Continuous Delivery - Agilidade Entre Desenvolvimento e Entrega de Valor

Marcio Luiz de Lima, Ricardo Bortolo Vieira

Faculdade Cidade Verde (FCV)
Maringá – PR – Brazil

marcimpriori@gmail.com, professor.ricardovieira@gmail.com

Resumo. *Este artigo transcreve a respeito da complexidade de desenhar e implantar todos os pontos envolvidos na Entrega Contínua de um software. Identificando pontos importantes e cruciais no processo. Permite discutir sobre a estrutura de uma gerência de configuração e seus itens que envolvem desde o sistema operacional até ambientes escaláveis de serviços na nuvem. Apresenta uma visão sobre Integração Contínua e os desafios de aplicar a cultura ágil reduzindo o trabalho operacional repetitivo e concentrando na qualidade e valor ao usuário. Reconhecendo que um desenvolvimento e entrega de sistema precisa considerar as pessoas, e entender que a melhoria contínua precisa ser mantida durante toda a manutenção do produto.*

Palavras chaves: *Entrega Contínua, Integração Contínua.*

Abstract. *This article transcribes about the complexity of designing and deploying all points involved in the Continuous Delivery of software. Identifying important and crucial points in the process. Let's discuss a structure of a configuration management and its items that involve the operating system to scalable services environments in the cloud. It presents a vision of Continuous Integration and the challenges of applying an agile culture by reducing repetitive operational work and focusing on quality and value to the user. Recognizing that a system development and delivery needs to consider people, and to understand that continuous improvement needs to be maintained throughout the maintenance of the product.*

Keyworks: *Continuous Delivery, Continuous Integration.*

1. Introdução

Em projetos de *software* um ponto que é de forte relevância para os *stakeholders*, é o *lead-time* de um valor. Uma das maiores dúvidas em um projeto de *software* é, como podemos transformar o planejado em sistema e entregar valores aos usuários o mais rápido possível?

Segundo Cruz (2013), projetos ágeis é um conjunto de conceitos e práticas que se encaixam permitindo um autogerenciamento dinâmico e eficiente, focando na entrega de valor de um negócio no menor tempo possível.

Grande parte das atividades realizadas, que necessitam de entrega para gerar valor ao produto, são feitas com alguma incompatibilidade proporcional ao tempo de entrega, ou seja, quanto mais tempo gasto entre o planejamento e a entrada em produção, maiores serão as chances de o produto entregue não atender as necessidades ou objetivo do usuário.

Existem diversos desafios na entrega das mudanças do *software*, mesmo com um planejamento exímio e uma execução pontual, fazer a entrega pode destruir todo um sistema. Muitos problemas permanecem ocultos durante o desenvolvimento, e só se revelam no instante em que uma versão é colocada em produção.

Humble e Farley (2014) identificaram alguns antipadrões na entrega de *software*: Implantar *software* manualmente, implantar em um ambiente similar ao de produção somente quando o desenvolvimento estiver completo, e a gerência de configuração manual dos ambientes de produção.

Implantação realizada manualmente, todas as vezes que uma versão do *software* é entregue, todo o processo de fechamento, teste, preparação e atualização é realizado manualmente.

Quando realizado por uma pessoa que possui total conhecimento técnico da versão, pode ser que poucos problemas ocorram, mas no momento em que mais de uma pessoa está envolvida no processo, a possibilidade se multiplica exponencialmente.

A utilização de uma documentação manual causa uma predisposição à falhas, pois não é comum que esses documentos sejam atualizados conforme as entregas são realizadas.

O processo torna-se custoso, afinal pode se arrastar por horas ou dias, agravando o risco de falhas, neste ponto o quesito humano fica mais destacado, uma vez que a equipe de implantação pode passar dias e madrugadas resolvendo problemas.

Nesta abordagem ainda se identifica a possibilidade de *rollback* ou *downgrade* da versão, isto é, a versão não está funcionando e será necessário retornar para a última versão que funcionava até o momento.

O que dependendo da situação, significa um investimento de tempo e recursos, correndo ainda o risco de não ser possível tal ação, uma vez que de acordo com o tipo de mudança ela pode não ser facilmente revertida.

