Ivan de Moura Miranda

Projeto de Automação de Operações e Entrega Contínua para Empresa de Desenvolvimento de Software.

Conselheiro Lafaiete

2016

Ivan de Moura Miranda

Projeto de Automação de Operações e Entrega Contínua para Empresa de Desenvolvimento de Software.

Relatório técnico apresentado como trabalho de conclusão de curso para obtenção do diploma do curso de Engenharia de Computação, Faculdade Presidente Antônio Carlos de Conselheiro Lafaiete.

Faculdade Presidente Antônio Carlos – FUPAC

Engenharia de Computação

Prof. Me. Jean Carlo Mendes

Conselheiro Lafaiete

2016

Resumo

Este trabalho descreve o projeto de otimização do processo de entrega de uma empresa de desenvolvimento de *softwares*. O objetivo é identificar as etapas de baixa produtividade para implementar ferramentas e aplicar metodologias que melhoram a qualidade dos produtos e serviços desenvolvidos. Durante o processo de desenvolvimento diversas tarefas repetitivas são executadas manualmente, com a automatização destas é possível reduzir o tempo investido no projeto, e com a aplicação de boas práticas de desenvolvimento, também consegue-se aumentar a estabilidade do sistema reduzindo o número de *bugs* encontrados em produção.

Palavras-chaves: entrega contínua, integração contínua, *devops*, metodologias ágeis.

Lista de tabelas

Lista de abreviaturas e siglas

AWS	*Amazon Web Services*
BaaS	*Back-end as a Service*
BDD	*Behavior Driven Development*
bug	*Defeito, falha ou erro no código de um programa que provoca seu mau funcionamento.*
CPU	*Central Processing Unit*
DDD	*Domain Drive Design*
deploy	*Implantação de uma versão do software em ambiente de produção*
devops	*Development and Operations*
DNS	*Domain Name System*
IaaS	*Infrastructure as a Service*
IPMI	*Intelligent Platform Management Interface*
JMX	*Java Management Extensions*
PaaS	*Platform as a Service*
RAM	*Random Access Memory*
SaaS	*Software as a Service*
SNMP	*Simple Network Management Protocol*
SSH	*Secure Shell*
TDD	*Test Driven Development*
XP	*Extreme Programming*

Sumário

1 Introdução

De acordo com uma pesquisa publicada pelo Stack Overflow (2016), os maiores desafios encontrados por desenvolvedores em seu ambiente de trabalho são a documentação ruim, os requisitos mal especificados e o processo de desenvolvimento ineficiente.

Diante dessa realidade percebe-se que o ambiente de desenvolvimento de *softwares* é negativamente afetado pela ineficiência do processo que muitas das vezes é excessivamente burocrático e pouco produtivo.

> Mas ser burocrático não é o problema, pois as regras são necessárias para nortear qualquer tipo de trabalho. O problema reside, é claro, em ser excessivamente burocrático, gerando atividades/produtos que não agregam valor e consomem desnecessariamente tempo da equipe. (E-BUSINESS, 2015)

Com o objetivo de resolver estes e outros problemas, várias metodologias foram criadas, como os métodos ágeis, a cultura *devops*, e as ferramentas que objetivam a automação dos processos de operação, permitindo a entrega contínua de *software* com a mínima intervenção humana, em busca de agilidade e qualidade do sistema entregue. O problema é que essa automação exige conhecimento de muitas tecnologias e ferramentas diferentes, o que é visto na maioria das vezes como barreira por desenvolvedores que não tem familiaridade com especificações de infraestrutura.

Na tentativa de resolver estes problemas, várias empresas viram uma oportunidade de negócio atrelada a consultoria e prestação de serviços nessa área, incentivando o surgimento de diversas plataformas, como o *Developer Cloud Service* (ORACLE, 2016), *Open Shift* (RED HAT, 2016) e o *AWS Lambda* (AMAZON, 2016). Estas soluções buscam facilitar a vida dos desenvolvedores oferecendo infraestrutura, plataforma ou *back-end* como serviço para as empresas de desenvolvimento, e algumas ainda oferecem ferramentas para gestão dos projetos do time de desenvolvimento.

Apesar destas soluções semi prontas, se levarmos em conta apenas as ferramentas de automação fornecidas, boa parte delas são gratuitas e de código fonte aberto como o *Git*, *Jenkins* e *Kubernetes*. O próprio *Open Shift*, por exemplo, possui uma versão de código aberto que pode ser instalado em qualquer empresa com acesso a todos os recursos. Então as vantagem de pagar por esses tipos de serviço se devem a hospedagem na nuvem, a consultoria especializada e ao baixo custo de manutenção provido pela falta de necessidade de contratação de um profissional dedicado, porém este custo ainda não é totalmente viável para micro e pequenas empresas, que de acordo com uma pesquisa realizada pela ABES

(2016) representam 95,1% das empresas de desenvolvimento e produção de tecnologia no Brasil.

Diante desse cenário, boa parte do mercado nacional pode se sentir desmotivado a contratar um serviço desse tipo, pois é muito difícil prever o benefício real proporcionado, afinal, o custo do investimento é fácil calcular, mas o lucro estimado é complicado, uma vez que eficiência e eficácia precisam ser convertidas em dinheiro baseadas na confiança de que a automação irá realmente resolver os problemas da empresa.

Uma alternativa a este investimento ainda existe, que é a capacitação profissional do time de desenvolvimento para a utilização das ferramentas gratuitas em servidores locais, que não precisam ser computadores muito potentes devido a baixa necessidade de escalabilidade e performance, uma vez que os usuários desses sistemas serão os próprios colaboradores da empresa.

Para implantação desse ambiente automatizado que utiliza apenas ferramentas gratuitas, é necessário o estudo de uma gama de soluções a fim de identificar a que mais atende a real necessidade da empresa.

1.1 Objetivos

Em uma tentativa de validar os benefícios da automação e entrega contínua num cenário real, graças ao apoio da DevelOP, uma empresa de desenvolvimento de *software* situada em Ouro Preto, este trabalho se propõe a modificar o ambiente de desenvolvimento da empresa baseando-se no estudo de boas práticas e ferramentas gratuitas buscando a automação das tarefas de operações e o aumento da qualidade dos *softwares* desenvolvidos. Objetiva-se também avaliar a eficácia dos métodos propostos graças a um comparativo de performance da empresa antes e depois da implementação do projeto.

1.2 Organização dos capítulos

Este trabalho foi dividido em duas partes, a primeira apresenta os procedimentos preparatórios, incluindo a metodologia utilizada, o levantamento bibliográfico, a apresentação da empresa e os procedimentos experimentais realizados em tentativa de propor soluções para os problemas identificados.

A segunda parte apresenta os resultados obtidos e realiza um comparativo entre o cenário da empresa antes e depois da implantação das metodologias sugeridas.

Parte I

Desenvolvimento

2 Metodologia

A fim de avaliar os benefícios das soluções propostas, este trabalho foi fundamentado na investigação a respeito do tema. As situações referentes ao objeto de estudo, que no caso se trata do processo de desenvolvimento de *softwares* da empresa DevelOP, foram examinadas com olhar investigativo buscando atingir a maior veracidade possível.

O projeto aborda conhecimentos a respeito da gerência de projetos, desenvolvimento e entrega de *softwares* e para isso se fez necessário direcionar a abordagem com base na utilização de material teórico, estabelecendo uma linha de investigação pela qual foi conduzido o trabalho para levantar todo o material necessário com o intuito de estabelecer uma avaliação prática dos resultados obtidos.

O projeto foi divido em 3 etapas. A primeira se baseou no estudo do cenário atual da empresa coletando e analisando dados em busca de identificar pontos problemáticos e levantar propostas de soluções. A segunda fase teve por objetivo implementar as soluções propostas e analisar seus efeitos no ambiente da empresa. A terceira fase teve a responsabilidade de mostrar os resultados e avaliar as soluções oferecidas e as vantagens que propõe.

Este trabalho atuou utilizando o método experimental de forma a estudar os fatores que influenciam o tempo e a qualidade do processo de desenvolvimento, desta forma avaliando a veracidade das hipóteses levantadas.

2.1 Procedimentos para coleta e análise dos dados

A coleta dos dados foi feita através do acompanhamento das rotinas da empresa para que o conhecimento da área de estudo seja ampliado, assim preenchendo melhor as lacunas no projeto.

