isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

DEFINITIVO

Estudo e aplicação de processos DevOps numa empresa de TI

Autor

João Alexandre Miranda Pequeno

Orientador

João António Pereira Almeida Durães

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA

Coimbra, julho 2022

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Estudo e aplicação de processos DevOps numa empresa de TI

Relatório de Estágio de Natureza Profissional para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

João Alexandre Miranda Pequeno

Orientador

João António Pereira Almeida Durães

Supervisor na empresa NOESIS

Márcio Filipe Alves Carvalho

Coimbra, julho 2022

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA

Agradecimentos

A elaboração do presente relatório, referente à obtenção do grau de Mestre em Informática e Sistemas, não seria possível sem o apoio de algumas pessoas e entidades. Assim sendo, pretendo agradecer de forma individual a todos os que sempre me apoiaram e, de alguma forma, contribuíram para a realização e concretização desta etapa final do mestrado. Deste modo, resta-me agradecer:

Aos meus pais, João e Fátima, por acreditarem sempre em mim e nas minhas capacidades. A vossa luta e esforço diário fizeram crescer a pessoa que sou hoje e eu estarei eternamente agradecido por isso. Obrigado por tudo, nunca vou conseguir retribuir tudo o que vocês já me deram. Amo-vos muito.

À restante família pelo apoio e confiança com que sempre me brindaram. Queria deixar um agradecimento especial ao meu irmão pelo seu apoio incondicional.

À Rita, que esteve sempre ao meu lado nos bons e maus momentos e que faz com todos os desafios se tornem mais fáceis. Agradeço por todo o apoio, toda a força e toda a paciência.

A toda a equipa da empresa Noesis, principalmente ao meu supervisor de estágio, Márcio Carvalho, pela oportunidade e ajuda, e a todos os meus colegas de trabalho pela disponibilidade e paciência que sempre mostraram para me ajudar em todas as dificuldades ao longo deste percurso.

Ao meu orientador do ISEC, professor João Durães, pela sua preocupação e ajuda em todas as questões que surgiram ao longo do estágio, como na construção deste relatório.

A todos os docentes que contribuíram para a minha formação pessoal e educacional ao longo de todos estes anos. Foi, sem dúvida alguma, importantíssimo a dedicação e o contributo de cada um.

A todos os enunciados, muito obrigado!

Resumo

Atualmente as empresas vivem períodos de grande mudança na forma como as suas equipas desenvolvem *software* devido ao facto de existir cada vez mais dificuldades na comunicação entre a equipa de desenvolvimento e a equipa de operações e na necessidade de entregar *software* de forma mais rápida. Através da necessidade de eliminar estas barreiras e colocar o desenvolvimento e a entrega de *software*, surge DevOps como uma metodologia que vem complementar a ainda predominante metodologia ágil, nomeadamente em áreas onde esta não atua, procurando aumentar ainda mais a rapidez e a eficiência das organizações no desenvolvimento dos seus projetos.

O DevOps é frequentemente referido na área da tecnologia como uma nova abordagem de entrega de *software*, através da colaboração entre a equipa de desenvolvimento e equipa de operações. Apesar das semelhanças existentes entre a metodologia de desenvolvimento ágil e o DevOps, ambos os conceitos possuem algumas diferenças a nível da entrega de *software*, no sentido em que o DevOps aproxima todos os intervenientes responsáveis, com o apoio de um conjunto de ferramentas que permitem automatizar e otimizar todos os seus processos.

O presente relatório de estágio pretende dar a conhecer os conceitos, práticas e ferramentas de DevOps e é conduzido à aplicação prática de um modelo automatizado para o processo de análise, desenvolvimento, testes e entrega de *software* num projeto desenvolvido pela empresa Noesis. Como principais resultados identificam-se a revisão da literatura efetuada e a aplicação das práticas e conceitos de DevOps que se tornaram relevantes para a componente prática realizada.

No final desta aplicação, foram analisadosos resultados obtidos a um conjunto de métricas como tempo de entrega do *software*, tempo de deteção de falhas, tempo de reparação de falhas, satisfação do cliente, tempo de consideração de uma falha e qualidade do *software* , com o objetivo de avaliar os benefícios da implementação de DevOps.

Palavras-Chave: DevOps; *Software*; Sistemas de informação; Tecnologias de Informação; Ágil; Desenvolvimento de *software*; Operações

Abstract

Currently companies are going through periods of great change in the way their teams develop software due to the increasing difficulties in communication between the development team and the operations team and the need to deliver software faster. The need to eliminate these barriers and put software development and delivery in first place led to the emergence of DevOps as a methodology that complements the still predominant agile methodology, particularly in areas where it doesn't work, seeking to further increase the speed and efficiency of organizations in the development of their projects.

The DevOps methodology is often referred to in the technology field as a new approach to software delivery, through collaboration between the development team and the operations team. Despite the similarities between the agile development methodology and DevOps, the two concepts have some differences in terms of software delivery, in the sense that DevOps brings together all responsible parties, with the support of a set of tools that allow automating and optimizing all its processes.

This internship report intends to present the concepts, practices and tools of DevOps and leads to the practical application of an automated model for the process of analysis, development, testing and delivery of software in a project developed by the company Noesis. Therefore, the main results are the literature review and the application of DevOps practices and concepts that were relevant to the practical component.

In the end of this application, the results obtained were analyzed to a set of metrics such as software delivery time, failure detection time, failure repair time, customer satisfaction, time to consider a failure and software quality, with the aim of evaluating the benefits of implementing DevOps.

Keywords: DevOps; *Software*; Information systems; Information technologies; Agile; Software development; Operations

Índice

Conteúdo

Índice de figuras

Índice de tabelas

Acrónimos

Abreviatura/Sigla	Descrição
AI	*Artificial Intelligence*
ALOC	*Software* de simulação de alocações e custos
CI	*Continuous Integration*
CD	*Continuous Delivery*
DEV	*Development*
IDE	*Integrated Development Environment*
ISEC	Instituto Superior de Engenharia de Coimbra
ISO	*International Organization Standardization*
MAT	*Software* de gestão de material
MTTD	*Mean time to detect*
MTTR	*Mean time to resolve*
NTX	*Ngine Testing Experience*
POM	*Project Object Model*
PROD	*Production*
QA	*Quality Assurance*
SI	Sistemas de Informação
TI	Tecnologias de Informação
XML	*Extensible Markup Language*
XP	*Extreme Programming*

1 Introdução

As empresas do sector da TI dependem cada vez mais das metodologias de desenvolvimento de *software*, uma vez que estas procuram as boas práticas e a rapidez de desenvolvimento de *software* e consequente diminuição de custos, procurando garantir o foco no desenvolvimento de *software* de qualidade e a satisfação dos seus clientes. A metodologia de desenvolvimento Ágil procura envolver o cliente no trabalho da equipa de desenvolvimento, visando produzir o *software* de forma mais rápida e iterativa. Nesta metodologia de desenvolvimento de *software* ocorrem muitas mudanças e implementações de risco em curtos espaços de tempo, devido ao facto de a equipa de desenvolvimento trabalhar no mesmo repositório e ter necessidade de integrar recursos com muita mais frequência.

Na fase da entrega de *software*, em grande parte das situações, a equipa de desenvolvimento e a equipa de operações entram em conflito entre si. A equipa de desenvolvimento está focada em garantir a criação de novas soluções, para que os sistemas se adaptem às alterações das necessidades de negócio dos seus clientes. Por outro lado, a equipa de operações coloca estas soluções no ambiente de produção. Por vezes,estas soluções podem comprometer a estabilidade e fiabilidade do sistema que se encontra em produção, e consequentemente do negócio da empresa, daí existirem alguns conflitos entre estas duas equipas. O que acontece em grande parte dos casos é que as equipas de operações, responsáveis por disponibilizar a infraestrutura para o desenvolvimento, não atendem totalmente às necessidades do negócio, o que resulta em atrasos no ciclo de desenvolvimento.

A incompatibilidade entre estas equipas deve-se ao facto da equipa de desenvolvimento desejar que as funcionalidades por si desenvolvidas sejam entregues rapidamente e a equipa de operações em garantir a estabilidade do sistema, implicando não haver muitas alterações no ambiente de produção. Por outro lado, a equipa de operações passa o ponto de situação do ambiente de produção e caso existam alguns problemas relacionados com a equipa de desenvolvimento, esta será chamada a intervir. Associando a incompatibilidade entre as duas equipas e a necessidade de colocar mais frequentemente novas versões de *software* em produção sem comprometer a estabilidade do sistema, surgiu a metodologia DevOps.

O DevOps tem como objetivo integrar todo o ciclo de desenvolvimento de *software*, desde a análise de requisitos até à implementação do *software* em produção (Humble & Farley, 2010). Para que a entrega do *software* seja feita sem problemas, toda a integração a que o DevOps se propõe é feita através de procedimentos automáticos, onde os erros e problemas do *software* são rapidamente detetados e corrigidos pela equipa de desenvolvimento e, posteriormente colocados novamente em produção pela equipa de operações. Para que esta tarefa avance sem problemas

e atrasos é necessário que exista ligação entre as equipas de desenvolvimento e operações, pelo que esta metodologia é interpretada como uma nova forma de trabalho e partilha, onde estas duas equipas trabalham pelo mesmo objetivo, que é garantir a qualidade do *software*, a qualidade da estrutura e, principalmente, a satisfação do cliente.

A metodologia DevOps preza pela utilização de uma série de mecanismos que auxiliem o processo de desenvolvimento de *software* permitindo que o desenvolvimento ocorra com o mínimo de interrupções e problemas possíveis. Colocando em ação as suas características, práticas e ferramentas que serão abordados no capítulo 2, será possível garantir uma série de benefícios, que serão abordados no subcapítulo 2.5.

1.1 Contextualização do estágio

O presente relatório pretende dar a conhecer o trabalho realizado pelo estagiário João Alexandre Miranda Pequeno, no âmbito da disciplina “Projeto ou Estágio” do Mestrado em Engenharia Informática, no ramo de Desenvolvimento de *Software*, do Departamento de Engenharia Informática e Sistemas do Instituto Superior de Engenharia de Coimbra, na empresa Noesis, em Coimbra.

O estágio enquadra-se na aplicação da metodologia DevOps e o estagiário foi integrado numa equipa composta por elementos multidisciplinares, experientes e capazes de transmitir o seu *know-how*. Este estágio teve como principal objetivo melhorar a cultura da empresa e a qualidade do *software* desenvolvido através da implementação de práticas relativas à metodologia DevOps. O acompanhamento técnico e pedagógico do estagiário, bem como a supervisão do seu progresso face às atividades propostas, ficaram a cargo do Professor Doutor João António Pereira Almeida Durães (DEIS-ISEC) e do Engenheiro Márcio Filipe Alves Carvalho (Noesis). Após a revisão da literatura, o estagiário foi integrado na equipa de desenvolvimento do projeto selecionado, onde aplicou as práticas e ferramentas descritas nos capítulos que se seguem, com a finalidade de atingir os seguintes objetivos:

- Reduzir o *time-to-market* das funcionalidades implementadas;
- Reduzir o tempo de entrega do *software*;
- Diminuir os problemas encontrados no *software* em ambiente de produção;
- Tornar as equipas de desenvolvimento e operações mais eficientes;
- Melhorar a satisfação do cliente;
- Melhorar a qualidade do *software* desenvolvido;

1.2 Entidades envolvidas

De seguida serão apresentadas as entidades envolvidas na realização deste projeto de estágio.

1.2.1 Instituto Superior de Engenharia de Coimbra

O ISEC (figura 1), unidade orgânica do Instituto Politécnico de Coimbra para o ensino da tecnologia e engenharia, é uma instituição de ensino superior que tem como missão “a criação, transmissão e difusão de cultura, ciência e tecnologia”. O seu objetivo é “ser uma referência de excelência no ensino, reconhecido nacional e internacionalmente por serviços de qualidade e relevância social, com práticas flexíveis, criativas e inovadoras” (ISEC, 2022).

O ISEC tem como visão ser uma referência de excelência no ensino, reconhecido nacional e internacionalmente por serviços de qualidade e relevância social, com práticas flexíveis, criativas e inovadoras de ensino. Pretende ainda ser um parceiro privilegiado das organizações empresariais e das famílias da região onde se insere pela orientação eminentemente prática, fundada num rigoroso conhecimento teórico, que imprime a todas as suas atividades.

O ISEC é instituto da área da ciência e tecnologia que ministra a preparação para o exercício de atividades profissionais no domínio da engenharia e promove o desenvolvimento da região de Coimbra. Em termos de oferta de formação, o ISEC disponibiliza licenciaturas em diversas áreas de engenharia, tais como, Engenharia Biológica, Engenharia Biomédica, Engenharia Civil, Engenharia Eletromecânica, Engenharia Eletrotécnica, Engenharia e Gestão Industrial, Engenharia Informática, Engenharia Mecânica e Engenharia Química. Esta instituição tem cerca de 2700 alunos e, quanto à empregabilidade, destaca-se no pódio a nível nacional, com mais de 98% de alunosempregados. (Wikipedia, ISEC, 2022)

Figura 1 – Logotipo Instituto Superior de Engenharia de Coimbra

1.2.2 Noesis

A Noesis é uma consultora tecnológica multinacional, fundada em 1995, especializada em fornecer serviços e soluções tecnológicas para diversos clientes que buscam aumentar a sua competitividade, facilitar a sua gestão, reduzir os seus custos e otimizar os seus processos, com o intuito de os ajudar a acelerar para a transformação digital (Noesis, 2022).

Desde o início da sua fundação até aos dias de hoje, a Noesis está em contínua expansão, fixando-se em inúmeros países e operando em diversos mercados (Financeiro, Tecnológico, Serviços & Indústria, Fármacos, Retalho, Energia e Público). Conta atualmente com mais de 800 colaboradores distribuídos por 8 escritórios: Lisboa, Porto, Coimbra, Bruxelas, São Paulo, Dublin, Roterdão e, mais recentemente, em Boston (Noesis, 2022).

A Noesis presta os seus serviços e desenvolve soluções diferenciadoras e inovadoras, através de oito unidades de negócio especializadas:

• ***Low-Code Solutions*** – Desenvolve aplicações web e mobile inovadoras e de forma rápida, ao longo de todas as unidades de negócio, com recurso à plataforma de *lowcode Outsystems*.

• ***Data Analytics* & AI** – Providencia soluções que permitem às organizações analisar e organizar os seus dados de forma a tirarem um melhor partido dos mesmos e, com recurso à inteligência artificial e *machine learning* prever o seu progresso.

• **DevOps *& Automation*** – A estratégia DevOps permite simplificar os fluxos de trabalho de forma a fornecer recursos de qualidade aos clientes de maneira rápida e eficaz. A área de automação ajuda os clientes a automatizar os seus processos de negócio, proporcionando ferramentas para que as empresas potencializam ao máximo os seus recursos humanos e reduzam custos e tempo de produção.

• ***Enterprise Resource Planning*** – Utiliza a tecnologia SAP de forma a ajudar os clientes a acelerarem os seus processos de negócio, oferecendo um modelo de dados simplificado e funcionalidades preditivas e de simulação, *on premise* ou na *cloud*. A solução tecnológica de *process mining*, *celonis*, permite aos clientes obterem uma melhoria da performance de negócio ao proporcionar a capacidade de compreender e analisar cada processo em tempo real.

• ***Enterprise Solutions*** - Desenvolve soluções que garantem a gestão centralizada e inteligente da informação e dos ativos das organizações, proporcionando uma experiência de negócio ampla e interligada em todos os processos internos e externos.

• ***Infrastruture Solutions*** - Desenha e implementa as mais modernas soluções de infraestrutura *end-to-end*, em ambientes *on premise*, *cloud* ou *híbridos*.

• ***Professional Services*** – Dispõe de uma equipa de consultores especializados em várias áreas chave de negócio, que fornecem as melhores soluções para os projetos dos clientes Noesis.

• ***Quality Management*** – Garante a qualidade dos projetos em todo o seu ciclo de desenvolvimento e assegura a eficácia, o rigor e a redução do risco nos procedimentos de teste. (Noesis, 2022).

Figura 2 – Logotipo Noesis

1.3 Visão e objetivos

A competição entre as empresas ligadas às TI está constantemente a aumentar, o que resulta na necessidade de responder às mudanças que os mercados exigem, de forma a oferecer uma melhor experiência aos seus clientes e inovar com os seus serviços e produtos (Soni, 2015). Desta forma, a necessidade de assegurar ao cliente a qualidade do *software* e prazos de entrega tornam-se pontos fulcrais na competitividade e luta pela sobrevivência que existe, atualmente, no mercado global. Estas exigências de mercado levam este tipo de empresas a adotar metodologias de desenvolvimento de *software* para desenvolverem os seus produtos de forma mais rápida e económica, procurando sempre garantir o foco no propósito do desenvolvimento e respetiva qualidade do *software.* Após várias metodologias de desenvolvimento de *software*, surge em 2008 o DevOps como metodologia que vem complementar a ainda predominante metodologia ágil, nomeadamente em áreas onde o ágil não atua, procurando aumentar ainda mais a rapidez e a eficiência das empresas nos seus projetos. A metodologia DevOps consiste, principalmente, num conjunto de práticas destinadas a aumentar a qualidade do código desenvolvido e a preencher as lacunasexistentes na comunicação entre a equipa de desenvolvimento e a equipa de operações. Esta filosofia tem vindo a conquistar várias empresas no sentido em que engloba o ciclo de vida dos projetos e melhora a experiência da comunicação e a satisfação dos clientes.

O primeiro passo para aplicar a metodologia DevOps é mudar a mentalidade das equipas, com o objetivo de aprimorar a rapidez e a qualidade das entregas.Para isso é necessário orientar as equipas de *software* a tornarem-se mais eficientes e a trabalharem como um todo, com o objetivo de otimizar o desenvolvimento e corrigir

problemas continuamente. Para a Noesis se manter na vanguarda da tecnologia é necessário procurar as melhores tecnologias e ajustá-las à sua realidade dos seus clientes.

Assim, o objetivo geral deste estágio focou-se na aplicação dos processos e práticas associadas à metodologia DevOps num *software* internamente desenvolvido pela empresa Noesis, com o objetivo de melhorar os métodos de desenvolvimento e entrega ao cliente final. Partindo deste objetivo, procurou-se identificar os obstáculos e necessidades da aplicação de DevOps. Desta forma foi feito o levantamento de um conjunto de métricas para comparar o antes e o depois da implementação de DevOps. Como resultado, é pretendido que se comprove que esta metodologia aumenta não só o valor da empresa, como também o valor dos seus colaboradores.

A aplicação de DevOps assenta num entendimento de múltiplas vertentes: a compreensão das suas características, o entendimento dos processos da empresa onde vai ser aplicado e o conhecimento técnico das ferramentas adequadas. Para além do ganho de produtividade, uma boa aplicação de DevOps permite também melhorias significativas na qualidade do produto final conseguida através de uma maior padronização de práticas, uma gestão de testes ágil e entrega contínua. Estes ganhos podem ser observados e quantificados através dos procedimentos habituais de gestão de projeto de *software*. Para medir os ganhos através da implementação da metodologia DevOps é necessário efetuar o levantamento de um conjunto de métricas para aferir e medir as diferenças em diversos aspetos de qualidade e desempenho.

1.4 Plano de trabalhos

Para atingir os objetivos descritos no subcapítulo anterior, as tarefas desenvolvidas foram divididas nas seguintes fases:

- **Fase 1** – Revisão da literatura sobre DevOps
  - Análise do estado da arte;
  - Pesquisa acerca das melhores práticas e características de implementação da metodologia DevOps;
  - Análise dos benefícios de DevOps

- **Fase 2** – Análise de ferramentas DevOps
  - Análise de ferramentas típicas de DevOps;
  - Comparação entre as ferramentas pesquisadas;

- **Fase 3** – Estudo dos processos para implementação de DevOps
  - Relação existente com a metodologia ágil
  - Implementação do DevOps
  - Obstáculos existentes

- **Fase 4** – Identificação do caso de estudo
  - Identificação dos requisitos para o caso de estudo;
  - Levantamento dos projetos existentes;
  - Comparação entre os projetos;
  - Definição do caso de estudo;

- **Fase 5** – Aplicação DevOps ao caso de estudo
  - Implementação das práticas de DevOps ao caso de estudo;

- **Fase 6** – Análise de resultados
  - Análise dos resultados obtidos em cada métrica;
  - Comparação entre a metodologia de desenvolvimento anteriormente utilizada com a metodologia implementada;

- **Fase 7** – Escrita do relatório final

O planeamento do trabalho e o seu faseamento são apresentados de seguida:

INI		**Início dos trabalhos**
Fase 1	(INI + 4 semanas)	Tarefa T1 terminada
Fase 2	(INI + 6 semanas)	Tarefa T2 terminada
Fase 3	(INI + 8 semanas)	Tarefa T3 terminada
Fase 4	(INI + 14 semanas)	Tarefa T4 terminada
Fase 5	(INI + 16 semanas)	Tarefa T5 terminada
Fase 6	(INI + 20 semanas)	Tarefa T6 terminada
Fase 7	(INI + 24 semanas)	Tarefa T7 terminada

Tabela 1 – Plano de trabalhos

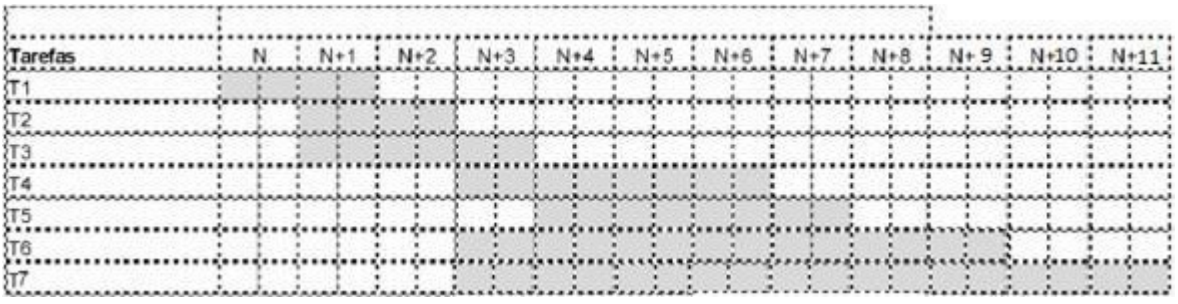


Tabela 2 – Faseamento do trabalho

1.5 Estrutura do relatório

O presente relatório de estágio está organizado em 7 capítulos:

O capítulo 1, “Introdução”, inicia-se com a motivação do trabalho realizado durante o Estágio, seguindo-se uma breve descrição da entidade de acolhimento Noesis e da instituição de ensino ISEC, a que pertence o estagiário. Nas secções seguintes são listados os objetivos definidos, é apresentado o plano de trabalhos previsto e termina com esta secção que descreve a estrutura do documento.

