Testes de API para um framework de backend

Gerson P. Vieira, Rodrigo A. V. Miranda

Departamento de Ciências Exatas

Universidade Federal da Paraíba (UFPB)

Av. Santa Elisabete, 160, Rio Tinto - PB, 58297-000

gerson.pires@dcx.ufpb.br

Abstract. *Software testing is the error search process through the execution of computer systems. The purpose of the tests is not to guarantee the absence of errors, but to minimize defects, help with project costs and increase the quality of the system. This article aims to implement and evaluate a battery of tests for a backend framework with implementation on Spring Boot and .NET implementations of the same application and ensuring that both meet the same requirements. Due to not being able to test a framework directly there was a need to implement an application using the framework in Spring Boot before testing the endpoints together with the code coverage of the framework. To ensure the validity of the tests, the same .NET application was implemented, where bugs and inconsistencies were found between the way the frameworks worked.*

Resumo. *Teste de software é o processo de procura por falhas ao executar sistemas computacionais com a finalidade de minimizar defeitos, reduzir os custos do projeto e aumentar a qualidade do sistema. Este artigo tem como objetivo, implementar, executar e avaliar uma bateria de testes para um [1] framework de backend com implementações em Spring Boot e .NET de uma mesma aplicação e garantir que ambas atendem os mesmo requisitos. Devido a não ter como testar um framework diretamente implementou-se uma aplicação utilizando o framework Spring Boot para então testar os endpoints em conjunto com a cobertura de código do framework. Para garantir a validade dos testes foi implementada a mesma aplicação em .NET onde encontrou-se bugs e inconsistências entre os modos em que os frameworks funcionavam.*

Palavras chave: Testes automatizados; Qualidade; Reuso de Software; Framework.

[1] Trabalho de conclusão de curso, sob orientação do professor Rodrigo de Almeida Vilar de Miranda submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO.

1. Introdução

Reusabilidade de software é o processo de criação de sistemas de software a partir de software existente em vez de criá-los a partir do zero (Krueger, 1992). Frakes e Fox (2002) explicam que reusabilidade é muitas vezes confundida com portabilidade, fazendo assim uma diferenciação entre os dois conceitos e mostrando que *reusabilidade* é o uso de ativos de uma aplicação em outra ou em si mesma e *portabilidade* é mudar uma aplicação para um ambiente ou plataforma diferente.

Conceitos de reutilização de código apareceram por volta dos anos 60, porém só receberam atenção do mercado e indústria nos anos 80 devido à exigência de mais eficiência em seus processos. No ano de 1980 foi iniciado o primeiro projeto de pesquisa na área de reutilização de código pela Universidade da Califórnia, tendo como coordenador o professor Peter Freeman no projeto *Reusable Software Engineering: A statement of long-range research objectives (1980)*. De acordo com Dresch A. apud Ezran et al. (2002), estudos de reutilização de software mostram que: i) 40% a 60% do código pode ser utilizado de um sistema para o outro; ii) 60% do projeto e código são reutilizáveis em aplicações de negócios; iii) 75% das funções do programa são comuns a mais de um programa, e que, iv) apenas 15% do código na maioria dos sistemas é exclusivo e novo para uma aplicação específica.

A definição de reuso de software tem vários pontos de vista. Peter Freeman (apud Ezran et al. 2002) define reutilização como o uso de qualquer informação que um desenvolvedor pode precisar no processo de criação de software. Para Frakes & Isoda (1994) é o uso de artefatos de softwares existentes ou conhecimento para a criação de um novo software que promove o aumento de qualidade do software e produtividade. Ezran (2002) diz que a reutilização de software é a prática sistemática de desenvolver software a partir de um estoque de

blocos de construção, de modo que as semelhanças de requisitos e arquitetura entre os aplicativos possam ser exploradas para obter benefícios substanciais em produtividade, qualidade e desempenho de negócios.

O Virtus[2] tem como uma das suas funções produzir sistemas de informação web junto aos projetos de P&D que realiza para indústrias nacionais e internacionais. A partir dos conceitos de reuso de código e de seus benefícios, no que diz respeito à produtividade da instituição, foi criado um *framework* chamado **Core** para fomentar o reúso das funções mais comuns em sistemas de informação web.

