Relato de Experiência Sobre a Implantação de um Processo de Entrega Contínua em uma Organização da Indústria Financeira

Everton Gomede, Rafael T. Silva, and Rodolfo M. Barros

Departamento de Computação
Universidade Estadual de Londrina
Londrina, Brasil
{evertongomede,rafaelthiago}@gmail.com,rodolfo@uel.br
http://www.uel.br

Resumo Entregar *software* de qualidade continuamente é um desafio para muitas organizações. Isto deve-se a fatores como gerenciamento de configuração, controle de código fonte, revisão aos pares, planejamento de entregas, auditorias, *compliance*, integração contínua, testes, implantações, gerenciamento de dependências, migração de bancos de dados, criação e gerenciamento de ambientes de testes e produção, entre outros. Para superar esses desafios, este artigo apresenta um relato de experiência sobre a implantação de um processo de entrega contínua em uma organização da indústria financeira. Este processo promove os artefatos produzidos por desenvolvedores de maneira gerenciada para o ambiente de produção, permitindo a rastreabilidade fim a fim entre requisitos e executáveis. Como resultado, obteve-se um ecossistema de ferramentas e técnicas avaliadas, testadas e colocadas em produção com o objetivo de suportar o processo. Adicionalmente, as lições aprendidas e recomendações foram catalogadas para que possam contribuir a projetos similares.

Keywords: Entrega Contínua, Gerenciamento de Configuração, Qualidade de *Software*, Rastreabilidade

1 Introdução

O Processo para Entrega de *Software* ou *Software Delivery Process* (SDP) consiste de várias tarefas com o objetivo de promover os artefatos criados para o ambiente de produção [1]. Estas tarefas podem ocorrer tanto do lado produtor quanto consumidor. Pelas características únicas de cada produto de *software*, um processo que possa ser utilizado em vários contextos, provavelmente não possa ser definido. Portanto, um SDP deveria ser interpretado como um *framework* a ser customizado de acordo com os requisitos e características de cada produto.

Esta customização, geralmente, faz com que o SDP seja executado de forma manual [2]. O ambiente de produção é configurado de maneira artesanal pela equipe de infraestrutura utilizando terminais e/ou ferramentas de terceiros. Os

artefatos são copiados de um servidor de integração contínua para o ambiente de produção e, eventualmente, alguns dados e/ou metadados são ajustados para que, então, o *software* seja liberado para operação.

Todavia, este processo possui algumas fragilidades. A primeira delas está relacionada com a previsibilidade, o que pode agravar o risco e aumentar o tempo de indisponibilidade, caso ocorra alguma situação inesperada. Adicionalmente, o fator de reprodutibilidade pode comprometer o diagnóstico de problemas ocorridos pós-implantação. Finalmente, este processo não é auditável e não permite uma recuperação precisa de todos os eventos realizados para as entregas efetuadas em um determinado período.

Adicionalmente, a indústria financeira está condicionada a regras de *compliance*, que em português seria o equivalente a aderência à norma. Dentre elas, a mais conhecida e aplicada mundialmente é a Lei Sarbanes-Oxley. Esta lei visa garantir a criação de mecanismos de auditoria e segurança confiáveis nas empresas, de modo a mitigar riscos aos negócios, evitar a ocorrência de fraudes ou assegurar que haja meios de identificá-las quando ocorrem, garantindo assim, a transparência na gestão [3]. Portanto, estas regras se aplicam, também, aos produtos de *software* que suportam a operação e gestão de empresas desse tipo.

Existe um interesse crescente em práticas para superar estes problemas [5]. Tais práticas são conhecidas como Entrega Contínua de *Software ou Software Continuous Delivery* (SCD), ou ainda *DevOps*, sendo definida como a capacidade de publicar *software* sempre que necessário. Esta publicação pode ser semanal, diária ou a cada alteração enviada ao repositório de código. A frequência não é o foco principal, mas sim, a habilidade de entrega quando necessário [2].

Este tipo de abordagem tem uma importância relevante no desenvolvimento de *software*, pois ajuda os responsáveis pela entrega a compreenderem bem seu processo e, consequentemente, melhorá-lo. Tais melhoras podem ser em termos de automação, redução do tempo de entrega, redução de retrabalho, redução de risco, entre outras. Dentre elas, a principal está na capacidade de dispor de uma versão do *software*, pronta para entrega, a cada novo código adicionado ao repositório [2].

Neste contexto, este trabalho apresenta a experiência de implantação de um processo de entrega contínua em uma organização da indústria financeira. O objetivo principal é contribuir com um conjunto de lições aprendidas e apresentar um possível *setup* que pode ser utilizado em abordagens desta natureza. Adicionalmente, algumas recomendações e questões futuras são discutidas. Questões de arquitetura, gerenciamento de projeto e demais dimensões relacionadas ao desenvolvimento de *software* foram omitidas para que o trabalho concentre-se nas questões de entrega contínua.