Quantidade de valor entregue: devido ao elevado custo da entrega, muitas vezes às empresas optam por fazer uma entrega com vários itens. Este tipo de opção pode aumentar a complexidade da aplicação da versão em produção, quando o responsável vai realizar a atualização, o sistema muitas vezes apresenta inconsistência, podendo não reagir conforme o esperado e até apresentar falhas no momento de sua execução.

Quando realizamos uma entrega é normal que os responsáveis façam testes nos componentes entregues, validando se as funcionalidades envolvidas apresentam os resultados esperados; quando falamos de uma entrega grande, o tempo desta validação pode se estender por horas, inclusive pode causar desconforto aos usuários devido ao grande impacto que a versão incide ao sistema.

Uma entrega grande com muitos recursos ainda possui uma tendência em não ter todos os valores conhecidos, isto ocorre porque nem tudo o que está sendo modificado na versão foi documentado ou mapeado nos testes, o que aumenta a possibilidade de falhas no momento da liberação em produção.

Com a possibilidade de problemas ocorrerem aumentando de acordo com o tamanho de uma versão, muitas vezes, após uma entrega realizada, surge o retrabalho, que foi definido por Sterman (1992), como: inconformidade, refeita, causada por falhas no planejamento e na análise dos requisitos afetando o projeto e seu objetivo.

Conforme a velocidade em que a tecnologia evolui, a exigência de qualidade e assertividade do cliente segue no mesmo ritmo. Para atender essa necessidade de qualidade e rapidez, surgem diversos conceitos e ferramentas que colaboram no processo de desenvolvimento e entrega. Com esse advento, surge algo que envolve técnicas, rotinas, processos e metodologia.

Continuous Delivery ou Entrega Continua, segundo Humble (2014) é a capacidade de se obter mudanças de todos os tipos - incluindo novos recursos, mudanças de configuração, correções de bugs e experiências – em produção ou nas mãos dos usuários, de forma segura, rápida e sustentável.

Durante o processo de desenvolvimento de um *software* é necessário definir a Gerência da Configuração, a *Continuous Integration* ou Integração Contínua e a Estratégia de Testes.

O trabalho na entrega contínua se inicia na definição do objetivo, em geral é o de reduzir custo. Para essa redução, muitas vezes precisamos identificar os pontos em que o processo se torna caro, a entrega de mudanças pequenas, que valorizem o sistema sem aumentar o seu risco atrai os clientes.

Para trabalhar no caminho da Entrega contínua é necessário ter a consciência de que a melhoria deve ser contínua, ou seja, não existe um momento em que todo processo estará completo, e todas as pessoas envolvidas precisam ter conhecimento.

Mapear e automatizar os processos exige muito esforço, causando grande cansaço, mas entregar *software* 6 vezes ao ano se tornou passado. Muitas vezes, utilizamos *softwares* que são atualizados semanalmente ou diariamente e nem percebemos. Na entrega contínua as pessoas se tornam o centro do processo, onde o conhecimento é valorizado e a repetição é automatizada.

Leszko (2017) apresenta que a Amazon demora 11,6 segundos (em média) nos seus *deploys*, o Facebook entrega duas releases por dia, e a Atlassian afirma que 65% de seus usuários praticam Entrega Contínua.

Com a velocidade na entrega de uma mudança, o tempo de feedback, na maioria, é instantâneo. Inconformidades são identificadas facilmente, e pelo tempo de resposta, quase todas as vezes, são corrigidas em minutos, praticamente erradicando problemas como o desenvolvedor ter esquecido o que foi feito devido ao tempo passado desde a programação.

2. Gerência da Configuração

O principal gerenciamento que existe na entrega contínua é a Configuração. É imprescindível que seja gerenciado, quase tudo, que for utilizado para construir, compilar, implantar, testar e liberar o sistema em produção, como código-fonte, scripts de compilação, configuração de ambientes e aplicações.

Cada empresa utiliza de uma estratégia de gerência própria, podendo assemelhar em alguns aspectos. A gerência da configuração precisa permitir à empresa replicar a

implantação de um ambiente com as mesmas características. Com essa gestão é necessário que todos os itens de configuração sejam controlados em versões, e que seja possível determinar os impactos e dependências de cada mudança e entrega.

2.1. SVC – *System Version Control*

O SVC (*System Version Control*) ou, Sistema de Controle de Versão é um dos principais mecanismos da gerência da configuração. O objetivo deste sistema é permitir que documentos, tenham seu histórico de alterações controlados em versões.

