Por que automatizar testes de software com Ruby?

Em projetos da área de Tecnologia da Informação desenvolvidos nos dias atuais, é difícil (e, muitas vezes, inviável) pensar em um processo de desenvolvimento de software que não inclua testes automatizados. Contudo, o leitor atento pode estar se perguntando: mas sobre qual tipo de teste de software estamos falando? Há diversos tipos e estes se subdividem basicamente em Testes Funcionais e Não Funcionais.

Falando inicialmente sobre testes não funcionais, eu gostaria de citar Testes de Performance. Quando nos referimos a Testes de Performance (nos quais desejamos simular carga sobre o ambiente da solução), a automação sempre foi imprescindível. Há vários motivos para isto e só não vou detalhá-los neste texto para não fugir do assunto principal que pretendo abordar. Era comum, há alguns anos, existir nas empresas a execução automatizada dos Testes de Performance e a execução manual dos Testes Funcionais. A automação dos Testes Funcionais era considerada muito trabalhosa e seu retorno financeiro era duvidoso. Eu comecei a trabalhar com Testes de Performance em 2006 e lembro-me bem de ter vivido esta realidade durante alguns anos.

O crescimento da adoção de práticas de DevOps e de desenvolvimento ágil nas empresas, além das necessidades de Integração Contínua (Continous Integration) e de Entrega Contínua (Continuous Delivery), causaram grandes mudanças em projetos de desenvolvimento de software. É neste contexto que a automação dos Testes Funcionais ganhou força e hoje ela é largamente utilizada pelas empresas. Não dá mais para depender apenas do processo manual de execução de Testes de Software, principalmente quando estamos nos referindo a Testes de Regressão.

Em paralelo a esta transformação, as ferramentas e bibliotecas voltadas para automação de Testes Funcionais evoluíram muito. Nos dias atuais, estas ferramentas e bibliotecas oferecem diversos recursos que resolvem vários tipos de problemas e não estou me referindo apenas à aplicações web. Penso que está cada vez mais difícil a realidade do Analista de Teste que escolheu se especializar em Testes de Software por não gostar de desenvolver código. Para comprovar este fato, basta verificar os requisitos que temos atualmente na maioria das vagas de trabalho atualmente disponíveis e que são voltadas à profissionais da área de Testes de Software.

A maioria dos profissionais de Testes de Software já percebeu esta tendência e está cada vez mais engajada em aperfeiçoar sua qualificações no que diz respeito a automação de Testes Funcionais. Neste processo de aperfeiçoamento como profissional, muitas vezes se busca o conhecimento de ferramentas e bibliotecas baseadas em linguagens de programação muito populares (como Java) e o Selenium WebDriver acaba sendo uma das primeiras escolhas.

Aprender Selenium WebDriver é fundamental, não discuto isto. Trata-se de uma biblioteca que possui versões voltadas a diversas linguagens de programação e Java é

uma destas opções. Há muito código de testes automatizados já desenvolvido com Selenium WebDriver e é fundamental ao Analista de Teste estar qualificado para trabalhar em projetos que envolvam a escrita de código usando esta biblioteca.

Contudo, hoje há alternativas (inclusive gratuitas) que podemos usar para automatizar testes funcionais de software que devem ser consideradas pelos profissionais e lideranças da área. Eu gostaria de abordar o uso da linguagem Ruby para este objetivo. Segundo a Wikipedia: "Ruby é uma linguagem de programação interpretada e multiparadigma, de tipagem dinâmica e forte, com gerenciamento de memória automático, originalmente planejada e desenvolvida no Japão, em 1995, por Yukihiro Matsumoto, para ser usada como linguagem de script. Matsumoto queria uma linguagem de script que fosse mais poderosa do que Perl e mais orientada a objetos do que Python".

A definição acima é interessante, porém o que Ruby tem que a torna uma opção valiosa quando falamos de testes automatizados de software? O uso da linguagem Ruby para automatizar testes de software torna o processo mais eficiente e mais produtivo sem sacrificar a eficácia. Desta forma, conseguimos trabalhar, ao mesmo tempo, com foco no problema e na solução, construindo rapidamente código de excelente qualidade que resolve os problemas e é executado de forma estável. A sintaxe da linguagem Ruby é sucinta e, pelo menos no que se refere à elaboração de scripts de testes automatizados de software, possui recursos similares a linguagens de programação mais verbosas como Java e C#.