Para isto, este artigo está dividido em cinco seções elementares, incluindo esta introdução. Na Seção 2, apresentam-se os conceitos fundamentais e os trabalhos relacionados. Na Seção 3, apresenta-se a experiência de implantação do processo para Entrega Contínua (EC). A Seção 4 apresenta os resultados e um conjunto de lições aprendidas. Finalmente, a Seção 5 apresenta as conclusões, recomendações e sugestões para trabalhos futuros.

2 Conceitos Fundamentais e Trabalhos Relacionados

Há uma relação entre a qualidade dos produtos de *software* e qualidade do processo utilizado para construí-los. A implantação de um processo tem como objetivo o aumento na qualidade do produto, a redução do retrabalho, o aumento de produtividade, a redução do tempo de entrega, aumento da rastreabilidade, maior previsibilidade e precisão das estimativas [2]. De forma geral, um processo de desenvolvimento de *software* contém as atividades mostradas na Figura 1.

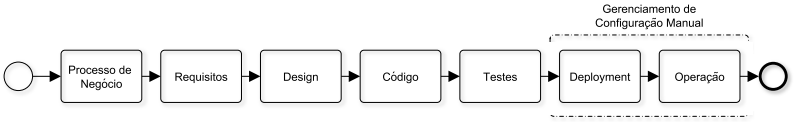

Figura 1. Processo de desenvolvimento de *software* [2]

As tarefas de gerenciamento de configuração das atividades *deployment* e operação, destacadas na Figura 1, geralmente são executadas de forma manual [2]. Tal prática, segundo Humble e Farley [2], é acompanhada dos anti-padrões:

1. Implantar *software* manualmente: deveria haver somente duas tarefas a serem executadas manualmente; (1) escolher a versão e (2) escolher o ambiente.
2. Implantar depois que o desenvolvimento estiver completo: é preciso integrar todas as atividades do processo de desenvolvimento e fazer com que os envolvidos trabalhem juntos desde o início do projeto.
3. Gerência de configuração manual dos ambientes de produção: todos os aspectos dos ambientes configurados deveriam ser aplicados a partir do controle de versão, em um processo automatizado.

Deste contexto, surgem as práticas de Entrega Contínua de *Software* (SCD). Esta é uma disciplina de desenvolvimento na qual constrói-se o *software* de maneira que este possa ser liberado para produção a qualquer momento, reduzindo ao mínimo o tempo de ciclo [2]. Para auxiliar este tipo de abordagem de entrega de *software*, desde a construção até a implantação, Humble e Farley apresentaram o *Deployment Pipeline* (DP), um padrão para automatizar o processo de SCD. Embora cada organização possa ter uma implementação deste padrão, de forma genérica ela é constituída das atividades mostradas na Figura 2.

A cada mudança, os artefatos são promovidos para a próxima instância do *pipeline* por meio de um conjunto de tarefas automatizadas. O primeiro passo do *pipeline* é criar os executáveis e instaladores a partir do repositório de código, em um processo conhecido com Integração Contínua (IC). As demais atividades executam uma série de testes para garantir que os executáveis possam ser publicados. Se a versão candidata passar em todos os testes e critérios, então, ela pode ser publicada [2].

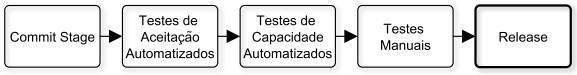

Figura 2. O *deployment pipeline* [2]

Para implementar este *pipeline*, algumas abordagens foram apresentadas. Dentre elas, Krusche e Alperowitz [5] descreveram a implementação de um processo para SCD em um ambiente de múltiplos projetos. O objetivo dos autores foi obter a habilidade de publicar *software* para seus clientes com apenas alguns cliques. A principal contribuição deste trabalho foi mostrar que, os desenvolvedores que trabalharam nos projetos com SCD entenderam e aplicaram os conceitos sendo convencidos dos benefícios do mesmo.

Bellomo et al. [4] apresentaram um arcabouço arquitetural em conjunto com táticas para projetos que tenham como objetivo SCD. A principal contribuição deste trabalho é uma coleção de táticas de SCD com o objetivo de obter os produtos de *software* executando com um maior nível de confiabilidade e monitoramento no ambiente de produção.

Fitzgerald e Stol [6] publicaram as tendências e desafios relacionados ao que os autores chamaram de "*Continuous* *", ou seja, todos os tópicos vinculados a entrega de *software* que possam ser classificados como contínuos. Os autores abordaram assuntos como; Integração Contínua (IC), Publicação Contínua (PC), Testes Contínuo (TC), *Compliance* Contínuo (CC), Segurança Contínua (SC), Entrega Contínua (EC), entre outros. Um ponto importante deste trabalho é a distinção entre Entrega Contínua e Publicação Contínua. Para os autores, Publicação Contínua é a capacidade de colocar produtos de *software* em produção de maneira automatizada. Tal definição é complementar a definição de entrega contínua de *software* dada anteriormente.

