10-2019

DevOps – foundations and perspectives

Leandro Sousa

Antonio Trigo

João Varajão

DevOps – fundamentos e perspetivas

DevOps – foundations and perspectives

Leandro Sousa, Instituto Politécnico de Coimbra, ISCAC, Quinta Agrícola, Bencanta, 3040-316 Coimbra, Portugal, leandro.sousa1993@gmail.com

António Trigo, Instituto Politécnico de Coimbra, ISCAC, Quinta Agrícola, Bencanta, 3040-316 Coimbra, Portugal / Universidade do Minho, Centro ALGORITMI, 4804-533 Guimarães, Portugal, antonio.trigo@gmail.com

João Varajão, Universidade do Minho, Centro ALGORITMI, 4804-533 Guimarães, Portugal, varajao@dsi.uminho.pt

Resumo

O *DevOps* é frequentemente referido pelos profissionais da área das tecnologias da informação como um movimento, cultural ou profissional, que apresenta uma nova abordagem de entrega de aplicações informáticas (*software*), através da colaboração entre as equipas de desenvolvimento e das equipas de operações. Tem subjacente um conjunto diversificado de princípios, relacionados com cultura, automatização, *lean*, monitorização e partilha; e práticas, tais como, *continuous integration* e *continuous deployment*. Este artigo procura fazer uma reflexão sobre as várias vertentes do *DevOps*, evidenciando os principais benefícios e barreiras à sua adoção.

Palavras-chave: *DevOps*; Desenvolvimento; Operações; Profissionais de TI; CALMS.

Abstract

DevOps is often defined by information technology professionals as a movement, cultural or professional, that brings a new approach to the delivery of software applications through the close collaboration between the development and operations teams. It is grounded by a diverse set of principles, such as culture, automation, lean, monitoring and sharing; and practices, such as, continuous integration and continuous deployment. This article aims to review various aspects of DevOps, highlighting the main benefits and barriers of its adoption.

Keywords: *DevOps; Development; Operations; IT Professionals; CALMS.*

1. INTRODUÇÃO

As empresas focadas no desenvolvimento de aplicações informáticas *(software)* necessitam de continuamente procurar a melhoria da gestão dos seus projetos, para que os seus produtos/serviços atinjam patamares superiores de qualidade e cheguem mais depressa ao mercado. O objetivo primordial é satisfazer os seus clientes, dado que este um é importante indicador do sucesso dos projetos (Varajão, 2018).

ISSN 2183-489X

É neste contexto que surge, há não muitos anos, o fenómeno do desenvolvimento ágil, o qual procura envolver o cliente no trabalho da equipa de desenvolvimento, visando produzir *software* de forma mais rápida e, sobretudo, mais iterativa. Uma vez que o cliente acompanha de forma próxima o desenvolvimento dos projetos, tal resulta num aumento da sua satisfação (Nuottila, Aaltonen, e Kujala, 2016). Associado ao desenvolvimento ágil e à necessidade de colocar frequentemente as novas versões (parciais) de *software* em produção, surgiu um outro fenómeno: o *DevOps*.

O *DevOps* procura integrar todo o processo de desenvolvimento de *software*, desde a análise de requisitos até à colocação do *software* em produção (Humble & Farley, 2010). Esta integração é desejavelmente feita através de procedimentos automáticos, incluindo, por exemplo, a realização de testes, que procuram garantir que o *software* satisfaz os requisitos de qualidade. Para além da automatização da entrega de *software*, o *DevOps* procura também garantir, através de processos automáticos (como, por exemplo alarmísticas), que os erros detetados no *software* em produção chegam rapidamente ao conhecimento das equipas de desenvolvimento. Assim, estas podem corrigir os eventuais erros e colocar rapidamente em produção uma nova versão do *software*.

De modo a que estes mecanismos funcionem, é necessário que as equipas de desenvolvimento e as equipas de operações colaborem e sejam solidárias entre si. É este tipo de envolvimento entre os elementos das diferentes equipas que faz com que o *DevOps* seja interpretado como um novo movimento dos profissionais de Tecnologias da Informação (TI), em que todos têm como princípio base do seu trabalho a partilha e o trabalhar para um objetivo comum. É assim facilmente compreensível que, para que o *DevOps* tenha sucesso, é necessário que o mesmo faça parte da cultura organizacional (Humble & Farley, 2010; Kornilova, 2017; Walls, 2013).

Este artigo exploratório, de revisão, procura fazer uma reflexão sobre as várias vertentes do *DevOps*, evidenciando os principais benefícios e barreiras da sua adoção, os quais são explanados nas próximas secções. Termina com algumas considerações finais e apontadores para trabalho futuro.

2. DESENVOLVIMENTO E ENTREGA DE SOFTWARE

Desde o desenvolvimento até produção, existem várias etapas que devem ser respeitadas quando se desenvolve *software*. Uma das grandes diferenças entre duas das abordagens principais do desenvolvimento de *software*, cascata (*waterfall*) e ágil (*agile*), é a forma como os requisitos são definidos (Huckabee, 2015). Em *cascata* os requisitos são identificados inicialmente no projeto e são esses requisitos que conduzem ao produto final, não sendo facilmente acomodadas alterações. Em *ágil*, os requisitos identificados podem ser alterados ao longo do projeto, conforme o *feedback* do cliente. Este trabalha colaborativamente com a equipa de desenvolvimento de forma a criar um produto com maior qualidade e que vá de encontro às suas reais necessidades (as quais podem evoluir no decorrer do projeto).