O capítulo 2, “Conceitos e estado da arte”, apresenta a descrição detalhada sobre a metodologia utilizada, as suas práticas, as suas características e os desafios à sua implementação.

No capítulo 3, “Caso de estudo”, é apresentado o caso de estudo que será utilizado e de que forma foi feita a sua seleção. Também serão apresentadas as métricas que serão utilizadas para a recolha de resultados e quais foram as ferramentas utilizadas.

No capítulo 4, “NTX e processos aplicados pela Noesis”, é ilustrada a arquitetura do NTX e são apresentados alguns processos aplicados pela Noesis em outros projetos.

No capítulo 5, “Implementação”, é apresentada a metodologia e as práticas desenvolvidas para aplicar DevOps no caso de estudo selecionado.

No capítulo 6, “Análise e discussão de resultados”, são apresentados os resultados obtidos na aplicação prática da metodologia DevOps e a sua análise comparativa.

No capítulo 7, “Conclusões”, são apresentadas as conclusões do trabalho efetuado bem como o trabalho futuro que poderá ser realizado futuramente.

2 Conceitos e estado da arte

Este capítulo aborda os conceitos importantes sobre a filosofia DevOps com o objetivo de dar a conhecer a metodologia que assenta neste estágio. Desta forma serão apresentados os seguintes subcapítulos: características de DevOps, benefícios de DevOps, ferramentas típicas de DevOps, benefícios de DevOps, relação com a metodologia ágil e, por fim, a adoção e implementação da metodologia, assim como os seus desafios.

2.1 Características de DevOps

Uma das características da metodologia DevOps é a melhoria na integração da equipa de desenvolvimento com a equipa de operações, no entanto, esta integração não é realizada de forma *ad-hoc*, sem qualquer metodologia ou linhas orientadoras. Assim, o conceito DevOps baseia-se em quatro características (Fitzgerald & Stol, 2014) que, ao serem seguidas corretamente, transformam a curva de implementação desta metodologia o mais linear possível. As características associadas à metodologia DevOps são definidas nos próximos subcapítulos.

2.1.1 Mudança cultural

A metodologia DevOps requer uma mudança cultural, o que significa que as interações entre as equipas de desenvolvimento e operações vão ter algumas mudanças. O principal desafio para algumas organizações, face à cultura DevOps, passa por descartar os silos existentes entre estes grupos e melhorar a comunicação interdepartamental. Desta forma, o DevOps aprimora uma comunicação saudável entre toda a empresa permitindo beneficiar de um ambiente eficaz, livrando-se das barreiras de comunicação do modo convencional. Para isso, é fundamental seguir os seguintes pontos para alcançar a desejada mudança cultural:

- Ter capacidade de detetar falhas rapidamente, de forma a antecipá-las e corrigi-las atempadamente;
- Ter capacidade de experimentar novos métodos, como novas ideias que possam melhorar e tornar mais eficiente o trabalho que é desenvolvido;
- Ter capacidade de receber sugestões de melhoramento em qualquer fase do processo de desenvolvimento.

2.1.2 Automação

Tipicamente, os processos de desenvolvimento de *software* são marcados por uma série de processos manuais, cujo resultado é o aumento do custo de desenvolvimento e entrega o *software*, uma cadência reduzida do ciclo de vida das *releases* e a qualidade reduzida. O DevOps baseia-se na automação completa da construção, implementação, testes e instalação nos diferentes ambientes, com o objetivo de aumentar a velocidade e qualidade do que é desenvolvido, obter prazos de entrega curtos e, portanto, *feedback* rápidos por parte dos clientes. (Alice Njenga, 2021)

2.1.3 Controlo

Através do controlo dos processos, ferramentas e dados obtidos, é possível manter um acompanhamento próximo não só do estado atual do desenvolvimento de *software* como também de eventuais melhorias implementadas através da utilização da metodologia DevOps. A identificação de métricas e efetuando as consequentes medições são não só essenciais para perceber o estado atual do desenvolvimento, como também para identificar certas lacunas e possíveis melhorias que possam ser feitas. Tomando certas decisões baseadas nos dados obtidos através das medições podem servir de chave para melhorar o desenvolvimento e a satisfaçãodo cliente.

2.1.4 Comunicação

O quarto princípio foca-se na criação de ciclos de *feedback* eficientes, no sentido da equipa de desenvolvimento para a equipa de operações e vice-versa, que resultam na validação rápida e deteção de erros. Através do *feedback* contínuo é possível manter o foco nos problemas, eliminando-os à medida que eles surgem, tornando a deteção de eventuais problemas mais facilitada.

A metodologia DevOps pretende que o utilizador final tome contacto o mais rapidamente possível com a *release* que foi entregue para que o seu *feedback* acerca dos desenvolvimentos efetuados seja dado o mais rapidamente possível. A partilha de informações e ideias não se adequa apenas ao utilizador final, mas também a todos os participantes nas mais diversas fases do ciclo de desenvolvimento, pois esta é promovida através de uma maior comunicação e aproximação interna entre as diferentes pessoas e equipas, que é um dos fatores principais da metodologia DevOps.

Assim, a deteção e a correção de problemas tornam-se mais simples, dado que os problemas são mais pequenos, mais baratos e mais fáceis de resolver, evitando, dessa forma, problemas maiores.

2.2 Práticas de DevOps

O aumento da rapidez e qualidade do desenvolvimento de *software* que a metodologia DevOps procura atingir é em grande parte alcançado por um conjunto de práticas de desenvolvimento de *software* ágil. Estas práticas focam-se no planeamento e adaptação contínua às rápidas mudanças nos requisitos e podem ser usadas para estender o fluxo contínuo de atividades, desde o cliente até o desenvolvimento, passando pelas operações, enquanto acrescenta qualidade ao *software* que está a ser desenvolvido (Fitzgerald & Stol, 2015).

A *pipeline de* DevOps é composta por diversas fases que podem ser observadas na *pipeline* de *DevOps*, ilustrada na figura 3 (Atlassian, 2022), segundo Manish Virmani (Virmani, 2015). A *pipeline* de DevOps trata-se de um conjunto de processos e ferramentas que permite que a equipa de desenvolvimento e a equipa de operações trabalhem de forma coesa para desenvolver e entregar o código no ambiente de produção. Cada uma das práticas da *pipeline* desempenha uma função especifica, portanto cada uma será avaliada antes de passar para a etapa seguinte. Em caso de falha, a *pipeline* é interrompida e o *feedback* é fornecido às equipas envolventes.

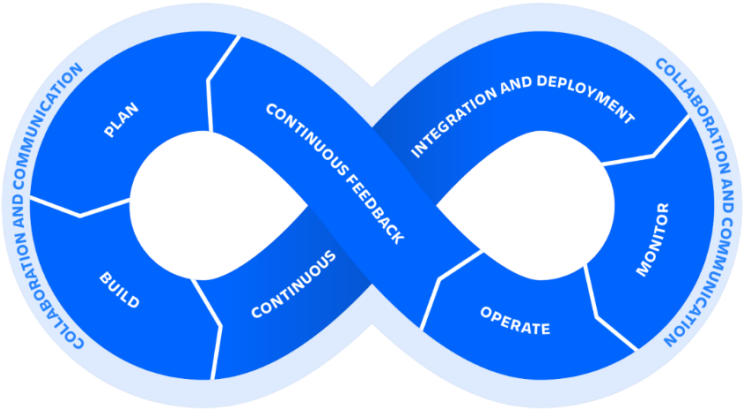


Figura 3 – Pipeline de DevOps

A seguinte lista é composta pelas práticas de DevOps mais conhecidas atualmente:

- Integração contínua
- Entrega contínua
- *Feedback* contínuo do cliente*;*
- Planeamento contínuo;
- Monitorização contínua;

Nos subcapítulos seguintes serão abordadas cada uma das práticas acima mencionadas.

2.2.1 Integração contínua

Tradicionalmente, os programadores desenvolvem o código, executam os testes unitários necessários e passam imediatamente para o próximo componente a desenvolver, sendo que estes componentes serão posteriormente integrados com todos os restantes que se encontram no repositório de código-fonte. Quando esta integração não é feita continuamente, existe um risco acrescido no que toca ao surgimento de um maior número de erros, o que numa fase adiantada do desenvolvimento poderá resultar em atrasos significativos e eventuais custos adicionais.

A prática de desenvolvimento de *software* de integração contínua (CI), de acordo com Beller, Gousios e Zaidman foi originalmente criada pela Microsoft em 1995 e mais tarde ficou conhecida como uma das principais práticas de desenvolvimento de *software em Extreme Programming* (XP) (Beller, Gousios, & Zaidman, 2017). Esta prática surgiu como uma melhor prática para as equipas de *software* que geralmente trabalham separadamente e, em determinado momento, necessitam de integrar as suas alterações com o restante código-fonte. Segundo Fitzgerald e Stol (Fitzgerald & Stol, 2015), a utilização da integração contínua permite que as tarefas de desenvolvimento de *software* sejam desenvolvidas de forma independente e paralela entre os elementos que fazem parte da equipa de desenvolvimento. Assim que uma dessas tarefas for concluída, um dos elementos da equipa apresentará o código ao sistema de CI para ser integrado ao restante projeto.

Quando o sistema de CI entra em ação, o seguinte conjunto de atividades serão realizadas: criação de um pacote, compilação do código, execução de testes unitários, validação do código e montagem do código. Todas essas atividades auxiliam o processo de entrega do *software*, no sentido em que ajuda a integrar o código, a executar testes automatizados e a fazer o *deploy* em qualquer ambiente. A figura 4 (Fitzgerald &Stol, 2015) ilustra as atividades de integração contínua.

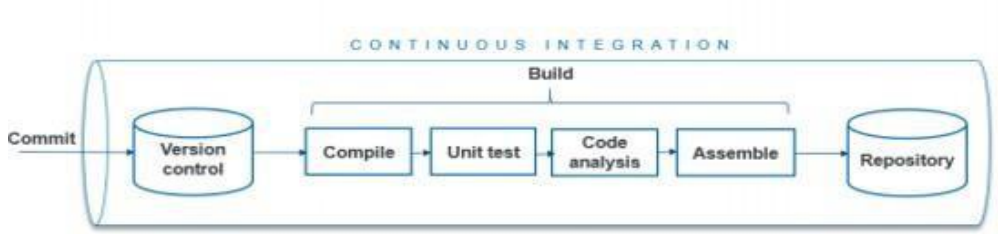


Figura 4 – Atividades de integração contínua

2.2.2 Entrega contínua

A prática de entrega contínua, tal como o seu nome indica, tem como principal objetivo a entrega contínua de valor ao cliente, sob a forma de versões estáveis e preparadas para entrar em produção a qualquer momento. Esta fase exige que o código seja entregue com mais frequência e maior qualidade possível e, para isso, é necessário utilizar ferramentas de gestão de código-fonte e ferramentas de gestão de controlo de versões, de forma a contribuir para as entregas de *software* com muito mais qualidade.

A entrega contínua (CD) tem por base a prática da integração contínua, mas desta vez associados à entrega do *software*. Esta prática tem como objetivo resolver o problema que as metodologias mais antigas tinham na fase da entrega do *software*, pois este processo era feito apenas no final do ciclo de desenvolvimento. A entrega contínua permite entregar o *software* em qualquer momento, o que significa que os utilizadores não precisam de esperar tanto tempo até que os novos recursos estejam disponíveis em produção. Para além do benefício da rapidez da entrega de *software*, a integração contínua também permite:

- Diminuir os problemas associados à entrega de *software*- É relativamente simples fazer implementações sem que o utilizador tenha a noção disso;
- Entregar *software* mais rapidamente - Automatizar o processo de implementação no ambiente de produção;
- Melhorar a qualidade do *software* - Facilidade de utilização de *pipelines* de entrega contínua, o que torna a execução de testes automáticos contínua, melhorando a qualidade dos produtos e serviços.
- Reduzir custo - Qualquer produto ou serviço de *software* bem-sucedido evoluirá significativamente ao longo do tempo. Desta forma, ao investir na automação, testes e ambientes, reduzimos substancialmente o custo de desenvolvimento das alterações do *software*.
- *Feedback* mais recorrente – É possível obter *feedback* dos utilizadores com base no *software* que se encontra em produção.

2.2.3 Monitorização contínua

O objetivo da prática de monitorização contínua, tal como o nome indica, é monitorizar continuamente a infraestrutura, de forma a criar um processo fácil e limpo de criação de relatórios. Para isto são necessárias ferramentas de monitorização e registos, permitindo às empresas obter uma visão geral do estado dos ambientes existentes. Com o conjunto de práticas anteriormente abordadas, é possível recolher de modo automático todo um conjunto de informação acerca dos diferentes processos, através da medição de um conjunto de métricas à partida definidas. Deste modo, é possível identificar problemas de modo mais rápido, preciso e simples, sobrando mais tempo para a definição e implementação de melhores soluções. Para além disso, esta visão contínua permite também definir e quantificar riscos com maior celeridade e precisão, possibilitando assim antecipar eventuais problemas.

2.2.4 Feedback contínuo do cliente

O *feedback* contínuo do cliente tem como base a recolha contínua de *feedback* por parte do utilizador final, como uma forma de procurar garantir que as suas necessidades sejam correspondidas da forma mais rápida e eficiente possível. Para se obter o *feedback* do utilizador final, é necessário que a equipa garanta a entrega continua dos componentes para que estes possam ser posteriormente validados (Soni, 2015).

Esta prática de DevOps permite que qualquer negócio se torne mais ágil e proativo no sentido em que responde continuamente às necessidades dos seus utilizadores. A resposta que é dada de forma contínua permite que as equipas se mantenham focadas em trabalhar nas questões que representam efetivamente interesse para o utilizador final, transformando o desenvolvimento em valor acrescentado para o *software*. O *feedback* que é recebido é um dos fatores importantes para uma outra prática da metodologia DevOps, o planeamento contínuo.

2.2.5 Planeamento contínuo

A prática de planeamento contínuo é essencial para garantir uma constante e rápida adaptação às alterações existentes no mercado (Virmani, 2015) ou às alterações originadas pelo *feedback* recebido por parte do cliente e equipa de operações. Esta prática está ligada às metodologias de desenvolvimento de *software*

ágil que defendem planos curtos e flexíveis que podem ser adaptados às mudanças repentinas. Através da utilização do planeamento contínuo, o plano de tarefas é atualizado sempre que é necessário efetuar uma alteração. Apesar da sua constante alteração, este plano por ser mais preciso, dado que é atualizado rapidamente à medida que os projetos evoluem e os dados que contém acabam por ser mais detalhados.

O DevOps pressupõe e possibilita a execução da prática de planeamento contínuo através da existência de um *product backlog*, priorizado conforme o *feedback* contínuo recolhido por parte dos utilizadores finais. O *feedback* contínuo permite adaptar constantemente a priorização de atividades do *backlog*, garantindo assim que as prioridades de desenvolvimento estão em linha com as necessidades dos seus utilizadores finais (Virmani, 2015).

2.3 Ferramentas típicas de DevOps

As ferramentas utilizadas para a aplicação da metodologia DevOps divergem de empresa para empresa. O ciclo de vida de DevOps é composto por 6 fases que envolvem a utilização de vários tipos de ferramentas que auxiliam na aplicação das suas práticas. Para cada uma das fases que se seguem, existe uma grande variedade de ferramentas que servem para o mesmo efeito, mas apenas foram selecionadas as ferramentas *open-source* que foram estudadas e testadas para o caso de estudo escolhido. A escolha para as ferramentas que serão referidas de seguida foi feita com base em dois fatores: preço e impacto no mercado de IT.

2.3.1 Planeamento

O planeamento da metodologia DevOps é muito semelhante ao planeamento realizado nas metodologias ágeis. O planeamento é muito importante na medida em que possibilita a divisão do trabalho das equipas em partes menores e a sua gestão torna-se mais fácil. Para facilitar a gestão do trabalho das equipas, é necessário utilizar ferramentas que permitam a sua colaboração e o planeamento de tarefas, tal como o *Jira software*. O *Jira software* (Figura 5) (Wikipedia, Jira (software), 2022), foi desenvolvido pela empresa *Atlassian* com o objetivo de permitir a monitorização de tarefas e acompanhamento de projetos, garantindo a gestão de todas as atividades e permitindo que toda a equipa esteja alinhada quanto ao planeamento das ações a serem executadas.

Figura 5 – Logotipo Jira software

2.3.2 Desenvolvimento

Durante o processo de desenvolvimento de *software*, é importante que o código seja entregue com mais frequência, rapidez e melhor qualidade. Para isso, são necessárias ferramentas de controlo de versões e revisão de código. Uma das ferramentas mais utilizadas nas fases de codificação, compilação e integração contínua de DevOps é o *Git*. O *Git* (Figura 6) (Wikipedia, Git, 2022), é um *software* de controlo de versões e gestão de código fonte e é utilizada para verificar mudanças em ficheiros de código, o que permite controlar o trabalho da equipa de desenvolvimento, visando aumentar a velocidade de desenvolvimento, suporte e integridade do *software*.

Figura 6 – Logotipo Git

Uma das ferramentas que também auxilia na fase de desenvolvimento é o *Docker*. O *Docker* (Figura 7) (Wikipedia, Docker, 2022) é uma ferramenta *open-source* projetada para criar e executar aplicações de uma forma bastante facilitada, através da utilização de *containers* e surgiu como alternativa à utilização de máquinas virtuais. Os *containers* permitem "empacotar" uma determinada aplicação, como bibliotecas e outras dependências adjacentes e, em seguida, executa essa mesma aplicação. Assim, é possível garantir a execução de uma determinada aplicação numa máquina hospedeira Linux, independentemente da máquina ter configurações personalizadas diferentes da máquina usada para escrever e testar o código. Esta ferramenta garante que o mesmo ambiente de desenvolvimento de *software* se mantenha em todas as etapas do ciclo de vida de DevOps, desde o planeamento até a fase de *feedback*. Para além do *Docker*, também foi estudada e testada a ferramenta *Kubernetes*, ferramenta destinada à gestão de *Docker containers*. A aplicação desta ferramenta ao trabalho desenvolvido acabou por não avançar porque o número de *containers* criados era reduzido.

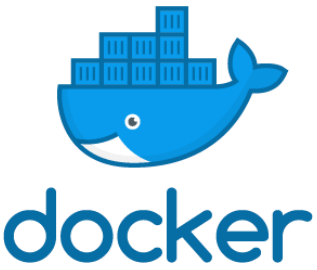


Figura 7 – Logotipo Docker

2.3.3 Testes

Para criar qualidade no *software* é necessário executar os testes automatizados e manuais durante o processo de entrega. Uma das ferramentas mais conhecidas de destinadas à criação de *scripts* de teste é o *Selenium*.

Selenium (Figura 8) (Wikipedia, Selenium, 2022) é uma ferramenta de teste automáticos *open-source* usada principalmente para validar aplicações *web* em vários *browsers*. Esta ferramenta permite utilizar várias linguagens de programação como *Java, C #, Python etc* para criar os seus *scripts* de teste. O componente mais importante da ferramenta *Selenium* é o *WebDriver*.

A ferramenta *Selenium WebDriver* é usada para criar testes *web* e verificar se eles funcionam conforme o esperado. Esta ferramenta oferece suporte a muitos tipos de browsers, como *Firefox, Chrome, IE* e *Safari*. No entanto, através da utilização do *Selenium WebDriver*, pode-se verificar que esta ferramenta é independente de plataforma, pois o mesmo código pode ser utilizado em diversos sistemas operativos.