Embora todos estes trabalhos tenham uma natureza prática, nenhum deles apresentou quais ferramentas foram utilizadas, quais as recomendações para cenários similares e quais foram as lições aprendidas durante a implantação. Portanto, o trabalho apresentado neste artigo procura suprir estas lacunas.

3 Descrição da Experiência

A organização, na qual o processo de SCD foi implantado, atua na indústria financeira e possui uma participação no mercado brasileiro de aproximadamente 3%, com mais de 100 mil clientes e mais de 2,5 bilhões de recursos administrados. Ambos contextos, político e econômico, são favoráveis à empresa, proporcionando assim, um crescimento acelerado e, consequentemente, uma exposição maior a fraudes e demais fragilidades tratadas na Lei Sarbaney-Oxley [3].

Em conjunto a este cenário, do ponto de vista de entrega de produtos de *software*, os problemas também foram agravados pelo crescimento. Dentre eles, os principais estão relacionados à baixa produtividade, pouca visibilidade e principalmente muitos pontos de auditoria relacionados à baixa rastreabilidade e

baixa maturidade do processo de desenvolvimento de *software*. As causas destes problemas foram levantadas e catalogadas por meio de sessões de *brainstorming*, com a equipe de desenvolvimento e infraestrutura. Algumas causas estão presentes em todos os problemas. Em uma análise de Pareto pode-se observar que as causas (1) Falta de processo, (2) Falta de ferramentas e (3) Falta de cultura correspondem a mais de 50% dos problemas identificados. Isto é mostrado na Figura 3.

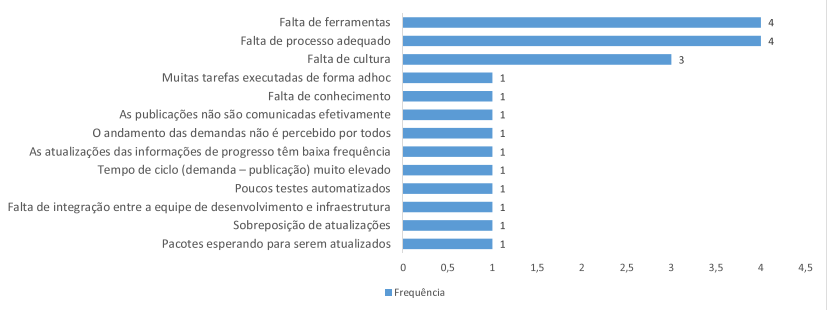

Figura 3. Causas identificadas e suas frequências

Portanto, para resolver estas causas, a implantação de um processo em conjunto com ferramentas adequadas pode auxiliar na mudança da cultura e, consequentemente, aumentar o nível de maturidade em publicação de produtos de *software*. Para isto, o primeiro passo foi identificar em qual o nível de maturidade o processo da organização estava. Para esta classificação, utilizou-se o modelo de maturidade sugerido por Humble e Farley [2]. Este modelo classifica o *pipeline* de entrega em 3 dimensões; (1) processo, (2) automação e (3) colaboração e em 5 níveis; (1) inicial, (2) gerenciado, (3) definido, (4) controlado e (5) otimizado. A Figura 4 mostra este modelo.

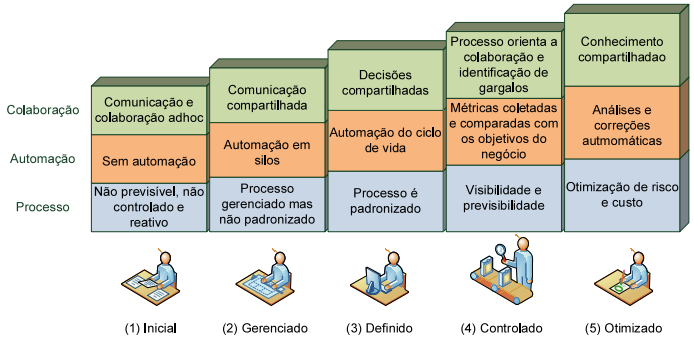

Figura 4. Modelo de maturidade de entrega contínua de *software* [2]

A utilização deste modelo foi motivada pela necessidade de uma métrica para medir o antes e o depois da implantação e, principalmente, por ser o modelo aceito pelas organizações para medir o nível de maturidade dos processos de SCD [2]. Desta forma, o processo de entrega de *software* da organização foi classificado como (1) inicial pelos seguintes motivos:

- Processo
  - O processo não é previsível em termos de tempo e efeitos colaterais
  - A publicação dos produtos de *software* é feita de forma manual
  - O processo é reativo, sendo disparado por necessidades pontuais
- Automação
  - Somente a integração dos códigos é feita de maneira automatizada
- Colaboração
  - A comunicação entre as equipes (devel e infra) é *adhoc*
  - A equipe de infraestrutura não tem acesso a todas as informações necessárias para a publicação de um produto de *software*

Assim, o processo foi planejado para alcançar o nível (2) Gerenciado de maturidade nas dimensões processo, automação e colaboração. Como resultado obteve-se o *setup* [1] exibido na Figura 5.

