INTERFACE TECNOLÓGICA

UM ESTUDO DE CASO SOBRE O DESENVOLVIMENTO DE UMA APLICAÇÃO SCRUM E DEVOPS

CASE STUDY ON THE DEVELOPMENT OF A SCRUM AND DEVOPS APPLICATION

Hygor Podgornik – hppodgornik@gmail.com
Faculdade de Tecnologia (Fatec) – Taquaritinga – SP – Brasil

Daniela Gibertoni – daniela.gibertoni@fatectq.edu.br
Faculdade de Tecnologia (Fatec) – Taquaritinga – SP – Brasil

DOI: 10.31510/infa.v17i2.978
Data de publicação: 18/12/2020

RESUMO

O mercado muda constantemente para atender as necessidades dos usuários, e constantemente a necessidade dos usuários muda e com isso, nenhum software terá os requisitos muito bem definidos logo ao início do projeto, pois eles estão em constante mudança. Dessa forma, torna-se necessário que o desenvolvimento de software não fique engessado em metodologias de desenvolvimento que não permitam tais mudanças. Para sanar esse problema, surgiram as metodologias de desenvolvimento ágil, que permitem com que os times sejam autogerenciáveis e auto adaptáveis às necessidades, tornando as mudanças menos custosas. Porém, o desenvolvimento é uma parte do processo, e ele ainda depende do time de operações para que as aplicações sejam publicadas corretamente, e o time de operações também sentiu a necessidade de tornar a infraestrutura ágil. Com isso, surgiu o DevOps, uma cooperação entre o time de desenvolvimento e o time de operações, para que não apenas o time de desenvolvimento fosse ágil ao desenvolver as necessidades, mas o time de operações também pudesse ser ágil na hora de entregá-las ao usuário final, de forma que ainda haja uma qualidade no produto que está sendo entregue. Este artigo aborda, por meio da pesquisa bibliográfica exploratória experimental, a experiência de um time scrum acadêmico aplicando a cultura DevOps durante o desenvolvimento de uma aplicação para uma instituição filantrópica. Como principais resultados obtidos, destacam-se uma maior qualidade do produto publicado e uma motivação maior da equipe ao ver a cada semana que seu trabalho estava sendo publicado na internet.

Palavras-chave: Desenvolvimento Ágil. Scrum. DevOps. Aplicações Web. Operações.

ABSTRACT

The market constantly changes to meet the needs of users, and constantly the needs of users change and with this, no software will have the requirements very well defined right at the beginning of the project, as they are constantly changing. Thus, it becomes necessary that software development is not limited to development methodologies that do not allow such

changes. To solve this problem, agile development methodologies have emerged, which allow teams to be self-managing and self-adaptable to needs, making changes less costly. However, development is a part of the process, and it still depends on the operations team for applications to be published correctly, and the operations team also felt the need to make the infrastructure agile. With that, DevOps emerged, a cooperation between the development team and the operations team, so that not only was the development team agile in developing needs, but the operations team could also be agile when delivering them to the end user, so that there is still a quality in the product being delivered. This article addresses, through experimental exploratory bibliographic research, the experience of an academic scrum team applying the DevOps culture during the development of an application for a philanthropic institution. As main results obtained, we highlight a higher quality of the published product and a greater motivation of the team to see every week that their work was being published on the internet.

Keywords: Agile Development. Scrum. DevOps. Web applications. Operations.

1 INTRODUÇÃO

De acordo com Pressman (2011), os métodos de desenvolvimento ágeis se desenvolveram para sanar pontos fracos dos métodos de desenvolvimento convencionais.

Atualmente, é muito difícil conseguir prever qual será a evolução de um sistema de computador. O mercado sofre constantes mudanças, as necessidades dos usuários também mudam conforme o tempo e novos competidores entram no mercado, por isso, é muito difícil que todos os requisitos de um projeto sejam definidos logo no início, portanto, é necessário ser ágil para se adaptar. No entanto, adaptação requer mudança, e mudanças sem controle e mal gerenciadas podem ser muito caras (PRESSMAN, 2011).