Figura 8 – Logotipo Selenium

Após a execução do conjunto de testes desenvolvidos na ferramenta *Selenium*,é necessário armazenar os resultados obtidos e as evidências de teste. O *Confluence* (Figura 9) (Wikipedia, Confluence, 2022) funciona como um *wiki*, tornando as tarefas de criar e editar conteúdo muito mais facilitadas. Esta ferramenta funciona como um espaço de trabalho onde as equipas estruturam, organizam e partilham o trabalho, para que cada membro da equipa tenha acesso às informações de que precisa num único espaço.

Existe muito trabalho relacionado com os projetos de *software*, desde requisitos do produto, planos de projeto, documentação técnica, notas de reuniões, *feedback* dos clientes, evidências de testes, entre outros. Assim sendo, esta ferramenta ajuda a equipa a planear e controlar o trabalho que gira em torno do produto, assim como o organizar todo o conteúdo adicional que é criado ao longo desse caminho através das *Confluence Pages*, eliminando a necessidade de armazenar documentos em vários locais, como drives compartilhadas ou documentos Word.

Figura 9 – Logotipo Confluence

2.3.4 Integração e entrega contínuas

A integração contínua e entrega contínua garantem a redução de custos, tempo e riscos da entrega de *software*. Quando estas duas práticas são automatizadas, passa a ser possível lançar uma nova versão do *software* com o mínimo de intervenção manual. Dessa forma, são necessárias ferramentas que agilizem os processos que estas duas práticas enfrentam até que o *software* seja entregue ao cliente final. Uma das ferramentas mais conhecidas para a integração e entrega contínuas é o *Jenkins*.

O *Jenkins* (Figura 10) (Wikipedia, Jenkins, 2022) é um servidor de automação *open-source* que auxilia na automatização de processos de desenvolvimento de *software*, tais como integração contínua, entrega contínua e testes. Esta ferramenta insere-se na fase de integração contínua e entrega contínua do ciclo de vida de DevOps e facilita as equipas a monitorizar certas tarefas que se tornam repetitivas e é bastante fácil de se integrar com outras ferramentas, através dos *plugins* que fornece.

Figura 10 – Logotipo Jenkins

2.3.5 Monitorização

A monitorização é uma prática útil para detetar e evitar eventuais incidentes no ambiente de produção. Uma das ferramentas mais usadas para a monitorização do *software* é o *Dynatrace*. O *Dynatrace* (Figura 11) (Wikipedia, Dynatrace, 2022) recorre à inteligência artificial para monitorizar continuamente a infraestrutura e apresenta os resultados numa *dashboard* em tempo real. O *Dynatrace* permite encontrar mais rapidamente respostas aos problemas existentes no *software* e torna as equipas mais produtivas através da gestão de incidentes.

Figura 11 – Logotipo Dynatrace

2.3.6 Feedback

É possível levantar o *feedback* do cliente através de ferramentas como *Slack*, via *e-mail* ou formulários. Estas ferramentas são bastante usadas pelas empresas da área de TI e através delas é possível efetuar as trocas de *feedback* de forma instantànea. Apesar disso, existem alguns problemas no que toca à reunião das informações levantadas e é necessário uma ferramenta que faça a gestão das trocas rápidas de *feedback*. Desta forma, surgiu no mercado a ferramenta *Jira Service Management* (Figura 12) (Wikipedia, Jira Service Management, 2022) que permite receber, gerir e resolver as solicitações recebidas por parte dos clientes. Esta ferramenta armazena todo o *feedback* num único local e mantém a equipa no caminho certo para resolver os problemas encontrados. Para além disso, como também é um serviço desenvolvido pela *Atlassian,* a integração ao *Jira software* pode ser feita de forma bastante simplificada.

Figura 12 – Logotipo Jira Service Management

2.4 Benefícios de DevOps

A adoção de DevOps resulta numa grande variedade de benefícios, que resultam da boa aplicação das práticas descritas anteriormente. Os benefícios podem ser categorizados em benefícios técnicos, culturais e comerciais (Kim et al., 2016). Os benefícios técnicos estão relacionados com as práticas de CI/CD e com a velocidade na resolução de problemas. Os benefícios culturais tornam as equipas mais motivadas a contribuir para os objetivos da empresa, uma vez que ajudam a melhorar a comunicação entre elas. Por fim, os benefícios comerciais estão relacionados com o cliente final e com a velocidade de entrega de valor, garantindo maior estabilidade do negócio e maior espaço para inovação.

Os seguintes subcapítulos abordam os principais benefícios de DevOps e descrevem como é que os benefícios têm impacto positivo nos fluxos de trabalho e nos negócios das organizações.

2.4.1 Entrega de software

Um dos benefícios da implementação de DevOps é a diminuição do tempo de entrega do produto (Callanan & Spillane, 2016) (Elberzhager, Naab, Arif, & Süß, 2017) (Fazal-Baqaie, Güldali, & Oberthür, 2017). A metodologia DevOps permite que o desenvolvimento seja dividido em várias fases o que possibilitou as equipas a entregar o produto em pequenos lotes ajudando na minimização de riscos, dado que a deteção de erros é antecipada.

A automatização de processos está diretamente relacionada com a rapidez da entrega de *software*, uma vez que a equipa de operações possui uma gestão de ambientes simplificada. Desta forma, a publicação das novas alterações de *software* nos vários ambientes existentes torna-se mais fácil, permitindo assim que as novas alterações sejam visíveis mais rapidamente.

A entrega mais rápida do *software* é considerada uma mais valia significativa para o negócio, principalmente em setores de elevada concorrência onde a exigência, a inovação e a rapidez de resposta às tendências do mercado é elevada. Este fator aliado à redução do risco de falhas em ambiente de produção representa uma grande vantagem competitiva em relação a outras empresas que não adotem este tipo de metodologia.

2.4.2 Comunicação entre as equipas

Através das alterações do ponto de vista cultural a que o DevOps se propõe, a equipa de desenvolvimento e a equipa de operações torna-se uma só e começa a existir melhor comunicação e cumplicidade, quebrando a barreira existente nos processos antigos. É não só necessário que a equipa de operações seja envolvida desde o início do ciclo de desenvolvimento para que haja uma maior visibilidade do projeto e uma maior previsibilidade de potenciais problemas, como também que exista a partilha de capacidades e experiências entre estas duas equipas.

Em resultado de uma maior comunicação e colaboração, não só entre estas duas equipas, como também com o cliente final, através da prática de *feedback* contínuo, é possível encontrar uma maior coordenação interna e as equipas mais focadas nas necessidades do utilizador final.

2.4.3 Qualidade do software

Segundo Callanan e Spillane (Callanan & Spillane, 2016), a melhoria da qualidade do *software* está diretamente ligada aos testes de *software*, desde testes unitários até à monitorização no ambiente de produção, a fim de garantir que todos os requisitos são atendidos corretamente. A automação é também um fator importante na qualidade do produto, já que ela garante que nenhum passo é deixado para trás, garantindo assim que todos os requisitos sejam cumpridos.

A implementação de DevOps resulta em certas mudanças culturais e práticas que são reconhecidas como um fator importante na qualidade da solução entregue (Elberzhager, Naab, Arif, & Süß, 2017). As mudanças culturais passam por direcionar os programadores a atender aos pedidos dos clientes, enquanto a equipa de operações tem como função gerir e melhorar a acessibilidade e estabilidade do que é produzido pela equipa de desenvolvimento, visando manter uma comunicação saudável entre estas duas partes. A facilidade de comunicação entre a equipa de desenvolvimento e a equipa de operações resulta na melhoria da moral do trabalho em equipa, especialmente porque permite às equipas lançar *software* com mais qualidade. A utilização de práticas de integração contínua e entrega contínua contribuem para o aumento do nível de maturidade da empresa no sentido em que permitem detetar e corrigir possíveis erros de forma antecipada, reduzindo assim o tempo necessário para aperfeiçoar a qualidade do *software*. Desta forma, o aumento da eficiência das equipas e a automação de processos representam, mais uma vez, uma vantagem competitiva que o DevOps permite alcançar.

2.5 Relação com a metodologia ágil

Petersen (Petersen, 2011) afirma que a metodologia ágil tem como objetivo entregar continuamente *software* funcional e que este possa ser consultado pelo cliente, de forma que este averigue se a solução entregue está de acordo com o que foi pedido. Atravésdo *feedback* contínuo por parte do cliente, é possível garantir um maior alinhamento do desenvolvimento em relação às necessidades do cliente, o que resulta num *software* menos propenso a erros e problemas no momento da entrega final, já que este foi sendo sucessivamente aprovado ao longo das diferentes iterações.

Os benefícios do desenvolvimento ágil estão principalmente relacionados com a capacidade de responder aos requisitos em constante mudança. As atualizações de *software* são feitas em lotes menores, tornando a deteção de possíveis problemas mais rápida, reduzindo os custos que possam estar relacionados. No desenvolvimento ágil de *software*, os programadores podem desenvolver alterações imediatamente a partir do *feedback* do cliente e inserir o código do *software* no ambiente de produção maisrapidamente e com mais frequência. Através da aceleração da entrega de

software, as equipas de operações sentem-se impotentes porque deixam de conseguir acompanhar a velocidade da entrega. Desta forma, o ciclo de espera dos componentes desenvolvidos para entrar em produção torna-se maior, representando assim um claro desperdício, devido ao facto destes componentes não terem impacto no negócio imediatamente após a sua criação.

Em suma, a metodologia ágil foca-se na fase de desenvolvimento de *software*, deixando de lado a entrada em produção, que tipicamente é responsabilidade da equipa de operações. É neste momento que surge o upgrade ao ágil: a metodologia DevOps, que automatiza a entrega de *software* e, ao mesmo tempo, fornece uma melhor colaboração entre estas duas entidades organizacionais, enfatizando a colaboração, comunicação e integração entre ela (Condo, Mines, Seguin, Homan, & Neuburg, 2017).

2.6 Adoção da metodologia DevOps e seus desafios

A implementação da metodologia DevOps numa empresa não é uma tarefa fácil. Nas próximas duas subsecções serão abordadas a metodologia de adoção de DevOps e os obstáculos à implementação de DevOps.

2.6.1 Adoção da metodologia DevOps

No modelo tradicional cascata, cada fase do processo desde o levantamento de requisitos até à fase de manutenção do *software*, cada fase apenas é iniciada quando a fase anterior estiver totalmente finalizada (Sarker, Faruque, Hossen, & Rahman, 2015). É exatamente neste ponto que a metodologia ágil começa a divergir do modelo tradicional cascata, pois realiza inúmeros ciclos até o desenvolvimento estar completo e onde a entrega éfeita por fases, dado que os requisitos do projeto e os processos e o *feedback* do cliente são obtidos durante o desenvolvimento em vez de na primeira fase do projeto (Boehm & Turner, 2005).

Para além das dificuldades acima descritas, existe uma mudança não só de trabalho como também à forma como o processo de desenvolvimento deverá ocorrer. Surgido a partir da necessidade de otimizar a entrega de serviços e *software* com um nível de eficiência e agilidade elevados, o movimento Devops enfatiza a integração, comunicação e colaboração entre equipas. O DevOps é uma solução que fornece um conjunto de características e práticas que deverão ser aplicadas a uma empresa, mas é imprescindível que a forma de trabalhar dentro da empresa seja reestruturada (Sharma, 2017) (Erich,Amrit, & Daneva, 2014).

No relatório *State of DevOps* de 2018 (DORA, 2018) é feita uma distinção das etapas ao longo da adoção de DevOps nas empresas. Segundo a figura que se segue, na primeira etapa as empresas começam a construir as práticas fundamentais que serão adotadas e melhoradas durante as etapas seguintes. Nesta etapa, as equipas começam a unir forças e a aprender a utilizar as novas ferramentas de desenvolvimento ágil, integrando o controlo de versões. Na segunda etapa, as equipas de desenvolvimento e operações começam a operar com as ferramentas definidas de forma a acelerar a velocidade de entrega e reduzir erros de implementação. Na terceira etapa, as organizações começam a enfrentar alguns problemas, devido ao facto que as equipas de operações não conseguem lidar com as implementações contínuas da equipa de desenvolvimento. Para resolver este problema, é necessária uma mudança a nível cultural, onde é formada uma cultura de confiança entre estas duas equipas, criando liberdade suficiente para trabalharem sem aprovações manuais. Na fase 4, a empresa está apta para fazer entradas automatizadas, o que faz com que o trabalho seja entregue de forma mais rápida, segura e sem erros.

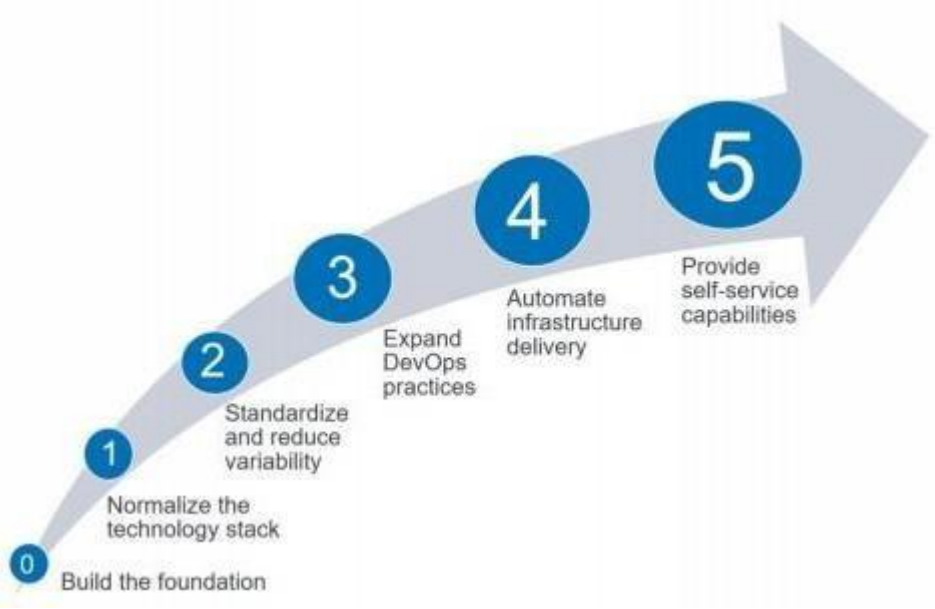


Figura 13 – Etapas da adoção de metodologias DevOps

2.6.2 Desafios à implementação de DevOps

Qualquer que seja a metodologia que se pretende implementar num determinado projeto de *software*, existem desafios inerentes e a metodologia DevOps não é exceção. Apesar da metodologia DevOps servir como resolução dos problemas existentes em vários níveis, é possível observar que existem mudanças que podem implicar alterações radicais na estrutura e processos de uma empresa.

O primeiro obstáculo à implementação de DevOps está relacionado com os problemas organizacionais. A falta de orientação que as equipas tinham ao longo da adoção de DevOps foi considerado um desafio, porque as equipas não possuíam qualquer plano estratégico que as pudesse auxiliar e orientar durante a adoção desta metodologia. Numa segunda fase, a adoção da metodologia DevOps veio trazer outros desafios para algumas equipas devido ao facto de estarem organizadas em silos e, como já é sabido, o DevOps valoriza o trabalho conjunto entre equipas.

O segundo desafio da implementação da metodologia DevOps consiste na falta de suporte. A falta de suporte proveniente dos níveis superiores da empresa, traz graves problemas no que toca ao apoio das equipas de desenvolvimento. A construção do compromisso e confiança pode trazer importantes benefícios para as empresas, portanto, o nível de gestão deve estar constantemente alinhado com a equipa de desenvolvimento durante todas as operações do dia-a-dia.

O terceiro desafio da implementação de DevOps é a adaptação às novas ferramentas. A falta de tempo para implementar a metodologia DevOps é considerada também um desafio, devido ao facto de geralmente a gestão das empresas não estar disposta a apoiar as equipas de desenvolvimento com tempo e recursos necessários para aprenderem novas ferramentas, bem como automatizar os processos de entrega de *software*. Não existe maneira de implementar uma metodologia de forma rápida, pelo que, a melhor forma de realizar uma mudança organizacional é começar a alocar tempo e recursos para fazer a transição para esta metodologia. Desta forma, os programadores têm mais tempo para se habituarem às novas ferramentas e aos novos métodos de trabalho.

Por fim, o quarto e último desafio da implementação de DevOps é a adaptação às novas mudanças. A tentativa de procurar por um melhor método de desenvolvimento de *software* pode tornar-se desconfortável, pois exige que as equipas saiam da sua zona de conforto e mudem a sua cultura organizacional e a sua mentalidade, implicando a resistência à mudança.

3 Caso de estudo

De forma a poder aferir experimentalmente os ganhos de produtividade da aplicação da metodologia DevOps, é necessário um caso de estudo. De forma a fazer a seleção deste caso de estudo é necessário analisar de forma pormenorizada as métricas que precisam de ser analisadas e comparadas. Os objetivos do caso de estudo neste trabalho são observar e medir as diferenças em diversos aspetos de qualidade e desempenho através da aplicação da metodologia DevOps a um projeto anteriormente realizado pela empresa Noesis.

3.1 Especificação das métricas

Com base nas características, práticas e benefícios de DevOps acima definidos, é necessário medir a qualidade das alterações impostas no projeto que será desenvolvido. As métricas de DevOps são dados usados para analisar o resultado da implementação da metodologia DevOps. As métricas permitem que as equipas de DevOps façam as medidas necessárias e avaliem os resultados obtidos.

As métricas que serão avaliadas no caso de estudo que será selecionado são: tempo e frequência das entregas de *software*, tempo médio de deteção de falhas no ambiente de produção, tempo médio de reparação de um problema em produção, *feedback* do cliente, tempo que um problema reportado pelo cliente demora a ser considerado e, por fim, a qualidade do *software*. De seguida será feita a descrição detalhada de cada uma destas métricas definidas.

3.1.1 Tempo e frequência das entregas em produção

Entregar atualizações, novos recursos e melhorias no *software* que se está a desenvolver com maior frequência é crucial para criar e manter a vantagem competitiva. A capacidade de melhorar a frequência de implementação leva a uma maior agilidade e maior resposta às necessidades dos clientes. Medir a frequência da entrega pode ajudar a obter uma melhor visão sobre quais foram as mudanças mais benéficas. A identificação de uma diminuição repentina nesta frequência pode resultar no desequilíbrio do fluxo de trabalho, o que poderá atrasar as entregas ao cliente, portanto, o ideal será manter ou aumentar gradualmente esta frequência de forma a não perder o ritmo de desenvolvimento e entrega.

O ambiente desenvolvimento, tal como o seu nome indica, pertence à equipa de desenvolvimento que trabalha diariamente para acrescentar valor ao *software*, através da criação de novas funcionalidades. Sempre que existem novas alterações no código, elas passam para o ambiente teste para serem testadas, seja manual ou automatizadamente. Depois de testadas, o processo ideal é passar o *software* para o ambiente de produção, e é neste momento que o *software* fica disponível para o cliente. Caso existam algumas correções para efetuar, o *software* é novamente submetido para alterações, ou seja, passa novamente para o ambiente de desenvolvimento. Ao efetuar a entrega do *software* para um determinado ambiente de forma mais frequente, o risco de existirem erros em ambiente de produção diminui porque a sua identificação é antecipada. A entrega de *software* mais frequente significa que as alterações entre as implementações são em menor quantidade. Uma vez identificados, os erros podem ser corrigidos mais rapidamente e isso torna a revisão total de *software* desnecessária.

3.1.2 Tempo médio de deteção de falhas em produção

O tempo médio de deteção de uma falha em produção ou MTTD (Pure Storage, 2022) é uma métrica bastante útil para fornecer indicações acerca da estabilidade do sistema. Uma maneira de detetar defeitos no ambiente de produção passa por automatizar as práticas de testes,que muitas vezes deixam que os defeitos entrem em produção, prolongando os ciclos de desenvolvimento. Por mais que frequência de entrega de *software* para produção seja elevada, é importante ter em consideração as falhas que podem ocorrer.

O tempo médio de deteção de falhas em produção poderá ser calculado tendo em conta os seguintes fatores: total de horas do *software* em funcionamento contínuo e número total de problemas. Esta métrica é muito importante para as equipas de DevOps que desejam monitorizar a eficácia das suas ferramentas e processos de gestão de incidentes. O ideal é que esta métrica tenha o valor o mais baixo possível, o que significa que as suas ferramentas e processos funcionam corretamente. Assim que é detetada uma falha em produção, passa-se para a sua resolução. Com isto, passamos para a próxima métrica, e não menos importante, tempo médio de reparação de um problema em produção.

3.1.3 Tempo médio de reparação de um problema

O tempo médio de reparação de um problema ou MTTR (DATTA, 2022) em produção começa assim queuma falha é detetada e abrange o tempo de diagnóstico, tempo de reparação, testes e todas as outras atividades até que o serviço volte ao seu normal funcionamento. Reparar um problema pode ser algo complicado, principalmente se for um problema complexo que comprometa a execução de um determinado *software* em ambiente deprodução. Em maior parte dos casos, o que se faz é efetuar *rollback* à última versão estável de um programa e voltá-la a colocar em produção. A forma ideal de corrigir um problema em ambiente de produção seria resolvê-lo quase imediatamente após a sua deteção, mas isso é muito difícil porque não é vantajoso ter o ambiente de produção em baixo.

