UNIARA
UNIVERSIDADE DE ARARAQUARA

ANÁLISE COMPARATIVA ENTRE PROCESSOS DE ENTREGA DE SOFTWARE DO TRADICIONAL À ENTREGA CONTÍNUA

João Victor Peria Aloia[1]
José Eduardo Ribeiro[2]

RESUMO: A intensa pressão sobre as fábricas de software para conseguir demonstrar resultados mais rapidamente, vem revolucionando seu processo de desenvolvimento de *software*. O que no passado a entrega tradicional era sinônima de atrasos e aumento nos custos dos projetos, atualmente é referência em métodos que propõem agilidade e qualidade no ciclo de desenvolvimento nas entregas contínuas. Dentro desse processo do desenvolvimento ágil é nítida a importância de dois serviços, são eles, integração contínua, que promove um *feedback* ao incorporar mudanças no software, mostrando a equipe o que funciona e o que está faltando para chegar até a implantação em produção, e entrega contínua (CD) que está atrelada ao (CI). O presente trabalho tem como objetivo analisar e avaliar as comparações entre os processos "Entrega tradicional" versus "Entrega Contínua". Como resultados, o estudo de caso realizado em duas fábricas de software, apresenta as diferenças de como é feita a entrega de software funcional em produção. Conclui-se que o processo "Entrega Contínua" tem uma grande correspondência em valor ao negócio devido a sua agilidade e qualidade no ciclo de desenvolvimento, teste e implantação do sofware.

Palavras-chave: integração contínua (CI). entrega contínua (CD). *software.*

[1] Graduando em Sistemas de Informação da Universidade de Araraquara – UNIARA, ARARAQUARA – SP, joaoaloia@gmail.com
[2] Orientador docente do curso Engenharia da Computação e Sistemas de Informação da Universidade de Araraquara – UNIARA, ARARAQUARA – SP, jeribeiro@uniara.com.br

COMPARATIVE ANALYSIS BETWEEN TRADITIONAL SOFTWARE DELIVERY PROCESSES AND CONTINUOUS DELIVERY

ABSTRACT *: The intense pressure on software factories to be able to demonstrate results faster has revolutionized their software development process. What in the past the traditional delivery was synonymous with delays and increase in the costs of the projects, is currently a reference in methods that propose agility and quality in the development cycle in continuous deliveries. Within this process of agile development is clear the importance of two services, they are, continuous integration, which promotes feedback by incorporating changes in the software, showing the team what works and what is missing to reach the deployment in production, and continuous delivery (CD) that is tied to (CI). The present work aims to analyze and evaluate the comparisons between the processes "Traditional delivery" versus "Continuous delivery". As a result, the case study carried out in two software factories presents the differences in how the delivery of functional software in production is made. It is concluded that the process "Continuous Delivery" has a great correspondence in value to the business due to its agility and quality in the cycle of development, testing and deployment of the software.*

Keywords: *continuous integration (CI). continuous delivery (CD). softwares.*

1 INTRODUÇÃO

As empresas de TI adotam como melhores práticas o uso das metodologias como *ITIL* (*Information Technology Infrastructure Library*), *CMMI* (*Capability Maturity Model Integration*) entre outras, porem tal metodologia tem um processo muito complexo e lento para disponibilizar uma versão de *software* em produção, esse processo pode levar muito tempo entre análise de requisitos, plano de projeto, desenvolvimento, homologação e produção. Nesse caminho podem ocorrer muitas falhas entre desenvolvedores e infraestrutura, sendo que a mais comum é a equalização de ambiente, cobertura de testes e garantia de qualidade. Nem sempre o ambiente de homologação vai ser igual ao de produção, existem algumas particularidades entre parametrizações e configurações que podem ocasionar essa imparcialidade. Além das diferenças no ambiente, existe outro fator, que no qual é a falha humana (resultante da execução manual do processo), crucial na baixa qualidade do serviço, no aumento de códigos quebrados,

nas divergências entre as versões do código fonte que está em produção com o código disponibilizado para o início do desenvolvimento, além de *rollbacks*3, tomados e nas insatisfações dos gestores em relação ao aumento do *backlog*4 de incidentes e problemas.

