadas as atividades de teste para as estórias: criar e revisar casos de testes, automatizar testes, executar, cadastrar defeitos, validar defeitos, executar regressão e documentar resultados.

O testador fazia parte do time Scrum, suas atividades do *Sprint* se dividiam em tarefas que eram acompanhadas pela equipe na reunião diária *Scrum*, o que permitiu maior interação com o time de desenvolvimento e qualquer impedimento para o andamento das tarefas era identificado e toda equipe era envolvida.

Na primeira etapa do processo, o testador escolheu ferramentas que se adequavam à plataforma de desenvolvimento e a realidade da organização, teriam que ser ferramentas abertas, de fácil configuração e execução.

Pelo Sistema (OCSS) se tratar de uma aplicação Web, a ferramenta escolhida para os testes funcionais foi o *Selenium Core e IDE* [Selenium 2009], pois é um software *open source* que utiliza a abordagem *rec-and-play* e oferece suporte a testes em aplicações Web. O *Selenium IDE* é um plugin do Mozilla Firefox. Com ele é possível gravar ações do usuário no *browser* criando scripts de teste em *HTML* e executá-las, posteriormente, na ferramenta.

Contudo, houve também a automação no gerenciamento, planejamento e elaboração dos casos de testes, a fim de, minimizar o tempo de documentação, visto que os resultados obtidos precisavam também ser reportados aos desenvolvedores de maneira rápida e prática. Para isso, de acordo com as pesquisas na web entre os grupos de teste, escolhemos outra ferramenta *open source*, chamada *TestLink* [TestLink 2009]. Com esta ferramenta é possível escrever Casos de Teste e gerar relatórios com base nos resultados obtidos a partir da execução dos testes.

Para o Cadastro de defeitos encontrados a ferramenta escolhida foi o *Mantis* [Mantis 2009], que também é *open source* e oferece bons recursos para este controle e já era usada na empresa.

O ciclo de teste que era executado a cada *Sprint* se baseava nas seguintes atividades:

1) Configuração do ambiente de teste, instalação de ferramentas e acesso aos usuários;
2) Planejar os casos de teste para as estórias do *Sprint* atual, se baseando nos critérios de aceitação e dando ênfase para criação de casos de teste dirigidos a falhas, quando os casos eram terminados o analista da equipe revisava para garantir cobertura correta das funcionalidades;
3) Com base no Caso de Teste, eram criados os scripts de teste executáveis. Para isso o sistema desenvolvido foi inicialmente executado no *browser* Mozila Firefox, onde o *Selenium IDE* trabalhava gravando toda a ação do testador e transformando-as em scripts *HTML*. Estes scripts eram editados, reunidos em uma suíte de testes e reaproveitados para atender a outros casos de testes;
4) Através do *Selenium Core*, os scripts eram executados tanto no *browser* Firefox como no Internet Explorer. A execução gerava defeitos e o testador cadastrava no *TestLink* o resultado da execução dos casos de teste. No *Mantis* o defeito era informado e direcionado ao desenvolvedor responsável, este recebia uma notificação do defeito por e-mail;
5) Relatórios eram gerados automaticamente a partir das execuções informadas. O *TestLink* oferece diversas opções para analisar os resultados gerados com base nas execuções de casos de Teste. Os relatórios mais utilizados foram: *General Test Plan Metrics*, *Failed Test Cases*, *Charts* e *Total Bugs For Each Test Case*. Eles podem ser acessados a partir da opção *Results* do menu superior do *TestLink*;
6) Quando o desenvolvedor sinalizava no *Mantis* que os defeitos tinham sido corrigidos, toda suíte de teste era executada pelo *Selenium*, validando os defeitos e fazendo a regressão na aplicação, garantindo que outra parte do sistema não foi afetada pelas alterações;
7) Caso os critérios de aceitação fossem satisfeitos, as mesmas atividades do processo recomeçavam para o próximo *Sprint*.

Na entrega final do projeto, além de serem executados e reportados todos os testes funcionais, conseguimos tempo para realizar testes de desempenho no ambiente de produção e testes de estresse, porém não de maneira automática.

4.1. Análise de Resultados

Com a experiência prática da automação do processo de teste, foi observado no decorrer do projeto ganhos, primeiramente, no sentido de aumentar a segurança da equipe quanto aos testes, pois com o ambiente automático configurado com sucesso, conseguimos cobrir 100% das funcionalidades descritas no backlog, além de realizarmos testes de interface automáticos onde foram encontrados 65% dos defeitos da aplicação. Em segundo, mesmo com pouco tempo estimado de projeto, o ciclo de teste era completamente executado, dando tempo também de realizar testes de regressão a cada *Sprint*, onde encontramos muitos defeitos gerados com as modificações no código, como deslocamento de elementos da interface, scripts dinâmicos (*javascript*) que paravam de responder aos comandos, entre outros.

Através da adoção de ferramentas para a especificação de teste e execução, economizou-se tempo na geração de relatórios de testes, com mais tempo disponível para teste, foi possível encontrar inúmeros defeitos de compatibilidade entre os diferentes *browsers*, que corresponderam a 90% dos defeitos encontrados na interface, como por exemplo, o posicionamento de uma tabela ou botão que no Firefox e no Internet Explorer apareciam com alinhamentos diferentes, o mesmo com o posicionamento de textos, figuras e tamanho de letras. Estes defeitos eram identificados, registrados e repassados à equipe de desenvolvimento que tratava de acordo com a prioridade da entrega, mas sempre em algum momento foram corrigidos.