Para melhorar o tempo de reparação de um problema é necessário utilizar uma ferramenta que permita efetuar a cobertura de código. Os testes de cobertura de código mostram qual foi o código que foi executado e qual a percentagem de testes concluídos com sucesso, reduzindo assim o tempo necessário para devolver o *software* ao ambiente de produção. Este tipo de testes ajudam os programadores a determinar mais rapidamente quais são as partes do código que precisam de mais atenção, permitindo assim uma diminuição do tempo de reparação do problema, a colocação da versão corrigida em produção e, consequentemente, a proteção da satisfação do cliente.

3.1.4 Nível de satisfação do cliente

O nível de satisfação do cliente é um aspeto muito importante na implementação de qualquer *software* porque é uma forma de obter informações acerca da sua opinião sobre o trabalho que está a ser desenvolvido. As informações obtidas através do *feedback* são utilizadas para planear e implementar novas melhorias no *software*, com o objetivo de manter a confiança e satisfação por parte do cliente. O atendimento é um dos fatores mais importantes, juntamente com o serviço que está a ser desenvolvido para o cliente, pois se a empresa fornecer um excelente serviço e possuir um péssimo atendimento, irá receber um *feedback* negativo.

O objetivo de obter *feedback* dos clientes é fornecer informação não só para obter a visão geral do trabalho que tem vindo a ser realizado até ao momento, como também ter uma ideia se empresa que está a prestar o serviço está ou não num bom caminho. Existem muitas abordagens e metodologias para obter resultados a nível desta métrica, tais como entrevistas, email, mensagens, formulários, etc.

O levantamento do *feedback* do cliente permite avaliar a sua satisfação, sejam críticas, sugestões ou elogios e irá ter um impacto muito importante no desenvolvimeto de *software*, porque a equipa de desenvolvimento irá trabalhar de acordo com o *feedback* recebido.

3.1.5 Tempo médio de consideração de um problema

Consoante a carga de trabalhos da equipa de desenvolvimento, esta pode não ter tempo de resposta imediata para corrigir os erros detetados pelo cliente. Desta forma, será importante identificar o tempo entre o instante em que os erros são reportados pelo cliente e o instante em que eles começam a ser corrigidos.

Uma forma de medir o tempo que um problema demora a ser considerado é criar um item no *product backlog* quando esse problema é reportado por parte do cliente. Assim, a cada item será atribuída a sua data de criação e quem será o responsável pela sua resolução. No fim do problema se encontrar resolvido, o responsável irá definir esse item como concluído e irá atribuir-lhe o tempo que demorou a corrigi-lo. O tempo que esse problema demorou a ser considerado e corrigido é dado pelo dia de início, e dia de término desse item do *backlog*.

3.1.6 Qualidade do software

A *International Organization Standardization* (ISO) define qualidade de *software* como a totalidade de características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades explicitas e implícitas (Devmedia, 2022). O principal objetivo de quem desenvolve o *software* é atender todos os requisitos obtidos pelo *feedback* dado pelo cliente e entregar o *software* com o número mínimo de problemas possíveis.

Um dos métodos mais utilizados pelas empresas de TI para medir a qualidade do seu *software* são os testes de *software*. A etapa de desenvolvimento e execução de testes de *software* é uma etapa muitíssimo importante para o controlo de qualidade do *software*, porque serve para assegurar que o *software* está a seguir todas as funcionalidades esperadas e que estas estão a funcionar corretamente. Desta forma, é importante criar cenários de teste que cubram toda a estrutura do *software*, para não ficar nenhum problema por identificar, resultando na melhoria da qualidade do *software*.

3.2 Seleção do projeto

O projeto ideal para a aplicação de DevOps deverá ser selecionado com critério pois a aplicação da metodologia DevOps que será previamente projetada deverá ser algo observável. Assim, é importante que o projeto escolhido tenha o seu prazo de entrega definido, porque a metodologia DevOps, segundo as suas práticas, foca-se em desenvolver um projeto com um orçamento e prazo fixo. Outro requisito e não menos importante é a complexidade do projeto. Num projeto com algum nível de complexidade, as práticas de integração contínua e entrega contínua tornam-se quase como de "uso obrigatório". Quando se desenvolve um projeto complexo, é difícil prever o que se pode passar duas ou três semanas após o início do seu desenvolvimento. Dado a sua complexidade, pode ser necessário estimar novamente tarefas que já deveriam ter sido desenvolvidas nesse espaço de tempo e, por algum motivo não foi possível desenvolver, apesar de, obviamente, a equipa de desenvolvimento fazer os possíveis para concluir todas as tarefas propostas para a sprint que decorre naquele momento.

A complexidade dos projetos está diretamente relacionada com a constante mudança dos requisitos. Atualmente, devido ao avanço da tecnologia, é cada vez mais normal que os projetos sofram constantes mudanças durante o seu desenvolvimento. A metodologia DevOps suporta um processo no qual os requisitos devem mudar e evoluir a qualquer momento. Assim, num projeto com maior complexidade, onde o tempo de entrega será maior e as alterações nos requisitos são constantes, o DevOps será a melhor abordagem a seguir. É importante que seja possível aplicar a metodologia DevOps a um projeto em que também seja possível aplicar a metodologia ágil. O DevOps baseia-se na metodologia ágil e grande parte das práticas são comuns em ambas as metodologias. Inicialmente foram propostos três projetos, desenvolvidos por equipas de desenvolvimento da empresa Noesis, para servirem à aplicação de práticas associadas à metodologia DevOps, são eles: *software* de gestão de material da empresa, designado por MAT, *software* de simulação de alocações e custos, designado por ALOC, e o *software* de automação de testes, designado por NTX.

O projeto MAT e ALOC foram desenvolvidos exclusivamente para uso interno da Noesis. Estes projetos *software* de pequena dimensão são compostos por equipas de dois elementos e seguem a metodologia de desenvolvimento em cascata, também conhecida por abordagem *top-down*. Neste tipo de metodologia, o modelo de direção segue um método de sequência que inclui as seguintes fases: definição dos requisitos, projeto de sistemas e de *software*, implementação e testes, integração, operação e manutenção. Neste modelo, somente no final do projeto é que será analisada uma possível mudança nos processos, pois cada fase só é iniciada após a conclusão da sua antecessora. Desta forma, é essencial existir uma especificação rigorosa dos requisitos no início do projeto porque este modelo não

promove a realização de mudanças necessárias não identificadas nas fases iniciais, ficando as mesmas em standby. Ambas as aplicações foram desenvolvidas em Java e existe um repositório de código para cada uma delas.

O projeto NTX procura responder à necessidade dos clientes na execução de testes de regressão, a qualquer momento, e de forma automática. Este projeto trata-se do desenvolvimento de uma ferramenta que funciona como plugin da ferramenta *Jira software* e que permite fazer testes mobile, web e aplicações *desktop* de uma forma bastante simplificada. A equipa de desenvolvimento deste projeto é composta por cinco elementos que partilham um repositório de código para que todas as versões do *software* sejam guardadas. Neste projeto foi utilizada a metodologia *Scrum*, baseada na metodologia ágil. Este tipo de metodologia defende que se deve procurar satisfazer as necessidades dos clientes através práticas contínuas, mantendo sempre a comunicação constante e mantendo o foco na comunicação entre todos os membros das equipas envolvidas. Ao contrário da metodologia em cascata, as alterações dos requisitos devem ser aceites a qualquer momento, mesmo que seja numa fase tardia do desenvolvimento. Os processos da metodologia ágil não só promovem um desenvolvimento sustentável aumentando a produtividade e a velocidade de entrega, como também permitem ter momentos de retrospetiva, onde as equipas poderão fazer os ajustes necessários e tornar o desenvolvimento mais eficaz.

Quaisquer uns projetos mencionados anteriormente podem servir de base à aplicação das metodologias DevOps, mas como esta tarefa inclui o gasto de tempo e outro tipo de recursos, é necessário saber qual é o projeto que permite aferir os benefícios da implementação de DevOps e o resultado das métricas que se pretendem avaliar. Foram considerados três fatores para efetuar a seleção do caso de estudo: risco, valor e dificuldade.

A aplicação de metodologias DevOps no projeto MAT e ALOC não envolve grande risco porque são projetos de pequena escala e o *software* não altera ao longo do tempo. Desta forma, não seria necessário alterar repentinamente a forma de trabalhar das equipas e o receio de dar o passo em frente relativamente a esta metodologia seria menor. Para além disso, os projetos seguem a metodologia de desenvolvimento de *software* em cascata o que iria trazer mais algumas dificuldades às equipas envolventes no que toca à alteração das suas práticas de trabalho, devido à diferença entre as duas metodologias de desenvolvimento. Devido ao facto destas duas aplicações serem desenvolvidas para uso interno, não existe atualização constante do *software*, portanto a implementação das metodologias DevOps nestes dois projetos não iria trazer benefícios a nível da velocidade da entrega do *software*, mas sim a nível de estabilidade e consistência.

O projeto NTX é desenvolvido através de metodologias ágeis principalmente devido à velocidade da entrega e às atualizações frequentes que são feitas. Desta forma, a implementação da cultura DevOps seria mais facilitada pois existe semelhança quanto às práticas de desenvolvimento de *software*, mas , por outro lado, o cliente poderia ficar comprometido se algumprocesso da implementação corresse mal. Desta forma, não haveria grande dificuldade na transição da metodologia ágil para a metodologia DevOps, mas o risco de implementação seria maior em relação aos projetos anteriormente referidos. A adição de valor ao projeto NTX, a sua entrega ao cliente e a sua qualidade podem ser ainda mais rápidas e observáveis através da utilização da metodologia DevOps. Para auxiliar na seleção do projeto, foi desenvolvida a seguinte tabela que apresenta a comparação entre os três projetos e quais são asmetodologiaspraticadas pelas suas equipas.

Detalhes / Projeto	MAT	ALOC	NTX
Metodologia de desenvolvimento	Cascata	Cascata	Ágil
Tamanho da equipa	2	2	5
Complexidade	Simples	Simples	Complexo
Repositório central	Sim	Sim	Sim
Integração contínua	Não	Não	Sim
Entrega contínua	Não	Não	Sim
Monitorização	Não	Não	Não
***Feedback* contínuo**	Não	Não	Não
Testes de *software*	Não	Não	Unitários

Tabela 3 – Detalhes dos projetos existentes

Os clientes do NTX estão envolvidos em diferentes setores do mercado global e possuem a necessidade de ter um *software* de testes para garantir a qualidade do seu *software*. Deste modo, a aplicação NTX não pode ter grandes intervalos de manutenção porque os seus clientes dependem diretamente dela para vender os seus produtos. Assim, a análise do risco, valor e dificuldade de cada um dos projetos existentes, pode-se verificar que o projeto NTX requer maior risco para aplicação dasmetodologias DevOps em relação aos projetos MAT e ALOC. Apesar do risco de implementação ser maior, a dificuldade de implementação será menor, porque as metodologias ágil e DevOps são semelhantes em algumas das suas práticas e o valor acrescentado ao *software* será bastante maior, no sentido em que os projetos MAT e ALOC são projetos mais pequenos onde a tarefa de implementação de novos requisitos será reduzida.

Desta forma, pode-se concluir que o projeto NTX tem mais vantagens na aplicação da metodologia de DevOps em relação aos outros dois projetos. Assim, o projeto selecionado para efetuar a implementação de DevOps e verificar os seus benefícios e o resultado das métricas, será o projeto NTX. O facto de a entrega contínua já ser colocada em prática no projeto NTX em nada influencia a implementação de DevOps neste projeto, porque, certamente muitos outros aspetos relacionados com esta prática serãomelhorados e a entrega ao cliente final será mais rápida e com menos problemas. De seguida será abordada a forma de operacionalização das métricas especificadas no capítulo 3.1

3.3 Ferramentas utilizadas no caso de estudo

O DevOps envolve a utilização de várias ferramentas de vários tipos, para vários propósitos. Neste caso de estudo foi necessário selecionar ferramentas para acompanhar as fases de planeamento e acompanhamento do projeto, gestão de código, integração e entrega contínuas, testes, documentação e gestão de *containers*. Nospróximos subcapítulos são referidas as ferramentas utilizadas.

3.3.1 Planeamento e acompanhamento do projeto

Para planear e acompanhar o projeto e as tarefas das equipas envolvidas, utilizou-se a ferramenta *Jira software*, referenciada no subcapitulo 2.4.1, porque é uma ferramenta que permite obter:

- Maior transparência - mais visibilidade sobre os requisitos;
- Mais priorização - mais facilidade na deteção e priorização dos requisitos, consoante a necessidade do cliente;

- Menos complexidade - através da divisão dos requisitos em pequenos componentes, facilitando a gestão e o desenvolvimento.
- Mais rastreabilidade - mais facilidade na visibilidade dos requisitos, de forma a verificar se eles foram ou não atendidos, aumentando assim o desenvolvimento.

Durante a fase Pre-DevOps, a ferramenta *Jira Software* era utilizada apenas para servir a aplicação NTX, pelo que a funcionalidade que esta ferramenta fornece para planeamento e acompanhamento do projeto não era utilizada. Neste sentido e após se dar início à migração do planeamento e à utilização por parte das equipas envolventes da plataforma *Jira Software*, foi possível centralizar o planeamento com a ferramenta NTX num só local.

O *Jira software* foi utilizado no projeto NTX principalmente para:

- Escolher tarefas para fazer parte da próxima *sprint;*
- Criar tarefas e sub-tarefas;
- Organizar tarefas por estado (ex: ativo, em progresso, concluído);
- Atribuir ficheiros e imagens a tarefas posteriormente criadas;
- Gerar *burndown chart* com progresso da sprint;

3.3.2 Gestão de código

Durante esta fase, é importante que o código seja entregue com mais frequência, rapidez e melhor qualidade. Para isso, são necessárias ferramentas de controlo de versões e ferramentas de revisão de código. Para esta fase, a ferramenta selecionada foi o Bitbucket.

O Bitbucket é uma ferramenta da empresa Atlassian destinada à gestão de repositório *Git* projetada para fornecer um local centralizado para colaborar com o código-fonte de uma aplicação e guiá-la através do fluxo de desenvolvimento. Esta ferramenta utiliza um sistema de controlo de versões que permite desenvolver projetos onde vários elementos da equipa de desenvolvimento contribuem ao mesmo tempo onde podem criar e editar arquivos sem permitir que outras pessoas escrevam por cima das suas alterações.

O Bitbucket é uma ferramenta muito útil para gerir o controlo de versões das alterações que foram realizadas à plataforma NTX e aos testes que foram realizados. Assim, foi necessário criar um repositório que armazena todo o código-fonte da aplicação. Desta forma, a utilização desta ferramentatrouxe benefícios a vários níveis como minimização de erros e facilidade na gestão do repositório existente.

3.3.3 Integração e entrega contínuas

O objetivo desta fase é ter um processo de melhoria contínua. Um processo em que os ambientes são configurados sem intervenção humana e o código é entregue automaticamente seria o ideal. Para isto, são necessárias ferramentas para integração e entregas contínuas. Para as práticas de integração contínua e entrega contínua, a ferramenta selecionada foi o *Jenkins*. Tal como já foi abordado no subcapítulo 2.4.4, o *Jenkins* é uma ferramenta que podeacelerar o processo de desenvolvimento de *software* através da automação, permitindo controlar os processos do ciclo de vida de desenvolvimento. A utilizaçãodesta ferramenta permitiu:

- Criar ambientes de desenvolvimento – Criação do ambiente de desenvolvimento, testes e produção.
- Obter resultado dos testes – Executar os testes no ambiente de testes.
- Criar pipeline - Criar a pipeline para a entrega do *software* nos diversos ambientes.
- Histórico de artefactos – Guardar versões antigas do *software*

3.3.4 Testes

Durante a fase de testes, o objetivo é ter um processo automatizado, reduzindo o erro humano e otimizar a entrega. Sempre que um conjunto de requisitos está pronto para evoluir, o sistema executa a análise de código e um conjunto de testes para garantir a consistência e o desempenho do *software*. Os benefícios pretendidos quando se utilizam ferramentas de testes de *software* são as seguintes:

- Redução de custos – Inicialmente, o investimento pode ser maior, mas, após a implementação, o custo dos testes automatizados torna-se menor comparando-se ao teste manual devido à otimização dos seus procedimentos.
- Eficiência nas operações - Ao analisar a eficiência dos testes manuais, é percetível que por vezes estes podem levar tempo para serem executados. Neste caso, o período de execução para o teste automatizado é muito menor e os resultados podem ser averiguados rapidamente trazendo agilidade para as operações de qualidade.
- Aumento da produtividade - Como o teste automatizado é mais eficiente, as respostas são dadas mais rapidamente para os programadores corrigirem os erros. Isto aumenta a produtividade da equipa devido ao feedback que os testes proporcionam.

- Segurança – Os testes automatizados podem ser criados para validações não funcionais como segurança e desempenho. Deste modo, a automação realiza verificações constantes de forma a reduzir o risco de possíveis ataques.

Para os testes funcionais, a ferramenta selecionada foi o *Selenium* (subcapítulo 2.4.3). Através da criaçãode scripts de teste com esta ferramenta, foi possível:

- Criar testes – Criar *scripts* de teste para a ferramenta NTX;
- Testes automatizados – Integrar os testes com a ferramenta *Jenkins;*
- Report de testes – Obter *template* de *report* de uma execução;
- Identificar objetos web– Identificar objetos da página como *links*, botões, *dropdowns*, entre outros

3.3.5 Documentação

Para a documentação deste projeto, utilizou-se a ferramenta *Confluence*, já abordada no subcapítulo 2.4.3. O *Confluence* foi útil para:

- Criar notas de reunião
- Criar planos de projeto
- Criar requisitos
- Guardar versões antigas dos ficheiros
- Manter os ficheiros seguros

3.3.6 Gestão de containers

Foi necessário recorrer à gestão de *containers* para executar a ferramenta *Jira software*, *Jenkins* e *Sonarqube* num ambiente isolado (container) e instalar a versão do NTX na máquina hospedeira. Para isso selecionou-se a ferramenta *Docker* (subcapítulo 2.4.2), porque fornece:

- Fácil configuração – Permite configurar o sistema de forma fácil e rápida;
- Isolamento de execução – Permite executar as aplicações em modo *sandboxing*, contendo os recursos em uso por estas do resto do sistema que aloja o *container*.
- Memória – Fornece um espaço menor do sistema operativo;

A tecnologia *Docker* usa o Kernel do Linux e os recursos que este fornece para isolar processos. O objetivo do container é criar essa independência, ou seja, ser capaz de executar vários processos e aplicações de forma independente para fazer o melhor uso possível da infraestrutura, enquanto mantém a sua segurança. Ao contrário das máquinas virtuais, o *Docker* permite que um conjunto de processos possam ser executados de uma forma isolada, através da partilha do *kernel*. Outro aspeto importante que este isolamento traz é a organização das aplicações e das suas dependências, pois torna-se possível isolar todas as aplicações e as suas dependências são organizadas e correm dentro do seu container.

3.4 Operacionalização das métricas

Esta secção descreve a forma como foram obtidos os valores que serão usados para as métricas especificadas no subcapítulo 3.1.

3.4.1 Determinação do tempo e frequência das entregas em produção

Para tornar a entrega de *software* uma tarefa fácil de ser realizada, é necessário tornareste processo totalmente automático, através da entrega contínua. Para isso irão ser criados três ambientes de desenvolvimento: desenvolvimento (DEV), testes (QA) e produção (Prod). O primeiro ambiente, tal como o nome indica, é o local onde o *software* é moldado pela equipa de desenvolvimento para atender às necessidades do cliente. Assim que a equipa de desenvolvimento possui uma determinada funcionalidade ou uma nova versão de *software* para acrescentar em produção, faz- se a entrega do pacote do *software* para o ambiente de QA. O ambiente de QA é o local onde a equipa de testes trabalha no *software* proveniente do ambiente de DEV. Se eventualmente o *software* não passar em um ou mais cenários de teste, ele volta para o ambiente de DEV para posteriormente ser corrigido. Após essa correção, voltaao ambiente de QA para ser novamente testado e, quando passar em todos os requisitos de teste e todas as equipas envolvidas estiverem satisfeitas com o funcionamento do *software*, ele será colocado em produção e ficará visível para o cliente.

Figura 14 – Ambientes Jenkins

Para a entrega contínua e gestão das entregas para o ambiente de produção utilizou-se a ferramenta *Jenkins*. Com esta ferramenta foi possível criar os três ambientes (através de *Jenkins jobs*) acima descritos e fazer a entrega de *software* para qualquer ambiente. Esta métrica que se pretende avaliar está relacionada com a iteração do *software*, mas é facilmente levantada através da ferramenta *Jenkins* porque é possível obter um ficheiro *log* com todas as informações acerca da entrega que foi feita, tais como, data e hora, o tempo que esta demorou, o ambiente, entre outros.

3.4.2 Determinação do tempo médio de deteção de falhas em produção

Sempre que uma determinada falha em produção é detetada, é criado um cartão no quadro *Scrum*. Um cartão do quadro *Scrum* é um elemento visual de uma unidade de trabalho que representa o trabalho que foi requisitado para ser desenvolvido. Este cartão contém informações detalhadas sobre a tarefa que se irá realizar, como o seu estado, o tipo de tarefa a realizar, a sua descrição, a pessoa responsável, prazo, entre outro tipo de detalhes.