Falha humana é muito comum no processo de *deploy*, o desenvolvedor obtém uma versão a qual está em produção, executa o processo de desenvolvimento que resulta o pacote que deverá ser entregue em produção. Esse processo ocorre em empresas que não possuem políticas e processos bem definidos, boas práticas, conhecimento técnico e geralmente sem ferramenta específica de controle de versão, de geração do *build*, validação de ambiente e de *deploy*5.

Em contrapartida, a indústria de software vem evoluindo e lidado com grandes desafios, como entregas mais frequentes, ágeis, com alto padrão de qualidade e flexibilidade nos requisitos aliados a baixos custos de produção. Em consequência deste ambiente, que demanda por produtos inovadores e um processo mais produtivo às mudanças, os métodos ágeis despontam como uma solução para que as equipes de desenvolvimento obtenham sucesso e sobrevivência no mercado (MATHARU et al., 2015). As *startups*, por exemplo, vem utilizando dos métodos ágeis para se adaptar melhor à estratégia de negócios (GIARDINO et al., 2014). Os seus objetivos em curto prazo, por sua vez, trazem uma constante pressão para conseguir demonstrar resultados mais rápido.

1.1 METODOLOGIA

Foi realizado um estudo de caso de duas empresas (por questão de integridade das empresas, os nomes informados serão meramente ilustrativos) desenvolvimento de *software*, onde a XRY *Software* opera com o processo tradicional de entrega de *software* e a AEF Solution que trabalha de forma contínua em suas entregas de pacotes. Foram realizadas pesquisas bibliográficas sobre tecnologias e ferramentas *como Git, Jenkins, Sonar, Nexus, Bitbucket* e SVN, entre outros. Durante o estudo de caso, foi analisado o desempenho das combinações de

[3] **Rollback**. É uma palavra de informática usada para definir encerramento da transação, descartando (desfazendo) todas as alterações realizadas durante a transação.
[4] **Backlog**. Backlog é uma espécie de fila de chamados de clientes.
[5] **Deploy**. O termo deploy significa uma implantação em um ambiente local, na própria máquina do desenvolvedor ou dentro de um servidor.

ferramentas e as tendências para o futuro, e procurou-se questionar sobre a supervalorização do processo de entrega contínua de *software*, que se tornou um dos mais maduros e estáveis processos CI do mercado.

1.2 PROCESSO ENTREGA DE SOFTWARE TRADICIONAL

As metodologias tradicionais são também chamadas de pesadas, cascata ou orientadas a documentação (Royce, 1970). O processo tradicional atingiu uma maturidade muito alta, porém foi observado um engessamento entre as áreas de desenvolvimento, *tester* e operação de TI em relação a alocação do recurso, onde o *tester* só consegue iniciar sua atividade após o desenvolvedor finalizar o desenvolvimento e o time de operação implantar o *software* no ambiente de teste.

Segundo dados do *CHAOS Report*, apresentados na Tabela 1, em 2015, mostram que apenas 29% dos projetos foram entregues respeitando os prazos e os custos e com todas as funcionalidades especificadas *(successful*). Aproximadamente 19% dos projetos foram cancelados antes de estarem completos e 52% foram entregues, porém com prazos maiores, custos maiores e com menos funcionalidades do que especificado no início do projeto (*Challenged).*