Na conferência Ágil, realizada em 2008 em Toronto, um administrador de sistemas, Patrick Debois, falou sobre incorporar o Scrum as operações, em uma palestra intitulada de "Agile Operations and Infrastructure: How Infra-gile are you?". Debois trabalhou em um projeto para migração de ambientes de um data center, e em um dia ele estava desenvolvendo alguma necessidade e no outro "apagando incêndios" juntamente a equipe de operações, e essa mudança de ambientes estava se tornando cansativa para ele. Debois se juntou a Andrew Clay Shafer, outro interessado no assunto de infraestrutura ágil, que havia inscrito uma palestra com esse assunto na conferência de 2008, para discutir sobre o assunto. Ao mesmo tempo, algumas empresas já davam passos para alcançar maior agilidade nas entregas de software, uma delas foi o Flickr, que em 2005 precisou migrar todos os seus dados do Canadá para os Estados Unidos. John Allspaw e Paul Hammond apresentaram em 2009, na

conferência Velocity Santa Clara, a mudança que eles obtiveram ao juntar o time de desenvolvimento com o time de operações, permitindo que a equipe chegasse ao objetivo mais rapidamente (DAVIS; DANIELS, 2016).

Dessa forma, Bass, Weber e Zhu (2015) definem que DevOps, de muitas maneiras, é uma resposta aos problemas de demora nas releases. Quanto mais tempo uma novidade demora para chegar ao mercado, menos resultados positivos trará para a empresa. Neste contexto, este artigo tem como objetivo investigar e identificar as principais características no uso da metodologia de desenvolvimento ágil Scrum e o uso da cultura DevOps, durante o desenvolvimento de uma aplicação web.

Para alcançar o objetivo definido, a metodologia de pesquisa utilizada neste artigo é a pesquisa bibliográfica exploratória e experimental. Dessa forma, foi realizada uma abordagem prática dos temas apresentados e da cultura DevOps em um time Scrum, sendo possível elaborar um estudo de caso durante o desenvolvimento de uma aplicação web para uma entidade filantrópica.

Este artigo foi estruturado em cinco seções, sendo a primeira delas a introdução, trazendo a contextualização da proposta da pesquisa realizada. Na seção dois é apresentada a fundamentação teórica contendo as informações sobre o tema central da pesquisa. Na seção três é apresentada a metodologia que foi utilizada para a realização da pesquisa. Na seção quatro é apresentado o estudo de caso, o desenvolvimento de uma aplicação web para uma entidade filantrópica com um time Scrum e a cultura DevOps. E por fim, na seção cinco, são apresentados os resultados obtidos com a pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados os conceitos referentes ao tema central da pesquisa deste artigo, o Scrum e a cultura DevOps, que foram utilizadas no estudo de caso em que foi desenvolvido uma aplicação web para uma entidade filantrópica.

2.1 Scrum

Segundo Schwaber e Sutherland (2013), o Scrum vem sendo utilizado para gerenciar o projeto de produtos complexos desde o início dos anos 1990. O Scrum trata-se de um

framework, onde dentro dele podem ser utilizados processos e técnicas que melhor se adaptarem para o desenvolvimento do produto. Ele enfatiza o uso de um conjunto de padrões de processos de software que provaram ser eficazes no desenvolvimento de projetos cujo prazos de entrega são apertados, os requisitos do software são mutáveis e críticos. Cada um desses padrões define um conjunto de ações a serem tomadas no desenvolvimento (PRESSMAN, 2011).

Segundo Schwaber (2004) problemas complexos são problemas que tem um comportamento imprevisível, e a maneira como ele se mostrará também será impossível de prever. Quando se tem problemas complexos precisamos aplicar um grau de precisão muito grande para resolvê-lo, e quando necessitamos de um grau de precisão grande, é importante guiar o processo para resolução do problema passo a passo, garantindo a precisão desejada. Caso o grau de precisão não seja alcançado, é necessário que se façam correções até que ele seja alcançado. Estabelecer um processo para produzir algo repetidamente com um resultado aceitável pode ser chamado de controle de processo definido. Quando o controle de processo definido não produz os resultados esperados, devemos empregar o controle de processo empírico. Nos processos de controle definidos é possível aumentar a produção, pois todo o processo de resolução do problema já foi definido anteriormente, porém no empirismo isso não é possível, pois ele afirma que o conhecimento sobre determinada situação é proveniente da experiência e resultados obtidos em uma fase anterior.