À identificação dos requisitos segue-se o desenvolvimento dos produtos. Posteriormente, o *software* resultante é tipicamente testado por equipas capacitadas para o efeito, que definem e executam casos de testes. O resultado dos testes é usado para dar *feedback* à equipa de desenvolvimento acerca de possíveis erros e correções que sejam necessárias. Os testes são normalmente executados num ambiente de controlo da qualidade (Visser, Rigal, Wijnholds, & Lubsen, 2016). Dependendo do produto e da criticidade que este tenha para o modelo de negócio, podem existir mais ambientes de teste, como, por exemplo, pré-produção ou aceitação (Visser et al., 2016). Desta forma, garante-se a entrega de *software* com mais qualidade. Depois dos vários testes serem realizados com sucesso, o *software* pode ser colocado em produção e fica pronto a ser utilizado pelo cliente. Caso surjam erros em produção, podem ser efetuadas correções, que frequentemente não passam por todas as fases do ciclo de desenvolvimento de forma a resolver os problemas com maior rapidez.

O ciclo de vida de desenvolvimento de *software*, em inglês *Software Development Life Cycle* (SDLC), é distinto conforme a abordagem utilizada seja *cascata* ou *ágil*. Ambos têm o mesmo objetivo, ou seja, a entrega de valor para o cliente, mas organizam de forma diferente as atividades de desenvolvimento, como pode ser observado na Figura 1.

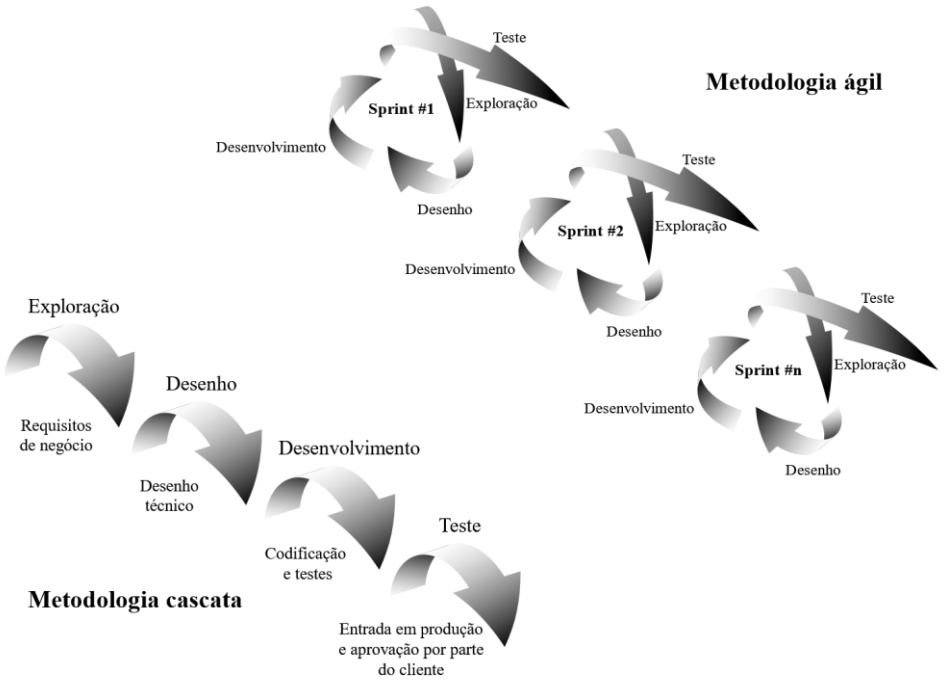

Figura 1 – Ciclo de vida de *Waterfall vs. Agile* SDLC.
Adaptado de: (STH, 2019)

A metodologia *ágil* aplica as mesmas atividades que são aplicadas em *cascata*, porém com diversas iterações (mais) curtas do ciclo de atividades. Assim, traduz-se numa entrega de valor e *feedback* contínuo (Cois, Yankel, & Connell, 2015).

De forma a suportar as várias etapas de desenvolvimento de *software*, Jobs et al. (2016) defendem que é necessário criar uma infraestrutura com ambientes distintos, mas semelhantes (de forma a que os processos e automatismos não constituam um impedimento na entrega de *software*). Estes autores defendem que devem existir quatro ambientes: Desenvolvimento; Qualidade; Aceitação; e Produção (regra geral, o ambiente de aceitação é um espelho do ambiente de Produção). Contudo, as designações e a quantidade de ambientes podem variar de organização para organização, com base nas regras estabelecidas por estas. Quando existe um encadeamento de estes ambientes, com conceitos de *build/release/deploy* associados, podemos afirmar que estamos perante um *pipeline* de desenvolvimento de *software* que suporta o SDLC (Virmani, 2015). No âmbito do *DevOps* o *pipeline* é automatizado surgindo os conceitos de *Continuous Integration* e *Continuous Deployment*, que visam colmatar as lacunas existentes nas metodologias cascata e ágil (Cois et al., 2015).

3. DEVOPS

Não obstante ter uma popularidade crescente, o *DevOps* ainda não se encontra bem definido, pois alguns autores identificam-no como sendo uma metodologia de desenvolvimento de *software*, outros como sendo uma abordagem de desenvolvimento de *software*, outros como sendo um método para a gestão de projetos de desenvolvimento de *software*, outros como sendo uma cultura organizacional, e outros ainda como sendo um movimento dos profissionais de TI, como se apresenta seguidamente.