Tabela 1 - Dados do *CHAOS Report* de 2011 até 2015

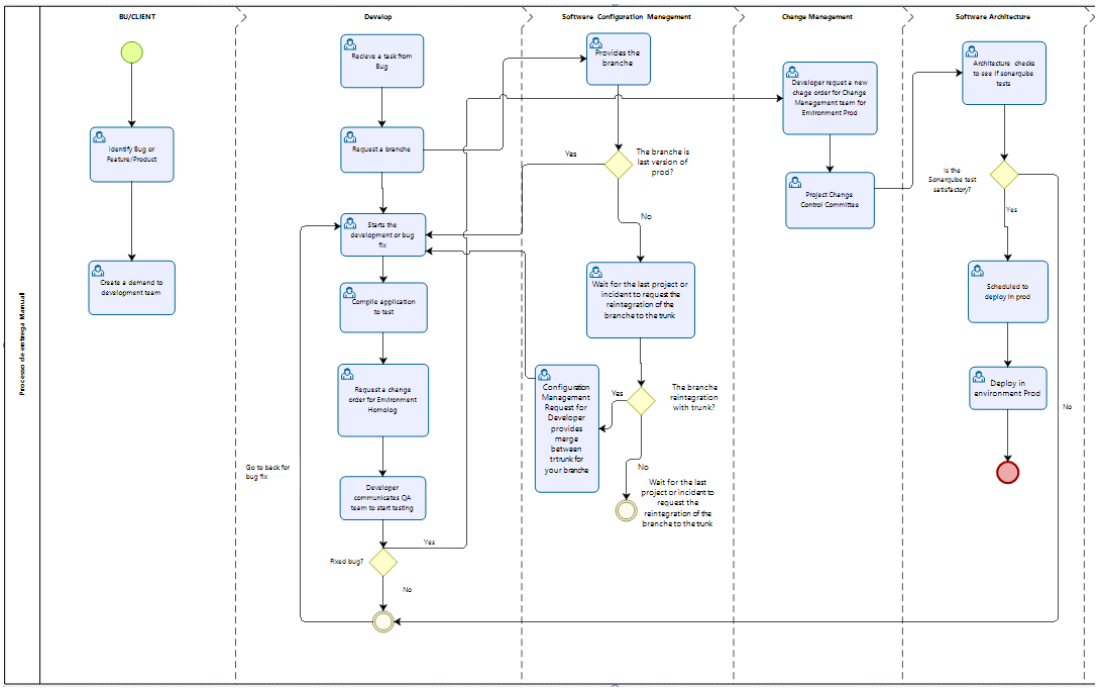
	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

Fonte: *Standish group 2015 chaos report - q&a*

Mesmo os projetos cuja entrega é feita respeitando os limites de prazo e custo, possuem qualidade suspeita, uma vez que, provavelmente foram feitos com muita pressão sobre os desenvolvedores, o que pode quadruplicar o número de erros de *software*, segundo a mesma pesquisa. As principais razões destas falhas estavam relacionadas com o modelo tradicional. A recomendação final foi que o desenvolvimento de *software* deveria ser baseado em modelos incrementais, o que poderia evitar muitas das falhas reportadas.

Foram realizados estudos em duas empresas de *software*, uma com processo tradicional e outra com processos ágeis. Será ocultado o nome das empresas a fim de preservação de imagem. Para entrega tradicional a empresa Demora *Sotware* possui o CM6 que é responsável pela tarefa de controle de versão e aplicação no ambiente de homologação/produção. Dentro desse processo a sequência de cada tarefa envolve muitas pessoas e isso pode ser um problema muito grande onde podem ocorrer diversos tipos de falhas. Abaixo consta a figura do fluxograma do processo tradicional de entrega de *Software.*

Figura &2 - fluxograma do Processo Tradicional de *Software*



Fonte: Elaborado pelo autor (2018)

A interpretação do fluxograma citado na Figura 1, representa o seguinte cenário, com as especificidades da empresa XRY *Software*, mas que contempla um padrão na maioria das empresas que possui o fluxo de entrega manual. A Unidade de negócio identifica o *bug* ou uma nova solução, cria a tarefa para com as informações necessárias para correção ou criação do produto, o desenvolvedor solicita o código fonte para a equipe Configuração de *Software*, que retorna a tarefa para o analista iniciar o desenvolvimento. O repositório de código fonte pode estar desatualizado com a versão atual de produção, isso ocorre frequentemente devido ao fluxo centralizado do repositório. Após realizar o desenvolvimento, o analista realiza o *commit*7 da

[6] **CM.** Configuração de software.
[7] **Commit.** Registrar, salvar as alterações no código fonte.

mudança no sistema de controle de versão SVN, gera o pacote localmente utilizando as dependências que contêm em sua própria máquina.