O Core é um *framework* que contém os componentes mais comuns tanto no *backend* como no *frontend* e aborda os seguintes tópicos: segurança entre cliente/servidor, integração com o banco de dados (CRUD e migração), versionamento de API, relacionamento entre as entidades do sistema, tratamento de erros e internacionalização. O *framework* foi lançado com uma versão para *backend* em Java e *frontend* em Angular e, depois de um tempo, foram criadas outras versões com *backend* em .NET e *frontend* em React Native.

Em experimentos internos, as funcionalidades que foram implementadas a partir do Core apresentaram uma produtividade uma ordem de grandeza maior do que as mesmas que eram feitas sem esse *framework*. Desse modo, ele passou a ser utilizado em diversos projetos do Virtus com seus parceiros. Inclusive um dos pontos da Capacitação que o Virtus realiza junto aos alunos da UFPB - Campus IV é justamente o Core nas versões Java e Angular, onde os alunos de graduação estudam essas versões do Core e as utilizam para desenvolver projetos.

Devido à criação de novas especificações do Core surgiram questões relacionadas à legitimidade das novas implementações do *framework*:

- Como garantir igualdade de funcionamento entre as implementações do Core, sejam elas entre diferentes versões de uma tecnologia ou entre tecnologias diferentes?

[2] O Virtus é um Núcleo de Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação, Comunicação e Automação da UFCG - https://www.virtus.ufcg.edu.br

- Como identificar possíveis erros antes de liberar o uso de alguma implementação do Core para os projetos?

Em um mercado tão competitivo é imprescindível que os sistemas desenvolvidos pelo Virtus sejam entregues no mais pleno funcionamento. Nesse contexto tem-se a importância dos testes de software.

- Os testes de software reduzem os prejuízos financeiros, pois o produto chega ao cliente com menos defeitos e evitam o retrabalho pela equipe de desenvolvimento.

Atualmente o Core não possui testes automatizados para validar todas as suas funções. E isso gera certos dilemas tanto na criação de novas abordagens do Core quanto em uma evolução de uma abordagem existente.

Tendo em vista a importância da testagem, o Core necessita de uma abordagem de teste que garanta que suas novas versões funcionem de forma adequada e que futuras alterações não comprometerão as versões existentes.

Todavia a realização de testes para um *framework* não é uma tarefa trivial, pois os *frameworks* por si sós não são executáveis nem testáveis diretamente. Em vez disso, é preciso criar uma ou mais aplicações que em teoria exercitem todo o código do *framework* e realizar testes para a aplicação, observando se os testes cobrem de fato todo o *framework* e não apenas a aplicação.

Este trabalho tem como objetivo projetar, implementar e validar uma bateria de testes para a especificação em *backend* do *framework* Core Virtus com os seguintes objetivos específicos:

1. Implementar uma aplicação base que consiga utilizar de todas as funções do Core como CRUD, paginação, segurança, busca e tratamento de erros;
2. Testar todos os end-points criados pela aplicação e certificar que funcionam de forma adequada. Para isso será necessária a criação de uma bateria de testes automatizados baseada na API REST da aplicação base;
3. Aferir a cobertura do código para garantir que 100% do Core foi coberto pela bateria de testes além de identificar possíveis erros de implementação, códigos inúteis e omissões. Caso a cobertura esteja incompleta, retornar

aos passos 1 e 2 para criar mais funcionalidades na aplicação base e seus respectivos testes;

4. Avaliar se a bateria de testes criada é válida para outras implementações de API do Core, a saber a versão .NET, para a qual será criada uma aplicação base semelhante a da versão em Java.

Na Seção 2, será apresentada a fundamentação teórica que embasou esta pesquisa. A Seção 3 contempla a metodologia que foi utilizada para efetuar as atividades. A Seção 4 contém os resultados obtidos. E a Seção 5 apresenta as conclusões deste trabalho.

2.Fundamentação teórica

Nesta Seção, são apresentados os conceitos de sistemas de informação, *frameworks*, testes para esses sistemas e o ciclo de vida dos testes.