A definição mais consentânea encontrada na literatura é a de que *DevOps* é uma nova abordagem de entrega de aplicações informáticas (*software*), a qual ocorre através da colaboração entre as equipas de desenvolvimento e de operações, em oposição à abordagem tradicional em que o desenvolvimento e as operações estão separados nos seus silos organizacionais. Embora a palavra ágil não surja muito associada a esta abordagem, a verdade é que o primeiro nome que lhe foi atribuído pelos seus proponentes iniciais, Patrick DuBois e Andrew Clay Schafer, foi "*Agile Infrastructure*" (Debois, 2008).

DevOps deriva de *Dev* + *Ops* (*Development* + *Operations*), o que remete para a unificação de equipas, transformando os silos existentes num conjunto de equipas que trabalham em prol da organização e não em torno da sua atividade dentro dela, ligando assim todas as diretrizes que já existem no desenvolvimento de *software*. Nas metodologias tradicionais, a área de desenvolvimento está preocupada apenas com uma coisa: desenvolver (incluindo o desenvolvimento contínuo de novas *features* e a correção de problemas em produção). Por outro lado, as operações são responsáveis pelo aprovisionamento de *hardware* e *software*, estando "apenas" preocupadas com a estabilidade e manutenção dos sistemas. O caminho para produção deve ser desimpedido e sem obstáculos, de forma a que as operações consigam realizar o seu trabalho através de *releases* mais rápidas (Ravichandran et al., 2016; Fallis, 2013; Kim, Humble, Debois, & Willis, 2016).

3.1. *Pilares do DevOps*

Em 2010 Wills (Wills, 2010) propôs a caracterização do *DevOps* baseada em quatro pilares: cultura (*culture*); automatização (*automation*); medição (*measurement*); e partilha (*sharing*). A sigla CAMS surge desta quatro expressões (*Culture, Automation, Measurement e Sharing*) (Humble & Molesky, 2011). Mais tarde, Jez Humble adicionou a estes quatro pilares, o pilar *Lean* (L), passando a sigla a ser CALMS (Riley, 2014). Ambas as siglas são referenciadas na literatura, mais profissional do que académica, o que realça a natureza eminentemente profissional do movimento. De seguida apresentam-se os cinco pilares do CALMS.

3.1.1.*Cultura (Culture)*

A implementação do *DevOps* traz consigo a proposta de uma nova cultura e mentalidade organizacional, que incentiva a empatia e a colaboração entre as diferentes equipas envolvidas nos processos de negócio, de desenvolvimento e de operações, evitando os silos organizacionais (Hamunen, 2016; Humble & Molesky, 2011; Laihonen, 2018; Luz, 2018; Riley, 2014). O *DevOps* propõe que esta nova cultura seja intrínseca a todos os elementos da organização, desde a administração aos programadores, passando pelas operações. As pessoas e processos são a prioridade, mas devem ser suportados por metodologias e ferramentas que permitam, de forma contínua e com ciclos de feedback, garantir um processo de melhoria contínua. Toda a organização deve ser um ecossistema cujo resultado é a entrega contínua de valor para o cliente final. A união das equipas deve ser a preocupação número um, aliada à boa comunicação, de forma a que o modelo de negócio flua naturalmente mais rápido, para não surgirem silos empresariais. Os canais de comunicação possuem um papel crucial na adoção e implementação da cultura, pois estes coordenam e conduzem a organização para os seus objetivos (Walls, 2013).

3.1.2.*Automatização (Automation)*

A automatização dos processos associados ao desenvolvimento e à entrega de *software* é a principal tarefa das organizações que querem adotar o *DevOps*. Os processos que se podem automatizar são inúmeros. Desde o processo de compilação, testes, disponibilização e instalação (*build*, testes, *release* e *deploy*) até à monitorização, o que permite um *feedback* mais rápido e o reduzir dos tempos de espera, por exemplo, com alarmísticas para as equipas de operações (Hamunen, 2016). A automatização permite, por um lado, eliminar tarefas monótonas no sentido que são transformadas em ações automáticas, evitando o erro humano, e melhorando a consistência da infraestrutura (Lwakatare, Kuvaja, & Oivo, 2016). Por outro lado, permite libertar recursos humanos da área de TI, tipicamente com competências valiosas (Trigo et al., 2010), para a realização de outras tarefas, que criem valor para a organização (Laihonen, 2018). De facto, os testes automatizados retiram uma grande parte do esforço das equipas de testes para que estas se possam focar em melhorar e criar

testes passíveis de serem automatizados, dentro do âmbito dos vários tipos de testes já referidos. Os testes automáticos são os que proporcionam a integração das práticas de *Continuous Integration* e *Continuous Deployment*, pois só através destes é que é possível obter feedback de forma a que o *software* possa prosseguir para produção (caso os testes tenham sido validados com sucesso).

3.1.3.*Lean*

O *Lean* surgiu no setor da indústria automóvel, através da Toyota (*Toyota Production System*). Suporta a produção em larga escala e procura a melhoria contínua tendo por base a redução de desperdícios, de diferentes naturezas, como, por exemplo, de tempo e de dinheiro, por forma a entregar mais valor ao cliente (Krafcik, 1988). A cultura *DevOps* está fortemente associada aos processos ágeis de desenvolvimento de *software*, que pressupõem o desenvolvimento de *software* com base no *feedback* do cliente e a entrega de *software* em pequenas iterações e de forma regular (Kim et al., 2016). As metodologias *Lean*, focadas na melhoria contínua e sistemática, são um pilar importante do *DevOps* (Kim et al., 2016), tornando o processo de desenvolvimento e de entrega de *software* mais robusto, eficaz e eficiente.