Subversion ou SVN é um sistema de controle de versão de código aberto. Fundado em 2000 pela *CollabNet*, Inc., o projeto e o *software Subversion* obtiveram um sucesso incrível na última década. O *Subversion* desfrutou e continua a gozar de ampla adoção, tanto na arena do código aberto quanto no mundo corporativo. Essa ferramenta é indicada para pequenas empresas de *software,* pois o controle de versão é centralizado, dificultando a forma de trabalho entre equipes de projetos diferentes. (*Subversion*. Disponível em *https://subversion.apache.org/)*

O binário8 é disponibilizado em um diretório dentro de um servidor de forma manual feito através de uma ferramenta FTP (*File Transfer Protocol*). Realizado a etapa anterior, o pacote está pronto para ser implantado em homologação, nesta fase acontece uma elevada execução de tempo, pois necessita de aprovação de equipes como infraestrutura, Gerente de teste e Gestão de configuração por não possuir automação. Quando é realizada a implantação, o time de qualidade de software inicia os testes do produto, após passar da fase de teste e ser homologado, o produto está pronto para ser implantado em produção. A tarefa retorna para o analista onde se faz necessário submeter o código para alguma ferramenta de análise de código com o intuito de verificar a cobertura de teste. As principais medidas de um teste incluem a cobertura e a qualidade.

A cobertura é a medida da abrangência do teste e é expressa pela cobertura dos requisitos e casos de teste ou pela cobertura do código executado e a qualidade é uma medida de confiabilidade, estabilidade e desempenho do objetivo do teste (sistema ou aplicativo em teste). Ela se baseia na avaliação dos resultados do teste e na análise das solicitações de mudança (defeitos) identificadas durante o teste.

Depois de obter o relatório gerado na etapa anterior, é solicitado ao time Gerenciamento de Mudanças que inclua sua demanda na janela de implantação. A equipe de Arquitetura de Software analisa os resultados das métricas do código gerado pela ferramenta de cobertura de código que aprova ou não a implantação. Neste contexto, a equipe de Arquitetura aprova a implantação em produção, após realizar o *deploy* no ambiente, o analista volta a atuar no acompanhamento e na realização dos testes e homologação pós-produção, após horas de

[8] **Binário**. Arquivo gerado após a compilação do código fonte, exemplo de binário ear/jar/war.

trabalho, o ambiente é liberado e o sistema volta a operar normalmente. Essa empresa implementa um pacote por mês, pois seu processo é manual e não possui estrutura adequada para aumentar a quantidade de *deploys.*

Quando a gestor de TI e as equipes evolvidas na entrega avaliam suas realizações, percebem que ficaram mais tempo fazendo operações manuais do que implementando melhorias no sistema. O grande desafio dessa empresa é diminuir bruscamente o tempo gasto na sua entrega e as falhas humanas, pois se ocorrer uma atualização errada do binário, o tempo gasto para solução é duas vezes maior.

1.3 PROCESSO ENTREGA CONTÍNUA DE *SOFTWARE*

O processo ágil deve ser muito bem executado e experimentado para surtir efeito na entrega contínua, as integrações são extremamente complexas e deve-se atender todas as aplicações e tecnologias que a empresa trabalha, com esse intuito deve-se preparar e possuir uma equipe qualificada para desenvolver a integração contínua.

A Entrega Contínua é um conjunto de práticas que tem como objetivo garantir que código funcional pode ser implantado no ambiente de produção a qualquer momento.

O fluxo de entrega contínua e o envio de código funcional acontecem em alguns ambientes, geralmente em ambiente de DEV (onde é feito o desenvolvimento e ou correção do código fonte e implantado para que o analista valide se o que foi realizado teve sucesso ou não), HOMOLOG/QA (o ambiente de homologação/*quality assurance*, que no português significa garantia de qualidade, é onde o teste de interface é realizado e a equipe de QA certifica-se de que o novo código não terá nenhum impacto na funcionalidade existente e testa as principais funcionalidades do sistema) e PRODUÇÃO (ambiente produtivo Atende usuários finais / clientes), Nesta fase do processo se está entregando código para uma base de usuários, entenda-se por base de usuários, *Testers* / QA, clientes para homologar ou até usuários finais. Semelhante à integração contínua, neste momento pode-se incrementar o processo de teste com os testes de comportamento (*Behavior Tests - BDD*) para testar a lógica de negócio ou até mesmo testes visuais.