Sistema de informação, como o nome sugere, é um sistema onde o objeto principal é a informação. Segundo Sequeira (2001), três atividades distintas são fundamentais para um sistema de informação:

- Receber dados de fontes internas ou externas.
- Processar os dados recebidos de acordo com critérios preestabelecidos.
- Emitir a informação de forma adequada para as necessidades de seus destinatários.

Diversos tipos de SIs são utilizados nas organizações, entre eles estão os sistemas rotineiros que tem a função de processar os dados, armazenar, ordenar e mostrar de forma simples, os SIGs (Sistemas de Informações Gerenciais) ou MIS (*Management Information Systems*) que tem como propósito facilitar gerentes em sua funções além de prover diagnósticos para tomadas de decisões, e os Sistemas de Informações Geográficas ou GIS (*Geographic Information Systems*) que são objetivados em integrar mapas, processar distâncias e localidades.

No desenvolvimento de sistemas de informação é bastante comum o uso de *frameworks*. Johnson (1997) diz que *framework* é uma técnica de reuso de

código onde compartilha de várias técnicas de reuso em geral e reuso em orientação a objetos em particular. Johnson (1997) , além disso, os define de duas formas, a primeira como um *design* reutilizável de todo ou parte de um sistema que é determinado por um conjunto de classes abstratas e o modo que suas instâncias interagem entre si. Na segunda definição, diz que um *framework* é o esqueleto de uma aplicação que pode ser customizado pelo desenvolvedor.

Ao se criar um sistema de informação, uma das partes fundamentais do processo é o teste de software que segundo (Silva, 2016) ".. é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado". O processo de testagem é bem definido e tem um ciclo de vida de seis etapas, definidas por Silva (2016) da seguinte forma :

- **Procedimentos iniciais**: Inicia-se um aprofundamento nos requisitos de negócio;
- **Planejamento**: Elaboração do Plano de Teste e a Estratégia de Teste;
- **Preparação**: Preparar o ambiente para a execução correta dos testes;
- **Especificação**:Elaboração e revisão dos casos de testes e roteiros de teste;
- **Execução**: Os testes são executados conforme descrito nos casos de teste e roteiros de teste;
- **Entrega**: Nessa etapa o software é entregue após passar por todas as etapas, reduzindo as chances de eventuais erros (bugs).

3.Metodologia

Nessa seção, será apresentada a metodologia escolhida para guiar a construção deste trabalho.

Toda a parte de metodologia seguirá o padrão de *design science research* que tem como estrutura um ciclo regulador (Figura 1), método de pesquisa que foi formalizado por Eekels e Roozenburg (1991), e o ciclo para resolução de problemas de Van Aken, Berends e Van der Bij (2012).

Figura 1: Design cycle proposto por Eekels e Roozenburg (1991, p. 199, apud apud Dresch A., 2015)

O ciclo para resolução de problemas (Figura 2) é dividido em 5 etapas. A primeira etapa consiste em compreender e definir o problema a ser resolvido. A etapa seguinte tem a finalidade de analisar e diagnosticar o problema para seguir para a próxima etapa que se trata de projetar a solução para a problemática. Na próxima etapa executa a intervenção que é a fase de implementação da solução proposta e por fim serão avaliados os resultados da proposta de solução e as

aprendizagens geradas pelo ciclo que poderão servir de base para o surgimento de novos problemas, assim iniciando um novo ciclo.

Figura 2: Ciclo para resolução de problemas (Dresch A., 2015).

Eekels e Roozenburg resumem o ciclo em 5 etapas distintas que serão a linha de base para os processos realizados neste trabalho.

Na primeira etapa será observada a problemática do Core não possuir testes para assim definir quais serão os objetivos que se deseja alcançar.

A segunda fase do ciclo é a parte da análise, onde analisa-se a situação original e as possíveis soluções para o problema, sempre buscando melhorias. Foram analisadas as ferramentas e abordagens de teste que poderiam ser utilizadas para permitir a solução do problema levando em conta que se deve garantir o teste de várias implementações do Core, portanto tem-se a necessidade de que a bateria de testes seja realizada independente de tecnologia e para isso será testada apenas a API REST e assim ficará independente da plataforma utilizada pelo *backend*.