O tempo médio de deteção de falhas em produção, tal como foi referido na sua especificação, é calculado através da seguinte fórmula:

$$Total\ de\ horas\ do\ software\ em\ funcionamento \div N\acute{u}mero\ total\ de\ problemas$$

Fórmula 1 – Determinação do tempo médio de deteção de falhas em produção

O total de horas de *software* em funcionamento contínuo é calculado pela diferença temporal entre o momento em que o *software* é colocado em produção e o momento em que uma determinada falha é detetada. O momento em que o *software* é colocado em produção é obtido na ferramenta *Jenkins*.

O momento em que uma determinada falha é detetada pode ser obtido no quadro *Scrum*, através dos detalhesda tarefa (Figura 15).

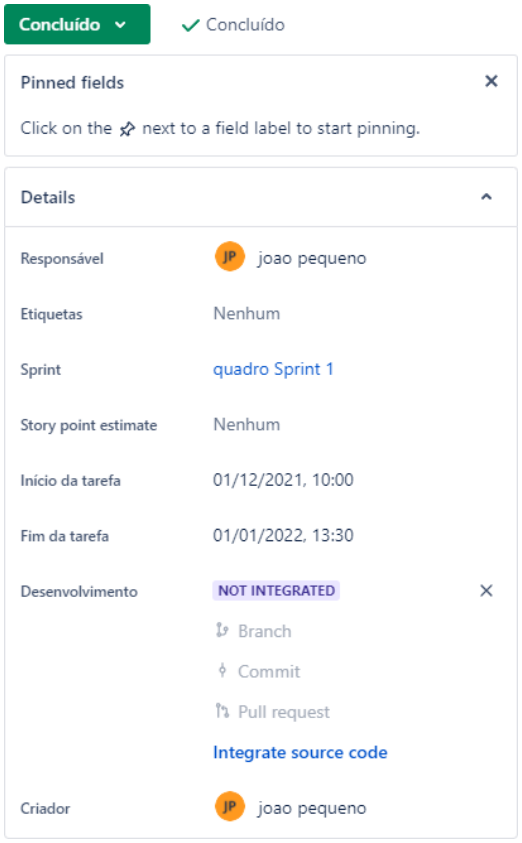


Figura 15 – Detalhes da tarefa

3.4.3 Determinação do tempo médio de reparação de um problema

O tempo médio de reparação de um problema em produção, à semelhança da métrica anterior, será calculado com base no quadro *Scrum*. Esta métrica é calculada através da seguinte fórmula:

$$Total\ de\ horas\ gastas\ em\ reparações\ de\ problemas \div Número\ total\ de\ problemas$$

Fórmula 2 – Determinação do tempo médio de reparação de um problema

O total de horas gastas na reparação de um determinado problema será a diferença temporal da mudança de estado do cartão de ativo para fechado. Assim que o cartão com a informação relativa ao problema encontrado estiver fechado, é atribuído num dos campos do cartão o tempo que se gastou para corrigir o problema (Figura 15). Estes cálculos são realizados semanalmente e os resultados obtidos podem ser consultados no Apêndice D.

3.4.4 Determinação do nível de satisfação do cliente

O *feedback* recebido por parte do cliente foi obtido por meio de formulários de satisfação. Os formulários de satisfação permitem obter as opiniões do cliente acerca da sua experiência com o *software*. Depois de coletar o seu *feedback*, foi necessário analisar os dados e verificar se é necessário alterar ou acrescentar algo ao *software*.

Os *feedbacks* foram recolhidos trimestralmente através da ferramenta *Google forms* e os resultados são guardados numa folha Excel. No Apêndice G pode-se observar um exemplo do formulário de satisfação entregue a um dos clientes.

3.4.5 Determinação do tempo médio de consideração de um problema

O tempo em que um problema reportado demora a ser considerado, à semelhança das métricas "tempo médio de deteção de falhas em produção" e "tempo médio de reparação de um problema em produção", será calculado com base no quadro *Scrum*. Esta métrica é calculada através da seguinte fórmula:

$$Momento\ do\ tratamento\ do\ problema - Momento\ do\ report\ do\ problema$$

Fórmula 3 – Determinação do tempo médio de consideração de um problema

Para efetuar o cálculo desta métrica é criado um cartão no *backlog* relativo ao problema que terá de ser corrigido. O momento do *report* do cliente é obtido da mesma forma que na métrica "tempo médio de deteção de falhas em produção", através do cartão existente no quadro *Scrum*. O momento do tratamento do problema será quando esse mesmo cartão passa do estado "novo" para o estado "ativo".

3.4.6 Determinação da qualidade do software

O levantamento da qualidade de *software* inclui diversas vertentes como funcionalidade, confiabilidade, usabilidade, eficiência, entre outros. Devido ao nível de complexidade deste tema e para simplificar a operacionalização desta métrica, optou-se por considerar apenas o resultado obtido na execução dos testes desenvolvidos no ambiente de testes pela ferramenta *Selenium*.

A execução dos testes de *software* vai servir para verificar o estado do funcionamento do *software* desenvolvido e vai permitir levantar a quantidade de falhas que ocorrem durante um determinado número de execuções, que será o valor usado para comparar o nível de qualidade do *software* entre o antes e o após a implementação de práticas de DevOps.

No próximo capítulo será detalhada a arquitetura do NTX e alguns processos aplicados pela Noesis em outros projetos.

4 NTX e processos aplicados pela Noesis

Os próximos três subcapítulos descrevem o projeto NTX e a metodologia de desenvolvimento usada para a sua implementação pela empresa Noesis.

4.1 Introdução ao NTX

O NTX “Ngine Testing Experience” é uma framework de automação de testes que permite à empresa Noesis responder à necessidade dos seus clientes na execução de testes de regressão, a qualquer momento e de forma automática (Apêndice B). Esta framework permite que qualquer pessoa sem conhecimentos técnicos (apenas funcionais), ao utilizar as funcionalidades disponibilizadas, consiga automatizar e executar um determinado conjunto de testes. A utilização desta ferramenta permite obter algumas vantagens, tais como:

- Criar passo-a-passo testes automáticos a partir da escolha dos objetos existentes na página web;
- Automatizar aplicações mobile;
- Duplicar casos de teste;
- Passar dados e objetos entre diferentes steps de teste;
- Calendarizar a execução de testes;
- Testar em vários browsers (ie, Firefox, Edge, Chrome, safari)

A arquitetura da ferramenta NTX é a seguinte:

- Servidor
    - *Tomcat*
- Bases de dados
    - *Oracle db*
    - *MS SQL Server*
    - *MYSQL*
- Repositórios de teste
    - *HP ALM*
    - *Xray*
    - *Azure DevOps*
- Testes web
    - *Software*
- Testes mobile (Android e iOS)
    - *Appium*

- Testes desktop
    - *SAP*

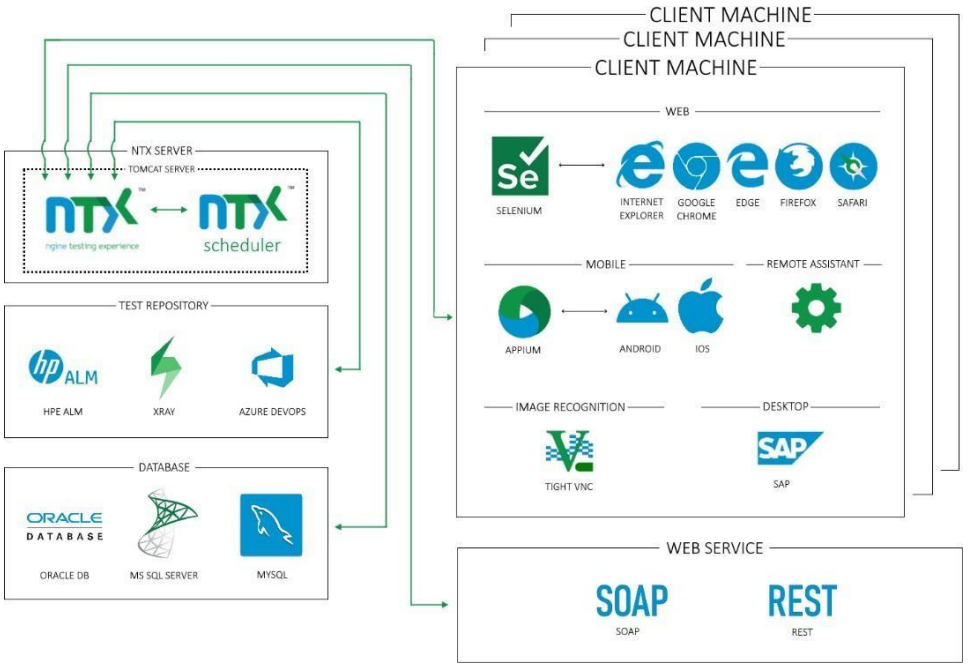


Figura 16 – Arquitetura NTX

4.2 Operacionalização das métricas pre-DevOps

Após a especificação das métricas, é necessário efetuar o seu levantamento, de forma a ser possível efetuar a comparação dos resultados entre as duas metodologias. De seguida será abordada de que forma se operacionalizaram as métricas antes da implementação da metodologia DevOps.

- Tempo e frequência das entregas no ambiente de produção

As entregas no ambiente de produção eram realizadas através da ferramenta *Azure*. As entregas não eram efetuadas de forma contínua, ou seja, o envio do *software* para produção era feito manualmente. *Esta* ferramenta permite obter um histórico de todas as entregas, portanto é possível obter o tempo e a frequência com que estas foram efetuadas.

- Tempo médio de deteção de falhas em produção

Os detalhes das falhas em produção eram armazenados numa folha Excel que continha o momento do *report* da falha, o tipo de falha, a criticidade da falha, a versão do *software*, o início de reparação do problema e o fim de reparação do problema. O tempo médio de deteção de falhas em produção seria a diferença entre o momento exato da conclusão da entrega do *software*, que poderia ser obtido na ferramenta *Azure DevOps*, e o momento do *report*, que era obtido nas folhas *Excel*.

- Tempo médio de reparação de um problema em produção

Como abordado no tópico anterior, o tempo de reparação de um problema em produção era registado no Excel das falhas em produção.

- Nível de satisfação do cliente

O nível de satisfação dos clientes era recolhido através de *feedbacks* recolhidos trimestralmente através de trocas de e-mail e as respostas eram guardadas em folhas *Excel*.

- Tempo em que um problema reportado demora a ser considerado

O tempo em que um problema reportado demora a ser considerado era calculado pela diferença entre o momento do fim de reparação do problema e o momento em que o problema foi reportado. Estes dois fatores podem ser consultados no documento *Excel* referido nas métricas “Tempo médio de reparação de deteção de falhas em produção” e “Tempo médio de reparação de um problema em produção”.

- Qualidade do *software*

A qualidade do *software* era medida de duas formas: através de testes unitários e através de testes manuais. Os testes unitários eram realizados no momento da implementação de uma funcionalidade no *software* e os testes manuais eram feitos a essa mesma funcionalidade, antes de se criar uma versão para entregar ao cliente. Os resultados desses testes, assim como os *inputs* usados, eram armazenados também numa folha *Excel*.

4.3 Processos aplicados nos projetos da Noesis

Os processos aplicados pela empresa Noesis no desenvolvimento dos seus projetos são: recolha de requisitos, planeamento e estimação, orçamento, desenvolvimento, e, por fim, planeamento de qualidade e risco.

No início de cada projeto é feita a recolha de requisitos, que serão as funcionalidades e outros aspetos importantes que serão desenvolvidos no projeto. O objetivo final deste processo é existir uma lista de requisitos bem definidos e aprovados pelo cliente. Esses requisitos serão divididos em tarefas e procede-se à estimação do esforço necessário para as desenvolver. O processo de orçamento é referente à entrega de orçamento e prazos ao cliente. Assim que o cliente decida avançar, inicia-se o desenvolvimento.

O desenvolvimento segue a metodologia ágil de desenvolvimento de *software*, mais propriamente a metodologia Scrum. A metodologia Scrum caracteriza-se pela inovação, flexibilidade, competitividade e produtividade. Estas são as chaves de uma metodologia que se baseia numa estrutura de desenvolvimento incremental. Os projetos desenvolvidos anteriormente através da metodologia Scrum foram bem-sucedidos, tanto a nível da satisfação dos clientes, como a nível da satisfação das equipas de desenvolvimento. Desta forma, a Noesis tem vindo cada vez mais a adotar e a melhorar os seus processos de desenvolvimento ágil, através dos processos de planeamento de qualidade e risco que permitem obter uma visão geral do que correu bem e menos bem no desenvolvimento dos projetos.

5 Implementação

Cada uma das fases do ciclo DevOps, referidas no capítulo 2.2, são suportadas por um conjunto de ferramentas. Não existe um guia das ferramentas a que deverão ser utilizadas devido ao fato do DevOps consistir numa mudança cultural e de colaboração entre as equipas de desenvolvimento e operações (Andy Lipnitski, 2021). Desta forma, não existe nenhuma ferramenta única para a aplicação do DevOps, mas sim um conjuntode ferramentas que interagem entre si, onde todos os estágios do ciclo de vida estãoconstantemente a transferir informação entre eles. No próximo subcapítulo serão referidos alguns detalhes relativos à implementação que será efetuada.

5.1 Detalhes da implementação

Considerando as características do projeto NTX, as ferramentas selecionadas e as práticas de DevOps, é necessário construir um conjunto de processos automatizados que permitem compilar, construir e entregar de maneira confiável todas as suas alterações de *software*. Desta forma, é necessário criar uma pipeline de DevOps com o conjunto de ferramentas selecionadas anteriormente para que elas interajam entre si, criando um ciclo automatizado entre as cinco fases da implementação do *software* abordadas em cima: requisitos, desenvolvimento, testes, entrega e monitorização. Deste modo, a arquitetura da pipeline que será implementada deverá seguir o modelo ilustrado na figura 17.

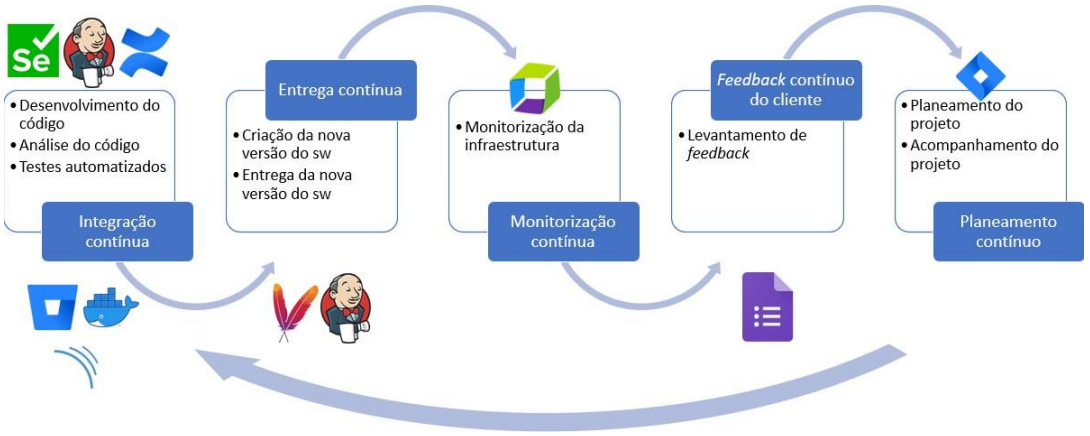


Figura 17 – Arquitetura da pipeline automatizada

5.1.1 Desenvolvimento e análise de código

O fluxo de desenvolvimento e análise de código trata-se de um conjunto de tarefas que são realizadas desde o momento em que um determinado elemento da equipa de desenvolvimento inicia o desenvolvimento de uma funcionalidade até ao momento em que essa funcionalidade é adicionada ao repositório de código.

A primeira etapa do fluxo de desenvolvimento e análise de código é a atualização do projeto que se encontra no computador do elemento responsável pelo desenvolvimento da nova funcionalidade. Assim que o elemento responsável obtiver a versão mais recente que se encontra no repositório partilhado, é iniciado o desenvolvimento da nova funcionalidade. A próxima etapa trata-se da atualização do repositório partilhado, ou seja, no fim do desenvolvimento da funcionalidade, o elemento responsável executa o *commit* para atualizar o repositório Bitbucket e é iniciada uma tarefa do *Sonarqube* que analisa o código que foi adicionado. Em caso de sucesso, é ativado um *job* do *Jenkins* que publica o relatório de análise de código que foi analisado na ferramenta *Confluence*. Em caso de falha, o Bitbucket gera um código de erro que será retornado ao elemento responsável e o repositório não será atualizado. A figura que se segue representa o funcionamento do fluxo de desenvolvimento e análise de código.

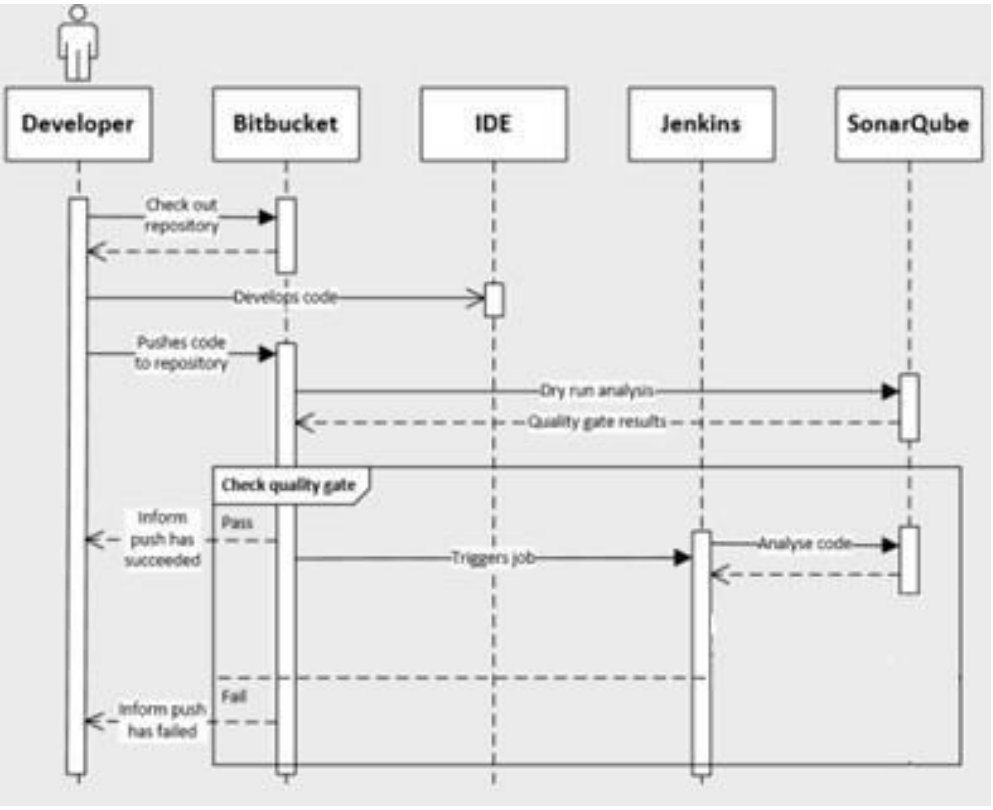


Figura 18 – Fluxo de desenvolvimento e análise de código

O fluxo de desenvolvimento implica a utilização da ferramenta *Git* para atualizar o repositório *Bitbucket*. Esta ferramenta fornece uma variada lista de comandos que permitem fazer toda a gestão do repositório central. Como se pode observar na figura anterior, o fluxo de desenvolvimento inicia com a atualização do projeto local. Após atualizar o projeto localmente, o programador responsável está apto para iniciar o desenvolvimento. Assim que terminar o desenvolvimento e os seus testes unitários, é necessário atualizar o código que se encontra no repositório remoto *Bitbucket*. Após a tentativa de atualização do código do repositório, será iniciado o fluxo de análise do código que foi desenvolvido que será detalhado na próxima subsecção.

Para além das tarefas que este fluxo de desenvolvimento envolve, foi também utilizada a ferramenta *Docker* única e exclusivamente destinada à criação de ambientes de desenvolvimento e testes localmente, através de containers, para desenvolver as tarefas por parte das equipas responsáveis e para gerir as aplicações *Sonarqube*, *Jenkins* e *Jira software*. A lista dos *containers* criados é ilustrada na seguinte figura e esta tarefa não incluiu a criação de volumes nem a utilização de *docker-compose* para orquestrar o arranque e a paragem dos *containers*. Ao invés disso foi utilizado um ficheiro *Dockerfile* que possui um conjunto de instruções ao qual o Docker é capaz de ler e com base nessas instruções criar, de forma automática, uma imagem. Uma imagem Docker é constituída por um conjunto de camadas. Cada uma destas camadas representa uma instrução na Dockerfile da imagem. É possível perceber que esta arquitetura simplifica bastante o uso de recursos e a sobrecarga do sistema para podermos configurar qualquer aplicação. A principal vantagem desta arquitetura é que não é necessário ter hardware extra para o sistema operativo hospedeiro, porque tudo isso será tratado dentro do container.