Muitas vezes quando o controle de versão é pautado, lembra-se apenas do código-fonte, mas no caso da gerência da configuração outros itens precisam ser considerados, como documentação, estrutura de banco de dados, testes e scripts de compilação.

Historicamente, o primeiro sistema de controle de versão foi o SCCS (*Source Code Control System*), escrito em 1972. Este, serviu como base para alguns outros, entre eles, o CVS *Concurrent Version System* (Sistema Concorrente de Versão) e o Subversion.

O CVS foi escrito na década de 80, sendo o mais popular por muitos anos, principalmente pela característica de ser gratuito e sua principal característica é que permitia a alteração concorrente de arquivos.

O Subversion surge com uma visão de CVS melhorado, ele possui características semelhantes como a arquitetura cliente-servidor, mas com uma proposta de facilidade aos usuários e maior capacidade no controle reduzindo espaço e aumentando a velocidade, é um dos sistemas centralizados mais utilizados na atualidade, as principais ferramentas é o Apache Subversion e o TortoiseSVN.

No final da década de 90, surge uma nova abordagem, os sistemas de controle de versão distribuídos, ou DVCS (*Distributed Version Control Systems*) rapidamente se tornando popular. Como no nome, a estratégia é que cada usuário possui seu repositório em seu computador, apesar de ser muito comum equipes de desenvolvimento definir um repositório central permitindo a integração do *software*.

Os sistemas distribuídos, possuem características que alavancam a sua popularidade. O sistema permite que em segundos seja possível ter um repositório

preparado; alterações individuais podem ser enviadas entre os usuários sem passar pelo repositório central; ideias podem ser testadas em *branchs* local sem a necessidade de envio ao servidor remoto; o servidor não tem tantas requisições uma vez que vários *commits* locais podem ser feitos sem precisar enviar ao servidor.

Uma das principais dificuldades encontrada no GIT é o entendimento de sua arquitetura, e seus comandos. Com isso diversas empresas adotam ferramentas e técnicas para facilitar nesse trabalho.

Para o controle na entrega contínua e a definição de versão e entrega, muitas vezes no sistema distribuído é comum definir uma estrutura de fluxo de trabalho, que permite identificar mudanças que serão entregues.

Outro ponto que alavancou os sistemas distribuídos foram os repositórios online com preços acessíveis e altamente compatíveis com integração contínua, os principais são GitHub, BitBucket e GitLab.

2.2. Configuração do *Software*

Humble e Farley (2014), dividem o *software* em três partes: dados, executáveis e configuração. Muitas das linguagens de programação possuem sua própria estrutura de arquivos para configuração, permitindo que ambientes se diferenciam apenas pelas configurações.

Para um processo de implantação válido, a equipe de desenvolvimento muitas vezes precisa identificar essas configurações e quais são os impactos e objetivos.

As configurações do *software* são feitas de forma que ajude no desenvolvimento, teste e implantação, flexibilizando os ambientes e permitindo características únicas.

Essa flexibilidade gerada pode ocasionar complexidade no processo, isso ocorre porque muitas vezes, para atender demandas de regras distintas do cliente, ao invés de optar por um código, o desenvolvimento determina uma configuração para identificar algum tipo de particularidade ou customização, isso ocorre diversas vezes em ambientes onde uma alteração no código-fonte é muito mais custoso do que uma configuração de ambiente.

Utilizando um tipo de abordagem como esta, geralmente pode fazer o sistema parar de funcionar da mesma que uma alteração na programação, contudo, algumas

configurações do ambiente não estão versionadas, o que dá à programação uma confiança quando a mesma possui testes e *builds* automatizados.

Para garantia e segurança da configuração do sistema normalmente é necessário que sejam feitos testes. Basicamente, é preciso verificar os sistemas externos estão se integrando de forma correta.

2.3. Configuração das Aplicações

A grande maioria dos *softwares* não funcionam em uma única aplicação, é normal que tenha ao menos um outro sistema ou serviço envolvido, por exemplo: aplicação de e-mail, fila, agendamento, API (*Application Programming Interface*).

Ao aplicar a Entrega Contínua, essa intensidade de atualização precisa garantir que a comunicação, com as aplicações, não seja impactada e que permaneça funcionando.