A síntese é a terceira parte do ciclo, onde foi visualizada toda a situação a ser resolvida ou melhorada. Ao final dessa etapa foi obtida uma proposta para solucionar o problema. Nessa etapa, inicialmente, foi decidido pela criação de uma aplicação (API Rest), desenvolvida em Java, devido ao Core ser um *framework* e não haver como testá-lo sem essa abordagem.

A quarta etapa foi justamente a parte de implementação onde a bateria de testes foi implementada e executada.

Na quinta etapa efetuou-se a avaliação dos resultados obtidos na intervenção, verificando se atendiam aos requisitos propostos anteriormente. Aqui foi realizada a coleta da cobertura de código e de possíveis correções para o *framework*.

Após cada um desses ciclos, verificou-se o valor dos resultados e foi decidido se a solução era suficiente. Caso a cobertura do código e os testes não fossem satisfatórios o ciclo seria reiniciado, voltando para a parte de análise do problema atual. Sendo assim, a bateria de testes foi implementada de forma incremental e regulada seguindo o ciclo proposto por Eekels e Roozenburg (1991).

4. Resultados

Nessa Seção, serão apresentados os resultados obtidos no trabalho como um todo.

Após a escolha do *IntelliJ* como *IDE* junto ao *Jacoco* para aferir a cobertura de código do Core, devido a simplicidade na forma de usar e coletar resultados que essas tecnologias proporcionam, e o *REST Assured* para os testes de API, iniciou-se o primeiro ciclo de implementação do aplicativo.

O **teste inicial** para averiguar o funcionamento da cobertura de código foi o de autenticação com a API, dado que o Core fornece uma função autenticação para esse fim. Após a execução do [3]teste, foi verificado (Figura 4) que a cobertura

[3] O *Jacoco* é uma biblioteca usada para cobertura de código - https://www.jacoco.org/
O IntelliJ é uma IDE da JET BRAINS - https://www.jetbrains.com/

de código do Core estava em 35% das linhas de código, 24% dos métodos e 61% das classes. Um resultado ainda bem baixo, portanto um novo ciclo foi iniciado.

No início do **segundo ciclo** de testes, foi estabelecido que seria feita a testagem dos *endpoints* do CRUD, testando assim a inserção, atualização e deleção de um elemento na API. Para que fosse possível implementar esses testes deu-se início à criação da aplicação base. Assim como mostra o UML da aplicação (Figura 3), a aplicação é constituída de Modelo chamado Carro, O DTO (Data Transfer Object) do Modelo, o repositório encarregado das ações com o banco de dados, o *Service* para tratar os requisitos de negócio e por fim o *Controller* responsável por tratar as requisições REST dos clientes.