No processo de entrega contínua o termo CI, descreve o conceito de algo junto “integrado” e feito sem parar “continuamente”. Esse é o princípio da CI: o desenvolvedor faz a tarefa criativa, o código e uma ferramenta de orquestração de tarefas gera *builds*, testes

integrados, análise das métricas na qualidade do código *(Sonarqube*), empacotamento e armazenamento (*NEXUS*) do binário gerado e, *deploys* e etc. Um exemplo de ferramenta é o *Jenkins*9, uma vantagem da automação é ter um sistema atualizado diversas vezes continuamente, íntegro sem quebra de código e ou perca de versão, garantindo que o início do desenvolvimento seja feito de forma íntegra evitando impedimentos manuais, pois o processo de CI fornece *feedback* como está o código, a qualidade e o que é preciso fazer para melhorar.

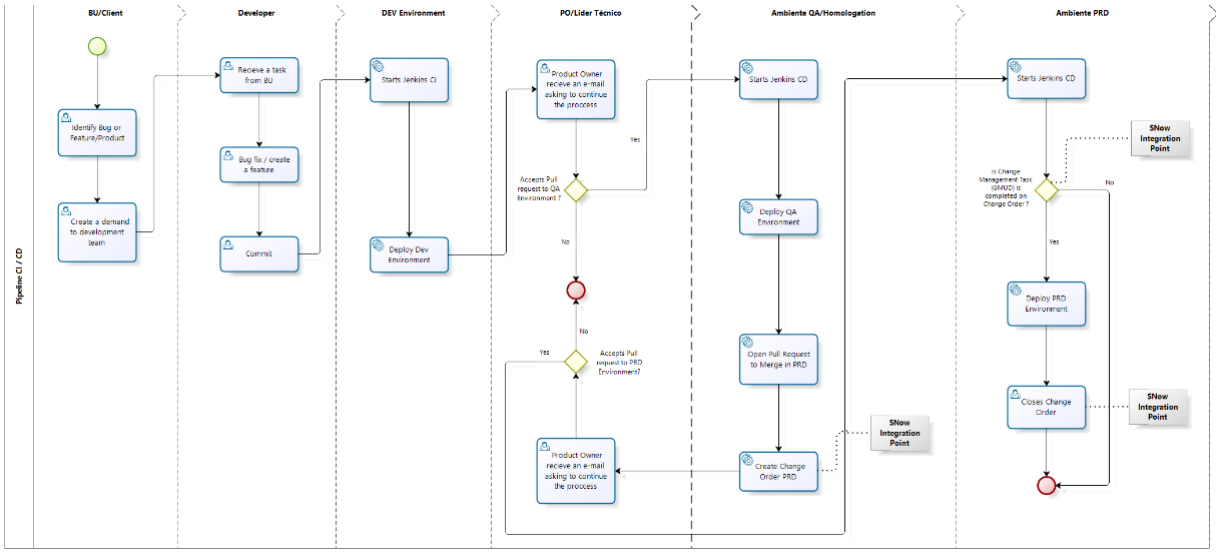
O *Sonar* (ou *SonarQube*) é um agregador de métricas que pode ser usado para medir a qualidade do código do seu sistema. Desenvolvido na plataforma Java, executa métricas para diversas linguagens (como Java, C++, *JavaScript* e CSS); a maioria delas de forma gratuita, e algumas pagas. (*SonarQube*. Disponível em *https://www.sonarqube.org/features/integration/*).

O *Nexus Repository* inclui duas opções de inteligência de componentes, Verificação de Saúde do Repositório e Verificação de Saúde do Aplicativo, cada um fornece uma lista detalhada de vulnerabilidade de segurança e problemas de conformidade de licença para todos os componentes de código aberto, encontrados dentro de seus repositórios ou aplicativos. Essa ferramenta é utilizada para armazenar o binário, código gerado a partir do *build.* (*Nexus Repository Manager. Disponível em https://www.sonatype.com/nexus-repository-oss*).

Com o aperfeiçoamento e conhecimento de processos ágeis, integração contínua e entrega contínua, a indústria de software foi se aperfeiçoando e a empresa fictícia Y implantou o processo de integração contínua, de acordo com o fluxograma que melhor se enquadra nos processos mencionados.