Para isso, os ambientes possuem testes que simulam a comunicação entre as aplicações, assim antes e depois que uma atualização ocorra a estrutura e a comunicação é testada permitindo garantir que a versão entregue não perca compatibilidade com a versão de outra aplicação.

2.4. Configuração dos Sistemas

A configuração dos sistemas envolvidos na produção, implantação e manutenção, deve ser realizada para garantir que o processo não passe por roteiros diferentes, apresentado resultados não esperado.

Para gerenciar os sistemas operacionais dos ambientes algumas ferramentas são utilizadas, como exemplo temos, o Puppet e o CfEngine.

Essa ferramenta permite fazer o controle dos usuários ao acesso dos sistemas e máquinas.

As definições utilizadas pelas ferramentas podem ser controladas nos sistemas de controle de versão, permitindo a garantia do controle da mudança, ajudando na identificação de problemas.

É possível ativar agentes nos sistemas que regularmente buscam as configurações mais recentes e atualizam os sistemas.

Virtualizar ambientes trouxe a possibilidade de garantir que as estruturas dos sistemas não sejam perdidas em atualização ou mudança de máquinas.

Na atualidade, muitas empresas estão seguindo para a utilização de containers no lugar das máquinas virtuais.

Trabalhar com containers permite o aproveitamento de recursos, reduzindo custos e melhorando a facilidade no controle da configuração.

Na configuração, a utilização de containers, permite um nível de controle dos recursos de sistemas e *softwares* mais acessível e leve, se comparado à virtualização.

A utilização de container permite que configurações básicas de sistemas operacionais e banco de dados sejam definidos uma única vez.

Com a virtualização muitas das configurações necessárias para controle dos ambientes exigem uma complexidade e quantidade de recurso muito superior do que na abordagem da utilização de containers.

3. DevOps

Ao realizar uma pesquisa, pode-se encontrar diversas definições diferentes sobre DevOps. A grande maioria define um engenheiro de DevOps como sendo um desenvolvedor responsável pela parte operacional, ou alguém do time de operações que pode codificar.

Yanaga (2017) destaca que o propósito do DevOps é devolver ao time de desenvolvimento a responsabilidade pelo seu trabalho.

Com o DevOps existe a aproximação do desenvolvimento de uma mudança até a entrega do valor. Quando desenvolver e entregar ficam separados por times, departamentos e até por unidades diferentes, a responsabilidade acaba recaindo apenas sobre um dos envolvidos.

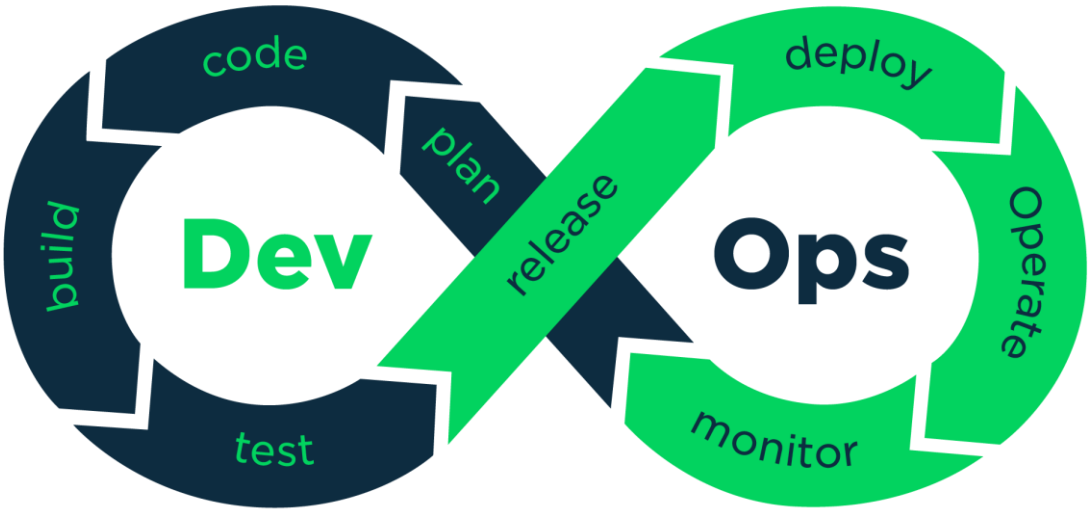

Figura 1 - Fluxo DEVOPS
Fonte: Yanaga (2017 p.89)

O processo de entrega contínua necessita de profissionais capazes de manter produto e processos funcionais.