3.1.4.Medição (Measurement)

A abordagem *DevOps* sugere que a medição e o controlo das equipas de desenvolvimento e de operações sejam feitos em simultâneo, sendo as equipas premiadas segundo os mesmos parâmetros, em que o objetivo comum é a produção de valor para o cliente. O sucesso da equipa de desenvolvimento é medido em função da aplicação que está em produção, utilizando a equipa de desenvolvimento o *feedback* da equipa de produção para a tomada de decisão quanto a novas melhorias e funcionalidades (Lwakatare et al., 2016). É possibilitada, assim, a criação de aplicações mais estáveis (Humble & Molesky, 2011). A grande inovação ao nível da medição e controlo do processo de desenvolvimento e de entrega de *software* é o mesmo ser visto de forma holística, em que todas as equipas envolvidas devem colaborar de forma a enriquecer estes dados e a sua qualidade.

3.1.5.Colaboração (Sharing)

Tradicionalmente as equipas envolvidas no processo de desenvolvimento de *software* não têm uma relação saudável com as equipas de operações: o desenvolvimento está mais preocupado com a entrega do *software*; enquanto que as operações querem garantir que existe estabilidade nas aplicações em produção. A cultura *DevOps* propõe a melhoria desta relação, com a ideia de que todos devem colaborar, evitando os silos organizacionais, e que as lições aprendidas por uns sejam partilhadas com todos, possibilitando melhorias na cultura (Hamunen, 2016; Humble & Molesky, 2011; Kim et al., 2016; Laihonen, 2018; Luz, 2018; Riley, 2014). Desta forma, a cultura *DevOps* promove uma comunicação mais frequente e eficiente, envolvendo todas as equipas no processo do

início ao fim. Como confere mais responsabilidade a todas as equipas, estas têm de colaborar mais eficientemente. Apesar de não ser esperado que os *developers* executem o trabalho das operações, e vice-versa, é possível que estes colaborem de forma a corrigirem os eventuais problemas mais rapidamente. O maior desafio é quebrar estas barreiras entre as equipas, sendo importante a utilização de ferramentas que facilitem a comunicação. A produção de documentação de forma colaborativa de suporte a todo o processo de entrega e desenvolvimento é outro aspeto importante, pois permite que novos elementos sejam integrados na equipa de forma mais eficiente (Walls, 2013).

3.2. *Princípios*

Existem princípios básicos que devem ser seguidos de forma a que a curva de implementação do *DevOps* seja a mais linear possível (Kim et al., 2016):

- Testar sempre que for oportuno, de modo a garantir a integridade de todo o ciclo de desenvolvimento e do próprio *software*;
- Melhoria contínua dos processos implementados, com a adoção de novas técnicas e ferramentas que visem criar uma maior qualidade do produto, e para que as entregas sejam ainda mais rápidas;
- Automatizar tudo o que for possível, para evitar o erro humano e investimentos desnecessários em recursos humanos;
- Trabalhar de forma a criar uma equipa única, para trabalhar em torno dos mesmos objetivos, eliminando assim silos existentes;
- Separar o processo por áreas e atividades, não de forma disruptiva que vá contra o princípio anterior, mas de forma a que se saiba onde começa e termina cada uma das fases do ciclo;
- Estar sempre a par das novas tecnologias de forma a que se consiga criar *Minimum Viable Products* (MVP) com novas soluções que contribuam para o modelo de negócio.

3.3. *Práticas*

Existem várias práticas que têm de se ter em conta quando se aborda *DevOps*, tais como, *Continuous Integration (CI), Continuous Delivery* e *Continuous Deployment (CD)* (Shahin, Ali Babar, & Zhu, 2017), as quais são descritas de seguida.

- *Continuous Integration* é a mais comum das três práticas: visa integrar o trabalho em progresso de forma a que toda a equipa tenha *feedback* constante do desenvolvimento. Consiste em estar ligado continuamente a um repositório de código, de forma a detetar novas alterações (*webhook*) num determinado ramo (*branch*). Quando são detetadas novas alterações, é lançado um processo onde o código é compilado, podendo ou não

ser testado através de testes unitários para uma primeira validação, de forma a confirmar que as alterações efetuadas estão corretas.

- *Continuous Delivery* é a prática que garante que o código compilado está pronto para ser disponibilizado nos diversos ambientes referidos na secção 2. Depois de passar todos os controlos de qualidade que visam garantir que o risco do *deploy* falhar é mais baixo (numa determinada máquina), pode então ser disponibilizada a nova alteração no ambiente em questão. Nesta prática, o código é disponibilizado depois de verificado o sucesso nos dois primeiros ambientes. Existe um *gatekeeper* que decide se as alterações avançam para produção. Esta prática pressupõe obrigatoriamente o uso de um passo manual no processo.
- *Continuous Deployment* é a prática que visa eliminar o passo manual referido anteriormente. Desta forma, sempre que um ciclo é concluído com sucesso, isso significa que existem novas alterações em produção. Tal também significa que as alterações efetuadas por uma equipa podem ser instaladas, independentemente do estado das alterações de outra equipa. Desta forma, é possível aumentar a produtividade e diminuir os ciclos de *release*.

Estas três práticas são as mais comuns e seguidas pela comunidade *DevOps* e, se bem implementadas, são o caminho para o sucesso na adoção da cultura *DevOps*. Em linhas gerais, pode-se afirmar que CI/CD leva à produção de novas versões de *software* (*releases*) mais rapidamente e com mais frequência, de forma a eliminar os obstáculos que existem entre o desenvolvimento e a produção. O cliente é beneficiado, pois obtém um acesso mais rápido e frequente a novas funcionalidades.