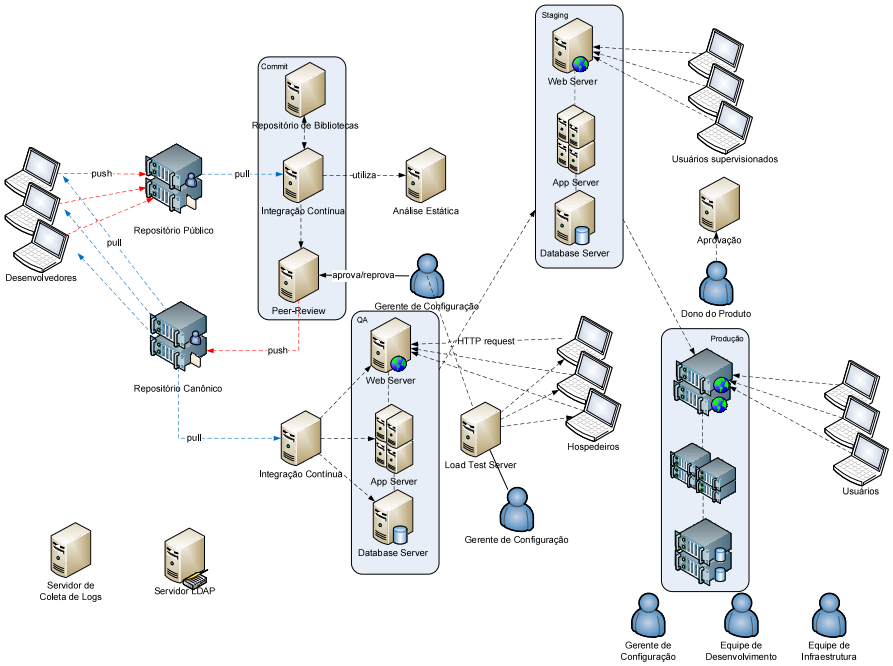

Figura 5. *Setup* do servidores e serviços para SCD

[1] Todas as configurações necessárias para o funcionamento do processo

Este *setup* possui 4 áreas; *Commit Stage* (CS), *Quality Assurance* (QA), *Staging* (ST) e Produção (PD). A área de *Commit* ou *Commit Stage* (CS) tem a responsabilidade principal de executar a integração contínua de todas as revisões de código enviadas ao repositório. Esta área é composta pelos serviços:

- Repositório de código público
  - Finalidade: receber as revisões de código que ainda não foram aprovadas
  - Ferramenta: Git
  - Funcionamento: possui um único *branch*, chamado *master,* que recebe todas as revisões de todos os desenvolvedores
- Integração contínua
  - Finalidade: integrar todos as revisões de código enviadas ao servidor
  - Ferramenta: Jenkins e Maven
  - Funcionamento: executa o trabalho de integração realizando testes unitários e adicionando o primeiro passo de aceitação no servidor de *peer-review*
- Análise estática
  - Finalidade: efetuar a análise do código enviado gerando relatórios de qualidade
  - Ferramenta: SonarQube
  - Funcionamento: a cada integração executa uma série de análises como métricas de tamanho, complexidade, cobertura de testes, cálculo de dependências, entre outras. Cria-se um *baseline* da qualidade do projeto neste momento
- *Peer-Review*
  - Finalidade: permitir a promoção ou rejeição dos códigos enviados ao repositório público para o repositório canônico
  - Ferramenta: Gerrit
  - Funcionamento: aprovação de dois passos, sendo o primeiro efetuao pelo servidor de integração contínua e o segundo pelo gerente de configuração. Caso a revisão passe por ambos, o código é promovido para o repositório canônico
- Repositório canônico
  - Finalidade: receber as revisões de código aprovadas
  - Ferramenta: Git
  - Funcionamento: possui um único *branch*, chamado *master*, que recebe todas as revisões do servidor de peer-review.
- Repositório de bibliotecas
  - Finalidade: armazenar as bibliotecas e componentes utilizados na integração
  - Ferramenta: Nexus
  - Funcionamento: as bibliotecas e componentes são instalados de forma automática ou manual no servidor ficando disponível para utilização no momento da integração

A Área de Quality Assurance (QA) tem a finalidade principal de realizar todos os testes automatizados e permitir que o Gerente de Qualidade execute testes manuais, como os testes exploratórios [2]. Esta área é composta pelos serviços:

- Integração contínua
  - Finalidade: obter uma cópia do código fonte e executar os testes de integração, funcional e carga automatizados
  - Ferramenta: Jenkins e Maven
  - Funcionamento: obtêm uma cópia do repositório canônico gera os executáveis, instala-os no servidor de bibliotecas e nos servidores de aplicação, página e banco. Depois disso, executa os testes de integração, funcional e carga
- Servidores de página, aplicação e banco
  - Finalidade: hospedar a aplicação para a execução dos testes
  - Ferramentas: pode variar de acordo com a tecnologia utilizada
  - Funcionamento: pode variar de acordo com a tecnologia utilizada
- Load test
  - Finalidade: executar o teste de carga contra os servidores de página, aplicação e banco
  - Ferramenta: Jmeter e Vagrant
  - Funcionamento: executa o script criado pelo gerente de qualidade alocando os hospedeiros conforme a necessidade do teste. Gera-se um *baseline* da carga suportada

A área de *Staging* tem o objetivo de prover aos usuários supervisionados e donos de produto um ambiente o mais próximo possível do ambiente de produção, para que os mesmos possam realizar os testes de aceitação. Estes testes estão relacionados com a experiência de usuário e a percepção dos mesmos com relação a como o produto de *software* atende aos requisitos especificados. Esta área possui uma cópia dos ambientes de operação, tanto em termos de sistemas operacionais, ferramentas e configurações, quanto em termos de dados. Os usuários supervisionados são usuários escolhidos para realizar os testes de aceitação. Eventualmente estes usuários estão no papel de dono de produto.

Embora existam técnicas para promover os artefatos do ambiente de QA para o ambiente de *Staging* de forma automática, a organização optou por manter este processo manual, mas ao mesmo tempo rastreado, pois publicação envolve a comunicação com os usuários supervisionados e donos de produto. Portanto, sempre antes de uma publicação uma comunicação ocorre entre o gerente de configuração, usuários supervisionados e donos de produto.

Finalmente, a promoção dos artefatos da área de *Staging* para a área de Produção é feita novamente de forma manual pelo gerente de configuração. Entretanto, membros da equipe de desenvolvimento e infraestrutura participam das tarefas de publicação. Novamente, a opção pela publicação manual foi de escolha da organização pelo mesmo motivo de comunicação, citando anteriormente.

Para manter a rastreabilidade do processo, no momento da publicação o determinado artefato recebe uma marcação com o seu código MD5 [2]. Desta forma, pode-se identificar qual a versão está sendo executada em produção.

Existem dois servidores que não foram mencionados; (1) o servidor de Coleta de Logs e (2) o servidor LDAP. O primeiro tem uma função muito importante

[2] O MD5 (Message-Digest algorithm 5) é um algoritmo de hash de 128 bits

no *setup*; obter todos os eventos que ocorreram por meio da indexação de logs. Isto facilita o diagnóstico, prove informações para criação de relatórios, alertas e *dashboards*. A ferramenta utilizada neste caso foi o Splunk.

O segundo servidor possui a função de permitir a autenticação e autorização à todos os servidores do *setup*. Isto é necessário pois torna-se custosa a manutenção de usuários ao longo de todos os servidores envolvidos de maneira individualizada, além de aumentar as falhas de segurança. A ferramenta utilizada neste caso foi o OpenLDAP.

A Tabela 1 mostra todas as ferramentas utilizadas no setup, explicando o motivo de sua escolha.

Tabela 1. Ferramentas Utilizadas

Finalidade	**Nome**	**URL**	**Motivo**
Integração Contínua	Jenkins	jenkins-ci.org	Ferramenta mais utilizada pela comunidade de desenvolvedores Java para integração contínua
Repositório de Código	Git	git-scm.com	Repositório de código flexível e de maior crescimento, em termos de usuário, no mundo
Build	Maven	maven.apache.org	Ferramenta mais utilizada pela comunidade de desenvolvedores Java para gerenciamento de build
Coleta de logs	Splunk	splunk.com	Ferramenta de coleta de logs mais utilizada pela indústria de tecnologia da informação
Peer-Review	Gerrit	code.google.com/p/gerrit	Escolhida pela facilidade de integração com o Git
Análise Estática	SonarQube	sonarqube.org	Ferramenta mais utilizada pela comunidade de desenvolvedores Java para análise estática
Teste de Carga	Jmeter	jmeter.apache.org	Ferramenta simples, open source e com as funcionalidades mais utilizadas para testes de carga
Repositório de Biblioteca	Nexus	sonatype.org/nexus	Ferramenta open source e mais completa para gerenciamento de bibliotecas
ALM	Redmine	redmine.org	Ferramenta com suporte a vários plug-ins e bem conhecida para gerenciamento de ALM
Database Migration	Flywaydb	flywaydb.org	Ferramenta open source, simples de configurar e utilizar
Instalação Automatizada	Vagrant	vagrantup.com	Ferramenta gratuita e utilizada por grandes corporações como a Nokia e Mozilla
Autenticação e autorização	OpenLDAP	openldap.org	Ferramenta open source e bem conhecida na indústria de tecnologia da informação

Com o Splunk e SonarQube, muitos indicadores puderam ser obtidos de forma automática. Entretanto, alguns indicadores tiveram a necessidade de serem coletados manualmente. Para isto a estratégia utilizada foi armazenar os dados em um banco de dados e fazer com que o Splunk o indexasse obtendo assim, a possibilidade de utilização dos *dashboards* da ferramenta. Estes indicadores estão relacionados à testes exploratórios e requisitos.