De acordo com Sabbagh (2013) o desenvolvimento dos softwares deve ocorrer de forma empírica. Nesta abordagem, aprendemos sobre o meio de produção que está sendo utilizado, como também sobre o produto que está sendo desenvolvido. Dessa forma, busca-se uma maximização de habilidades do time que está desenvolvendo o produto em se adaptar o mais rápido possível a mudanças, ao mesmo tempo que busca tornar-se mais produtivo.

O empirismo possui três pilares fundamentais, sendo eles: transparência: onde as fases do processo que afetam no resultado devem estar visíveis. Por exemplo, pode haver divergências no entendimento do que pode ser considerada uma funcionalidade como concluída. Nesse caso deve ficar explícito a toda equipe do projeto o que será entendido quando uma funcionalidade for marcada com esse status; inspeção: onde é feita uma inspeção para avaliar se o grau de precisão na resolução do problema está sendo alcançado durante o processo ou se é necessário que sejam feitas mudanças para que o resultado final seja como o esperado; e adaptação: onde o processo será adaptado de acordo com o resultado obtido na

inspeção. Se o inspetor avaliar que o processo não está surtindo o efeito esperado e o resultado não será aceitável, devem ser feitas adaptações no processo o mais rápido possível para adequá-lo ao grau de complexidade do problema e minimizar desvios futuros (SCHWABER, 2004).

Sendo assim, podemos afirmar que o Scrum é baseado em transparência, inspeção e adaptação, tanto no produto que está sendo desenvolvido, como nos processos que estão sendo utilizados para desenvolvê-lo, pois dessa forma é possível gerar o maior retorno possível para o cliente (SABBAGH, 2013).

Os times Scrum são compostos de três figuras que entregam produtos de forma iterativa e incremental, o que garante que uma versão potencialmente usável do produto estará disponível ao stakeholders. Eles são compostos pelo Product Owner (PO): é o dono do produto. Ele é o responsável por maximizar o valor do produto. O PO é responsável por gerenciar o backlog do produto e para que ele tenha sucesso, é necessário que toda a organização respeite as decisões tomadas por ele; Time de Desenvolvimento (TD): é responsável por realizar o trabalho para entregar uma versão usável que agregue valor ao produto que está sendo incrementado ao final de cada sprint. O TD é estruturado e organizado para ser auto gerenciável, além de possuírem todas as habilidades necessárias para criar o incremento do produto; e o Scrum Master (SM): é responsável por garantir com que as pessoas entendam e apliquem corretamente o scrum (SCHWABER; SUTHERLAND, 2013).

2.2 DevOps

Segundo Hüttermann (2012) DevOps é a mistura de desenvolvimento, representando desenvolvedores de software, e garantia de qualidade, e operações, representando quem coloca o software em infraestrutura de produção, incluindo administradores de sistema, administradores de banco de dados e técnicos de rede. DevOps descreve práticas para agilizar o processo de entrega de software, melhorando o tempo desde o início do ciclo até o final dele. Segundo Bass, Weber e Zhu (2015) DevOps é um conjunto de práticas destinadas a reduzir o tempo entre a confirmação de uma mudança em um sistema e a mudança ser colocada em produção, garantindo a alta qualidade.

A qualidade da mudança implantada em um sistema é muito importante, pois a mudança afetará todos os stakeholders. Uma das maneiras para assegurar a qualidade da

mudança, é realizar uma tomada de testes automatizados que devem ser aprovados antes de a mudança ser colocada em produção (BASS; WEBER; ZHU, 2015).

Hüttermann (2012) descreve quatro aspectos essenciais na prática DevOps, sendo eles: Cultura, priorizando pessoas a processos e ferramentas, pois softwares são feitos por pessoas e para pessoas; Automatização: como característica essencial para que a prática DevOps gere ganhos expressivos para a corporação; Medição: por meio dela é possível aferir a qualidade; Compartilhamento: DevOps cria uma cultura onde as pessoas compartilham ideias, processos e ferramentas.