Foi realizado um comparativo seguindo algumas métricas como: tempo necessário para entregar uma nova versão do *software*; relação entre a quantidade de *builds* quebradas e bem sucedidas por dia; quantidade de *bugs* reportados por semana; a facilidade de manutenção do código; e a satisfação dos desenvolvedores com o processo de desenvolvimento. A partir desse comparativo pode-se concluir as vantagens e desvantagens da solução proposta bem como avaliar a eficiência e eficácia do novo processo.

3 Levantamento bibliográfico

Este capítulo apresenta os conceitos teóricos relacionados as metodologias e procedimentos utilizados no desenvolvimento deste projeto.

3.1 Metodologia ágil

As metodologias de desenvolvimento ágil foram criadas após o surgimento do manifesto ágil por Beck et al. (2001), onde um grupo de especialistas renomados em computação definiu um conjunto de doze princípios de desenvolvimento de *software* que deveriam ser seguidos por empresas que desejam melhorar a qualidade dos produtos e serviços oferecidos.

3.1.1 *SCRUM*

Uma das metodologias ágeis mais famosa para gestão e planejamento de projetos de *softwares* é o *SCRUM*. Nele as atividades a serem implementadas são mantidas em uma lista conhecida como *Product Backlog*, e o projeto é divido em ciclos chamados de *Sprints* onde as atividades registradas no *Backlog* são distribuídas entre o time de desenvolvimento de acordo com as prioridades levantadas pelo *Product Owner*.

Além dessa organização e controle de atividades, o *SCRUM* sugere diversas outras práticas como reuniões diárias, reuniões de revisão de *sprints* e reuniões de retrospectiva visando aumentar o nível de comunicação entre o time e garantir eficiência durante o desenvolvimento do projeto (SCRUM, 2013).

3.1.2 *Extreme programming, XP*

Enquanto o *SCRUM* é uma metodologia voltada para a gestão e planejamento de projetos de *softwares*, o *XP* por outro lado é um metodologia de desenvolvimento. O *XP* alcança seu objetivo de ajudar a criar sistemas de melhor qualidade através de conjunto de valores, princípios e práticas que se diferem substancialmente da forma tradicional de se desenvolver *software* (EXTREME. . . , 2013).

Os valores no *XP* servem para responder a seguinte pergunta: se todos na equipe se concentrarem naquilo que é importante para a equipe, em que devem se concentrar? E a resposta é um conjunto de cinco valores para guiar o desenvolvimento:

- Comunicação

- Coragem
- *Feedback*
- Respeito
- Simplicidade

Já os princípios existem para servir de ponte entre valores e práticas, dos quais pode-se citar auto-semelhança, benefício mútuo, diversidade, humanismo, fluidez, melhora, qualidade, dentre outros.

As práticas representam aquilo que você verá as equipes *XP* fazendo diariamente e são divididas em duas partes: primárias e corolárias.

As práticas primárias são práticas que você pode começar a adotar imediatamente de forma segura para melhorar seu esforço de desenvolvimento de *software* das quais pode-se citar o desenvolvimento orientado a testes, *design* incremental, programação em par e integração contínua.

As práticas corolárias são práticas difíceis ou perigosas de serem implementadas antes de se adotar as práticas primárias. Pode-se citar a base de código unificada, código coletivo, contrato de escopo negociável e envolvimento do cliente real como práticas corolárias.

3.2 Integração contínua

Segundo Silveira et al. (2011), integrar novas versões e alterações realizadas pela equipe ao repositório principal em uma frequência alta, sempre e o mais rápido possível, tentando minimizar a influência de janelas longas sem integração, é chamado integração contínua.

O autor diz que o tempo necessário para receber um *feedback* em relação a uma alteração realizada no código de um sistema é um fator extremamente importante para garantir a qualidade do que está sendo desenvolvido, uma vez que quanto mais tardio o *feedback* for recebido pelos desenvolvedores a respeito de um *bug* ou funcionalidade que não atinge o objetivo, mais difícil é de se lembrar o que, como e os motivos pelos quais foi tomada uma decisão no desenvolvimento que resultou no mesmo.

> É importante que a maior parte possível dos testes seja executada a cada *commit*, para que a informação do impacto das alterações na base de código venha rapidamente, permitindo ao desenvolvedor agir de maneira adequada para corrigir falhas e *bugs* que possam ter se introduzido. (SILVEIRA et al., 2011)

Dessa forma conclui-se que o resultado obtido através do uso de integração contínua no ambiente de desenvolvimento é a garantia de que o código existente no repositório é sempre estável, e a qualquer momento pode-se gerar um novo artefato com a garantia de que o sistema está passando em todos os testes automáticos definidos durante o desenvolvimento.

Para a automação da execução dos testes que garantem a qualidade do código durante as integrações, Humble e Farley (2014) apresenta um padrão chamado de *Build Pipeline* que é, em essência, uma implementação automatizada do processo de compilar todas as partes de uma aplicação, implantá-la em um ambiente qualquer, seja de homologação ou produção, testá-la e efetuar sua entrega final.

3.3 Entrega contínua

Segundo Humble e Farley (2014), a entrega contínua é uma extensão natural da integração contínua, é uma abordagem na qual as equipes asseguram que cada alteração no sistema é entregável, e que pode-se liberar qualquer versão com o simples toque de um botão. A entrega contínua visa reduzir a complexidade de construção de uma nova versão, o que nos permite gerar versões com maior frequência o obter *feedback* rápido sobre o que os usuários se importam.

3.4 *DevOps*

Os especialistas se referem a *DevOps* como uma cultura, um movimento e um ambiente devido ao fato de ser uma metodologia que envolve valores presentes desde o planejamento até a infraestrutura necessária para a implantação de aplicações.

> *DevOps* (um amálgama de desenvolvimento e operações) é um método de desenvolvimento de *software* que enfatiza a comunicação, a colaboração e a integração entre os desenvolvedores de *software* e os profissionais de operações de tecnologia da informação (TI). (NEW RELIC, 2016 apud GARTNER RESEARCH, 2012)

De acordo com New Relic (2016), os principais fundamentos estabelecidos pela *DevOps* são:

- **Colaboração:** os setores da empresa não devem criar disputas ou se isolarem entre si, a necessidade de colaboração abrange todos que participam do fornecimento de *software*, inclusive testes, gestão de produtos, executivos, desenvolvimento e operações.

- **Automação:** não se deve perder tempo com tarefas manuais que podem ser automatizadas.
- **Integração contínua:** a prática de forçar os desenvolvedores a integrar seu trabalho com o de outros desenvolvedores frequentemente, pelo menos diariamente, expõe problemas de integração e conflitos antecipadamente, o que contribui para melhoria da qualidade do que é desenvolvido.
- **Testes contínuos:** em um ambiente *DevOps*, todo mundo se envolve nos testes. Os desenvolvedores certificam-se de, além de entregar códigos sem erros, fornecer conjuntos de dados de teste. Eles também ajudam os engenheiros de teste a configurar o ambiente de teste de modo a ficar o mais parecido possível com o ambiente de produção.
- **Fornecimento contínua:** elevar o conceito de integração contínua em mais uma etapa, expandir a integração para toda cadeia de lançamento, incluindo controle de qualidade e operações.
- **Monitoramento contínuo:** devido à grande quantidade de lançamentos, não é possível implementar o tipo de teste rigoroso pré-lançamento que caracteriza o desenvolvimento em cascata. Portanto, em um ambiente DevOps, as falhas devem ser encontradas e corrigidas em tempo real.

4 Análise do cenário atual

Este capítulo apresenta uma análise de características em busca de descrever os processos e a estrutura organizacional da empresa no inicio do desenvolvimento deste projeto.

4.1 Associação de Desenvolvimento de Software, Produtos e Pessoas da Região dos Inconfidentes Mineiros - DevelOP

A DevelOP é uma associação de direito privado, constituída na forma de sociedade civil de fins não lucrativos, com autonomia administrativa e financeira, fundada em 2016 por um grupo de profissionais em tecnologia da informação na região dos inconfidentes mineiros. A Associação tem por finalidade central realizar ações sociais de utilidade pública na área de desenvolvimento de *software*, produtos de *software* e pessoas.

Situada no centro de Ouro Preto, a associação tem como principal atividade econômica o desenvolvimento de programas de computador sob encomenda, mas também atua em atividades de desenvolvimento e licenciamento de programas de computador customizável, consultoria em tecnologia da informação e atividades de apoio à educação.