Após a entrega do projeto, defeitos de interface e funcionalidades não foram encontrados pelo cliente que se mostrou satisfeito, sugerindo apenas melhorias e correções textuais nos e-mails e relatórios da aplicação.

Esta prática permitiu que a equipe ganhasse conhecimento tanto em ferramentas quanto em processo de teste e tornou viável a execução deste processo para os projetos posteriores.

5. Conclusão

Neste artigo, foi apresentado o Processo de Teste Automatizado adequado a realidade da empresa e ao projeto. Para que pudéssemos garantir o mínimo de qualidade deste, foi adotada a abordagem de automação das atividades de teste que colaborou para satisfazer os requisitos do sistema e a execução de todas as etapas do processo de teste sem adição de tempo e custo.

Ao longo da experiência, observamos algumas particularidades da metodologia que tomamos como lições aprendidas. Primeiro o fato de, no *Scrum* parte das funcionalidades são feitas separadamente (entre *Sprints*), assim alguns scripts tiveram que ser atualizados e retrabalhados a cada mudança de requisito. Em segundo, como as estimativas são feitas no planejamento do *Sprint*, observamos que o esforço maior de tempo era apenas na primeira iteração, nas seguintes a execução do processo de teste se tornou mais rápida, devido ao reaproveitamento de scripts de teste e execução automática. Outra lição foi a ocorrência de muitos defeitos encontrados nas rodadas de teste de regressão, fazendo com que a estimativa de tempo de correção de código definida pelo desenvolvedor não fosse suficiente e defeitos encontrados tiveram que ser postergados de acordo com a prioridade da entrega.

Contudo, a experiência mostrou bons resultados nos permitindo adquirir conhecimento não só em ferramentas de teste como no benefício que o processo de teste trouxe ao projeto, nos levando a pesquisa de novas ferramentas de automação que atendam a outras plataformas de desenvolvimento para posteriores projetos na empresa.

Agradecimentos

Os autores agradecem ao Instituto Nokia de Tecnologia, a todos os profissionais envolvidos no projeto OCSS e a Professora Tayana Conte da Universidade Federal do Amazonas por suas contribuições na revisão deste artigo.

Referências

Bernardo, P. e Kon, F. (2008) “A Importância dos Testes Automatizados”, http://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf, Outubro.

Bissi, W. (2007) “Scrum – Metodologia de Desenvolvimento Ágil”, http://revista.grupointegrado.br/revista/index.php/campodigital/article/view/312/146, Outubro.

Correia , S. e Silva, A. (2004) “Técnicas para Construção de Testes Funcionais Automáticos”, http://paginas.fe.up.pt/~jpf/teach/TQS0405/sc-Quatic2004.pdf, Novembro.

Costa, M. (2006) “Estratégia de Automação em Testes : requisitos, arquitetura e acompanhamento de sua implantação”, http://libdigi.unicamp.br/document/?down=vtls000389004, Novembro.

Dasso, A. and Funes, A. (2007) Verification, Validation and Testing in Software Engineering. Idea Group, 2007.

Leal, I. (2009) “Requisitos de Metodologias de Teste de Software para Processos Ágeis”, http://homepages.dcc.ufmg.br/~rodolfo/dcc823-1-09/Entrega2Pos/igor2.pdf, Outubro.

Mantis Testing Tool. (2009) “The MantisBT Manual (v1.2.x)”, http://www.mantisbt.org/, Abril.

OpenSource Functional Testing Tools (2009) “OpenSource Functional Testing Tools – News and Discussion”, http://www.opensourcetesting.org/functional.php, Outubro.

Presman, R. S. (1995) Engenharia de Software. São Paulo: Makron Books do Brasil, 1995.

Selenium Testing Tool (2009) “Selenium Documentation. Selenium HQ - Web Application Testing System”, http://seleniumhq.org, Março.

Schwaber, K. (2004) Agile Project Management with Scrum. Microsoft Press, 2004.

TestLink Testing Tool (2009) “TestLink Documentation. TEAMST - Home of TestLink developers Community”, http://www.teamst.org/, Novembro.

Ruby on Rails Documentation. (2009) “Ruby on Rails – Ruby on Rails Guides”, http://guides.rubyonrails.org/, Novembro.

Tavares, A. (2008) “Gerência de Projeto com PMBOK e SCRUM – Um Estudo de Caso”, http://pessoal.facensa.com.br/sidnei/files/TCCI-EmAndamento/aleckssandrotavares.pdf, Novembro.

Tuschling, O. (2008) “Software Test Automation”, http://www.stickyminds.com/getfile.asp?ot=XML&id=14908&fn=XDD14908filelistfilename1%2Epdf , Outubro.

Fantinato, M. et al. (2009) “AutoTest – Um Framework Reutilizável para a Automação de Teste Funcional de Software”, http://www.sbc.org.br/bibliotecadigital/download.php?paper=255, Outubro.

Vercauteren, T. (2009) “Agile development and functional testing: friend or foe?”, http://www.stickyminds.com/getfile.asp?ot=XML&id=15206&fn=XDD15206filelistfilename1%2Epdf, Outubro.

Wilson, G. (2009) The reality of software testing in an Agile Environment Testing Experience - The Magazine for Professional Testers 03/2009 (pág. 94 a 96).