4 Resultados

Após a implantação, uma nova avaliação do nível de maturidade utilizando o modelo da Figura 4 foi efetuada, obtendo-se os resultados:

- Processo
  - O processo passou a ser gerenciado mas não padronizado, pois somente um projeto da organização foi adicionado ao mesmo

  - A promoção dos artefatos produzidos foi automatizada na maior parte do processo. Os pontos que não foram automatizados, ficaram desta forma por escolha da organização
  - O processo tornou-se proativo, sendo executado a cada nova revisão adicionada ao repositório
- Automação
  - As automatizações das tarefas de desenvolvimento foram feitas. No entanto, as tarefas de publicação para os ambientes de *Staging* e Produção ainda são manuais
- Colaboração
  - O gerenciamento da comunicação entre todos os envolvidos passou a ser registrada na ferramenta de ALM gerando uma base histórica
  - A equipe de infraestrutura tem acesso a todas as informações necessárias para a publicação de um produto de *software*

Com isso o nível do processo foi classificado com (2) Gerenciado, resultando assim, em alguns benefícios para a organização. Relacionando estes objetivos com as causas mostradas na Figura 3, obteve-se os seguintes resultados:

1. Falta de processo adequado: o processo foi adequado para o segundo nível do modelo de maturidade, permitindo assim, uma evolução gradual tanto da equipe, quanto da organização.
2. Falta de ferramentas: foram analisadas, instaladas e configuradas as ferramentas mostradas na Tabela 3. Depois do período de adaptação, todas elas mostraram-se adequadas e úteis ao processo de SDC.
3. Falta de cultura: tanto a equipe de infraestrutura quanto a equipe de desenvolvimento adquiriram a cultura de entrega contínua, entendendo os princípios e benefícios relacionados.
4. Muitas tarefas executadas de forma *adhoc*: houve uma redução do número de tarefas *adhoc*, sendo a maior parte automatizada. Esta automatização resulta em um entendimento melhor do processo.
5. Falta de conhecimento: a equipe teve um aumento de conhecimento não somente de processo e ferramentas, mas um entendimento melhor dos produtos de *software* que produz.
6. As publicações não são comunicadas efetivamente: o gerente de configuração passou a avisar os usuários e donos de produto a cada promoção de produtos de *software*, tanto para os ambientes de *staging* quanto de produção.
7. O andamento das demandas não é percebido por todos: como a comunicação foi melhorada devido as atualizações na ferramenta de ALM terem maior frequência.
8. As atualizações das informações de progresso têm baixa frequência: as atualizações passaram a ser feitas sempre que uma nova revisão é adicionada de maneira automática à ferramenta de ALM.
9. Tempo de ciclo (demanda – publicação) muito elevado: reduziu-se o tempo de ciclo devido a automação de tarefas repetitivas e, também, pela eliminação de tarefas que agregavam pouco valor ao processo. Um exemplo deste tipo de tarefa é a autorização formal entre equipe de desenvolvimento e donos de produto.

10. Poucos testes automatizados: o número de testes automatizados aumentou devido a necessidade de garantia de promoção dos artefatos de uma fase para outra. Isto gerou um impacto negativo na velocidade inicial pós-implantação.
11. Falta de integração entre a equipe de desenvolvimento e infra: ambas as equipes estão envolvidas no processo de entrega desde o planejamento até a publicação. Isto reduziu as falhas de comunicação e aumentou a sinergia entre as equipes.
12. Sobreposição de atualizações: houve uma redução no número de sobreposições devido a cada revisão enviada ao repositório o processo ser executado. Isto fez com que reduzisse o número de colisões de código e o número de *builds* danificados.
13. Pacotes esperando para serem atualizados: os pacotes passaram a ser atualizados com maior frequência devido a automação de tarefas repetitivas e ao aumento da confiança da equipe em publicar os produtos de *software*.

Embora grande parte destes benefícios sejam qualitativos, todos poderiam ser coletados inicialmente para identificar o resultado de forma quantitativa. A produtividade da equipe, medida neste caso em números de publicações mês, nos 3 primeiros meses foi reduzida por alguns motivos; mudança de paradigma: anteriormente as publicações eram feitas de maneira *adhoc*, sem as devidas aprovações e sem a rastreabilidade exigida. Mudar para o novo paradigma fez com que a equipe gastasse mais tempo cuidado do processo ao invés de somente implementar o código. Curva de aprendizado relacionada as ferramentas: todos os desenvolvedores tiveram que aprender os conceitos e utilização das ferramentas instaladas, isto foi obtido de maneira gradual e de acordo com o aumento da experiência dos mesmos.