3.4. *Ciclo DevOps*

O ciclo de *DevOps* apresentado na Figura 2 ilustra todos os processos inerentes ao desenvolvimento de uma aplicação.

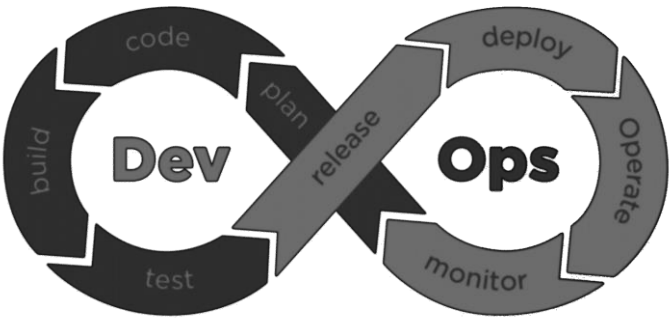

Figura 2 – Ciclo *DevOps*.
Fonte: (Kornilova, 2017)

O planeamento é considerado a primeira fase, sendo seguido pela codificação. Estas duas fases são normalmente definidas pelas equipas de desenvolvimento, com a participação da equipa de negócio (que define os requisitos a implementar). As fases de *build, test, release* e *deploy* são, por norma, automatizadas através de práticas de CI/CD. Nestas fases, o código desenvolvido anteriormente é compilado, testado e instalado nos ambientes, até chegar ao cliente final. A fase de *build* é onde ocorre a compilação do código e, também, alguns testes iniciais, como, por exemplo, testes unitários. A fase seguinte é a de testes e é no seu âmbito que são efetuados testes unitários, de aceitação, integridade, performance, e testes não funcionais. Destes testes, resulta a validação de código, de serviços, de escalabilidade, de segurança, de disponibilidade, entre outros. A fase de *release* está diretamente relacionada com a de *deploy* - enquanto que *release* é o ato de disponibilizar para o público um conjunto de alterações, *deploy* é a execução do processo de instalação num determinado ambiente. As duas fases finais são as de operações e de monitorização, nas quais as equipas garantem a integridade e a manutenção das aplicações. Também são recolhidas métricas para os próximos desenvolvimentos de forma a assegurar uma melhoria contínua. Através deste ciclo, com *feedback* constante em cada fase, é possível entregar *software* ao cliente final com mais valor.

Ao processo de *build* está inerente o uso de um repositório de código, um sistema de controlo de versões, que permite que os *developers* disponibilizem e visionem as suas alterações, para que estas possam seguir até ao cliente final. Estes sistemas são os responsáveis pelo desenvolvimento em larga escala. Fundamentalmente existem dois tipos de sistemas de controlo de versões: distribuídos e centralizados. Os sistemas centralizados têm o código centralizado num determinado servidor e todas as alterações são efetuadas diretamente lá - um exemplo é o *Subversion* (SVN). Os sistemas distribuídos, por outro lado, usam cópias locais do repositório em cada computador de cada *developer*, o que garante um histórico de alterações para que, quando as alterações forem submetidas ao servidor, exista um menor risco de surgirem problemas no código. Um dos exemplos mais utilizados mundialmente é o GIT, que é a base de um dos maiores repositórios que existem na *Cloud*, o *GitHub*.

3.4.1.Ferramentas de suporte

O ciclo *DevOps* integra uma grande diversidade de ferramentas, as quais são usadas nas diferentes fases.

Para o planeamento podem ser usadas ferramentas do tipo *Office*, incluindo ferramentas de planeamento e de gestão de projetos. A título de exemplo, é possível referir o *Jira*, que visa suportar todo o SDLC e tem por base métodos ágeis, como o Scrum ou o Kanban, utilizando quadros colaborativos com os estados base "*To Do, In Progress e Done"*. É também usado para o planeamento do processo e análise dos requisitos (Bass, Weber, & Zhu, 2015).

Para o *build* está implícito o uso de um repositório (como já referido anteriormente). Através de ferramentas de integração, como, por exemplo, o *Jenkins*, torna-se possível suportar a prática de *continuous integration*. O processo de *build* é normalmente definido pelos *developers*, escolhendo estes, conforme a tecnologia, qual o melhor processo para a aplicação em desenvolvimento. O *Jenkins* é uma ferramenta *open-source* que permite diversas personalizações, o que oferece uma grande versatilidade e o torna numa das ferramentas mais utilizadas para este efeito. Devido a essa versatilidade também pode ser utilizado para execução de *deploys*, apesar de existirem ferramentas mais direcionadas para o efeito, como, por exemplo, o Ansible e o XL Deploy (ou outras ferramentas "*low-code*" que permitem a automatização mais célere). Para grandes organizações este pode ser um problema, pois os sistemas *legacy* que eventualmente possuam podem não ser compatíveis com estas ferramentas, o que nesse caso obriga a que desenvolvam o seu próprio processo. Com esse desenvolvimento existe um maior controlo de toda a infraestrutura, o que permite uma maior agilidade no processo de *build, deploy e release*. Para *release*, pode recorrer-se a um ARA (*Application Release Automation*), que permite um controlo de *releases* sem codificação de uma plataforma ou infraestrutura (Bass et al., 2015).

A escalabilidade aplicacional deve ser aplicada de forma contínua, tirando partido de ferramentas de *conteinerização*, como é o caso do *Docker*. A portabilidade é vista como o pilar de este tipo de ferramentas. Esta ferramenta visa utilizar e partilhar os recursos de uma máquina, isolando aplicações em *containers*[1].