Apesar de ser uma empresa nova no mercado, a DevelOP já possui alguns projetos em seu portfólio, como o evento Empreenda. Em Ação! realizado em julho de 2016 onde equipes formadas por alunos de disciplinas de Empreendedorismo da Universidade Federal de Ouro Preto (UFOP) competem na idealização, planejamento e apresentação de modelos de negócio reais e o atual projeto em desenvolvimento, onde a empresa presta serviços a um cliente de São Paulo atuando na automação de processos de extração de conhecimento em Diários Oficiais.

4.2 Estrutura organizacional

As atividades econômicas desenvolvidas pela empresa apresentam características que exigem a divisão de times em projetos, mesmo que alguns funcionários algumas vezes participem de mais de um projeto simultaneamente, como é o caso dos analistas de negócio por exemplo. A empresa se organiza para alocar os recursos temporariamente até sua liberação.

Atualmente a empresa trabalha com especialistas em áreas definidas, como desenvolvedores, administradores, *designers*, analistas de negócio e jornalistas. A maioria dessas áreas é preenchida com apenas um profissional responsável por responder pelas demandas

relacionadas ao seu domínio. Quando há a alocação de recursos para projetos, o responsável pela gerência do projeto analisa quais áreas serão relacionadas em que momento, e então é feita a alocação dos times.

A comunicação é feita de maneira informal devido a proximidade dos funcionários e ao pequeno quadro de colaboradores.

4.3 Recursos disponíveis

O time de desenvolvimento é composto por um desenvolvedor pleno e três estagiários. As áreas de *design*, análise de negócios e jornalismo contam, cada uma, com apenas 1 funcionário.

Devido a parceria que a empresa tem com a Universidade Federal de Ouro Preto, algumas vezes por ano ela recebe inter cambistas de experiência e área de atuação diversificada.

Dos recursos que são relevantes para o objeto de estudo deste trabalho, a empresa terceiriza a hospedagem dos servidores de aplicação, contando com dois computadores em ambiente remoto cuja especificações podem ser encontradas na Tabela 1.

Tabela 1 – Especificações técnicas dos servidores da empresa

	Servidor A	**Servidor B**
CPU	Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz	AMD Opteron(tm) Processor 3280
Memória RAM	32 GB	32 GB
Disco rígido	500 GB	500 GB
Sistema Operacional	Ubuntu 14.04.3 LTS	Ubuntu 14.04.3 LTS

4.4 O processo de desenvolvimento

Esta sessão descreve o processo de desenvolvimento adotado pela empresa no início do projeto, com o objetivo de descrever as atividades realizadas pelo time de desenvolvimento no seu dia a dia facilitando a compreensão dos problemas existentes.

As metodologias envolvidas nos projetos da empresa são inspirados em métodos ágeis. Alguns utilizam *SCRUM* e outros apenas *Kanban*, isso varia de acordo com os requisitos e recursos disponíveis para cada projeto, mas de forma geral, a empresa sempre busca manter um quadro de atividades (*Kanban*) e realizar reuniões frequentes.

Através do acompanhamento do processo de desenvolvimento durante o período que este trabalho foi realizado, identifica-se um ciclo baseado em definição de requisitos,

implementação e execução de testes manuais. Inicialmente o coordenador do projeto realiza uma reunião com o time para especificar quais funcionalidades são prioritárias e detalhar o que precisa ser feito para seu desenvolvimento. Durante essa reunião é feita a divisão de atividades e a estipulação do tempo necessário por cada responsável. Uma pessoa é encarregada pelo registros das atividades no *software* de gestão, e em seguida cada membro do time começa a trabalhar na tarefa que lhe foi atribuída.

Diariamente o time realiza reuniões de acompanhamento, onde informam seu progresso ao coordenador do projeto e tiram dúvidas com os demais colaboradores. Caso uma atividade seja concluída, uma nova pode ser atribuída.

Durante o desenvolvimento, os programadores trabalham em *branches*[1] individuais com auxílio de um sistema de controle de versões chamado *Gogs*, que é hospedado em um dos servidores remotos. Eles testam manualmente cada funcionalidade em seu computador, e quando o time chega ao consentimento que existe um conjunto de funcionalidades pronto para ser entregue, algum membro do time compila o projeto em seu computador e realiza o envio do artefato gerado através de uma comunicação de rede para o ambiente de produção. Durante essa etapa de entrega, caso a versão do artefato gerado apresente *bugs*, o desenvolvedor responsável pela tarefa o corrige, gera uma nova versão e a envia novamente ao servidor de produção. Esta tarefa pode se repetir enquanto houverem *bugs* que impossibilitem o correto funcionamento da aplicação.

Caso as características do artefato gerado não influenciem negativamente no processo dos usuários do sistema, um teste manual é realizado no ambiente de produção.

Quando surgem demandas imprevistas durante o ciclo de desenvolvimento, uma reunião é convocada para re-ajustarem o cronograma semanal e redistribuírem as tarefas. Caso um *bugs* seja reportado pelo cliente, ele é avaliado quanto a sua criticidade e comparado à fila de tarefas pendentes. Caso seja um *bug* crítico, é tratado imediatamente, caso não seja critico e existam atividades prioritárias na fila, ele é registrado e levado em pauta na próxima reunião, onde o time reajusta o cronograma.

4.4.1 Dados coletados com a análise

A etapa de coleta de dados analíticos sobre o atual processo de desenvolvimento teve início no dia 8 de agosto de 2016. Durante essa etapa foram identificadas tarefas que fogem ao escopo de desenvolvimento e levantamento de requisitos, como atividades de compilação do código fonte, execução de testes manuais, configuração de ambientes de trabalho, etc. As execuções dessas atividades foram registradas a fim de sumarizar o tempo necessário e a quantidade de retrabalho envolvido. O acompanhamento e registro desses dados ocorreu até o dia 23 de setembro de 2016.

[1] Um *branch* é uma divisão independente da linha de desenvolvimento e permite que desenvolvedores mantenham versões diferentes do código fonte enquanto estão desenvolvendo atividades distintas.

As tabelas 2 e 3 apresentam dados quantitativos referentes as etapas do processo de desenvolvimento durante a fase de coleta. Graças a essas informações é possível identificar pontos de aprimoramento em potencial.

A Tabela 2 apresenta dados analíticos referentes aos repositórios de código fonte da empresa.

Tabela 2 – Dados do repositório de código fonte no início do projeto

Métrica	**Quantidade**
Número de *commits* realizados	125
Número de artefatos gerados	65
Número de mesclagem de código realizados	6
Número de *bugs* reportados pelo cliente	20
Número de atividades desenvolvidas pelo time que foram registradas no sistema de controle de atividades	70

A Tabela 3 demonstra o tempo necessário por entrega de uma nova versão do sistema. Consideramos o tempo necessário para construção de um artefato executável e a sua disponibilização em ambiente de produção. Esta etapa engloba tentativas má sucedidas de construção e entrega de uma versão estável e sem *bugs*.

Tabela 3 – Características das entregas no início do projeto

Data da entrega	**Tempo necessário**	**Número de tentativas má sucedidas até a conclusão da entrega**
16/08/2016	2 horas	0
23/08/2016	6 horas	3
30/08/2016	3 horas	1
07/09/2016	6 horas	5
14/09/2016	8 horas	7
19/09/2016	4 horas	2

Após a coleta de dados, uma entrevista foi realizada com o time de desenvolvimento onde foi solicitado que cada um citasse problemas marcantes que aconteceram com eles no decorrer do projeto com o objetivo de compreender a visão do time sobre o processo de desenvolvimento em busca de identificar aspectos impactantes na rotina de trabalho que precisam ser melhorados. As seguintes respostas foram obtidas:

- Algum membro do time esqueceu de apagar um diretório que estava em lugar indevido, o que ocasionou a sobrecarga do disco-rígido do servidor do cliente, pois havia uma unidade de baixa quantidade de armazenamento disponível alocada apenas para o sistema operacional, e esta ficou inoperante, exigindo a investigação e remoção dos arquivos indevidos.