Não é comum que uma política de culpados seja aplicada nas empresas, quando uma versão é entregue e problemas aparecem. Esforços são gastos em descobrir o responsável, prolongando o tempo de correção.

Com times divididos algumas dúvidas surgem, ao identificar um erro ou falha no sistema. Questionamento sobre o ambiente, se as configurações foram aplicadas, ou se os passos de implantação foram realizados corretamente.

No DevOps unimos as operações e o desenvolvimento, permitindo que com a automação estes riscos sejam mitigados. Recaindo sobre um mesmo time a responsabilidade de produzir e entregar o sistema.

Nas empresas, com maior experiência em processos ágeis, tem se tornado cada vez mais comum existir um time especialista em DevOps responsável na manutenção da melhoria contínua das ferramentas e técnicas, e os times de desenvolvimento por manter a qualidade dos processos com enfoque no produto.

4. *Continuous Integration* (CI)

Com o crescimento recente da demanda de *software* na grande maioria das empresas de grande e pequeno porte, muitos negócios aventuram em levar produtos para plataformas online.

Com essa necessidade do mercado, inúmeras metodologias e movimentos tendem à um desenvolvimento ágil e uma distribuição fácil e rápida. Um exemplo disso é a programação extrema (XP) que tem o objetivo de simplificar áreas do processo de desenvolvimento de *software*.

Com o desenvolvimento do *software* de forma ágil, outras necessidades foram surgindo no ambiente produtivo.

Abrir mão de documentos extensos, como manuais e diagramas, causa intenso desconforto em grandes empresas e analistas conservadores.

Um dos desafios encontrados no desenvolvimento, principalmente em projetos desenvolvidos por grandes equipes, está relacionado a entrega de valor ao usuário. Por vezes, na maioria do tempo de programação nenhuma entrega é gerada, principalmente em projetos que possuem *branches* de longa duração.

Uma abordagem ágil permite que pequenas mudanças sejam entregues reduzindo o impacto e mitigando ao máximo os riscos gerados.

Segundo Pathania (2016), o processo ágil no desenvolvimento permite uma alternativa aos processos tradicionais de desenvolvimento de *software*, baseado em princípios voltadas, principalmente, ao cliente, a qualidade na entrega e agilidade no feedback.

Integração Contínua ou *Continuous Integration* (CI) não é uma terminologia recente, a primeira aparição foi em 1999 por Beck.

Beck (1999) escreve que quando a programação se apresenta extrema e com resultados tão notáveis, deve-se trabalhar com integração frequente, ou seja, o tempo todo.

Integrar continuamente consiste em disponibilizar ao usuário em cada mudança uma nova versão.

Questionar se a estrutura da empresa permite tal intensidade não significa que o pensamento é conservador, contudo o movimento evolutivo da agilidade surge em passos pequenos que apresentam grandes resultados.

Integração Contínua possui o objetivo de automatizar processos repetitivos, muitas vezes realizados por interface humana.

Para implantação do processo, a empresa precisa reestruturar sua arquitetura, permitindo que ferramentas trabalhem em ambientes controlados e estáveis.

Além de uma estrutura, o procedimento requer uma mudança cultural, que tendenciosamente persiste em ser o ponto mais impactante no caminho da mudança.

Ao iniciar é necessário validar o Sistema de controle de versão, pois as ferramentas que trabalham no código-fonte necessitam de uma configuração que permita acessar, compilar, testar unidades do código, testar integrações e distribuir pacotes de versão.

Um único repositório contendo o código, os testes, scripts de banco, scripts de compilação e tudo o que é necessário para criar, instalar, executar e testar a aplicação, permite que falhas e mudanças sejam identificadas.

Pode parecer difícil de acreditar, mas ainda hoje algumas empresas que não utilizam um recurso para controlar a versão do sistema ou o seu código-fonte.

Automatizar o processo é a consistência principal da integração, neste ponto, surge a necessidade de realizar a compilação a partir de linhas de comandos nos sistemas operacionais.

Algumas linguagens e suas IDEs (*Integrated Development Environment*) acabam mascarando a possibilidade de automatização da compilação ou teste do projeto.