Nas fases de operação e monitorização são utilizadas ferramentas que possibilitam o *feedback* contínuo e devolvem alarmísticas para que a equipa seja capaz de responder a eventuais falhas ou problemas detetados. Exemplos destas ferramentas são o *Zabbix* e o *Nagios* (Bass et al., 2015)

3.5. Abordagem tradicional vs. DevOps

Na tabela abaixo encontra-se uma síntese das características da entrega de *software* segundo o modelo tradicional *vs. DevOps*.

[1] Unidade padrão de *software* que empacota o código e todas as suas dependências, de modo a que o aplicativo seja executado de maneira rápida e confiável, independentemente do ambiente de computação (Docker, n.d.)

Características	Tradicional	*DevOps*	Referência
Comunicação	Silos	Colaborativa	(Walls, 2013)
Definição de requisitos de *software*	Definidos apenas no início do projeto	Podem ser definidos/alterados ao longo do projeto	(Huckabee, 2015)
Deploy para produção	Tipicamente manual	*Continuous Delivery* ou *Continuous Deployment*	(Kim et al., 2016)
Automatização de processos	Podem existir processo automatizados	SDLC totalmente automatizado	(Erichm, & Amrit, 2017)
Colaboração entre equipas	Produto final com os requisitos desenvolvidos no início do projeto	Produto iterativo com valor crescente	(Erichm, & Amrit, 2017)
Equipas de desenvolvimento	Preocupadas apenas em satisfazer os requisitos	Colaboram ao longo de todo o SDLC para uma maior entrega de valor	(Riungu-Kalliosaari, Mäkinen, Lwakatare, Tiihonen, & Männistö, 2016)
Equipas de operações	Preocupadas apenas em manter as aplicações estáveis	Gestão colaborativa com as equipas de desenvolvimento	(Riungu-Kalliosaari et al., 2016)

Tabela 1 – Entrega de *software*: Tradicional vs. *DevOps*.

4. BENEFÍCIOS DO *DEVOPS*

Existem vários tipos de vantagens resultantes da adoção do *DevOps*, as quais podem ser agrupadas em benefícios técnicos, culturais e de negócio (Kim et al., 2016). Os benefícios técnicos estão relacionados com as práticas de CI/CD e com a velocidade de resolução de problemas. Os benefícios culturais contribuem para a melhoria da comunicação e para a criação de laços mais fortes entre os colaboradores das diferentes equipas e, consequentemente, para se ter colaboradores mais motivados, que contribuam para alcançar a missão e a visão da organização. Os benefícios de negócio estão relacionados com o cliente final e com a velocidade de entrega de valor, garantindo ainda uma maior estabilidade do negócio e mais espaço para inovar.

Os diversos benefícios são o resultado da boa aplicação dos princípios e práticas descritos anteriormente e decorrem de práticas de versionamento de código, integração contínua, testes contínuos e *deploy* contínuo, tudo com base numa monitorização contínua. Isto permite dar lugar a um princípio base do *Lean*: a melhoria contínua (Kim et al., 2016). A cultura *DevOps* deve ser vista como um acelerador de desenvolvimento de *software* que se baseia em metodologias *ágeis* de desenvolvimento. Podem adaptar-se as ferramentas aos processos ou vice-versa. Esta cultura permite uma abordagem híbrida em que podem ser privilegiados alguns princípios em detrimento de outros, para que seja possível adotar gradualmente todas as práticas.

Automatizar é um dos maiores benefícios de que se pode tirar partido. Assim, é possível providenciar as práticas já descritas, utilizar os mesmos métodos em todos os ambientes de forma a libertar recursos humanos (Laihonen, 2018).

A entrega de valor rápida e contínua é um dos grandes benefícios dos clientes (Shahin et al., 2017). O *feedback* que pode ser retribuído pelos clientes através de implementações de recolha de métricas para a construção de *dashboards* é também uma das mais-valias (Shahin et al., 2017).

O *DevOps* permite o aumento do desempenho e da produtividade organizacional do departamento de TI, a redução de custos no ciclo de vida de desenvolvimento de *software*, a melhoria na eficiência e eficácia operacional, e um maior alinhamento com o negócio e das equipas de desenvolvimento e de operações (Forsgren, Humble, Kim, Brown, & Kersten, 2017; Luz, 2018).

Benefícios	**Referências**
Melhoria da comunicação entre equipas	(Amaradri & Nutalapati, 2016; Lwakatare et al., 2019; Senapathi, Buchan, & Osman, 2018)
Maior partilha de conhecimento entre equipas	(Senapathi et al., 2018)
Maior motivação e alcance dos objetivos dos colaboradores	(Lwakatare, 2017)
Aumento do desempenho e da produtividade organizacional	(Forsgren et al., 2017; Luz, 2018; Lwakatare et al., 2019)
Integração contínua	(Amaradri & Nutalapati, 2016; Kim et al., 2016)
*Deploy c*ontínuo	(Amaradri & Nutalapati, 2016; Kim et al., 2016; Lwakatare et al., 2019)
Melhoria do tempo de entrega	(Lwakatare et al., 2019; Ozkaya, 2019)
Testes contínuos	(Amaradri & Nutalapati, 2016; Laihonen, 2018)
Monitorização contínua	(Amaradri & Nutalapati, 2016; Shahin et al., 2017)
Entrega de valor contínua	(Lwakatare, 2017; Shahin et al., 2017)
Feedback contínuo	(Amaradri & Nutalapati, 2016; Shahin et al., 2017)
Melhoria da qualidade do *software*	(Luz, Pinto, & Bonifácio, 2018; Lwakatare et al., 2019; Senapathi et al., 2018)
Melhoria contínua (*Lean)*	(Kim et al., 2016; Lwakatare et al., 2019)

Tabela 2 – Benefícios da cultura *DevOps*.