Ao falar do uso da linguagem Ruby no processo de automação de testes de software, é importante destacar o Capybara. O Capybara é um framework baseado no Selenium que facilita bastante o trabalho de criação de scripts de testes automatizados. Trata-se de uma Linguagem de Domínio Específico (DSL) voltada para a automação de testes funcionais de aplicações web. Ao trabalhar com Capybara, é possível desenvolver código de forma mais produtiva tanto pelo fato de trabalharmos com um framework do Selenium (o que encapsula diversos recursos desta biblioteca) como pelo fato de escrevermos código em Ruby (que, conforme já citado, possui sintaxe bem sucinta e que possibilita fazer mais com menos código).

Além das vantagens já citadas, é importante mencionar que é possível trabalhar com Capybara integrado ao Cucumber (gerando os códigos das “steps definitions” em Ruby). Desta forma, mesmo em projetos que usam o Desenvolvimento Orientado por Comportamento (BDD), podemos usufruir dos benefícios oferecidos pelo Capybara para automatizar as especificações (o Cucumber é voltado para automação de especificações e não especificamente para automação de testes).

Lembra do padrão de projeto que é utilizado por qualquer projeto de automação de testes de software que se preza? Sim, estou falando do padrão Page Object. Também é possível criar as classes referentes aos Page Objects usando código Ruby e e chamar estas classes em código do Capybara que implementa ou não “step definitions” (ou seja, refiro-me ao uso do Capybara integrado ou não ao Cucumber).

E como fica a Pirâmide de Testes? Quando pensamos em um processo completo de automação de Testes de Software, devemos considerar esta pirâmide e automatizar Testes Unitários, Testes de API e Testes Baseados na Interface do Usuário.

Para Testes Unitários, temos a opção do RSpec, sendo este o corresponde ao JUnit se considerarmos a plataforma Java. O RSpec está para o Capybara assim como o JUnit (ou o TestNG) está para o Selenium WebDriver para Java. Desta forma, o RSpec pode ser utilizado em Testes Unitários e também como uma ferramenta que trabalha integrada ao Capybara para determinar trechos de código que devem ser executados antes ou depois de cada caso de teste.

Ainda falando sobre a Piramide de Testes, podemos usar a biblioteca HTTParty para escrever código Ruby que automatiza testes funcionais de APIs. Novamente, temos aqui o benefício do ganho de produtividade por trabalharmos com Ruby.

Em qualquer nível da Pirâmide de Testes, é possível gerar relatórios em vários formatos diferentes contendo os resultados das execuções dos testes automatizados.

A construção de "scripts independentes" é uma boa prática que sempre deve ser considerada quando estamos construindo scripts de testes automatizados. "Scripts independentes" podem ser executados a qualquer momento, sem depender de precondições que, muitas vezes, referem-se à preparação do ambiente de testes. Um "script independente" consegue, por exemplo, sempre excluir um registro de uma tabela do Banco de Dados, pois ele sempre garante a presença deste registro nesta tabela antes de sua remoção. Quando trabalhamos com Ruby, podemos baixar várias "gems" (nome que se dá às dependências quando se trabalha com Ruby) que nos auxiliam na construção de "scripts independentes". Como exemplo, podemos citar a gem "pg" que permite escrever código para se conectar a um Banco de Dados PostgreSQL e executar uma instrução SQL antes da execução do script de teste automatizado.

Temos atualmente diversas ferramentas e bibliotecas para realizar automação de testes funcionais de software. Conforme relatado nos parágrafos iniciais deste texto, esta automação deixou de ser opcional nos contextos de vários projetos que temos hoje na área de Tecnologia da Informação. O uso da linguagem Ruby na construção de scripts de testes automatizados oferece as vantagens descritas neste artigo (além de outras que não foram detalhadas aqui) e não possui uma curva de aprendizado desfavorável quando o profissional em questão já tem sólidos conhecimentos de orientação a objetos e de conceitos gerais relativos à programação de computadores. No seu próximo projeto de software, considere automatizar os testes funcionais usando Ruby (Capybara, RSpec, HTTParty) e usufrua de seus benefícios!

Raphael Mantilha
Senior Automated/Performance Test Analyst at Cast group