[9] **Jenkins.** É um servidor de automação de código aberto escrito em Java. O Jenkins ajuda a automatizar a parte não humana do processo de desenvolvimento de software, com integração contínua e facilitando os aspectos técnicos da entrega contínua

Figura 3 - Fluxograma do processo Entrega Contínua



Fonte: Elaborado pelo autor (2018)

Segundo o fluxograma, a unidade de negócio identifica o *bug* ou uma nova implementação, abre a tarefa para o desenvolvedor, o analista faz o *checkout*10 do *branch*11 através do repositório de código fonte *Bitbucket* integrado com o *git*, realiza a correção, salvar a alteração e enviar para o repositório remoto. O processo de integração é disparado que faz toda a execução de validar o repositório da aplicação, o próximo passo é realizar o *checkout* no servidor de integração, realizar o *build*, testar o ambiente e fazer à análise da qualidade do código, em tempo real da integração todas as ações são informadas para o desenvolvedor.

O *Bitbucket* é mais do que apenas gerenciamento de código do *Git*, o mesmo dá às equipes um lugar para planejar projetos, colaborar em código, testar e implantar, uma ferramenta poderosa no controle de versionamento com excelente interface gráfica. Por não serem centralizados, equipes e projetos diferentes podem trabalhar ao mesmo tempo, pois é indicada para empresas de médio e grande porte. (*Bitbucket*. Disponível em *https://bitbucket.org/product/version-control-software)*

Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, de projetos pequenos a muito grandes, com velocidade e eficiência.O Git é fácil de aprender e tem uma pegada minúscula com desempenho extremamente rápido. Ele supera as ferramentas de SCM, como Subversion, CVS, Perforce e ClearCase, com recursos

[10] **Checkout**. O termo checkout utilizado na área de Tecnologia da informação significa realizar uma cópia de trabalho.
[11] **Branch.** É uma cópia do código derivado de um certo ponto do master que é utilizado para a aplicação de mudanças no código, preservando a integridade do código no master.

como ramificação local barata, áreas de preparação convenientes e vários fluxos de trabalho. (Git. Disponível em https://git-scm.com/

Na simulação dos testes são imputadas as stages12

Global: Lê o repositório, inicia o *checkout*, valida as informações como versão do *java*.

Before_build: Realiza o teste de cobertura de código.

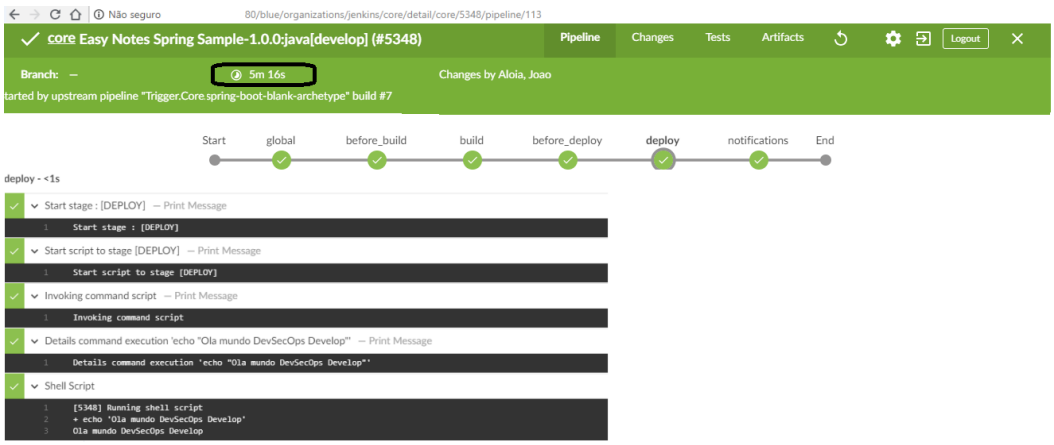
Build: Gera o pacote Binário como ex: (jar, ear, war).

Before_deploy: Pode ser colocada alguma validação como exemplo verificar se a aplicação está acessível.

Deploy: Atualiza o ambiente com o binário.