5. DESAFIOS E BARREIRAS À ADOÇÃO

Bass et al. (2015) questionam a razão do *DevOps*, que procura melhorar a comunicação entre as equipas de desenvolvimento e de operações, ainda não ser uma prática generalizada. Segundo estes autores, o principal problema é a resistência à mudança. Vista como uma das principais questões nas organizações, é necessário treinar e motivar as pessoas para que a adoção seja feita sem entraves e

que surja a vontade de transformar e melhorar os processos atuais, de forma a que a visão da empresa seja atingida mais rapidamente.

Outra barreira é a falta de comunicação, dado que as empresas, por norma, trabalham em silos. Negócio, Desenvolvimento e Operações, cada uma destas áreas está apenas preocupada com os problemas da sua área e em como solucioná-los. Para que isto mude, é necessário que cada equipa assuma responsabilidades novas e diferentes de forma a que o objetivo final, a entrega de valor contínua, seja o mesmo para todas as áreas (Riungu-Kalliosaari et al., 2016).

A dimensão da organização é vista como um desafio, sendo mais fácil implementar a cultura e práticas *DevOps* em organizações com poucos colaboradores e em equipas pequenas, como numa *startup*, do que em grandes organizações multinacionais. Independentemente da dimensão da organização, um desafio que se coloca é o tipo de abordagem para a implementação a seguir: *top-down* ou *bottom-up*. A primeira, responsabiliza a administração, que tem de incutir a cultura aos restantes colaboradores da organização. Se, por outro lado, for escolhida uma abordagem *bottom-up*, são os programadores os responsáveis por apresentar as vantagens da adoção aos seus responsáveis, e assim sucessivamente, até chegar à administração.

Quando se considera a adoção da cultura *DevOps* é necessário ter em conta os benefícios inerentes, como, por exemplo, a entrega de valor ao cliente final de forma mais rápida e contínua, em contraposição ao risco de surgirem problemas na implementação. É normal que estes aconteçam, o que torna necessário que a equipa implementadora esteja treinada para minimizar os riscos e que seja capaz de criar planos de contingência válidos.

É necessário ter em conta a constante evolução e mutação dos processos inerentes à cultura *DevOps*, dado que podem envolver a mudança de práticas, ferramentas e definições, e as organizações têm de estar preparadas para se ajustarem sempre que necessário (Riungu-Kalliosaari et al., 2016).

Barreiras	Referências
Prática não generalizada	(Bass et al., 2015; Lwakatare, 2017)
Resistência à mudança	(Amaradri & Nutalapati, 2016; Bass et al., 2015)
Falta de comunicação	(Amaradri & Nutalapati, 2016; Ozkaya, 2019; Riungu-Kalliosaari et al., 2016)
Evolução tecnológica	(Lwakatare, 2017; Riungu-Kalliosaari et al., 2016)
Mudança dos processos	(Lwakatare, 2017; Lwakatare et al., 2019; Riungu-Kalliosaari et al., 2016)
Mudança das ferramentas	(Lwakatare, 2017; Lwakatare et al., 2019; Riungu-Kalliosaari et al., 2016)
Dimensão da organização	(Amaradri & Nutalapati, 2016)
Falta de competências técnicas	(Lwakatare et al., 2019)

Tabela 3 – Barreiras à adoção de *DevOps*

6. CONCLUSÃO

Este artigo apresenta diversas vertentes do *DevOps*, destacando os benefícios e barreiras da sua adoção. É fundamental a compreensão das práticas, princípios e ferramentas que devem ser tidos em conta no contexto do *DevOps*, de modo a que a sua adoção possa ser devidamente ponderada de acordo com o tipo de organização e mercados onde opera. Os vários aspetos a ter em conta fazem emergir os riscos associados à adoção, reforçando a necessidade de planear devidamente a implementação e de serem criados planos de contingência e de mitigação.

Como trabalho futuro sugere-se o aprofundar do estudo das várias vertentes deste novo "movimento", que alguns autores apresentam como sendo uma metodologia de gestão de projetos (Banica, Radulescu, Rosca, & Hagiu, 2017). Propõe-se também a realização de estudos de casos em empresas que adotaram o *DevOps*, de modo a se obter uma compreensão abrangente das reais dificuldades e resultados da sua implementação.

AGRADECIMENTOS

Este trabalho foi apoiado pela FCT - Fundação para a Ciência e Tecnologia no âmbito do projeto: UID/CEC/00319/2019.

REFERÊNCIAS

Amaradri, A. S., & Nutalapati, S. B. (2016). *Continuous Integration , Deployment and Testing in DevOps Environment*. Blekinge Institute of Technology. Retirado de http://www.diva-portal.se/smash/get/diva2:1044691/FULLTEXT02.pdf

Aruna Ravichandran, Taylor, K., & Waterhouse, P. (2016). *Practical DevOps*. *DevOps for Digital Leaders*. CA Press. https://doi.org/10.1007/978-1-4842-1842-6

Banica, L., Radulescu, M., Rosca, D., & Hagiu, A. (2017). Is DevOps another Project Management Methodology? *Informatica Economica*, *21*(3/2017), 39–51. https://doi.org/10.12948/issn14531305/21.3.2017.04

Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A Software Architect's Perspective. https://doi.org/10.1017/CBO9781107415324.004

Cois, C. A., Yankel, J., & Connell, A. (2015). Modern DevOps: Optimizing software development through effective system interactions. *IEEE International Professional Communication Conference*, *2015-Janua*. https://doi.org/10.1109/IPCC.2014.7020388

Debois, P. (2008). Agile infrastructure and operations: how infra-gile are you? In *Agile 2008 Conference* (pp. 202–207). IEEE.