Figura 19 – Docker containers

5.1.2 Análise do código desenvolvido

A análise do código desenvolvido é feita pela ferramenta *Sonarqube*. Para se iniciar essa análise, é necessário que exista uma tentativa de atualização do repositório remoto do *Bitbucket*. A ferramenta que faz a ligação entre o código que se pretende adicionar ao repositório e o *Sonarqube* é o *Jenkins*. Para efetuar essa ligação, é necessário criar um *job* do *Jenkins* e efetuar duas configurações. A primeira configuração trata-se da ligação do *Jenkins* ao *Bitbucket,* através do *url* do repositório, e a segunda é o *trigger* desse *job* para efetuar a ligação ao *Sonarqube*, através da utilização de um *token* gerado pelo *Sonarqube* (Figura 20).

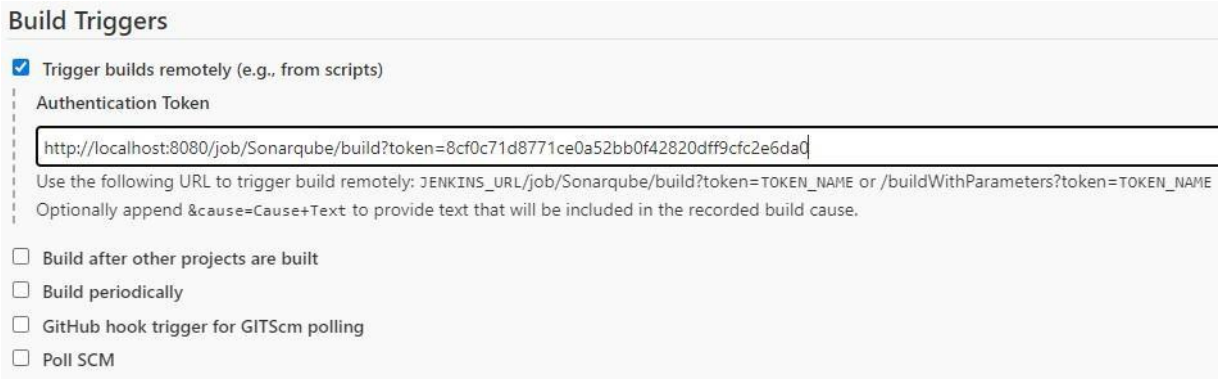


Figura 20 – Integração Jenkins com Sonarqube

A próxima etapa é executar a análise do código por parte do *Sonarqube*. Apenas os ficheiros que foram modificados pelo programador serão analisados e no final desta análise é criado um relatório com os dados da análise efetuada. Esta análise é efetuada utilizando o Sonar Scanner e a integração com o *Jenkins* foi feita através do identificador do projeto e o login (Figura 21). Caso o código seja adicionado ao repositório remoto significa que a análise feita pelo *Sonarqube* foi aprovada com sucesso. No próximo subcapítulo será abordado o fluxo dos testes automatizados.

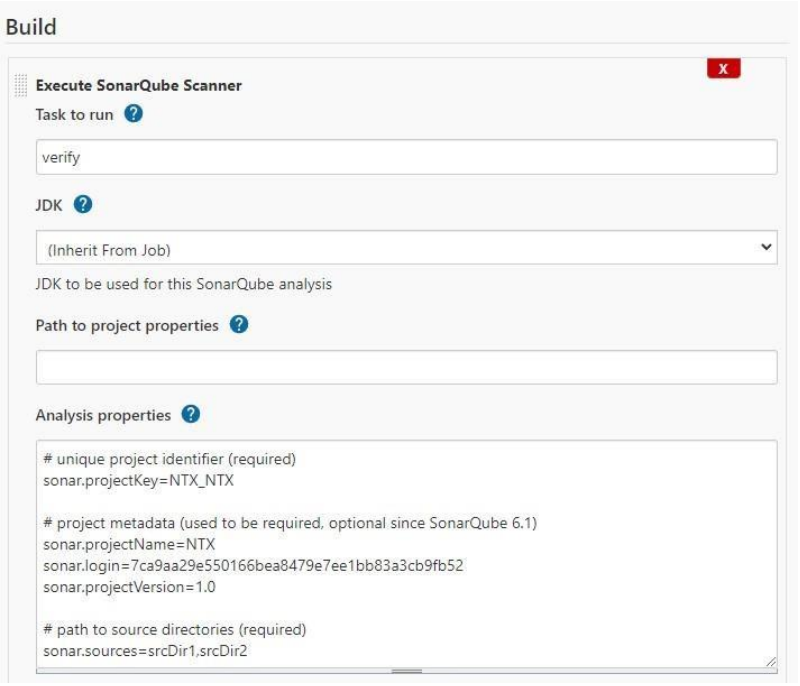


Figura 21 – Execução do Sonarqube Scanner

5.2 Testes automatizados

O fluxo de testes automatizados inicia quando o elemento da equipa de testes seleciona a lista de testes que deseja executar até ao momento em que os testes são executados.

A primeira etapa do fluxo de testes automatizados é a seleção do conjunto de testes que se pretende executar, na ferramenta *Jenkins*. Quando é feita essa seleção, o *Jenkins* procede à clonagem do repositório de testes e executa um conjunto de comandos para realizar os testes. A implementação do fluxo de testes automatizados dividiu-se em duas etapas principais, referidas nos pontos que se seguem, que serão ilustradas na figura que se segue e detalhadas nos subcapítulos seguintes:

- Seleção dos testes;
- Execução dos testes;

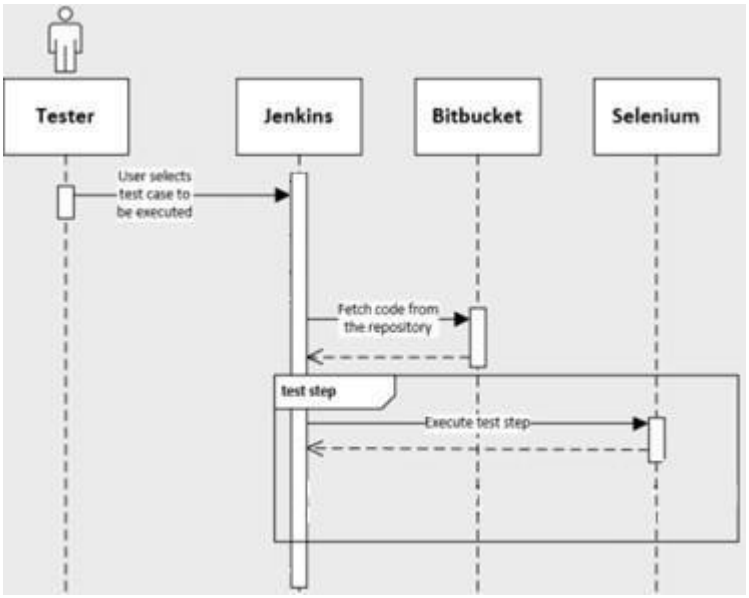


Figura 22 – Fluxo de testes automatizados

5.2.1 Seleção dos testes

A seleção dos testes é feita através da atualização do estado das tarefas no *Jira software*. Cada tarefa existente no *Jira software* possui o seu fluxo, que está ilustrado na figura que se segue. O fluxo possui quatro estados: “não iniciado”, “em progresso”, “automated test” e “concluído”. Sempre que uma tarefa é arrastada para o estado “automated test”, a execução do teste relativo a essa tarefa será iniciada.

Figura 23 – Jira software board

Neste caso específico, a integração entre o *Jenkins* e o *Jira software* é feita por um recetor que despoleta o *job* sempre que a tarefa “Navegar Menu Teste” passa para o estado “automated test”. Quando o *job* é acionado, inicia-se a execução dos testes que será detalhada na subsecção seguinte. A integração entre a mudança de estado do *Jira software* e o *job* do *Jenkins* é ilustrada na figura que se segue.

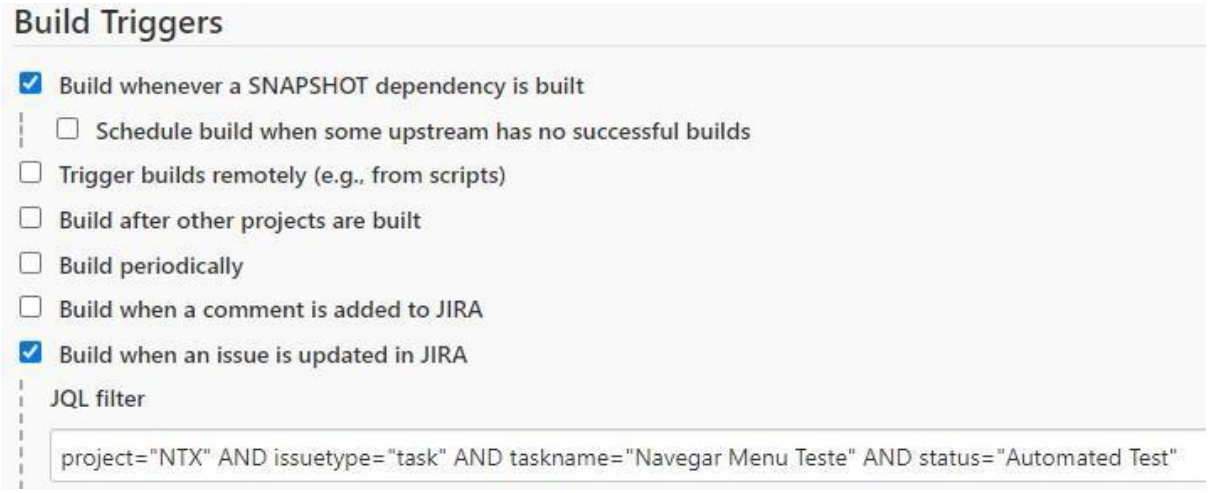


Figura 24 – Integração Jenkins com Jira software

5.2.2 Execução dos testes

A ferramenta *Jenkins* não permite fazer a integração com os scripts de teste diretamente devido ao facto de não existir, até ao momento, um plugin de *Software* no *Jenkins Marketplace*. Desta forma, foi necessário proceder à realização de um pequeno *workaround* que permitisse a execução dos testes *Software* através da execução de um *job* do *Jenkins*.

Para realizar essa integração foi necessário criar um esquema de suite de testes na *framework TestNG* para integrar com a ferramenta *Software*. A suite de testes é composta por um ficheiro pom (Apêndice F) e através da sua execução é possível executar o caso de teste (Navegar Menu Teste). A seguinte figura ilustra a integração entre o *Jenkins* e a suite de testes.

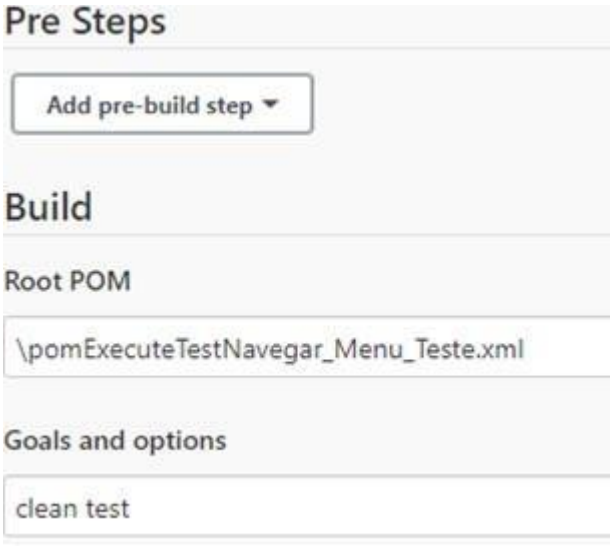


Figura 25 – Integração Jenkins com a suite de testes

Os exemplos de execução que foram ilustrados são referentes apenas a um teste (Navegar Menu Teste). Todos os restantes testes que foram criados são executados da mesma forma pois foi criado um *job* para cada um dos casos de teste e foi efetuada a alteração dos parâmetros dos *build triggers* e *build*. Sempre que existe uma atualização no repositório de testes é acionado um *job* de testes principal que despoleta todos os *jobs* que foram criados para os restantes *scripts* de teste.

5.3 Entrega de *software*

O fluxo de entrega de *software* inicia quando o *job* do *Jenkins* que executa os testes é concluído com sucesso, ou seja, assim que termina o fluxo de testes automatizados. Este fluxo inicia com a criação de uma nova versão ou pacote de *software* e termina com a sua entrega num servidor de aplicações.

A implementação do fluxo de entrega de *software* dividiu-se em duas etapas principais, que serão ilustradas na figura que se segue e detalhadas nos subcapítulos 5.3.1 e 5.3.2.

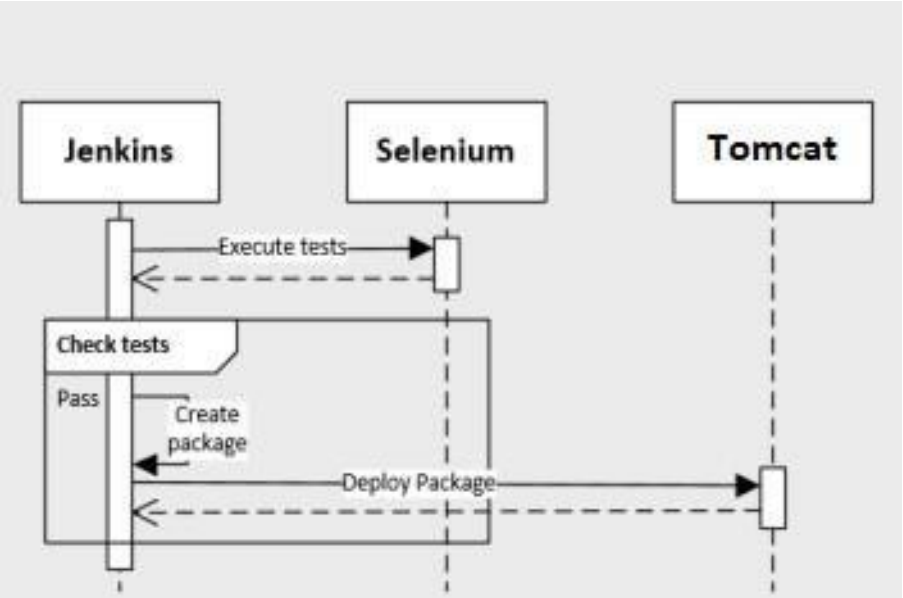


Figura 26 – Integração Jenkins com a suite de testes

5.3.1 Criação da nova versão do software

A partir do momento em que a execução dos testes termina com sucesso, procede-se para o empacotamento da nova versão do NTX. O empacotamento da aplicação será acionado pelo *job* responsável pela execução do ficheiro de dependências da aplicação, o ficheiro *pom.xml*. A *build* deste ficheiro permite criar um ficheiro *.war* (pacote) que contém a nova versão do NTX desenvolvida. A configuração do *job* no *Jenkins* é ilustrada na figura que se segue. O próximo passo será carregar esta nova versão no servidor *Tomcat*(ambiente de produção).

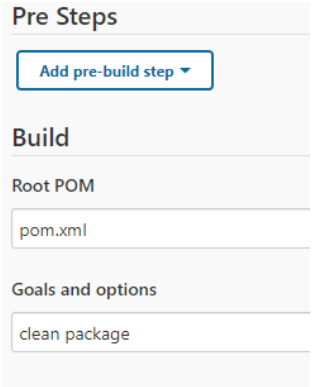


Figura 27 – Configuração do Jenkins para gerar o novo pacote NTX

5.3.2 Entrega da nova versão do software

A entrega da nova versão do NTX para o servidor *Tomcat* foi realizada através do *post build* do *job* descrito anteriormente. A entrega do novo pacote implica a desinstalação e a remoção da versão que se encontra no servidor *Tomcat*. A configuração do *job* é ilustrada na imagem que se segue e no Apêndice B encontram-se algumas imagens da aplicação NTX a executar no servidor de aplicações *Tomcat.* No apêndice E, é possível observar o tempo de entrega de *software* em ambiente de produção gerado pela ferramenta *Jenkins.*

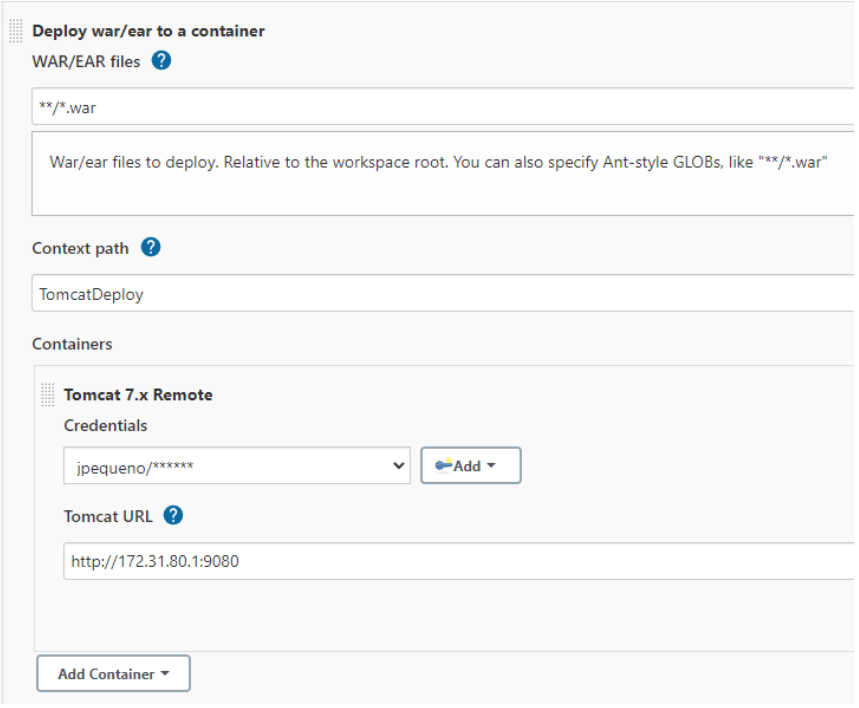


Figura 28 – Integração Jenkins com Tomcat

5.4 Monitorização

A monitorização da aplicação NTX, através da ferramenta *Dynatrace*, não é feita de forma automatizada devido ao facto de não existir nenhum *plugin* que permita fazer a integração do *Jenkins* com o *Dynatrace*. Desta forma, a monitorização é feita através da aplicação *Dynatrace One Agent*, que é inicializada manualmente. Os resultados das monitorizações efetuadas podem ser consultados no Apêndice A.

6. Análise e discussão de resultados

Após a pesquisa, estudo e aplicação da metodologia DevOps efetuadas nos capítulos anteriores, o presente capítulo pretende demonstrar a análise e discussão dos resultados obtidos à implementação efetuada. Desta forma, o objetivo é procurar identificar os principais resultados na implementação de práticas de DevOps, para que posteriormente possam ser analisados em comparação com as métricas especificadas no sub-capítulo 3.1.

A análise que se pretende elaborar, com a análise aos resultados obtidos, tem como principal objetivo a validação dos benefícios que a metodologia DevOps propõe após a sua adoção. Para validar os benefícios da implementação de DevOps é necessário efetuar a análise aos resultados obtidos face às métricas levantadas durante a realização do trabalho. O levantamento dos resultados é referente a dois momentos: antes da implementação DevOps (pre-DevOps) e após aimplementação DevOps (pos-DevOps). O objetivo será verificar os resultados e averiguar se os mesmos acrescentam melhorias através da aplicação da metodologia DevOps. Nas próximas duas secções serão demonstrados os resultadosobtidos antes e depois da implementação da metodologia DevOps, respetivamente.

6.1 Resultados obtidos

- Tempo e frequência das entregas no ambiente de produção

Tempo e frequência		
Métricas	Tempo de entrega	Frequência de entrega
Pre-DevOps	~40min	2x por semana
Pos-DevOps	~40 min	3x por semana

Tabela 4 – Métrica "Tempo e frequência das entregas no ambiente de produção"

Segundo os resultados referentes ao tempo e frequência das entregas no ambiente de produção, apresentados no Apêndice E, o resultado obtido no tempo médio de entrega do *software*, antes da implementação das metodologias DevOps, foi cerca de quarenta minutos e o resultado obtido no tempo medio de frequência de entrega foi duas vezes por semana. É importante referir que existiram 9 entregas referentes à fase Pre-DevOps e 21 entregas referentes à fase Pos-DevOps.

Após a comparação dos resultados, é possível verificar que a implementação DevOps melhorou a nível da frequência de entrega e otempo de entrega manteve-se o mesmo. A análise e a comparação dos resultados entre estes resultados serão efetuadas no subcapítulo 6.2

- Tempo médio de deteção de falhas em produção

Deteção de falhas	
Métrica	Tempo médio de deteção de falhas em produção
Pre-DevOps	~2 horas
Pos-DevOps	~1.5 horas

Tabela 5 – Métrica "Tempo médio de deteção de falhas em produção"

Na tabela anterior são apresentados os resultados obtidos à métrica do tempo médio de deteção de falhas em produção. O resultado obtido foi de aproximadamente 2 horas antes da implementação de DevOps e uma hora e meia após sua implementação. Desta forma, a diferença temporal entre o momento em que o *software* é colocado em produção e o momento em que uma determinada falha foi melhorada após a implementação de DevOps.