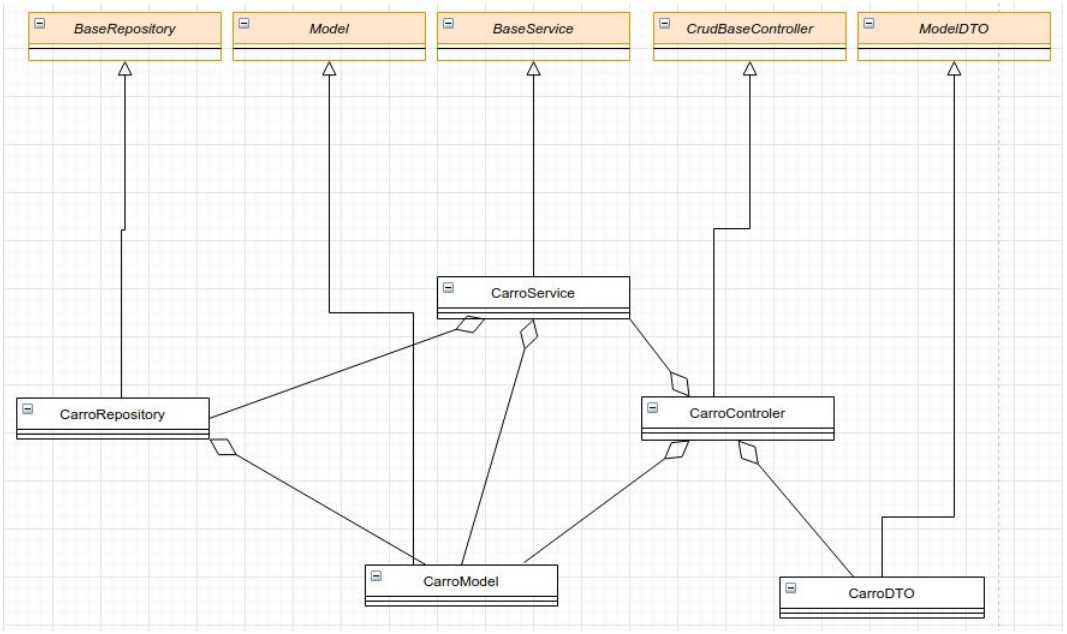

Figura 3

Diagrama da aplicação base

(Em branco as classes da aplicação e em amarelo as classes do Core)

Seguindo a implementação da aplicação base, foram criados e executados os testes. Verificou-se que a cobertura subiu de 35% para 71% por linha, 24%

O REST Assured é uma biblioteca para testagem e validação de serviços REST - https://rest-assured.io/

para 70% por método e de 61% para 86% por classe (Figura 4), mas ainda estava longe do objetivo deste trabalho.

No **terceiro ciclo** dos testes foram criados *endpoints* na API para verificação do tratamento de exceções. Nessa etapa foram criados os testes para as exceções e autenticação com credenciais incorretas. Foi registrado pela cobertura de código um aumento de 71% para 79% por linha, 70% para 80% por método e 86% para 95% por classe (Figura 4).

O **quarto ciclo** alcançou o nível de cobertura de código em 97% por classe, 94% por método e 91% por linha (Figura 4).Os testes criados nessa fase foram para utilizar de deleções e buscas personalizadas, assim como os testes para a autenticação e recuperação do token.

Nesta parte foi descoberta uma dupla utilização de captura de exceção em um fluxo do sistema sendo necessário apenas um único tratamento de exceção. Esse problema foi reportado para o time de arquitetura de software do Virtus.

No quinto e **último ciclo** de testes, foi necessário percorrer fluxos alternativos do *framework*, porém as linhas não cobertas restantes se tratavam de exceções de REST que não foram possíveis de executar, construtores privados que lançavam apenas exceções e fluxos de exceções muito específicos. Desse modo, decidiu-se encerrar a bateria de testes nesse ciclo, trazendo aumento apenas de 91% para 93% na cobertura por linha.

Como produto final, a bateria de testes realizou uma cobertura muito alta no código do Core, sendo portanto uma ferramenta significante para validá-lo, nas implementações atuais e futuras.

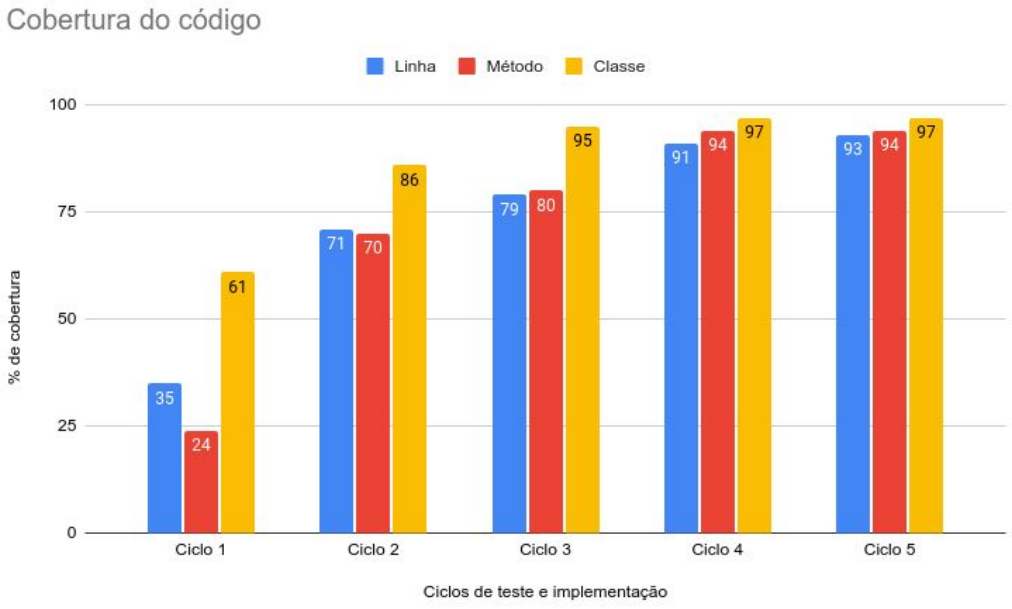

Figura 4

Evolução da cobertura de código a cada ciclo

Bugs, implementação e validação da bateria de testes na API .NET

Após a implementação satisfatória da bateria de testes, iniciou-se a implementação da aplicação (API) em .NET.