O risco das publicações também diminuiu por conta dos fatores: (1) mais pessoas cuidando do processo, (2) indicadores de qualidade, (3) aprovação por pares e (4) testes automatizados mais completos. O processo atendeu as regras de compliance garantido que qualquer alteração possa ser rastreada fim a fim, ou seja da demanda para o executável e do executável para a demanda, passando por qualquer um dos pontos intermediários. Este aumento de qualidade refletiu diretamente na maturidade da equipe de desenvolvimento, bem como na organização no geral.

Algumas lições foram aprendidas e catalogadas durante o projeto de implantação. As mais relevantes estão apresentadas na Tabela 2 relacionando o problema, consequência, causas, uma possível solução e resultados.

Dentre os resultados alcançados, pode-se destacar o aumento da qualidade do processo e do produto final, a redução do risco e aumento da disponibilidade do sistema. As consequências disso foram desenvolvedores mais focados em funcionalidade enquanto o processo tende a ser continuamente automatizado.

5 Conclusões

Este trabalho apresentou o projeto de implementação de um processo de entrega contínua, desde sua concepção até o pós-implantação, mostrando os dados de

Tabela 2. Lições Aprendidas

Título	**Treinar a equipe antes**
Problema	Curva de aprendizado alta
Consequência	Redução da produtividade pós-implantação
Causa	Falta de conhecimento em conceitos e ferramentas
Solução	Treinar todos os membros da equipe antes de implantar o processo e ferramentas
Resultado	Baixo impacto na produtividade pós-implantação
Título	**Projetar o processo antes de utilizar uma abordagem tentativa e erro**
Problema	Desenho do correto do processo
Consequência	Recursos gastos com retrabalho
Causa	Falta de planejar o processo como um todo
Solução	Desenhar o processo antes e com toda a equipe envolvida
Resultado	Redução do retrabalho
Título	**Deixar claro a todos os envolvidos os benefícios**
Problema	Explicar a todos, incluindo a gestão, os benefícios deste tipo de abordagem
Consequência	Conseguir recursos para implantação do processo
Causa	Falta de comunicação efetiva
Solução	Apresentar de maneira clara e formal os benefícios obtidos com SDC
Resultado	Facilidade de obtenção de recursos para implantação
Título	**Promover mudança de cultura**
Problema	Sem mudança de cultura o processo provavelmente não funcionara corretamente
Consequência	Poucos benefícios alcançados e recursos gastos desnecessariamente
Causa	Falta de entendimento de todos os elementos envolvidos
Solução	Tratar a mudança de maneira gradual e planejada
Resultado	Processo funcionando de maneira adequada

como o processo era antes, como ele foi planejado e como ele foi implementado. Isto gerou um *setup* que pode ser utilizado em processos similares.

Adicionalmente, dentre as contribuições pode-se citar (1) um conjunto de ferramentas avaliadas, (2) um conjunto de lições aprendidas e (3) um conjunto de recomendações que podem ser utilizadas tanto para organizações que não utilizam este tipo de abordagem, quanto para aquelas que já possuem um nível maior de maturidade. As recomendações foram organizadas de acordo com as dimensões do modelo de maturidade (processo, automação e colaboração). Dentre elas, pode-se citar:
Processo

1. Criar um processo repetitivo e confiável de publicação de *software*: deve-se criar o processo e o segui-lo, melhorando-o de forma incremental. Deve-se evitar tarefas *adhoc* pois as mesmas não auxiliam a evolução do processo.
2. Usar mensagens significativas de *commit*: deve-se vincular o commit a uma funcionalidade ou correção de *bugs*. Isto facilita a rastreabilidade e o diagnóstico de problemas.
3. Usar sempre o mesmo processo para colocar o *software* em produção: atualizações de emergência também devem passar pelo processo.
4. Publicações de zero *downtime*: deve-se ter uma estratégia de publicações com o menor tempo de indisponibilidade possível. A criação da estratégia ajuda no desenho do processo.

5. Manter o tempo de ciclo o menor possível: quanto menor o tempo de ciclo, mais fácil de publicar correções de emergência. Caso seja extremamente emergencial, deve-se fazer um *rollback*, pois provavelmente uma atualização emergencial pode adicionar mais erros do que soluções.
6. Não deve-se fazer mudanças diretamente no ambiente de produção: geralmente isto inclui artefatos ou configurações que não podem ser auditadas.
7. Mudanças de infraestrutura devem seguir o mesmo processo: deve-se utilizar alguma ferramenta para isto evitando o trabalho manual.
8. Basear-se um modelo de maturidade: deve-se utilizar um modelo para identificar quão maduro o processo está. Isto auxilia na tomada de decisão e, também, no desenho do processo.