Notifications: comunica o analista de todos os passos realizados da integração, como exemplo dar *feedback* de todas as fases da esteira.

Figura 4 - Passos da Esteira e o tempo gasto até implantar aplicação.



Fonte: Elaborado pelo autor (2018)

Para mudança ser implantada em homologação, o pipeline questiona o analista se deseja abrir uma *pull request* (Essa funcionalidade é para promover a alteração realizada) para o ambiente. E após o *deploy* ser efetuado, é feito novamente a mesma pergunta, porém, para entregar o código em produção.

[12] **Stages. Estagio de cada ação imputada**

2 Conclusão

A ilustração 1 demonstra as comparações realizadas entre os dois modelos depois da implantação da integração contínua e entrega contínua.

Ilustração 1 - Comparação entre os processos de entrega

Descrição	Tradicional	Contínua
Release com muitas funcionalidades	SIM	NÃO
Diminuição do tempo de *deploy*	NÃO	SIM
Equalização de ambientes	NÃO	SIM
Automação de testes	SIM	SIM
Controle de trabalho colaborativo	NÃO	SIM
Complexidade de Implantação	SIM	NÃO
Menor taxa de *rollback*	NÃO	SIM
Produto viável mínimo	SIM	NÃO
Trabalho de forma ágil	NÃO	SIM
Maior qualidade do produto entregue	NÃO	SIM

Fonte: Elaborado pelo autor (2018)

Os principais benefícios de ambas as abordagens, se referem ao fato de criar um processo de entrega confiável, previsível e passível de repetição, que, por sua vez, gera grandes reduções no tempo do ciclo do projeto, e entrega novas funcionalidades e correções aos usuários rapidamente. A redução de custos em si já é suficiente para cobrir todo o tempo investido no novo processo e na manutenção do mesmo, a restrição e impedimentos gerados pelos gestores operacionais relacionados as mudanças, é um dos motivos de impasse para implantar as boas práticas *devops* nas organizações, pois quebras de paradigmas e modelos de trabalhos já definidos e utilizados por longo prazo, demandam especialistas que possam compartilhar e demonstrar os benefícios na utilização de novas abordagens e métodos, mesmo gerando possível readequação do modo de trabalho de toda estrutura produtiva da empresa.

Neste trabalho, levantaram-se comparações entre processos de entrega contínua e tradicional por meio de um estudo de caso, com respectivos benefícios e performances, fatores como qualidade e agilidade foram colocadas em análises e evidenciadas através de resultados obtidos em duas empresas. O trabalho traz orientações e dados que podem facilitar as tomadas de decisões para gestores, profissionais, desenvolvedores e comunidade científica que precise, analisar ou arquitetar processos de entrega contínua com ferramentas de CI.

Conclui-se que a melhor forma de entregar um software em ambientes, seja de desenvolvimento, homologação ou produção, se faz utilizando o processo de entrega contínua, pois é obtida agilidade no tempo de entrega e uma qualidade muito maior.

Referências

COSTA, Eduardo; Entrega Contínua de Software - Revista. net Magazine 100. Disponível em: <http://www.devmedia.com.br/entrega-continua-de-software-revista-net-magazine-100/26312> Acesso em 24 de maio de 2017.

GIARDINO, C. et al. What do we know about software development in startups? IEEE software, IEEE Computer Society, v. 31, n. 5, p. 28–32, 2014.

HUMBLE, J.; FARLEY, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader). [S.l.]: Pearson Education, 2010.

LOELIGER, J.; MCCULLOUGH, M. Version Control with Git: Powerful tools and techniques for collaborative software development. [S.l.]: "O'Reilly Media,Inc.", 2012.

MATHARU, G. S. et al. Empirical study of agile software development methodologies: A comparative analysis. ACM SIGSOFT Software Engineering Notes, ACM, v. 40, n.1, p. 1–6, 2015.

ROYCE, W.W, Managing the development of large software systems: concepts and techniques. Proc. IEEE Westcon, Los Angeles,CA, 1970.

SHANE H., WOJEWODA S, Standish group 2015 chaos report - q&a with jennifer lynch, October 2015. Disponível em: <https://www.infoq.com/articles/standish-chaos-2015> Acesso em: 29 de maio de 2017.