Docker. (n.d.). What is a Container? | Docker.

Erich, F. M. A., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of Software: Evolution and Process*, *29*(6), e1885. https://doi.org/10.1002/smr.1885

Fallis, A. G. (2013). *Effective DevOps*. *Effective DevOps* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Forsgren, N., Humble, J., Kim, G., Brown, A., & Kersten, N. (2017). *2017 State of DevOps Report*. Retirado de http://www.berrykersten.nl/wp-content/uploads/2017-state-of-devops-report.pdf

Hamunen, J. (2016). *Challenges in Adopting a Devops Approach to Software Development and Operations*. Aalto University. Retirado de http://urn.fi/URN:NBN:fi:aalto-201609083476

Huckabee, W. A. (2015). Requirements Engineering in an Agile Software Development Enviroment. *Defense ARJ*, *22*(4), 394–415. Retirado de https://www.researchgate.net/profile/Allen_Huckabee/publication/281850655_Defense_Acquisition_Research_Journal/links/55fb466008ae07629e07bbd8.pdf

Humble, J., & Farley, D. (2010). *Continuous Delivery*. Pearson Education.

Humble, J., & Molesky, J. (2011). Why enterprises must adopt devops to enable continuous deliver. *Cutter IT Journal*, *24*(8).

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook : How to Create World-Class Agility, Reliability, and Security in Technology Organizations. *The DevOps Handbook*. https://doi.org/2016951904

Kornilova, I. (2017). DevOps is a culture, not a role! Acedido a 24 de junho, 2019, from https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

Krafcik, J. F. (1988). Triumph of the lean production system. *MIT Sloan Management Review*, *30*(1), 41.

Laihonen, P. (2018). *Adoption of DevOps Practices in the Finnish Software Industry: an Empirical Study*. Aalto University.

Luz, W. P. (2018). *Uma Caracterização da Adoção de DevOps Utilizando Grounded Theory*. Universidade de Brasília. Retirado de http://repositorio.unb.br/bitstream/10482/33950/1/2018_WelderPinheiroLuz.pdf

Luz, W. P., Pinto, G., & Bonifácio, R. (2018). Building a collaborative culture. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '18* (pp. 1–10). New York, New York, USA: ACM Press. https://doi.org/10.1145/3239235.3240299

Lwakatare, L. E. (2017). *DevOps adoption and implementation in software development practice : concept, practices, benefits and challenges*.

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., … Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. *Information and Software Technology*, *114*, 217–230. https://doi.org/10.1016/j.infsof.2019.06.010

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016). An exploratory study of devops extending the dimensions of devops with practices. In *The Eleventh International Conference on Software Engineering Advances (ICSEA 2016)* (pp. 91–99). Rome, Italy.

Nuottila, J., Aaltonen, K., and Kujala, J. (2016). Challenges of adopting agile methods in a public organization. *International Journal of Information Systems and Project Management*, *4*(3), 65–85. https://doi.org/10.12821/ijispm040304

Ozkaya, I. (2019). Are DevOps and Automation Our Next Silver Bullet? *IEEE Software*, *36*(4), 3–95. https://doi.org/10.1109/MS.2019.2910943

Riley, C. (2014). How to Keep CALMS and Release More! Retirado de https://blog.rapid7.com/2014/10/24/how-to-keep-calms-and-release-more/

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). DevOps Adoption Benefits and Challenges in Practice: A Case Study. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 10027 LNCS, pp. 590–597). https://doi.org/10.1007/978-3-319-49094-6_44

Senapathi, M., Buchan, J., & Osman, H. (2018). *DevOps Capabilities, Practices, and Challenges:*

Insights from a Case Study. https://doi.org/10.1145/3210459.3210465
Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*(Ci), 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629
STH. (2019). Agile vs. Waterfall: Which is the Best Methodology for Your Project? Acedido a 2 de junho de 2019, from https://www.softwaretestinghelp.com/agile-vs-waterfall/
Trigo, A., Varajão, J., Algoritmi, C., Molina-Castillo, F. J., Gonzalvez-Gallego, N., Soto-Acosta, P., & Barroso, J. (2010). IT professionals: An iberian snapshot. *International Journal of Human Capital and Information Technology Professionals*, *1*(1). https://doi.org/10.4018/jhcitp.2010091105
Varajão, J. (2018). The many facets of information systems (+projects) success. *International Journal of Information Systems and Project Management*, *6*(4), 5–13.
Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. *5th International Conference on Innovative Computing Technology, INTECH 2015*, (Intech), 78–82. https://doi.org/10.1109/INTECH.2015.7173368
Visser, J., Rigal, S., Wijnholds, G., & Lubsen, Z. (2016). *Building Software Teams: Ten Best Practices for Effective Software Development*. (O'REILLY, Ed.). O'Reilly Media.
Walls, M. (2013). *Building a DevOps Culture*. O´Reilly.
Wills, J. (2010). What Devops Means to Me. Acedido a 24 de junho de 2019, from https://blog.chef.io/2010/07/16/what-devops-means-to-me/