- Tempo médio de reparação de um problema em produção

Reparação de problema	
Métrica	Tempo médio de reparação de um problema
Pre-DevOps	~1.5 horas
Pos-DevOps	~1 hora

Tabela 6 – Métrica "Tempo médio de reparação de um problema em produção"

O tempo médio de reparação de um problema antes da implementação de DevOps era de, aproximadamente, uma hora e meia e, agora, após a implementação de DevOps, é cerca de uma hora. Os resultados que foram levantados referentes ao tempo médio de reparação de um problema encontram-se no Apêndice D.

- Nível de satisfação do cliente

***Feedback* do cliente**			
Métrica	*Feedback* do cliente		
***Feedback* geral**	Positivo	Neutro	Negativo
Pre-DevOps	~73.3%	~26.7%	0%
Pos-DevOps	~85.7%	~14.3%	0%

Tabela 7 – Métrica "Nível de satisfação do cliente"

Após a verificação da tabela anterior, é possível afirmar que, relativamente ao pre-DevOps, aproximadamente 73.3% dos clientes estavam satisfeitos com o NTX e 26.7% tinham opinião neutra. Relativamente ao pos-DevOps, aproximadamente 85.7% dos clientes estavam satisfeitos com o NTX e 14.3% tinham opinião neutra. O *feedback* foi recolhido trimestralmente e o número total de clientes aumentou na fase pre-DevOps, pelo que as percentagens apresentadas na tabela anterior não correspondem ao mesmo número de clientes. Apesar disso, como o *feedback* levantado não foi anónimo, foi possível levantar o *feedback* de alguns clientes que se mantiveram na fase pre-DevOps e pos-DevOps e a satisfação aumentou substancialmente, onde grande parte dos clientes que tinham o seu nível de satisfação "Neutro", acabaram por alterar o seu *feedback* para "Positivo" na fase pos-DevOps, como pode ser visivel no Apêndice C. Desta forma, é possível verificar que após a implementação de DevOps, os clientes ficaram mais satisfeitos. O formulário que foi utilizado para o levantamento da satisfação dos clientes pode ser consultado no Apêndice G.

- Tempo em que um problema reportado demora a ser considerado

Tempo de consideração	
Métrica	Tempo que um problema demora a ser considerado
Pre-DevOps	~10 horas
Pos-DevOps	~8 horas

Tabela 8 – Métrica "Tempo que um problema demora a ser considerado"

Na tabela anterior são apresentados os resultados obtidos à métrica do tempo médio de consideração de um problema. O resultado obtido foi de aproximadamente 10 horas antes da implementação de DevOps e oito horas após sua implementação.

- Qualidade do software

Qualidade do *software*	
Métrica	Qualidade do *software*
Pre-DevOps	Testes unitários
Pos-DevOps	Testes unitários + testes automatizados

Tabela 9 – Métrica "Qualidade do software"

Para concluir, na tabela anterior são apresentados os resultados referentes à qualidade do *software*, antes e após a implementação das metodologias DevOps. Antes da implementação de DevOps apenas eram executados testes unitários e, neste momento, após a implementação da metodologia DevOps, foram criados automatismos para executar testes automatizados em *Software*. Através desta implementação é possível garantir uma maior qualidade de *software* em pos-DevOps porque existe maior cobertura e maior quantidade de funcionalidades testadas.

6.2 Análise comparativa e discussão

Ao efetuar a comparação entre os resultados obtidos para a métrica "Tempo e frequência das entregas no ambiente de produção", pode-se afirmar que o tempo de entrega se mantém e que a frequência das entregas aumenta. O tempo de entrega ronda os mesmos valores no pre-DevOps e no pos-DevOps devido ao facto de que antes da aplicação de DevOps existia a utilização da ferramenta *Azure* para a entrega de *software* e, após a implementação das metodologias DevOps, existe a ferramenta *Jenkins* para o mesmo efeito. Como em ambos os momentos são utilizados ferramentas de CD, não se observa uma diferença significativa no tempo de entrega de *software*. No caso da frequência das entregas, o aumento no pos-DevOps deve-se, principalmente à automatização dos restantes processos DevOps e ao foco que as equipas passaram a ter para entregar o NTX em lotes mais pequenos.

Quanto à métrica "Tempo médio de deteção de falhas em produção", verifica-se também uma melhoria com a implementação da metodologia DevOps. Enquanto o valor obtido no pre-Devops é cerca de duas horas, no pos-DevOps é de uma hora e meia e isto deve-se, principalmente, ao facto de se ter verificado resultados significativos a nível do *feedback* e comunicação ao longo do ciclo de

desenvolvimento. Um outro fator importante que permitiu melhorar o resultado da deteção de falhas em produção após a implementação de DevOps foi a implementação de uma ferramenta de monitorização que valida a execução da aplicação em produção.

O resultado obtido para a métrica “Tempo médio de reparação de um problema em produção”, antes da implementação das metodologias DevOps, era de aproximadamente uma hora e meia e, atualmente, após a implementação de DevOps, esse tempo reduziu cerca de meia hora. A melhoria do resultado obtido pos-DevOps em relação ao pre-DevOps deve-se, principalmente à gestão de *containers* que foi criada através da ferramenta *Docker* e à divisão dos ambientes criada no *Jenkins*. Estes dois aspetos permitem desenvolver e testar o código em ambientes muito semelhantes ao ambiente de produção, resultando assim na diminuição do número de erros e diminuição de problemas relacionados com as adaptações aos ambientes.

Quanto aos resultados obtidos na métrica “Nível de satisfação do cliente”, é possível verificar que houve uma melhoria significativa nos *feedbacks* recebidos e o principal fator para este resultado deve-se às melhorias visíveis das entregas que foram efetuadas. O feedback na fase pos-DevOps foi levantado com outras versões de *software* em relação à fase pre-DevOps, pelo que o desenvolvimento colaborativo, a integração contínua e a entrega contínua permitiram não só são melhorar a qualidade do *software*, como também o aumento da satisfação do cliente. Apesar desta comparação ter sido feita com valores referentes a universos diferentes, os clientes pre-devops que se manteram na fase pos-devops aumentaram a sua satisfação e o aumento da satisfação de grande parte dos clientes é unânima.

A métrica “Tempo em que um problema reportado demora a ser considerado” também foi melhorada com a implementação de DevOps e o resultado obtido deve-se em muito à gestão mais eficiente das equipas e que, consequentemente, leva a uma maior eficiência global da empresa dado que as equipas envolventes podem agora focar-se em tarefas que primam pela qualidade do *software*.

A última métrica que será analisada é a métrica “Qualidade do *software*”. Através dos resultados obtidos, é possível comparar que a qualidade geral do NTX aumentou e esta melhoria deve-se principalmente aos resultados que foram obtidos aos testes de *software* realizados. Como já foi referido anteriormente, apesar desta métrica não ser completamente avaliada devido à complexidade da medição desta métrica, a utilização de ferramentas como *Sonarqube*, *Software* e os resultados obtidos na ferramenta *Dynatrace*, permitem afirmar que o NTX possui melhor qualidade em comparação ao desenvolvimento pre-DevOps.

7 Conclusões

De seguida serão apresentadas todas as conclusões retiradas acerca do trabalho realizado e alguns aspetos que poderiam ser melhorados neste projeto.

7.1 Principais conclusões

Após a investigação sobre a relevância do tema em análise e a clarificação dos objetivos do estudo, deu-se início a uma detalhada revisão da literatura, revisão essa que envolveu a análise profunda sobre o DevOps de forma a obter toda a compreensão sobre quais são os desafios da sua adoção ao nível organizacional e o paralelismo com a metodologia ágil, utilizada frequentemente como termo de comparação. Esta análise profunda permitiu obter o conhecimento e aptidões necessárias não só para a implementação do ciclo de DevOps, como também para moldar as equipas a adotarem este tipo de metodologia. A mudança da metodologia Scrum para a cultura DevOps está associada a processos de desenvolvimento ágeis, o que garante que os processos estão mais incorporados na empresa e são mais automatizados, conferindo a utilização de práticas de integração contínua e entrega contínua. Estas práticas garantem que os processos são executados de forma mais eficiente, são mais autónomos e com menor intervenção humana, o que leva a uma diminuição de falhas.

Existem várias barreiras, referidas na secção da revisão da literatura**,** que inibem ou atrasam a adoção de DevOps. A mudança de determinados processos leva as equipas a adotarem conhecimentos para utilizar as ferramentas e compreender os seus procedimentos. A falta de comunicação é considerada uma barreira porque, antes de se proceder à adoção do DevOps, as equipas trabalham sob a forma de silos. A adoção da metodologia DevOps possibilita uma grande melhoria a nível da comunicação, dando lugar a um ambiente mais colaborativo. Também deve existir formação para as equipas envolvidas, de forma a clarificar a utilização dos processos e ferramentas implementadas. Como já referido na análise de resultados, a equipa de desenvolvimento apenas teve formação com base em *self- learning* e criação de ambientes para testar o que foi aprendido. No entanto, a falta de formação certificada pode levar ao aumento do tempo de implementação e à não utilização total das ferramentas implementadas.

A empresa conseguiu quebrar a maioria das barreiras mesmo antes da adoção desta cultura, tudo através de uma boa visão e decisões estratégicas que culminam com a correta adoção de tecnologias de forma a habilitar as práticas de DevOps. A adoção da metodologia DevOps permitiu verificar o antes e depois da sua adoção dentro da empresa. Desta forma verificou-se um aumento generalizado da qualidade e transparência do serviço, uma redução dos tempos de manutenção e complexidade, uma melhoria da performance, acessibilidade e disponibilidade do sistema melhoradas e, não menos importante, a mitigação de certos riscos.

Como conclusão, é importante referir que não existe uma definição específica de DevOps, pois cada empresa adota uma cultura diferente com base no seu departamento de atividade e/ou nas práticas que melhor a beneficiam. As principais áreas de foco do DevOps são entrega, integração, automação e colaboração. O DevOps fornece às organizações uma abordagem inovadora no nível técnico, mas não há uma descrição geral de como implementá-lo. A mudança cultural da empresa e dos seus colaboradores também tende a ser difícil, mas no final, a colaboração entre as equipas e a melhoria da afinidade organizacional são geralmente reconhecidas.

Os objetivos foram maioritariamente cumpridos, mas, como se trata de um processo de melhoria contínua, ainda existe algum caminho a ser percorrido. Em geral, as pessoas envolvidas no projeto classificaram a adoção como relevante e positiva, sublinhando os benefícios conseguidos.

Como principais limitações à realização deste trabalho, destaca-se a dificuldade em comunicar com alguns dos elementos das equipas de desenvolvimento e alguns problemas na interpretação dos objetivos que foram inicialmente propostos, o que levou à falta de desenvolvimento de algumas das componentes da metodologia DevOps. Como pontos positivos, é de salientar a revisão da literatura, onde foram abordados todos os pontos importantes da metodologia DevOps e as práticas aplicadas no projeto NTX. Existem sempre muitas diversidades, sejam elas na implementação de uma nova metodologia ou no desenvolvimento de um determinado *software,* mas há sempre possibilidade de contornar as adversidades. Este projeto foi o exemplo disso.

7.2 Proposta de trabalho futuro

Em termos de recomendações para trabalho futuro, seria interessante iniciar novamente a adoção da metodologia DevOps desde o ponto de partida, ou seja, a fase de levantamento de requisitos e aplicar num novo projeto construido de raíz, de forma a passar por todo o ciclo de vida da aplicação e pelas fases do DevOps, de forma a refinar os resultados e comparações da pesquisa realizada. Além disso, existem também outros pontos que poderão ser explorados:

- Utilização de testes de carga, por exemplo, a ferramenta *Apache JMeter*
- Automatização do processo de publicação de evidências de revisão de código e testes na ferramenta *Confluence*.
- Automatização do processo de monitorização;
- Repetição desta aplicação prática em outras organizações de forma a comparar os resultados obtidos
- Repetição desta aplicação prática em outras organizações de forma a comparar os resultados obtidos

Referências bibliográficas

Alice Njenga (2021) Test automation and DevOps: What You Need to Know. Available at: https://www.cprime.com/resources/blog/test-automation-and-devops-what-you-need-to-know/ (Accessed: 18 Jan 2022).

Alsolamy, A. A., Khan, U. A., & Khan, P. M. (2014). IT-business alignment strategy for business growth. *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*.

Andy Lipnitski (2021) DevOps Implementation: Where to Start And How to Make It Result in Success? Available at: https://www.scnsoft.com/blog/devops-implementation-guide (Accessed: 20 Jan 2022).

Atlassian (2022) Criando uma cultura DevOps. Available at: https://www.atlassian.com/br/team-playbook/examples/devops-culture (Accessed: 18 Jan 2022).

Atlassian (2022) What is DevOps? Available at: https://www.atlassian.com/br/devops/what-is-devops (Accessed: 15 Jan 2022).

Beller, M., Gousios, G., & Zaidman, A. (2017). Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with *Git*Hub. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*.

Boehm, B., & Turner, R. (2005). *Management* Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software*.

Callanan, M., & Spillane, A. (2016). DevOps: Making It Easy to Do the Right Thing. *IEEE Software*.

Condo, C., Mines, D. L., Seguin, B., Homan, A., & Neuburg, S. (2017). Use DevOps And Supply Chain Principles To Automate Application Delivery Governance.

Datta, Ganesh (2022) Why MTTR is a Vital Metric for DevOps Teams. Available at: https://devops.com/why-mttr-is-a-vital-metric-for-devops-teams/ (Accessed: 30 Jan 2022).

Devmedia (2022) Qualidade de *software*. Available at: https://www.devmedia.com.br/qualidade-de-*software*/9408 (Accessed: 30 Jan 2022).

Ebert, C., Gallardo, G., Hernates, J., & Serrano, N. (2016). *DevOps.* IEEE *Software*.

Elberzhager, F., Naab, M., Arif, T., & Süß, I. (2017). From Agile Development to DevOps: Going To*war*ds Faster Releases at High Quality – Experiences from an Industrial Context.

Erich, F., Amrit, C., & Daneva, M. (2014). *Report: DevOps Literature Review.* Twente.

Fazal-Baqaie, M., Güldali, B., & Oberthür, S. (2017). To*war*ds DevOps in Multi-provider Projects. *2nd Workshop on Continuous Software Engineering*.

Fitzgerald, B., & Stol, K.-J. (2015). Continuous *Software* Engineering: A Roadmap and Agenda. *Journal of Systems and Software*.

Gruver, G., & Mouser, T. (2015). *Gruver & Mouser 2015Leading the Transformation: Applying Agile and DevOps Principles at Scale.* IT Revolution Press.

Hering, M. (2018). *DevOps for the Modern Enterprise: Winning Practices to Transform Legacy IT Organizations.* Oregon, Estados Unidos da América: It Revolution Press.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional.

Ikonen, M., Oza, N. V., Kettunen, P., & Abrahamson, P. (2010). Exploring the Sources of Waste in Kanban *Software* Development Projects. *Conference Proceedings of the EUROMICRO*.

ISEC (2022) Instituto Superior de Engenharia de Coimbra. Available at: https://www.isec.pt/pt/instituto/#lnkApresentacao (Accessed: 15 Jan 2022).

Kim, G., Debois, P., Humble, J., & Willes, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution Press.

Noesis (2022) Quem Somos. Available at: https://www.noesis.pt/pt/about-noesis/who-we-are (Accessed: 23 Jan 2022).

Noesis (2022) Os Nossos Clientes. Available at: https://www.noesis.pt/pt/about-noesis/clients (Accessed: 23 Jan 2022).

Park, H. Y., Jung, S. H., Lee, Y.-j., & Jang, K. C. (2006). The effect of improving IT standard in IT governance. *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*.

Petersen, K. (2011). What is MTTD?

Pure Storage (2022). s Lean Agile and Agile Lean? Available at: https://www.purestorage.com/br/knowledge/what-is-mttd.html (Accessed: 01 Feb 2022).

Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). *DevOps for Digital Leaders: Reignite Business with a Modern DevOps-Enabled Software Factory.* Apress.

Sarker, I. H., Faruque, F., Hossen, U., & Rahman, A. R. (2015). A Survey of *Software* Development Process Models in *Software* Engineering. *International Journal of Software Engineering and Its Applications*.

Sharma, S. (2013). *DevOps for Dummies.*

Sharma, S. (2017). *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise.* Wiley.

Soni, M. (2015). End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery. *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*.

Virmani, M. (2015). Understanding DevOps & bridging the gap from Continuous Integration to Continuous Delivery. *5th Int. Conf. Innov. Comput. Technol. INTECH 2015*.

Wikipedia Apache Maven (2022) Apache Maven Available at: https://pt.wikipedia.org/wiki/Apache_Maven (Accessed: 17 Jan 2022).

Wikipedia (2022) *Docker* (*software*). Available at: https://pt.wikipedia.org/wiki/*Docker*_(*software*) (Accessed: 15 Jan 2022).

Wikipedia (2022) Instituto Superior de Engenharia de Coimbra. Available at: https://pt.wikipedia.org/wiki/Instituto_Superior_de_Engenharia_de_Coimbra (Accessed: 15 Jan 2022).

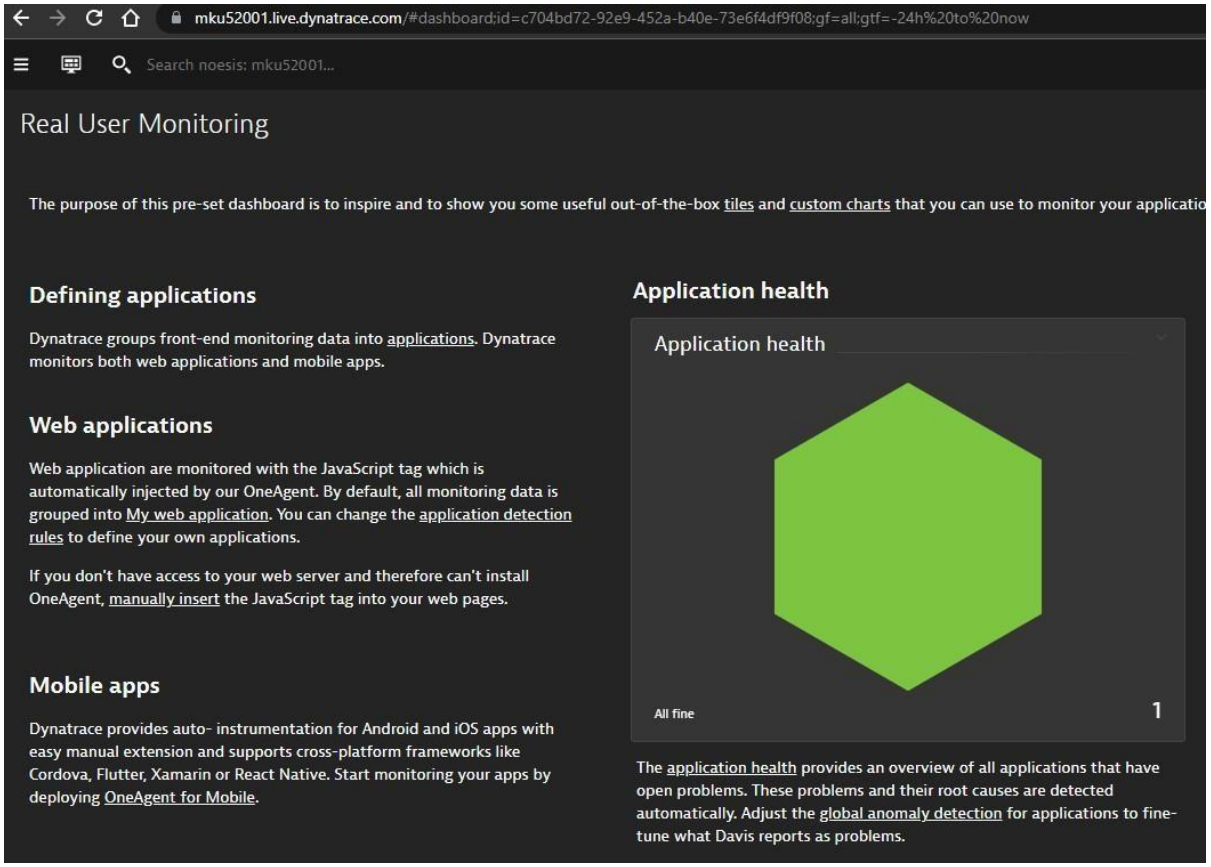
Wikipedia (2022) *Jira*. Available at: https://pt.wikipedia.org/wiki/*Jira* (Accessed: 15 Jan 2022).