Automação

1. Manter tudo no controle de versão: deve-se adicionar tudo o que for possível no controle de versão. Artefatos como código fonte, requisitos, documentos, arquivos de configurações, *scripts* de banco de dados. Além do gerenciamento, isto possibilita a exclusão de artefatos desnecessários sem risco.
2. Gerenciar dependências: não deve-se deixar as bibliotecas e componentes em qualquer lugar. Deve-se criar um repositório oficial para evitar a utilização de bibliotecas não autorizadas.
3. Gerenciar configurações (desenvolvimento, UAT, performance, produção): deve-se deixar scripts prontos para cada um dos ambientes. Isto evita o retrabalho de configurações *adhoc*.
4. Testar configurações: deve-se testar as configurações da mesma forma que os códigos são testados. Isto garante que os scripts são confiáveis e corretos.
5. Script pré *build*: sempre que possível, deve-se criar *scripts* para automatizar tarefas como *backups*, limpeza e alocação de espaço.
6. Script pós *build*: enviar de *e-mail* à equipe com o resultado de um *build* que falhou auxilia o diagnóstico e a correção de eventuais problemas.
7. Gerenciar ambientes: deve-se gerenciar os ambientes envolvidos na publicação de *software*. Este ambiente deve ser gerenciado da mesma forma que o ambiente de produção; com manutenções preventivas, alertas, entre outras.
8. Ter log de tudo o que for possível: ter log dos ambientes e todos os elementos possíveis facilita o diagnóstico e pode ser utilizado de maneira proativa.
9. Deve-se ter um *baseline* para diagnosticar problemas rapidamente: isto facilita a comparação entre versões anteriores e principalmente a evolução delas ao longo do tempo.
10. Preferir automação sobre documentação: deve-se preferir automatizar uma tarefa ao invés de documenta-la. Documentação tende a se deteriorar rapidamente e ter pouca utilidade para o pessoal técnico.

Colaboração

1. Entrega contínua não é para todos: existem organizações com um comitê de aprovação de mudanças ou a construção do *software* é feita por uma empresa terceira. Neste caso, deve-se automatizar as tarefas que forem possíveis.

2. Deve-se colocar as pessoas responsáveis pela entrega na construção do processo: isto as torna comprometidas com a execução do processo.
3. Criar um *dashboard* para o *pipeline*: deve-se mostrar onde as publicações estão no tempo. Isto facilita a comunicação e dá uma ideia clara de gargalos no processo.
4. Utilizar uma abordagem *Feature Driven Development*: Isto organiza a produção de uma funcionalidade do requisito à produção. Deve-se organizar o ciclo do processo orientado a funcionalidade.

Adicionalmente, alguns trabalhos futuros podem ser elaborados para evoluir o setup fornecido neste artigo. O primeiro deles tem como objetivo obter uma estratégia para publicação com o menor impacto, em termos de indisponibilidade, dos produtos de *software*. Analisando não somente publicações locais, mas em diferentes *timezones*. O segundo deles, está vinculado com as questões relacionadas a cenários com múltiplos projetos. Pode-se analisar como os artefatos, de vários projetos, são promovidos para produção por uma mesma equipe.

Demais questões tais como, melhorar o isolamento de código por meio de outros ramos, além do master; trocar o algoritmo MD5 por SHA1 ou mesmo SHA2, já que o MD5 apresenta falhas e não é resistente a colisão; apresentar os efeitos colaterais das ferramentas utilizadas; apresentar os dados qualitativos e quantitativos do pós-implantação; melhorar a abordagem para evitar problemas triviais de implantação, como treinamentos por exemplo; caracterizar quais tipos de projetos podem ser suportados por este tipo de abordagem.

Finalmente, este artigo tem um objetivo prático. Entretanto, implementar entrega contínua envolve mais do que instalar algumas ferramentas e automatizar algumas tarefas. Isto depende de uma colaboração efetiva entre todos os envolvidos no processo de entrega, suporte da alta gestão e, principalmente, do desejo das pessoas envolvidas em tornar as mudanças uma realidade.

Referências

1. Mantyla, M. V., & Vanhanen, J. (2011). Software Deployment Activities and Challenges - A Case Study of Four Software Product Companies. 2011 15th European Conference on Software Maintenance and Reengineering, 131–140.
2. Humble, J. and Farley, D., Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, ser. Addison- Wesley Signature Series. Pearson Education, 2010.
3. Ernawati, T., & Nugroho, D. R. (2012). IT Risk Management Framework Based on. International Conference on System Engineering and Technology, 11.
4. Bellomo, S., Ernst, N., Nord, R., & Kazman, R. (2014). Toward Design Decisions to Enable Deployability: Empirical Study of Three Projects Reaching for the Continuous Delivery Holy Grail. 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 702–707.
5. Krusche, S., & Alperowitz, L. (2013). Introduction of Continuous Delivery in Multi-Customer Project Courses Categories and Subject Descriptors, 335–343.
6. Fitzgerald, B. (2014). Continuous Software Engineering and Beyond : Trends and Challenges Categories and Subject Descriptors. RCoSE'14, 1–9.