Bass, Weber e Zhu (2015) identificam cinco práticas importantes para a cultura DevOps, sendo elas: Do ponto de vista de requisitos, integrar Ops ao time garantirá que requisitos de logs entendíveis sejam considerados; torne os devs mais responsáveis pelo tratamento de incidentes. Essa prática tem como objetivo encurtar o tempo em que uma falha for descoberta e é reparada; empregue o processo de implantação usado a todos. Essa prática tem o objetivo de prover alta qualidade de implantação já que em um processo normal de implantação, deve ser fácil de identificar o histórico do processo; faça uso da metodologia de implantação contínua. Ela encurta o tempo entre o código que o desenvolvedor escreve e o código implantado. A metodologia também enfatiza o uso de testes automatizados para garantir a qualidade da mudança; faça da infraestrutura um código. A adoção de scripts de implantação garante a qualidade das aplicações implantadas, e garantem que a implantação ocorra da forma como foi descrita. Scripts de implantação escritos de maneira incorreta podem causar erros na aplicação que está sendo implantada, portanto, é necessário aplicar práticas de controle de qualidade, que são aplicadas em softwares, a esses scripts de implantação, para garantir que as especificações dele estão corretas.

2.3 Controle de versão

Segundo Chacon e Straub (2020) os sistemas de controle de versão registram alterações em um arquivo ou em um conjunto de arquivos ao longo do tempo para que seja possível recuperar essas versões posteriormente. Esses arquivos podem ser desde o código fonte como outros documentos que façam parte de um projeto de software. Os desenvolvedores realizam mudanças nos arquivos e realizam commits. Os commits armazenam metadados como quem realizou a alteração, quando alterou, e porque alterou. Os

commits são feitos dentro do repositório do projeto de software, porém, geralmente, são feitos em outras branches, que são espécies de linha do tempo dos arquivos. Após feitos os commits, é possível que se mescle as branches a fim de juntar o que já existia no arquivo com as mudanças que foram feitas pelo desenvolvedor. O sistema de controle mescla os dois arquivos fazendo com que as mudanças que existiam na versão que foi alterada passem a fazer parte da branch que está sendo mesclada (DAVIS; DANIELS, 2016).

2.4 Integração contínua

A integração contínua (CI) é o processo de integrar novos códigos escritos pelos desenvolvedores a branch principal da aplicação frequentemente ao longo do dia. As ferramentas de CI quando recebem um pedido de integração pegam o código que está para ser integrado e realizam uma série de testes para garantir que aquele código novo que será admitido na branch principal não entrará em conflito com o que já existe, causando erros e quedas no serviço. Esse processo é realizado de forma automatizada, sem a necessidade de que uma pessoa os realize, evitando sobrecarga dos colaboradores. Essa prática é diferente de termos programadores trabalhando em cima de uma funcionalidade e só integrarem ela ao projeto ao final do desenvolvimento, pois funcionalidades grandes, que levam mais tempo para serem desenvolvidas, podem ter muitas mudanças de arquivos que podem levar a quebrar a aplicação quando for integrada, e com muitas mudanças realizadas pode ser difícil localizar qual foi o componente responsável por essa quebra. Com a integração contínua, como são feitas várias integrações ao longo do dia, as mudanças que são integradas são bem menores, sendo assim é mais fácil identificar pontos de quebra da aplicação (DAVIS; DANIELS, 2016). Segundo Swartout (2014) “contínua” indica o mais frequente possível, de preferência a cada mudança que ocorre. Caso algum teste falhe, a mudança não é integrada a aplicação evitando downtime.

2.5 Entrega contínua

A entrega contínua (CD) é um conjunto de princípios de engenharia de software que permite a entrega frequente de novas versões de uma aplicação, fazendo uso de testes automatizados e integração contínua. Está relacionada com a integração contínua, porém,

essa utiliza os testes automatizados para garantir que um novo bloco de código não quebrará o código atual quando for integrada, enquanto que em entrega continua os testes são realizados a fim de garantir que as mudanças não quebrarão o código atual e garantir que essa nova versão de código possa ser instalada no servidor para que seja utilizada pelos usuários (DAVIS; DANIELS, 2016).

2.6 Implantação contínua

A implantação contínua (CD) é o processo de implantar as mudanças no servidor de produção. Enquanto a entrega contínua se preocupa em deixar as mudanças testadas e validadas para serem implantadas no servidor de produção, a implantação contínua se preocupa em de fato instalá-las para produção. Quanto mais rápido esse processo for feito, mais rápido os desenvolvedores verão seu trabalho no ar para que os usuários possam utilizá-lo. Isso ajuda no trabalho, causando felicidade nos desenvolvedores envolvidos no desenvolvimento da funcionalidade e consequentemente levando eles a um melhor desempenho. Em contrapartida, isso também causa maior satisfação no cliente que verá as mudanças acontecendo mais rápido (DAVIS; DANIELS, 2016).