Devido ao conforto gerado pelas ferramentas muitos programadores desconhecem que funcionalidades existam em ambiente sem interface visual, e assim trabalham por anos sem saber que a IDE é uma ferramenta baseada em facilitadores de comandos básicos nos compiladores ou interpretadores das linguagens.

Quando o desenvolvedor terminar uma mudança e todos os artefatos estiverem prontos para ser submetidos à integração, uma premissa é que a cobertura de testes esteja adequada aos padrões definidos para a empresa ou o projeto.

Ao enviar as alterações para um sistema de controle de versão, de acordo com a maioria das empresas, estas mudanças no sistema passa por um fluxo.

Apesar de se apresentar de diversas formas, o fluxo mais comum que podemos encontrar nas empresas que utilizam a Integração Contínua como base, de acordo com a Figura 2.

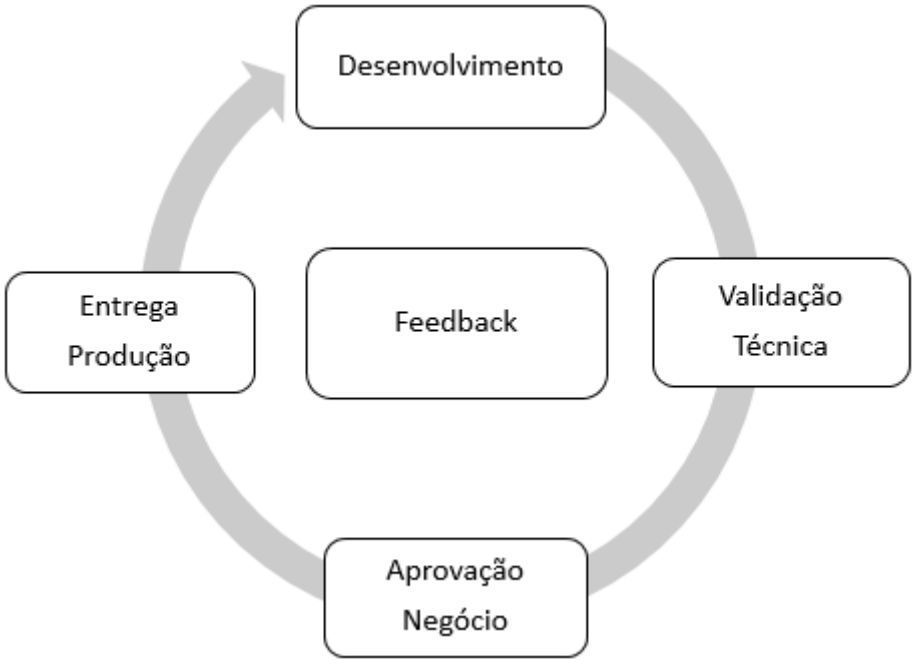

Figura 2 – Processo de desenvolvimento e entrega
Fonte adaptada: Humble(2014 p.30-70)

A Figura permite identificar de forma compacta, que o processo passa a ser reduzido e simples, facilitando o feedback constante em qualquer etapa.

No momento em que o desenvolvedor envia sua mudança para validação técnica, rotinas automatizadas de qualidade de código, considerando boas práticas e particularidades do projeto, fazem o trabalho de validar previamente as mudanças realizadas.

4.1. Validação Técnica

A validação técnica possui duas vertentes: Automática e manual. Quanto maior for a experiência e a automação aplicada nesse processo menor será a interação manual.

Na maioria das empresas o objetivo é garantir que a qualidade do código não sofra com imprevistos e inexperiência da equipe de desenvolvimento.

Os principais pontos verificados no passo de validação são: compilação do projeto, para garantir que não seja entregue código quebrado; cobertura de testes unitários, testes de integração e até testes de comportamento, neste ponto é possível identificar se alguma regra que já existiam no sistema pararam de funcionar.

Na validação dos testes, devido ao longo tempo de execução e a frequência que uma mudança é realizada, muitas empresas optam por fazer apenas testes unitários neste momento, e somente quando ocorrer o fechamento da versão do ambiente de qualidade, é que os testes de integração e de comportamento são executados.

Muitas empresas, com experiência e muitos casos de sucesso, ainda verificam através de passos mais complexos a qualidade do código, observando a complexidade, orientação à objetos, distribuição do código em arquivos.