- Em uma tentativa de entrega de uma nova versão, foi identificado a existência de um diretório chamado "teste"no ambiente de produção. Consultando o time sobre a necessidade daquele diretório e na tentativa de apaga-lo do sistema pois não deveria estar ali, foi comunicado que o ambiente de produção estava neste diretório e não no lugar correto onde deveria ser executado.
- Em uma tentativa de entrega foi identificada a configuração indevida de permissões de um certo diretório feita por outro membro do time.
- Por diversas vezes o time identificou problemas relacionados à implantação de uma nova versão devido a arquivos de propriedades configurados incorretamente.
- Vários problemas já aconteceram devido a execução de *scripts* no ambiente de produção sem os devidos testes serem realizados previamente.
- Diversos problemas aconteceram devido a necessidade da configuração manual do ambiente de produção, uma vez que diretórios precisavam ser manipulados por linha de comando.

4.4.2 Características do processo atual

Baseado nos dados coletados durante a primeira fase do projeto identificamos como principais problemas o tempo necessário para entrega de uma nova versão estável do sistema e a inexistência de uma garantia de qualidade sobre os artefatos entregues. A qualidade neste contexto pode ser melhorada aumentando a capacidade do *software* atender a necessidade de mudanças, o que poderá ser identificado pela redução da quantidade de *bugs*.

Segundo Reisswitz (2013), a qualidade de um *software* está diretamente relacionada à qualidade do processo de desenvolvimento, dessa forma, é comum que a busca por um *software* de maior qualidade passe necessariamente por uma melhoria no processo de desenvolvimento.

Relacionando as dificuldades reportadas pelo time com os problemas levantados, identificamos que as atividades que não são totalmente voltadas ao desenvolvimento nem ao levantamento de requisitos são os principais causadores dos problemas, provavelmente pela falta de especialistas em infraestrutura e pela falta de automação das atividades de implantação.

Os problemas citados se mostram relevantes uma vez que já foram identificados por especialistas, como Silveira et al. (2011, p. 136) que descrevem um ambiente com características similares ao ambiente da empresa alvo de estudo deste trabalho como um ambiente problemático e sujeito a diversas falhas.

A partir de todos esses dados levantados, nossa linha investigativa foi elaborada em busca de reduzir o tempo necessário para implantação de novas versões do código, estabelecer métricas de qualidade, garantir que a versão de produção é estável e reduzir o número de versões defeituosas entregues pelo time.

5 Procedimentos experimentais

Este capítulo descreve os procedimento realizados na empresa para atingir os objetivos do projeto, incluindo as dificuldades encontradas durante seu desenvolvimento.

5.1 Adaptando a rotina do time para aplicar as metodologias em prática

No início da aplicação das novas metodologias e boas práticas, buscamos encontrar formas de reduzir o tempo necessário para entrega de uma nova versão, para isso se fez necessário levantar as justificativas do tempo e esforço investidos no processo inicial.

O envio de novas versões ao servidor de produção de forma manual era um dos fatores que impactavam no tempo. Problemas com a conexão de internet do escritório faziam com que a pessoa responsável pelo envio do artefato ficasse esperando entre 15 minutos até uma hora, e as vezes acontecia da conexão de internet ser interrompida atrapalhando o progresso do envio, sendo necessário começar o processo outra vez.

Após o estudo de possíveis soluções, percebe-se que a melhor forma de otimizar o tempo de envio seria transferir a responsabilidade pela construção e envio do artefato a um servidor remoto com melhor conexão de internet, mas para que isso pudesse ser feito, era necessário que antes existisse a garantia de estabilidade da versão existente no repositório principal, só assim o processo poderia ser automatizado de forma eficiente. Decidiu-se que a prática de integração contínua deveria ser a primeira adotada pela empresa.

Para que a integração contínua apresente as vantagens de garantia de código estável, é necessário a existência de testes automatizados, requisito que o projeto não cumpria até então. Em conjunto com o time, o cronograma de atividades foi ajustado para que fosse possível separar aproximadamente uma semana para implantar os testes e refatorar trechos de código que precisavam ser otimizados para permitir o bom funcionamento dessa abordagem. Durante essa semana foram encontradas algumas dificuldades para dividir o tempo entre testes e continuar implementando novas funcionalidades, os colaboradores precisaram fazer algumas horas extras pra atualizar a aplicação e ainda automatizar estes testes.

Após esse período de automação de testes, o time percebeu que ainda não estava satisfeito com algumas implementações pois tinham dificuldades de fazer manutenção nelas, nesse momento foi identificado que uma grande refatoração seria necessária para melhorar a qualidade do sistema. Apesar dessa refatoração ser necessária, a aplicação

estava funcionando, e não era essencial refatorar o código do ponto de vista do cliente, então duas sugestões foram levantadas para mantermos a linha de desenvolvimento de novas funcionalidades operante e ainda sim conseguirmos melhorar a base de código existente. A primeira delas seria separar um dia inteiro por mês ou por semana para que o time ficasse por conta de refatorar o código, e chegamos a conclusão de que essa medida seria inviável por limitações de tempo e probabilidade do surgimento de emergências que comprometeriam essa abordagem. A segunda sugestão seria que o time, todos os dias na parte da manhã, assim que chegassem ao escritório, separassem entre trinta minutos e uma hora para fazer apenas um *commit* de refatoração ou de automação de testes, dessa forma, a passos lentos, chegaria uma hora onde sistema estaria mais estável. Essa abordagem foi aplicada durante todo o processo de implantação e apresentou ótimos resultados, com cerca de duas semanas o time já estava seguro de que a aplicação poderia aproveitar dos benefícios da prática de integração contínua.

Agora que haviam testes automatizados e que o código estava melhorando, um novo processo de desenvolvimento estava emergindo. Quando uma nova funcionalidade era implantada, o desenvolvedor fazia um *merge request* dentro da ferramenta de controle de versão, então outro desenvolvedor era acionado para fazer o *code review*. Este segundo desenvolvedor analisava as alterações feitas no repositório e podia escolher entre duas ações: recusar a alteração submetida justificando os problemas encontrados ou aceita-la e permitir que o sistema executasse a *build pipeline* do projeto. Essa prática de *code review* alem de fazer com que o time tenha mais conhecimento do código que constrói, também agiliza a correção de problemas caso algum *bug* seja injetado no sistema pelo desenvolvedor que solicita o *merge request*.

Nesse momento o time tinha garantia de que o código existente no repositório principal era estável e a integração de alterações era contínua, o próximo passo foi a automação do processo de implantação.

Devido a restrições de segurança impostas pelo cliente, um sistema de integração foi disponibilizado por ele no ambiente de rede privado do cliente para que este sistema executasse a implantação automaticamente. Depois que a *build pipeline* era executada em nosso servidor de integração contínua, bastava acessar o painel desse sistema disponibilizado no ambiente do cliente e apertar um botão que o processo de implantação era executado em pouquíssimos minutos.

5.2 Estudo comparativo entre ferramentas de DevOps

Para elaboração de um ambiente de desenvolvimento com tarefas de operações automatizadas, visando aumento da produtividade, economia de tempo e garantia de qualidade, foi realizado um levantamento comparativo entre ferramentas gratuitas ou

de código fonte aberto que possibilitem a aplicação de todos os métodos necessários especificados pelas metodologias apresentadas na sessão anterior.

É importante que as ferramentas sejam integradas para agilizar a comunicação do time e que tenham interfaces de utilização que estejam de acordo com as boas práticas indicadas com as metodologias de desenvolvimento desejadas.

Para aplicar essas metodologias no cenário atual da empresa, os seguintes tipos de ferramentas se mostraram necessários:

- Sistema de controle de versão com suporte a ramificações e pedidos de mesclagem de código submetidos a revisões preliminares.
- Sistema de execução de *build pipelines* com suporte a ambientes de testes.
- Sistema de automação de implantações.
- Sistema de monitoramento de recursos.

Além dessas ferramentas citadas, para implantação de um ambiente de entrega contínua completo, tolerante a falhas e altamente escalável, um série de ferramentas adicionais se fazem necessárias, como centralizadores de logs, provisionadores de ambientes, orquestradores de máquinas virtuais, dentre outros. Devido a limitações de tempo, nem todas as ferramentas foram avaliadas e aplicadas no ambiente da empresa. Como o objetivo deste trabalho é resolver os principais problemas relacionados a produtividade e qualidade do processo de desenvolvimento, a maior quantidade de esforço foi concentrada na implantação das principais ferramentas e na adaptação da metodologia de trabalho do time.