3 METODOLOGIA

Para esta pesquisa, foi realizada uma pesquisa bibliográfica exploratória experimental, composta por dados obtidos por meio de pesquisas em livros, sites, teses e artigos que também abordam sobre o tema central da pesquisa. Com os conhecimentos teóricos obtidos, foi possível realizar um estudo de caso ao realizar a implementação de uma aplicação web para uma entidade filantrópica, que foi desenvolvido por um time Scrum e a cultura DevOps.

4 ESTUDO DE CASO

O estudo de caso foi realizado pelo Grupo de Pesquisa em Engenharia de Software (GPES), com o time de desenvolvimento do projeto denominado Asilo Web App. O projeto refere-se ao desenvolvimento de duas aplicações para o Lar São Vicente de Paulo, uma Instituição de Longa Permanência para Idosos, situada na cidade de Ibitinga, interior do

estado de São Paulo, a fim de dar à instituição um meio para os interessados conhecerem mais sobre ela e sobre as atividades que lá acontecem.

A primeira aplicação, um site, que conta com uma seção onde a instituição pode contar mais detalhes sobre ela, uma seção onde pode publicar as dúvidas mais frequentes que os interessados na nela possam ter, uma seção onde pode divulgar notícias, ações beneficentes e eventos que foram realizados ou que ainda serão realizados. Também existe uma seção de portal da transparência, onde a instituição pode publicar documentos oficiais, prestações de contas, relatórios de atividades e quaisquer outros documentos que devem ser públicos a comunidade. Além do mais, o site conta com uma seção para que a instituição possa receber doações de forma totalmente online, rápida e segura, utilizando cartão de crédito ou boleto bancário.

Para gerenciar todas essas informações contidas no site, a segunda aplicação consiste em um painel administrativo, onde é possível gerenciar toda e qualquer informação que está presente no site. Por meio do painel administrativo também é possível acompanhar as doações que foram realizadas a entidade, por meio de um dashboard com gráficos que indicam as quantias recebidas por meio de doação. No painel administrativo também há uma seção onde a instituição pode acompanhar os acessos realizados ao site. Esta seção conta com gráficos que totalizam a quantidade de usuários que acessaram o site, qual tempo médio de cada seção, as notícias foram mais visualizadas, quais páginas são mais acessadas e por fim, uma relação de acessos a página de doações e o número efetivo de doações realizadas.

4.1 As tecnologias

Para o desenvolvimento de ambas as aplicações foi utilizado Javascript. Para o back-end foi utilizado o Node.js e para o front-end foi utilizado o framework Angular. Foi optado pelo Javascript, tendo em vista que ele ofereceria uma baixa curva de aprendizagem para os membros da equipe de desenvolvimento, além de que a mesma linguagem de programação seria utilizada em ambos os projetos. Para o banco de dados, optamos por utilizar um banco de dados não relacional, o MongoDB.

4.2 O desenvolvimento das aplicações

Para metodologia de desenvolvimento dessas aplicações, foi utilizado a metodologia de desenvolvimento ágil Scrum em um time de seis pessoas e o Product Owner, onde um era o Scrum Master, responsável por retirar impedimentos da equipe e garantir o sucesso do Scrum. As outras cinco pessoas eram do time de desenvolvimento, responsáveis pelos desenvolvimento das tarefas na sprint. E por fim, o Product Owner.

Além de utilizar o Scrum, foi introduzido ao time a cultura DevOps. Apesar de não termos um time focado na infraestrutura da aplicação, optamos por utilizar diferentes ambientes para que o desenvolvimento fosse testado em um ambiente parecido com o ambiente de produção.