Uma ferramenta muito utilizada para controle de qualidade de código é o SonarQube, de código aberto, de 2007 com o objetivo de garantir aos desenvolvedores qualidade em seus códigos.

O Sonar é facilmente integrado à sistemas de controle de versão distribuídos (GIT) ou ferramentas de CI, e auxilia na identificação de problemas como: cobertura do código por testes unitários, vulnerabilidade, indentação do código-fonte. A ferramenta trabalha com 20 linguagens de programação do mercado, como *Javascript, Java, Python, C# e PHP*.

Quando a mudança é validada pela automação, ainda é normal que o processo da empresa tenha uma etapa de revisão de código manual, onde o código da mudança será submetido à um olhar crítico de outro desenvolvedor, podendo este sugerir alterações na solução apresentada.

Durante o desenvolvimento é comum que uma equipe mais próxima faça uma divisão de conhecimento e soluções técnicas aplicadas no código, este tipo de

comportamento, e outras técnicas como *Pair Programming*, permitem que a etapa de revisão de código seja menos impactante no processo.

4.2. Aprovação Negócio

Com o movimento ágil envolvendo o ambiente de desenvolvimento, é comum que a área de negócio se envolva mais no processo, pois os resultados são impactados com sua presença.

A identificação de um analista do negócio pode ser interpretada de diversas maneiras na cultura ágil.

Segundo Humble e Farley (2014), testes de aceitação devem ser realizados em ambiente de homologação, onde a aplicação é colocada o mais similar possível do ambiente de produção considerando configuração e estado, permitindo *mock* de serviços externos.

Mais importante do que o título dado ao responsável pelo negócio, é entender que a definição de pronto da mudança deve ser clara e objetiva. Esta abordagem permite que o processo de aprovação não se prolongue ou seja interpretada de forma equivoca.

Este tipo de aprovação é realizado em um ambiente diferente do desenvolvimento, muitas vezes recebe o nome de Ambiente de Qualidade (QA) ou Ambiente de Homologação.

Assim que a mudança é aprovada tecnicamente, a ferramenta de implantação deve acessar os scripts e fazer a aplicação da nova versão no ambiente de qualidade.

O ambiente de qualidade precisa apresentar as mesmas configurações que o ambiente de produção, permitindo a simulação mais próxima do real possível.

Assim configurações como Sistema Operacional, versão de serviços, aplicações, plug-ins, extensões e todas as dependências precisam ser controladas. Isso permite a criação de instaladores reduzindo ao máximo a interface humana e o tempo de restabelecimento de um ambiente caso uma falha catastrófica ocorra.

4.3. Entrega do Produto

A definição de pronto de uma mudança precisa estar de acordo com o objetivo da mudança e as pessoas interessadas, o termo surge com o desenvolvimento Ágil e se torna mais comum na metodologia Scrum.

Segundo Visser (2016), o pronto é definido quando o produto contém tudo o que precisa ser feito para produzir o *software* em um estado liberável ao usuário final.

Definir que uma mudança está pronta, pode não necessariamente garantir que a mesma está aceitável ou que foi realizado com sucesso, contudo permite que seja verificado no momento da entrega.

Muitas das definições de pronto são acompanhadas das regras solicitadas pelo dono do produto, mas por vezes passam por uma revisão e entendimento do desenvolvimento.

A definição de pronto pode ser interpretada como uma lista de atividades no desenvolvimento.

Para o bom desempenho da Integração Contínua, na lista de definição de pronto pode estar contido uma regra de cobertura de testes do código.

No momento em que uma mudança possuir todas as características que à define como pronta, o caminho para a entrega final é realizado.

A entrega do produto ao usuário final, precisa passar pelo mesmo processo de integração da entrega em um ambiente de qualidade.

Quando o processo de integração contínua é automatizado por completo, este passo pode durar minutos ou segundos, variando de acordo com o impacto e tamanho da mudança.

Reduzir a indisponibilidade também é um dos objetivos da entrega contínua, para isso processos de entregas em períodos menos impactantes, ou roteiros que permitem contingenciamento, facilitam no alcance destes objetivos.

4.5. Ferramentas

Automação do processo permite garantir que as entregas seguem o mesmo roteiro, facilitando a identificação de falhas e *bugs*.

Com o crescimento da Entrega Contínua, surgem diversas ferramentas capazes de suprir os muitos tipos de *software*.