Uma pesquisa foi realizada para selecionar as ferramentas disponíveis no mercado que atendem os requisitos levantados, pois como foi apontado pela XebiaLabs (2016), existem centenas de ferramentas criadas para estes propósitos, e testar todas tornaria este trabalho completamente inviável por limitações de tempo e pelo surgimento constante de novas ferramentas similares. Devido a estes fatores, para cada tipo de ferramenta foram levantadas entre 4 e 5 opções mais populares baseadas em análises de sites especialistas em cada assunto.

Diante do conjunto de ferramentas identificadas, as consideradas não cruciais para a boa conclusão deste trabalho não foram implantadas neste projeto. Por isso algumas das ferramentas descritas nas seções seguintes não apresentam um detalhamento tão aprofundado.

5.2.1 Sistema de controle de versão

Para escolher qual sistema de controle de versão usar levamos em conta que o time já possuía experiência prévia com *git*, o que facilitaria sua utilização. Graças a análises feitas por DevMedia (2016), InfoQ (2008), SitePoint (2014), identifica-se que o *git* apresenta características que justificam ainda mais a sua escolha, mas ainda sim precisamos de uma ferramenta de controle de código fonte que tenha suporte a revisão e, de preferência, com uma interface gráfica para simplificar a gerência dos repositórios.

A Tabela 4 mostra os resultados da pesquisa para as ferramentas mais populares de controle de versão baseadas em git. A escolha dessas ferramentas foi feita baseando-se em recomendações de outros profissionais e recorrendo a análises feitas por sites como Slant (2016) e StackShare (2016a).

Baseando-se neste comparativo, chegamos a conclusão de que o *GitHub* e o *BitBucket* não representam a melhor escolha pelas limitações das versões gratuitas. Entre o *Gogs* e *GitLab* optamos pelo *GitLab* pela maior quantidade de recursos disponíveis.

5.2.2 Ferramentas de execução de *build pipelines*

As ferramentas de execução de *build pipelines* e de automação de rotinas de implantações podem ser encontradas na Tabela 5. A seleção dessas ferramentas também foi feita levando-se em conta as análises feitas pelo StackShare (2016b), porém também foram inclusos o *BitBucket* e o *GitLab* devido ao fato de serem ferramentas que, além de darem suporte a estes requisitos, também realizam o controle de código fonte. Uma vez que é vantajoso ter menos ferramentas para administrar, elas tem preferência sobre as demais.

Após a análise das funcionalidades disponíveis, o *BitBucket* e o *Travis CI* foram descartados pelas limitações das versões gratuitas. Ao experimentar o *Jenkins* e o *GitLab* na prática, percebe-se que a quantidade de *plugins* disponíveis para o *Jenkins* fazem com que ele se torne extremamente flexível em relação a diversidade de projetos e permitem análises profundas de quesitos como cobertura de testes, documentação e relatórios de compilação. Por outro lado o *GitLab* se mostra simples e prático para projetos menores e com menor diversidade.

Devido a estes fatores, optou-se pela utilização das duas ferramentas. A execução da *build pipeline* ocorreria no *GitLab* para projetos menores e menos complexos e o *Jenkis* foi escolhido para lidar com o processo de compilação de projetos maiores e mais complexos.

Tabela 4 – Comparativo entre ferramentas de controle de versão de código baseadas em git disponíveis no mercado.

	GitHub	**BitBucket**	**Gogs**	**GitLab**
Hospedagem remota	✓	✓	-	✓
Hospedagem local	-	-	✓	✓
Código aberto	-	-	✓	✓
Repositórios privados	Pago	Limite de 5 usuários por repositório	✓	✓
Gerência de organizações e times	✓	✓	✓	✓
Gerenciador de incidentes	✓	✓	✓	✓
Wiki	✓	✓	✓	✓
Pull requests	✓	✓	Apenas entre um *fork* e seu original	✓
Code Review	✓	✓	Não permite comentários em trechos de código	✓
Web Hooks	✓	✓	✓	✓
Build pipeline	-	✓	-	✓
Pesquisa por trecho de código	✓	-	-	-
Estatísticas de uso do repositório	✓	-	-	✓
Visualização gráfica das ramificações	✓	✓	-	✓
Snippets	✓	✓	-	✓

5.2.3 Ferramentas de monitoramento

As ferramentas de monitoramento de recursos, também identificadas com ajuda de análises de sites como DevOps.com (2015) e StackShare (2016c) e de indicações de profissionais. Elas podem ser encontradas na Tabela 6.

Devido a pequena diferença em relação a quantidade de funcionalidades entre as ferramentas de monitoramento, utilizamos como critério a experiência do time com as ferramentas em busca de reduzir o tempo de aprendizado e optamos pelo *Zabbix*.

Tabela 5 – Comparativo entre ferramentas de execução de *build pipelines* disponíveis no mercado.

	Jenkins	**GitLab**	**BitBucket**	**Travis CI**
Hospedagem remota	-	✓	✓	✓
Hospedagem local	✓	✓	-	-
Código aberto	✓	✓	-	-
Construção de branches	✓	✓	✓	✓
Construção de pull requestes	✓	✓	✓	✓
Nível de complexidade	Mediano	Simples	Simples	Simples
Suporte a múltiplos ambientes	✓	✓	Apenas variáveis de ambiente	Apenas variáveis de ambiente
Integração com Docker ou contêineres	✓	✓	-	-
Suporte a plugins	✓	-	-	-
Suporte a rotinas de deploy	✓	✓	✓	✓
Forma de definição da pipeline	Arquivo de configuração dentro do repositório ou através de uma interface gráfica	Arquivo de configuração dentro do repositório	Arquivo de configuração dentro do repositório	Arquivo de configuração dentro do repositório

5.2.4 Centralização de *logs*

As ferramentas de centralização de *logs* foram deixadas de lado durante a execução deste projeto devido a limitações de tempo e pelo fato do time já realizar um bom trabalho administrando os *logs* gerados pela aplicação graças aos *frameworks* de desenvolvimento utilizados. Porém uma breve análise de algumas das principais ferramentas de centralização de *logs* foi realizada para que um estudo de suas vantagens pudesse ser feito e essa decisão pudesse ser tomada.

Uma das ferramentas mais descomplicada e robusta para centralização de logs se chama *Kibana*, que é uma aplicação *web* com interface simples e amigável que permite a visualização de e análise de dados diversos em tempo real. Basicamente ele funciona como uma ferramenta de visualização em conjunto do *Elastic Search*, que é o responsável pelo armazenamento dos arquivos de *log* e outras métricas da aplicação. Para utilização do *Kibana* é necessário configurar sua aplicação para enviar e armazenar os *logs* no *Elastic Search*, e com algumas configurações, preparar o *Kiabana* para receber consultas e exibir painéis informativos sobre as informações registradas(KIBANA, 2016).

Tabela 6 – Comparativo entre ferramentas de monitoramento de recursos disponíveis no mercado.

	Zabbix	**Cacti**	**Prometheus**	**Nagios**	**Icinga**
Hospedagem remota	-	-	-	-	-
Hospedagem local	✓	✓	✓	✓	✓
Código aberto	✓	✓	✓	✓	✓
Geração de gráficos	✓	✓	✓	✓	✓
Suporte a protocolo SNMP	✓	✓	✓	✓	✓
Suporte a protocolo JMX	✓	✓	✓	✓	✓
Suporte a protocolo IPMI	✓	✓	✓	✓	✓
Cliente para coleta customizada	✓	✓	✓	✓	✓
Suporte a alertas e notificações de situações críticas	✓	-	✓	✓	✓

Uma boa alternativa ao *Kibana* é o *GrayLogs* (GRAYLOG, 2016), que também funciona em conjunto do *Elastic Search*, porém uma das principais diferenças entre eles é que o *GrayLog* utiliza também o *Apache Kafka* como sistema intermediário de controle de mensagens que permite o particionamento do fluxo de dados entre *clusters* de máquinas, o que é ideal para aplicações de analise de *big data*. Diversas arquiteturas distribuídas podem se beneficiar desse modelo expandindo seu nível de controle e monitoramento do estado da aplicação.

5.2.5 Conteinerização e provisionamento

Devido a limitações de tempo e analisando os resultados obtidos com os experimentos iniciais, apenas uma *build pipeline* com *deploy* automático configurado no *Jenkins* e no *GitLab* já se mostraram suficiente para elevar o nível de qualidade do processo atual, porem é preciso comentar sobre o uso de contêineres e provisionamento de máquinas virtuais, uma vez que essas tecnologias se mostram cada vez mais presentes em ambientes de desenvolvimento integrado, como pode ser observado em uma pesquisa publicada em 2016 pela RightScale (2016).