Como cada integrante do time era responsável pelo desenvolvimento de um módulo, foi imprescindível o uso do GIT no projeto. Foram utilizadas diversas branches para que um trabalho não testado não integrasse a parte do projeto que já havia sido testada e validada. O time de desenvolvimento realizava suas tasks e realizava os primeiros testes. Ao finalizar, realizava o merge para uma branch com o projeto completo para que fosse realizada mais uma tomada de testes. Ao finalizar essa tomada de testes, essa branch era mesclada com a branch principal que estava instalada em um ambiente idêntico ao ambiente de produção, onde seriam realizados mais testes para validar que todo o projeto estava íntegro. Todo esse processo era realizado em uma aplicação de CI/CD utilizando o Heroku.

4.3 Resultados obtidos

Ao final do desenvolvimento do projeto, foi possível observar um resultado positivo, pois a cada semana que passava, o time de desenvolvimento podia ver seu trabalho sendo publicado, o que os manteve motivados a continuar. A utilização do Scrum, juntamente com o GIT e a cultura DevOps maximizou a velocidade de desenvolvimento e a qualidade do produto, uma vez que o GIT era responsável por gerenciar todas as mudanças que ocorriam no código e realizar a mescla entre novas mudanças e código que já existia de forma prudente, e o processo de CI/CD responsável por realizar uma bateria de testes para garantir que um código novo não quebraria um código que já estava funcionando corretamente. A maior dificuldade foi no início do projeto para a equipe compreender como funcionava todo o

sistema de controle de versão do GIT e como as alterações feitas seriam publicadas de forma automatizada em um ambiente idêntico ao de produção.

5 CONSIDERAÇÕES FINAIS

Com a implantação do projeto, foi possível trazer à instituição filantrópica uma informatização, além de proporcionar uma demonstração pública das atividades que acontecem dentro dela e que podem ser compartilhadas com a comunidade da região. Além disso, a aplicação proporciona a instituição seu próprio portal da transparência para a publicação dos documentos que se enquadram na lei da transparência. Um grande ponto positivo é também a possibilidade de a instituição receber doações para ela de forma online.

Inicialmente, o maior desafio desse projeto foi com que os integrantes do time de desenvolvimento aplicassem corretamente os conceitos do GIT, porém, após algumas semanas de desenvolvimento esses conceitos já estavam bem mais claros. Dentre os pontos positivos, pode-se destacar a facilidade em se identificar falhas dentro do código, fazendo uso dos testes antes que um código novo fosse mesclado a um código já funcional.

Dessa forma, foi possível notar neste estudo de caso, que o uso do Scrum aliado com a cultura DevOps foi importante e positivo para o time e para o desenvolvimento da aplicação, pois com as entregas semanais e a aplicação de DevOps, era possível o time ir vendo parte do seu trabalho sendo publicado e mantendo-os motivados a continuar o desenvolvimento, e também para a aplicação, que não contava com erros de implementação, pois antes de novos códigos entrarem em produção, eles eram testados e validados para garantir que um código já funcional não iria quebrar.

REFERÊNCIAS

BASS, Len; WEBER, Ingo; ZHU, Liming. **DevOps**: a software architect's perspective. Westford, Massachusetts: Addison-Wesley, 2015. 425 p

CHACON, Scott; STRAUB, Ben. **Pro Git**: Everything you need to know about GIT. 2. ed. Mountain View: Apress, 2020. 521 p.

DAVIS, Jennifer; DANIELS, Ryn. **Effective DevOps**: Building a Culture of Collaboration, Affinity, and Tooling at Scale. Sebastopol, California: O'Reilly Media, 2016. 410 p.

HÜTTERMANN, Michael. **DevOps for Developers**. New York, New York: Appress, 2012. 184 p.

PRESSMAN, Roger S. **Engenharia de Software**: Uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011. 779 p.

SABBAGH, Rafael. **Scrum**: Gestão Ágil para Projetos de Sucesso. São Paulo: Casa do Código, 2013. 355 p.

SCHWABER, Ken. **Agile Project Management with Scrum.** Redmond, Washington: Microsoft Press, 2004. 187 p.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum**: Um guia definitivo para o Scrum: As regras do jogo. [S. l.: s. n.], 2013. 19 p. Disponível em: https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf. Acesso em: 23 set. 2020.

SWARTOUT, Paul. **Continuous Delivery and DevOps - A Quickstart Guide**: Deliver quality software regularly and painlessly by adopting CD and DevOps. 2. ed. Birmingham: Packt Publishing, 2014. 196 p.