Primeiramente foi feita a testagem do *login* na API. Nessa parte, foi identificado que a versão .NET do Core tem por configuração padrão um redirecionamento para uma URL HTTPS na qual o REST Assured não conseguia seguir tal redirecionamento. Neste momento, para a continuação da testagem, foi de suma importância a retirada do redirecionamento de URL. Na testagem de credenciais erradas para login foi encontrada uma divergência no erro enviado das duas APIs.

Na implementação do CRUD da aplicação .NET verificou-se que havia um erro na hora de montar o token da aplicação. A falha consistia na string “Bearer” que estava sendo duplicada o que ocasionava em erro na aplicação .NET. Porém, a aplicação Java aceitava o token. Após a resolução do erro da

bateria de testes foram aprovados os testes de CRUD, mostrando que as duas implementações foram feitas de forma correta.

Implementar os fluxos alternativos foi um grande desafio devido a discrepâncias em algumas formas que cada aplicação resolvia determinado problema e a falta de familiaridade com o .NET, mas os fluxos alternativos foram implementados com sucesso e os testes passaram.

5.Conclusões

A abordagem utilizada a fim de solucionar o problema da testagem se mostrou eficiente para a padronização e validação das diferentes abordagens do *framework* Core, uma vez que a bateria de testes é criada em cima de uma implementação validada e testada.

Os testes de cobertura com o Jacoco + IntelliJ no desenvolvimento da API em Java junto com a bateria de testes em REST Assured foi a combinação que garantiu a avaliação tanto interna quanto externa da API e a validade da bateria de testes para ser utilizada na mesma implementação da API em .NET.

Na etapa de testagem da API em .NET com a bateria de testes, foram encontrados bugs e incoerências entre as implementações do Core .NET e Core Java, confirmando o valor da abordagem proposta neste trabalho.

Como sugestões de trabalhos futuros, tem-se:

- Fazer uma bateria de testes de carga e *stress* seguindo a mesma abordagem deste trabalho e que seja independente de tecnologia;
- Evoluir a bateria de testes para novas funcionalidades que forem incorporadas ao core.

Referências

Dresch A., Lacerda D.P., Antunes J.A.V., “Design Science Research: Método de Pesquisa para Avanço da Ciência e Tecnologia”, Bookman Editora, páginas 204, 2015.

Silva J.S., Viana G.B., Machado G.B.G., Silva R.O., “O processo de teste de software”,Tecnologias em Projeção, volume 7, número 2, 2016, páginas 99-108.

Sequeira B.D., “Influências e Efeitos dos Sistemas de Informação e Tecnologias de Informação no Desempenho Profissional”, Serviços Administrativos da Universidade de Évora, 2001.

Batista O.E., “Sistemas de Informação: O uso consciente da tecnologia para o gerenciamento”, Editora Saraiva, 2ª edição, Páginas 369, 2017.

Krueger W.C., “Software Reuse”, ACM Computing Surveys, 1992.

Ramos F. G., “Reusabilidade em SOA: Um Mapeamento Sistemático da Literatura”, 2014.

Frakes W., Terry C., “Software reuse: metrics and models”, ACM Computing Surveys, 1992.

Freeman P., “Reusable Software Engineering: A Statement of Long-Range Research Objectives”, UC Irvine, 1980.

Johnson E.R., “Frameworks = (components + patterns)”, Communications of the ACM, 1997.

Frakes W. B., Fox C.J., “Sixteen Questions About Software Reuse”, Communications of the ACM, 1995.

Morisio M., Erzan M., Tully C., “Success and Failure Factors in Software Reuse”, IEEE Transactions on Software Engineering, 2002