Os contêineres são máquinas virtuais capazes de serem executadas sem a necessidade de carregamento de supervisores, o que as tornam mais leves que as máquinas virtuais convencionais. A utilização de contêineres para implantar um *software* em produção traz vantagens como controle total do ambiente de execução da aplicação, mais organização para configurações, versionamento de infra-estrutura, maior controle de dependências a nível de sistema operacional, maior aproveitamento do *hardware*, dentre outras (TURNBULL,

2016).

Com a utilização de contêineres, o time de operações de uma empresa pode trabalhar com provisionamento de máquinas virtuais onde não há a necessidade de configuração manual de servidores, basta criar os *scripts* de instanciação do contêiner, embutir a aplicação e inicia-lo no servidor de produção.

O provisionamento e implantação dos contêineres feito puramente com *Docker* muitas vezes não é tão simples quanto poderia ser, é aqui que entram as ferramentas de orquestração. As ferramentas de orquestração mais populares segundo a pesquisa da RightScale (2016) são o *Puppet* e o *Chef*, que muitas das vezes são utilizadas em conjunto. Além dessas duas, outras bem populares são o *Kubernetes* (KUBERNETES, 2016) e o *Open Shift* (OPENSHIFT, 2016a).

Essas ferramentas permitem a construção de arquivos de descrição de infraestrutura, onde o desenvolvedor pode especificar quais imagens de contêineres serão instanciadas, quais serão os volumes de dados persistentes que ficarão salvos em disco, quais são as dependências entre contêineres para que o serviço funcione corretamente, quantas réplicas de cada serviço devem ser instanciadas, qual política de reinicialização deve ser aplicada a cada serviço em caso de falha, quais os requisitos de consumo para o levantamento de mais réplicas, dentre várias outras especificações.

Outra vantagem dessa abordagem é a construção de ambientes de testes e homologação idênticos ao ambiente de produção com uma quantidade mínima de trabalho necessário, reduzindo os riscos de comportamentos inesperados devido a ambientes diversificados.

5.3 Implantação do ambiente de desenvolvimento integrado

No início dos procedimentos de implantação das ferramentas escolhidas, foi realizado uma tentativa de instalação de uma ferramenta chamada *Fabric8*. Essa ferramenta promete facilitar o processo de instalação e configuração de diversos componentes de *DevOps* como o *GitLab*, *Jenkins*, *Kibana* e etc. Feita para funcionar em conjunto do *Kubernetes* ou *Open Shift*, o *Fabric8* traz uma interface gráfica que permite a iniciação de contêineres que executam as ferramentas de *DevOps* com apenas um único clique, além de permitir a construção de novos serviços com a necessidade de pouquíssimas configurações.

Foram realizadas tentativas de instalação do *Fabric8* seguindo o passo a passo encontrado na documentação (FABRIC8, 2016). No dia que teve início a implantação desse ambiente existiam 4 formas diferentes de se realizar a instalação da ferramenta, cada uma dependente de um serviço adicional diferente. Uma delas utilizava o *Vagrant*, outra o *MiniKube*, outra o *MiniShift* e a última necessitava do *Open Shift Origin* instalado no servidor.

A primeira tentativa de instalação foi feita utilizando uma máquina virtual provisionada pelo *Vagrant* seguindo o próprio tutorial da página do *Fabric8*, porém após a instalação um erro ocorria sempre que uma tentativa de criação de projeto era realizada. Uma pesquisa foi realizada em tentativa de reparar esse erro e a melhor resposta obtida indicava que o problema acontecia devido a possível incompatibilidade entre a versão do sistema operacional do servidor e um *plugin* do *Vagrant* chamando *Landrush*, que é responsável pela descoberta de DNS dentro do ambiente virtualizado. Devido a impossibilidade de se alternar a versão do sistema operacional nessa etapa do projeto, essa possibilidade foi descartada.

Uma segunda tentativa de instalação foi feita utilizando o *MiniKube*, porém essa forma tinha uma característica negativa de não persistir os projetos e a informação criada após a reinicialização do sistema, ou seja, sempre que o servidor era reiniciado todas as informações registradas eram perdidas. Devido a este aspecto essa opção foi descartada.

A terceira tentativa foi realizada utilizando o *MiniShift*, mas este apresentou o mesmo comportamento do *MiniKube* e também foi descartado.

A quarta tentativa foi feita realizando a instalação do *Open Shift Origin* através de uma imagem de contêiner fornecida pela *Red Hat* no servidor(OPENSHIFT, 2016b). Nessa abordagem, um serviço chamado *Router* não conseguia as permissões adequadas para ser inicializado corretamente.

Devido a diversidade de problemas existentes em métodos alternativos de instalação dessa ferramenta ela foi descartada. Foi decidido que seria melhor ter um pouco mais de trabalho realizando a instalação de cada componente individualmente do que aplicar uma solução que parece ainda ser instável.

5.3.1 Instalação do *Docker*

Para simplificar a instalação das ferramentas e já preparar o servidor para execução de ambientes de testes, foi decidido instalar o *Docker* e optar sempre por versões conteinerizadas das ferramentas.

Os procedimento necessários para instalação do *Docker* podem ser encontrados na documentação oficial (DOCKER, 2016a). Para o ambiente da empresa, o *script* de instalação utilizado está disponível no Apêndice A.

5.3.2 Instalação do sistema de controle de versão

O passo a passo da instalação do *GitLab*, que foi o sistema de controle de versão escolhido, foi seguido de acordo com as instruções de um repositório da versão conteinerizada da ferramenta (GITHUB, 2016a).

O primeiro passo é a realização do *download* do arquivo docker-compose.yml a partir do seguinte comando no terminal:

```
$ wget https://raw.githubusercontent.com/sameersbn/docker-gitlab/
master/docker-compose.yml
```

Após o *download* do arquivo, foi necessário altera-lo ajustando algumas configurações indicando e usuário e senha administrativa e a porta em que o serviço estará disponível.

Depois bastou executar o *Docker Compose* no diretório onde se encontra este arquivo que a aplicação ficou disponível.

5.3.3 Instalação da ferramenta de integração contínua

O procedimento para instalação do *Jenkins* foi realizado de acordo com as instruções da imagem oficial disponibilizada no *Docker Hub* (DOCKER, 2016b).

Um único comando é necessário para instalação:

```
$ docker run -p 8080:8080 -p 50000:50000 \
  -v /your/home:/var/jenkins_home jenkins
```

5.3.4 Instalação da ferramenta de monitoramento

A instalação do *Zabbix* foi realizada utilizando um arquivo de configuração para o *Docker Compose* encontrado no *GitHub* (GITHUB, 2016b). Após o *download* do foi necessário altera-lo ajustando a porta em que a aplicação deve ser executada. Após o ajuste do arquivo de configuração basta executar o *Docker Compose* no diretório onde ele se encontra.

Uma ferramenta adicional foi instalada neste estágio para a criação de *dashboards* de monitoramento, graças a recomendações do próprio guia de instalação do *Zabbix* (ZABBIX, 2016).

A ferramenta se chama *Grafana*, e para sua instalação basta executar os comandos:

```
$ docker run -d -v /var/lib/grafana --name \
  grafana-xxl-storage busybox:latest
$ docker run \
  -d \
  -p 3000:3000 \
  --name grafana-xxl \
  --volumes-from grafana-xxl-storage \
  monitoringartist/grafana-xxl:latest
```

A *dashboard* foi configurada para exibir o monitoramento em tempo real do consumo de memória, espaço disponível em disco, uso de CPU e tráfego de rede na máquina que executa o ambiente de produção da aplicação. Foi utilizado um guia disponibilizado pelo Zabbix Brasil (2016) para realizar a customização da *dashboard*.

Parte II

Resultados

6 Análise do novo cenário

Este capítulo apresenta uma análise das características da empresa no fim do desenvolvimento do projeto, demonstrando as mudanças mais significativas e apresentando as vantagens e desvantagens do novo processo de desenvolvimento.