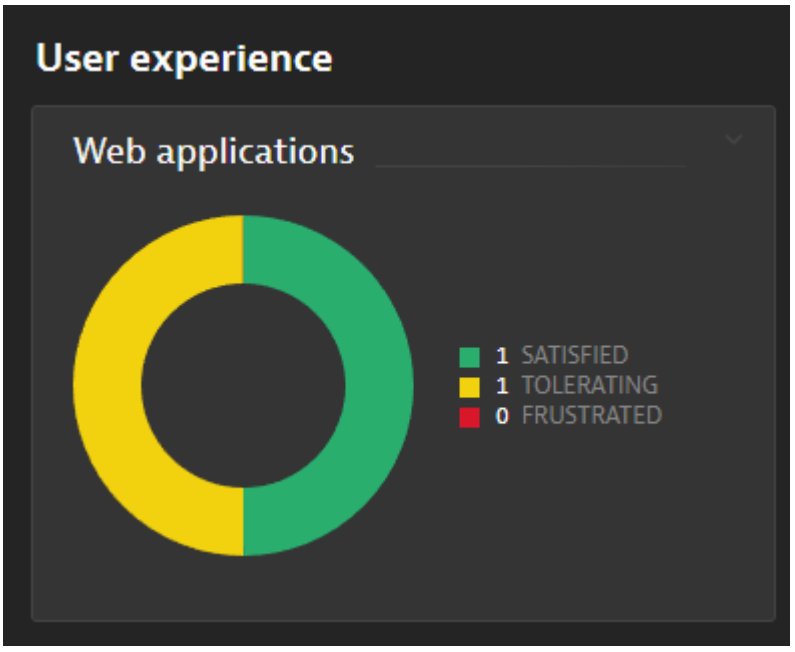
Wikipedia (2022) *Jenkins* (*software*). Available at: https://en.wikipedia.org/wiki/*Jenkins*_(*software*) (Accessed: 18 Jan 2022).

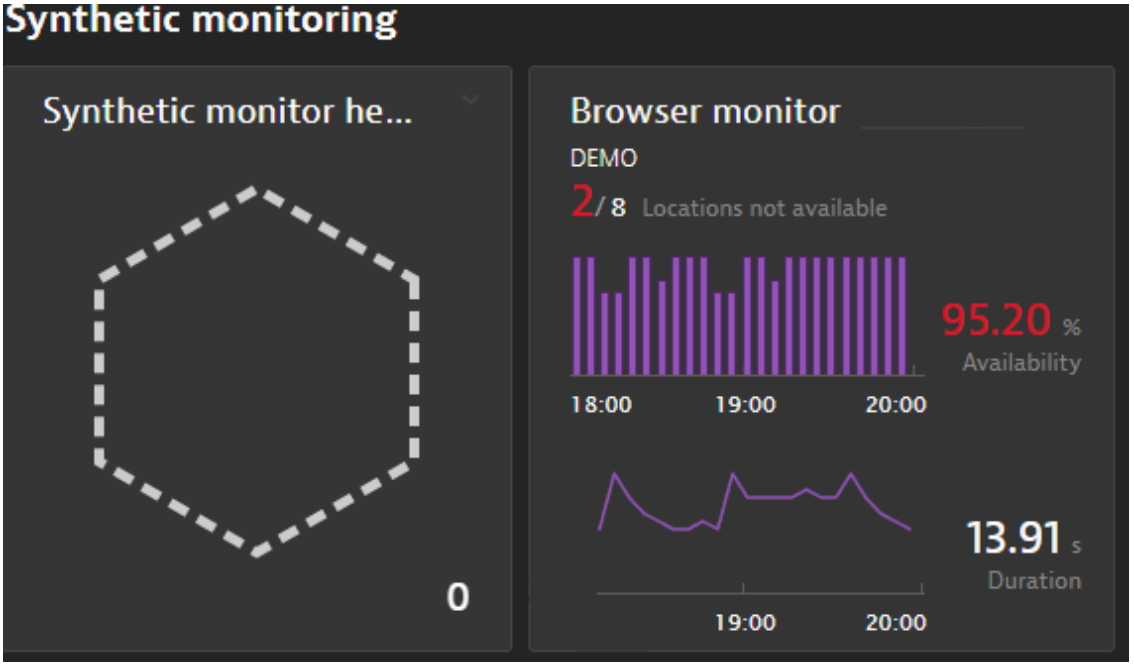
Wikipedia (2022) *Confluence* (*software*). Available at: https://en.wikipedia.org/wiki/Confluence_(software)*software*) (Accessed: 18 Jan 2022).

Apêndices

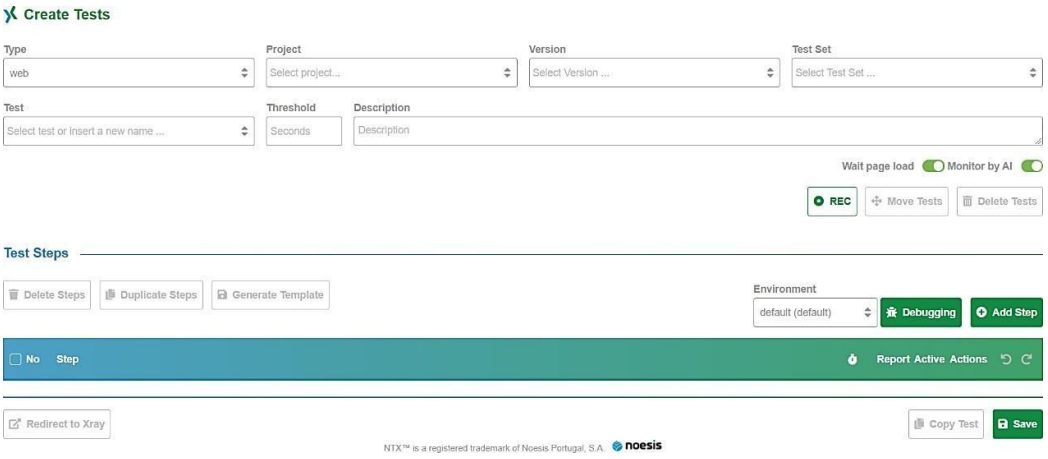
Apêndice A – Dynatrace

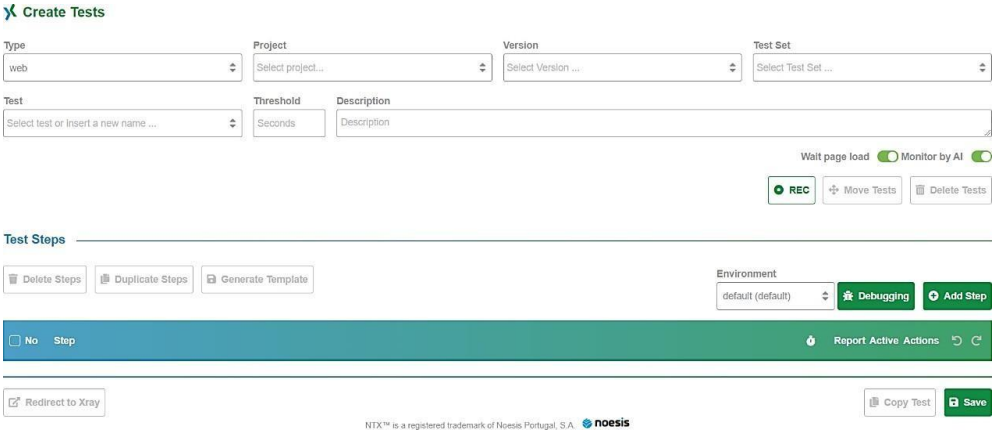


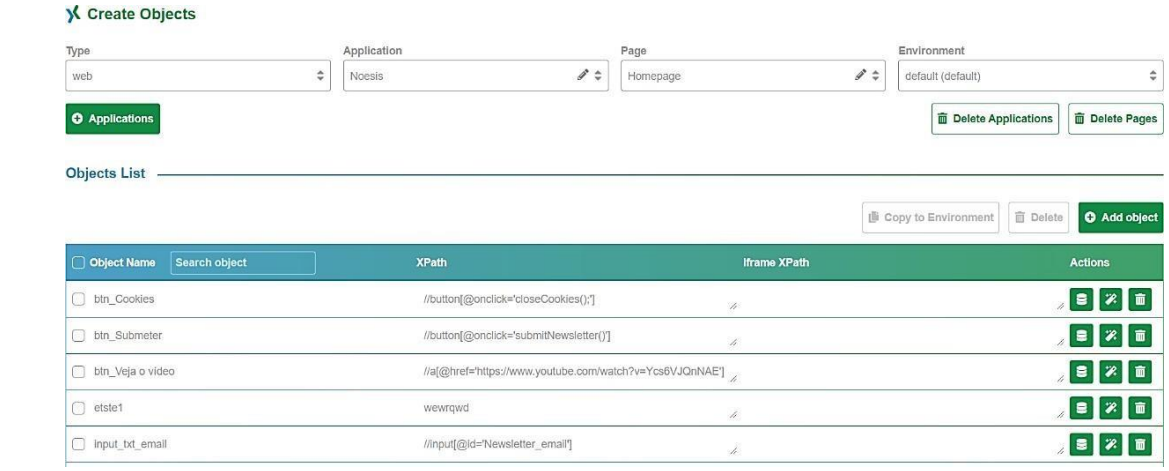


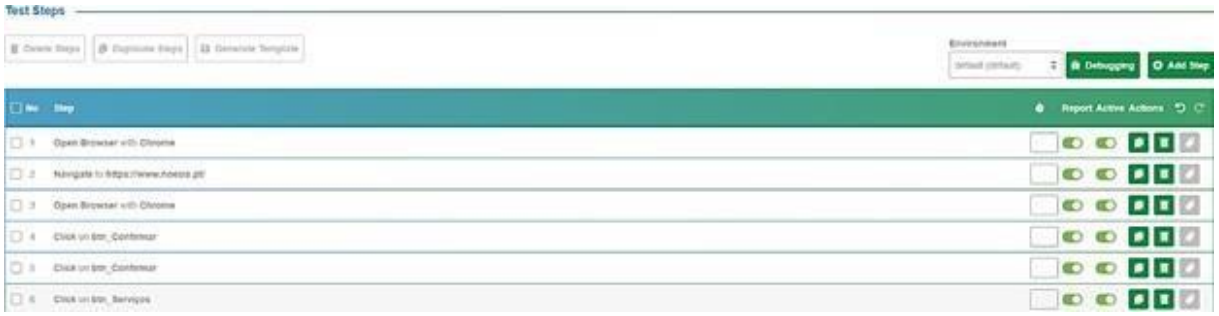

Synthetic monitoring
Synthetic monitor he...
0
Browser monitor
DEMO
2 / 8 Locations not available
95.20 %
Availability
18:00
19:00
20:00
13.91 s
Duration
19:00
20:00

Apêndice B – NTX após implementação de DevOps

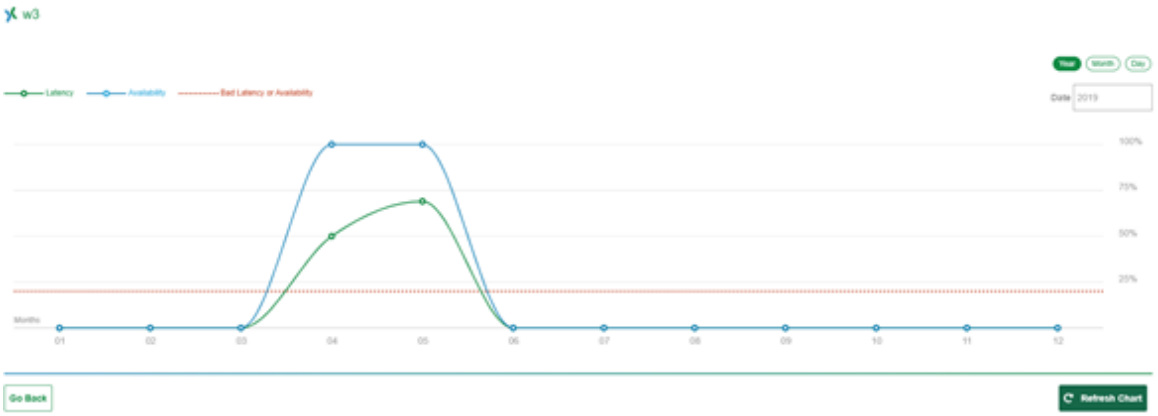





Create Objects
Type
web
Application
Noesis
Page
Homepage
Environment
default (default)
Applications
Delete Applications
Delete Pages
Objects List
Copy to Environment
Delete
Add object
Object Name
Search object
XPath
Iframe XPath
Actions
btn_Cookies
//button[@onclick='closeCookies();']
btn_Submeter
//button[@onclick='submitNewsletter()']
btn_Veja o video
//a[@href='https://www.youtube.com/watch?v=Ycs6VJQnNAE']
etste1
wewrqwd
input_txt_email
//input[@id='Newsletter_email']


Test Steps


Execution's Performance


w3
Latency
Availability
Bad Latency or Availability
Year
Month
Day
Date
2019
100%
75%
50%
25%
Months
01
02
03
04
05
06
07
08
09
10
11
12
Go Back
Refresh Chart

Apêndice C – Nível de satisfação do cliente

Versão NTX	Recomenda o NTX?	Feedback geral
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.3.2000	Neutro	Neutro
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.3.2000	Sim	Positivo
2.4.2000	Sim	Positivo
2.4.2000	Sim	Positivo
2.4.2000	Neutro	Neutro
2.4.2000	Neutro	Neutro
2.4.2000	Sim	Positivo
2.4.2000	Neutro	Neutro
2.4.2000	Sim	Positivo
2.4.2000	Sim	Positivo
2.4.2000	Sim	Positivo
2.4.2000	Sim	Positivo
2.4.2000	Sim	Positivo

Apêndice D – Reparações em produção

PRE-DEVOPS

FIX(url)	Time(h)
dev.azure	1
dev.azure	1
dev.azure	1
dev.azure	1.5
dev.azure	1
dev.azure	0.5
dev.azure	0.5
dev.azure	1.5
dev.azure	1.5
dev.azure	1.5
dev.azure	1.5
dev.azure	1.5
dev.azure	1.5
dev.azure	1.5
dev.azure	3
dev.azure	2
dev.azure	2,5
dev.azure	2,5

POS-DEVOPS

Problema (Jira Task)	Tempo de reparação (h)
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	3
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	2
https://ntxdevops.atlassia	4
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	0.5
https://ntxdevops.atlassia	1
https://ntxdevops.atlassia	0.5

Apêndice E – Entregas de software

Entrega (versão)	Tempo de execução Jenkins(h)
2.3.2000	0.6
2.3.2000	0.6
2.3.2000	0.6
2.3.2000	0.6
2.3.2000	0.6
2.4.2000	0.5
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.6
2.4.2000	0.7
2.4.2000	0.7
2.4.2000	0.7
2.4.2000	0.7
2.4.2000	0.7
2.4.2000	0.6
2.4.2000	0.6

Apêndice F – pomExecuteTestNavegar_Menu_Teste.xml

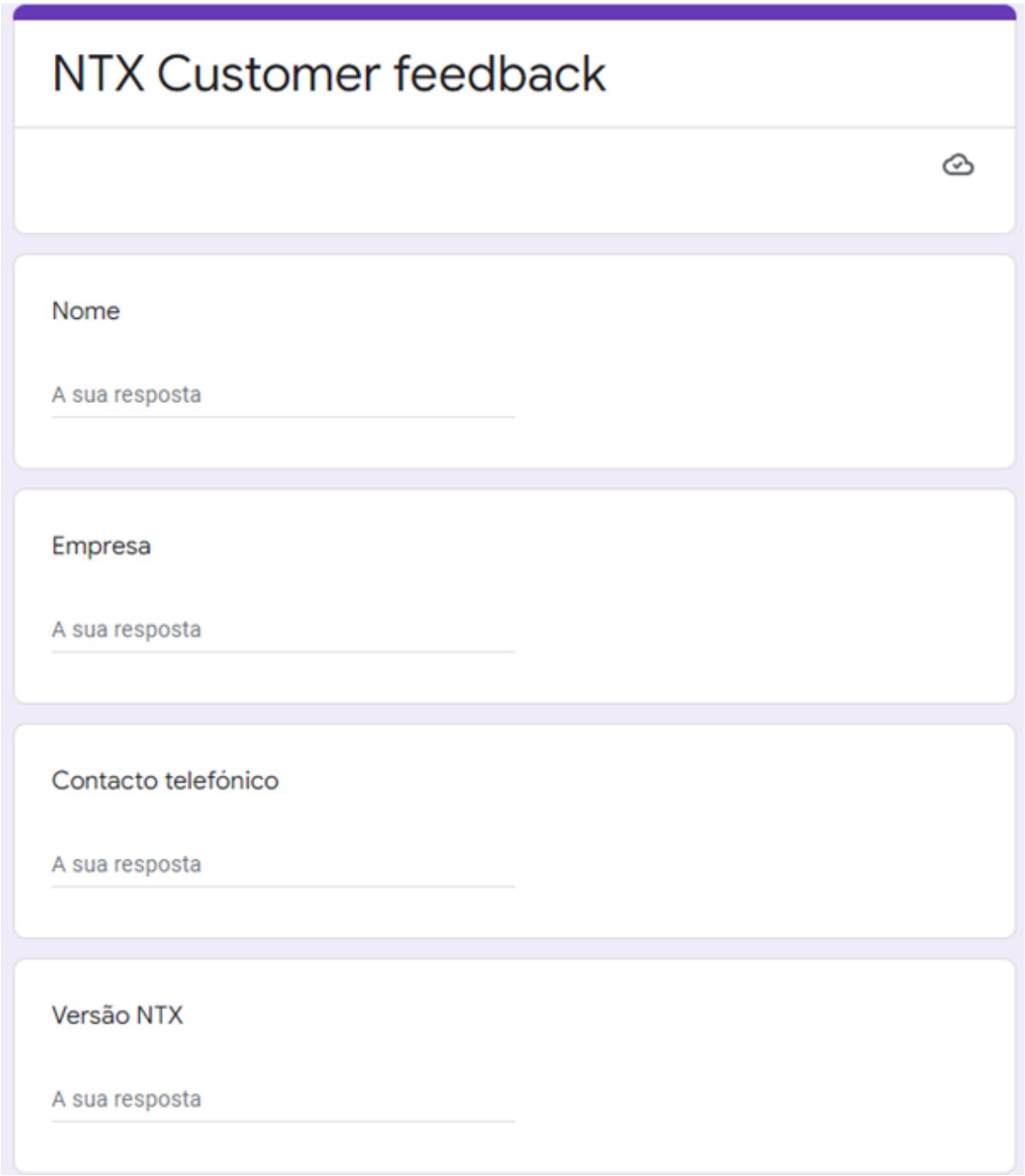
pomExecuteTestNavegar_Menu_Teste.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>NTX_Test_call</groupId>
    <artifactId>Demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <jre.level>1.7</jre.level>
        <jdk.level>1.7</jdk.level>
    </properties>

    <build>
        <plugins>
            <!-- Compiler plug-in -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>${jdk.level}</source>
                    <target>${jdk.level}</target>
                </configuration>
            </plugin>
            <!-- Below plug-in is used to execute tests -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.18.1</version>
                <configuration>
                    <suiteXmlFiles>
                        <!-- TestNG suite XML files -->
                        <suiteXmlFile>navegar_menu_teste.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <!-- Include the following dependencies -->
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.45.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
```

pomExecuteTestNavegar_Menu_Teste.xml

```
    <build>
        <plugins>
            <!-- Compiler plug-in -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>${jdk.level}</source>
                    <target>${jdk.level}</target>
                </configuration>
            </plugin>
            <!-- Below plug-in is used to execute tests -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.18.1</version>
                <configuration>
                    <suiteXmlFiles>
                        <!-- TestNG suite XML files -->
                        <suiteXmlFile>navegar_menu_teste.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <!-- Include the following dependencies -->
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.45.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.8.8</version>
        </dependency>
    </dependencies>

</project>
```

Apêndice G – Formulário de *feedback*



Recomenda o NTX?

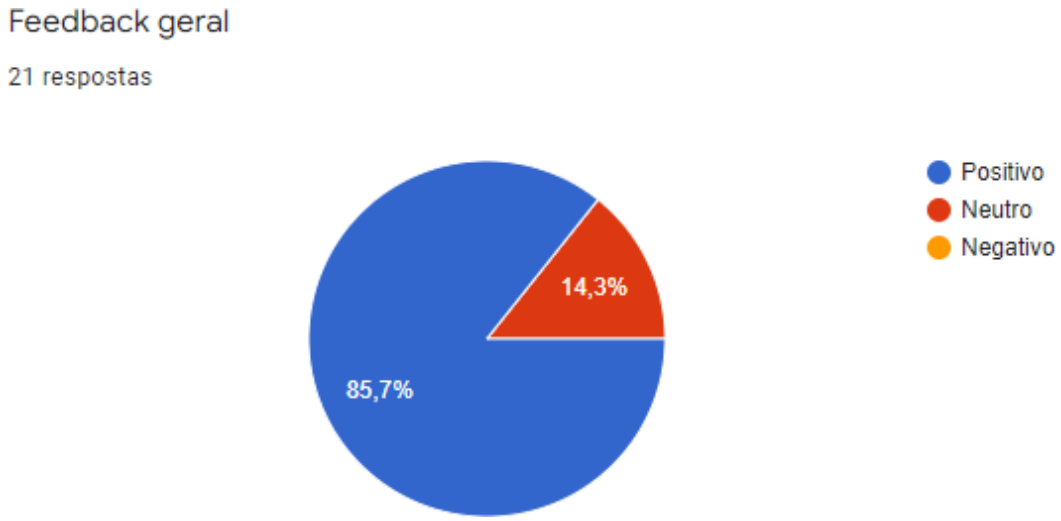
- ◯ Não
- ◯ Neutro
- ◯ Sim

Feedback geral

- ◯ Positivo
- ◯ Neutro
- ◯ Negativo

Sugestões

A sua resposta


Feedback geral
21 respostas
Positivo
Neutro
Negativo
14,3%
85,7%