O principal objetivo de uma ferramenta de Integração Contínua é ser um facilitador na aplicação do fluxo de entrega.

Um servidor CI precisa prover:

- Integrações com os SVCs, permitindo identificar sempre uma mudança realizada no código;
- API para integrar com *softwares* de gestão ou controle;
- Comunicação por chats, e-mails ou notificações para alertar sobre falhas ou entregas;
- Interação e manipulação de dados dos compiladores, permitindo a construção de versões e pacotes de componentes;
- Ambiente de monitoração avançada, dando aos desenvolvedores o máximo de informação possível a respeito de testes, falhas e processos.

No mercado atual, existem muitas ferramentas capazes de executar tais funcionalidades e muito mais, algumas delas se tornaram destaque.

Nas grandes corporações é comum que seja adotado diferentes ferramentas de acordo com o objetivo do sistema ou projeto.

Ferramentas de código aberto muitas vezes são utilizadas por empresas em projetos livres, ou que permitem sugestões de uma comunidade aberta.

5. Conclusão

Um *Software* criado e desenvolvido com sucesso, não é considerado valor se ainda está parado no processo antes de chegar em produção.

A automação do processo permite uma melhoria na qualidade e velocidade do *software*, mas é necessário que as falhas sejam identificadas e reconhecidas no processo.

Aplicar a Entrega Contínua é uma mudança de paradigma na equipe. O projeto exige intensa atenção e manutenção contínua.

A Gerência da Configuração é o principal ponto de organização, permitindo a garantia a recriação do sistema do zero.

A entrega de *software* foi, por muito tempo, deixada de lado e permanecia em uma área não muito bem definida entre o desenvolvimento e operações; o principal objetivo do DevOps é aproximar e deixar estes dois times mais responsáveis e próximos, permitindo que ambos esforcem seus trabalhos na qualidade e velocidade.

Fazer a Integração Contínua é garantir que a aplicação funciona desde o desenvolvimento até a produção. O processo cria um feedback contínuo permitindo a correção em um momento barato.

Na Integração Contínua a implantação pode ser de forma gradual, permitindo alcançar resultados favoráveis em um curto espaço de tempo.

Aplicar, validar, configurar e manter a Entrega Contínua, requer disciplina constante no time, o que não difere de outros projetos.

Com a Entrega Contínua, é possível fazer a monitoração quando o processo está perdendo qualidade, como validar a cobertura dos testes ou verificar se os fluxos estão mantendo seus roteiros.

6. Trabalhos Futuros

Para continuidade, existe a necessidade de identificar e implementar uma Estratégia de Testes, permitindo que a qualidade seja garantida em todos os aspectos do projeto.

É recomendado que nos trabalhos posteriores um estudo e detalhamento da Anatomia do *Pipeline* de Implantação seja realizado, dando a capacidade de identificar e desenvolver os melhores roteiros e *scripts* de compilação e implantação, além de suas métricas e estágio de *commit*, permitindo o *feedback* rápido e controle de equipes grandes.

Referências

Beck, Kent e Fowler, Martin (1999) "**Extreme Programming Explained**". Addison-Wesley Professional, Upper Saddle River, NJ.

Cruz, Fábio (2013) "**Scrum e PMBOK unidos no Gerenciamento de Projetos**". Brasport Livros e Multimídia, Rio de Janeiro, RJ.

Humble, Jez e Farley, David (2014) "**Entrega Contínua: Como Entregar Software de Forma Rápida e Confiável**" tradução: Marco Aurélio Valtas Cunha, Bookman, Porto Alegre.

Leszko, Rafael (2017) "**Continuous Delivery with Docker and Jenkins**", Packt Publishing, Birmingham, UK.

Pathania, Nikhil (2016) "**Learning Continuous Integration with Jenkins**". Packt Publishing, Birmingham, UK.

Sterman, J. D. (1992) "**System Dynamics Modeling for Project Management**". MIT Sloan School of Management, Cambridge, MA.

Visser, Joost (2016) "**Building Software Teams**". Software Improvement B.V. O'Reilly Media, Inc., Sebastopol, CA.

Yanaga, Edson (2017). "**Migrating to Microservice Databases: From Relational Monolith to Distributed Data**", O'Reilly Media, Inc., Sebastopol, CA.