6.1 Coleta de dados

A etapa de coleta de dados analíticos sobre o novo processo de desenvolvimento teve início no dia 11 de outubro de 2016. Durante essa etapa buscamos identificar quais foram as mudanças nas variáveis dependentes analisadas no começo do projeto, relacionadas a tarefas que fogem ao escopo de desenvolvimento e levantamento de requisitos, como atividades de compilação do código fonte, execução de testes manuais, configuração de ambientes de trabalho, etc. O acompanhamento e registro desses dados ocorreu até o dia 4 de novembro de 2016.

As tabelas 7 e 8 apresentam dados quantitativos referentes as etapas do novo processo de desenvolvimento. Graças a essas informações pode-se analisar os resultados obtidos com a implantação das novas metodologias e boas práticas de desenvolvimento.

A Tabela 7 apresenta dados analíticos referentes aos repositórios de código fonte da empresa.

Tabela 7 – Dados do repositório de código fonte no fim do projeto

Métrica	**Quantidade**
Número de *commits* realizados	434
Número de artefatos gerados	115
Número de mesclagem de código realizados	19
Número de mesclagem de código rejeitadas	3
Número de *bugs* reportados pelo cliente	6
Número de atividades desenvolvidas pelo time que foram registradas no sistema de controle de atividades	34
Tempo médio de execução da *build pipeline*	5 minutos

A Tabela 8 demonstra o tempo necessário por entrega de uma nova versão do sistema. Consideramos o tempo necessário para construção de um artefato executável e a sua disponibilização em ambiente de produção. Esta etapa engloba tentativas má sucedidas de construção e entrega de uma versão estável e sem *bugs*.

Tabela 8 – Características das entregas no fim do projeto

Data da entrega	**Tempo necessário**	**Número de tentativas má sucedidas até a conclusão da entrega**
14/10/2016	13 segundos	0
18/10/2016	17 segundos	0
23/10/2016	1 minuto e 21 segundos	0
25/10/2016	1 minuto e 17 segundos	0
26/10/2016	16 segundos	0
28/10/2016	12 segundos	0
31/10/2016	14 segundos	0

Apesar do tempo investido com a entregas no novo processo ser significativamente menor que no antigo, vale ressaltar que o time investe mais tempo no desenvolvimento de testes automatizados, porém o esforço é reduzido assim que cada teste é criado, pois não há necessidade de testes manuais.

Após a coleta de dados, uma entrevista foi realizada com o time de desenvolvimento onde foi solicitado que cada um dissesse seu ponto de vista sobre as vantagens e desvantagens do novo processo de desenvolvimento. As principais observações identificadas na entrevista estão pontuadas a seguir:

- O time notou uma significativa redução do tempo necessário para identificar e resolver problemas, uma vez que as alterações realizadas são sempre pequenas e constantes, o campo de busca pela alteração que injetou o *bug* é reduzido. E graças a automação do processo de entrega, as novas versões de correção são disponibilizadas rapidamente.

- Uma vez que as correções são entregues mais rapidamente, o time reconheceu maior satisfação do cliente, que apesar de ainda encontrar alguns *bugs*, se mostrou bastante satisfeito com o tempo de resposta.

- Apesar de todas essas vantagens algumas dificuldades ainda são encontradas por alguns membros do time que precisaram aprender a utilizar várias ferramentas novas.

- O processo de *code review* algumas vezes se mostra desnecessariamente burocrático, pois se há necessidade de uma correção ser disponibilizada imediatamente, é necessário interromper outro desenvolvedor para efetuar o *code review*. Apesar desse trabalho adicional, o time ainda se mostra satisfeito ao utilizar essa prática, pois isso fornece mais segurança de que as alterações são estáveis e economiza tempo de investigação futura.

6.2 Comparativos com o cenário anterior

A Tabela 9 apresenta um comparativo entre os dados coletados antes e depois da conclusão do projeto, evidenciando as melhorias obtidas. Os dados exibidos na tabela representam uma média mensal dos dados coletados.

Tabela 9 – Comparativo baseado em uma média mensal dos dados coletados antes e depois da conclusão do projeto.

	Antes	**Depois**
Tempo médio necessário para realizar a entrega de uma nova versão	5 horas	33 segundos
Média de tentativas má sucedidas até a conclusão de uma entrega	3	0
Número de commits realizados	62	434
Número de artefatos gerados	32	115
Número de mesclagem de código realizadas	3	19
Número de mesclagem de código rejeitadas	1	3
Número de bugs reportados pelo cliente	10	6
Número de atividades desenvolvidas pelo time que foram registradas no sistema de controle de atividades	35	34
Tempo médio de execução de build pipelines	-	5 minutos

Ao compararmos os dados colhidos no fim do projeto com os do início, percebe-se uma melhoria significativa no tempo investido com entregas de novas versões e com a qualidade do produto oferecido.

Lembrando que os dados no início do projeto foram coletados por dois meses e no fim apenas um mês, percebe-se que agora o time realiza 7 vezes mais *commits* e 6 vezes mais mesclagem de código, o que é reflexo da prática integração contínua. Como cada alteração é menor e é testada automaticamente graças as *build pipelines* do *Jenkins* e *GitLab*, o esforço necessário para identificar e corrigir *bugs* é reduzido.

Percebe-se também que o time realiza duas vezes mais entregas, e consegue ser em média 237 vezes mais rápido para disponibilizar uma nova versão do que era inicialmente, graças ao fato do código do repositório ser sempre confiável e do processo de entrega ser automatizado com auxílio do *Jenkins*.

A quantidade de *bugs* reportados pelo cliente também apresentou uma redução significativa. Devido ao tempo necessário de entrega ser menor, o tempo gasto para corrigir *bugs* e colocar a correção em produção também é reduzido. O *Zabbix* colabora nesse cenário

por apresentar indicativos de consumo de recursos no servidor, agilizando o processo de investigação graças as informações coletadas.

E por último, um fator positivo pode ser notado nas respostas dadas pelo time de desenvolvimento, que aparenta estar mais satisfeito com o trabalho realizado e tem mais tempo pra investir no que é realmente importante para o projeto.

7 Sugestões para outras empresas

De todas as atividades executadas durante este projeto, a mais complexa foi o ajuste realizado nas rotinas do time de desenvolvimento, e isso só pôde ser realizado graças ao apoio e incentivo de todos os colaboradores durante todo o processo. Se não houvesse o *feedback* que permitisse compreender porquê o problema existe, não seria possível chegar a um resultado efetivo.

A parte de escolha e instalação de ferramentas não apresentou dificuldades significativas, mas entender como elas funcionam e como utilizá-las para colocar as metodologias em prática foi um grande desafio.

É importante sempre compartilhar a visão do negócio com o time e permitir que todos compreendam o trabalho e o papel de todos dentro da empresa. Isso facilita a comunicação, evita conflitos desnecessários e colabora para que não haja disputa entre setores. É extremamente importante fazer com que todos trabalhem juntos.

E por último, o mais importante, incentivar as pessoas a aprender boas práticas e melhorar sempre. Todas as metodologias aplicadas aqui são explicadas em diversas literaturas, mas colocá-las em prática não é tão simples como parece, é necessário muito estudo e experiência, e às vezes, tentativa e erro.

8 Trabalhos futuros

Os próximos passos de aperfeiçoamento deste projeto envolvem a implantação contínua, a centralização de logs e avaliação em tempo real.

Na etapa atual de desenvolvimento dos projetos da empresa ainda não é um requisito fornecer aplicações altamente tolerante a falhas e escaláveis, mas sabe-se que chegará um momento onde isso será necessário, e para isso precisamos modificar a etapa final de implantação para suportar contêineres replicáveis, o que exigirá o uso de orquestradores e mais ferramentas.

Com o crescimento distribuído da aplicação, imaginamos também que chegará o ponto onde será necessário centralizar os logs e as informações de avaliação da aplicação. Hoje a quantidade reduzida de módulos não traz dificuldades de rastreio de *bugs* ou registros de acontecimentos, mas isso pode complicar ao longo do tempo.

Para os projetos desenvolvidos pela empresa que são oferecidos como produtos ou serviços proprietários, será necessário também a automação do processo de provisionamento de infraestrutura e *hardware*, para isso será necessário o uso avançado de orquestradores e contêineres.

9 Conclusão

Este trabalho mostra que é extremamente vantajoso a utilização de metodologias ágeis bem como a automação de operações e entrega contínua. Os resultados mostram que o tempo necessário com tarefas de entrega de *software* em um ambiente integrado e automatizado representa 0,4% do tempo necessário em cenário onde essas tarefas são realizadas manualmente.

Além do tempo economizado, percebe-se também uma melhoria na qualidade dos produtos e serviços oferecidos, uma vez que acontecem menos problemas no momento da entrega devido ao fato do código estar sempre estável.

Quanto a aplicação dessas práticas no ambiente de desenvolvimento, conclui-se que não faltam ferramentas gratuitas no mercado e que a instalação dessas ferramentas é o menor dos problemas diante da dificuldade de avaliar o processo de desenvolvimento e estabelecer mudanças que afetam a rotina dos colaboradores da empresa.

Concluímos que o principal fator para o estabelecimento de um ambiente de desenvolvimento eficaz é a colaboração de todos os membros do time e a busca constante por automação e melhoria do processo.

Referências

ABES. Mercado brasileiro de software: panorama e tendências. 2016. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/ABES-Publicacao-Mercado-2016.pdf>. Acesso em: 17 out. 2016. Citado na página 8.

AMAZON. Aws lambda, detalhes do produto. 2016. Disponível em: <https://aws.amazon.com/pt/lambda/details/>. Acesso em: 17 out. 2016. Citado na página 7.

BECK, K. et al. Manifesto para o desenvolvimento ágil de software. 2001. Disponível em: <http://www.manifestoagil.com.br/>. Acesso em: 12 out. 2016. Citado na página 11.

DEVMEDIA. Controles de versão para projetos java. 2016. Disponível em: <http://www.devmedia.com.br/controles-de-versao-para-projetos-java/26056>. Acesso em: 10 out. 2016. Citado na página 24.

DEVOPS.COM. 9 open source devops tools we love. 2015. Disponível em: <https://devops.com/9-open-source-devops-tools-love/>. Acesso em: 10 out. 2016. Citado na página 25.

DOCKER. Install docker on ubuntu. 2016. Disponível em: <https://docs.docker.com/engine/installation/linux/ubuntulinux/>. Acesso em: 14 out. 2016. Citado na página 29.

DOCKER. library/jenkins - docker hub. 2016. Disponível em: <https://hub.docker.com/_/jenkins/>. Acesso em: 14 out. 2016. Citado na página 30.

E-BUSINESS. Entendendo o manifesto Ágil. 2015. Disponível em: <http://www.ebusinessconsultoria.com.br/infonews/entendendo-o-manifesto-agil>. Acesso em: 10 out. 2016. Citado na página 7.

EXTREME Programming, XP: Metodologia de desenvolvimento ágil. 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/xp/>. Acesso em: 12 out. 2016. Citado na página 11.

FABRIC8. Getting started: Fabric8 documentation. 2016. Disponível em: <https://fabric8.io/guide/getStarted/index.html>. Acesso em: 12 out. 2016. Citado na página 28.

GARTNER RESEARCH. Big data drives rapid changes in infrastructure and $232 billion in it spending through 2016. Out. 2012. Citado na página 13.

GITHUB. Dockerized gitlab. 2016. Disponível em: <https://github.com/sameersbn/docker-gitlab>. Acesso em: 14 out. 2016. Citado na página 29.

GITHUB. zabbix-xxl/docker-compose.yml at master - monitoringartist/zabbix-xxl. 2016. Disponível em: <https://github.com/monitoringartist/zabbix-xxl/blob/master/Dockerfile/zabbix-2.4/docker-compose.yml>. Acesso em: 14 out. 2016. Citado na página 30.

GRAYLOG. Graylog: Open source log management. 2016. Disponível em: <https://www.graylog.org/>. Acesso em: 10 out. 2016. Citado na página 27.

HUMBLE, J.; FARLEY, D. *Entrega Contínua*: Como entregar software de forma rápida e confiável. [S.l.]: BOOKMAN, 2014. Citado na página 13.

INFOQ. Sistemas de controle de versão distribuído: Um guia não tão rápido. 2008. Disponível em: <https://www.infoq.com/br/articles/dvcs-guide>. Acesso em: 10 out. 2016. Citado na página 24.

KIBANA. Kibana: Explore, visualize, discover data. 2016. Disponível em: <https://www.elastic.co/products/kibana>. Acesso em: 10 out. 2016. Citado na página 26.

KUBERNETES. Kubernetes: Production-grade container orchestration. 2016. Disponível em: <http://kubernetes.io/>. Acesso em: 10 out. 2016. Citado na página 28.

NEW RELIC. Explorando a devops: O que ela é e por que é importante para sua empresa. 2016. Disponível em: <https://try.newrelic.com/rs/412-MZS-894/images/NavigatingDevOps_ptbr-A4.pdf>. Acesso em: 14 out. 2016. Citado na página 13.

OPENSHIFT. Openshift: Paas by red hat, built on docker and kubernetes. 2016. Disponível em: <https://www.openshift.com/>. Acesso em: 10 out. 2016. Citado na página 28.

OPENSHIFT. Setting up a cluster. 2016. Disponível em: <https://docs.openshift.org/latest/getting_started/administrators.html>. Acesso em: 13 out. 2016. Citado na página 29.

ORACLE. Oracle developer cloud service. 2016. Disponível em: <https://cloud.oracle.com/en_US/opc/developer-service/features>. Acesso em: 17 out. 2016. Citado na página 7.

RED HAT. Open shift features. 2016. Disponível em: <https://www.openshift.com/features/>. Acesso em: 17 out. 2016. Citado na página 7.

REISSWITZ, F. *Análise de Sistemas*: Qualidade de software. [S.l.: s.n.], 2013. v. 7. Citado na página 19.

RIGHTSCALE. State of the cloud report: Devops trends. 2016. Disponível em: <http://assets.rightscale.com/uploads/pdfs/rightscale-2016-state-of-the-cloud-report-devops-trends.pdf>. Acesso em: 10 out. 2016. Citado 2 vezes nas páginas 27 e 28.

SCRUM: Metodologia ágil para gestão e planejamento de projetos. 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso em: 12 out. 2016. Citado na página 11.

SILVEIRA, P. et al. *Introdução à Arquitetura e Design de Software*: Uma visão sobre a plataforma java. [S.l.]: Elsevier, 2011. Citado 2 vezes nas páginas 12 e 19.

SITEPOINT. Version control software in 2014: What are your options? 2014. Disponível em: <https://www.sitepoint.com/version-control-software-2014-what-options/>. Acesso em: 10 out. 2016. Citado na página 24.

SLANT. What are the best alternatives to github for open source projects? 2016. Disponível em: <https://www.slant.co/topics/5335/~alternatives-to-github-for-open-source-projects>. Acesso em: 10 out. 2016. Citado na página 24.

STACK OVERFLOW. Stack overflow developer survey 2016 results. 2016. Disponível em: <https://stackoverflow.com/research/developer-survey-2016>. Acesso em: 10 out. 2016. Citado na página 7.

STACKSHARE. Code collaboration & version control. 2016. Disponível em: <http://stackshare.io/code-collaboration-version-control>. Acesso em: 10 out. 2016. Citado na página 24.

STACKSHARE. Continuous integration. 2016. Disponível em: <http://stackshare.io/continuous-integration>. Acesso em: 10 out. 2016. Citado na página 24.

STACKSHARE. Continuous integration. 2016. Disponível em: <http://stackshare.io/monitoring-tools>. Acesso em: 10 out. 2016. Citado na página 25.

TURNBULL, J. *THE DOCKER BOOK*: Containerization is the new virtualization. [S.l.: s.n.], 2016. Citado na página 28.

XEBIALABS. Periodic table of devops tools. 2016. Disponível em: <https://xebialabs.com/periodic-table-of-devops-tools/>. Acesso em: 10 out. 2016. Citado na página 23.

ZABBIX. Dockerized zabbix. 2016. Disponível em: <https://www.zabbix.org/wiki/Dockerized_Zabbix>. Acesso em: 14 out. 2016. Citado na página 30.

ZABBIX BRASIL. Integração do zabbix com grafana. 2016. Disponível em: <http://zabbixbrasil.org/?p=1674>. Acesso em: 14 out. 2016. Citado na página 31.

Apêndices

APÊNDICE A – Script de instalação do Docker no Ubuntu 14.04.3

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 \
   --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main" \
   | sudo tee /etc/apt/sources.list.d/docker.list
sudo apt-get update
sudo apt-get install linux-image-extra-$(uname -r) \
   linux-image-extra-virtual
sudo apt-get install docker-engine
sudo service docker start
```