Universidade Federal de Pernambuco
Centro de Informática

*Graduação em Sistemas de Informação*

---

# Automatização na escolha de ferramentas em projetos DevOps

José Eudes de Souza Júnior

Trabalho de Graduação

Recife, Julho de 2017

Universidade Federal de Pernambuco
Centro de Informática

José Eudes de Souza Júnior

# Automatização na escolha de ferramentas em projetos DevOps

*Trabalho apresentado ao programa de GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO como requisito parcial para obtenção do grau de bacharel em SISTEMAS DE INFORMAÇÃO.*

Orientador: Prof. Dr. Vinícius Cardoso Garcia

Recife, Julho de 2017

Universidade Federal de Pernambuco

Centro de Informática

José Eudes de Souza Júnior

# Automatização na escolha de ferramentas em projetos DevOps

Aprovado em ___/___/___

BANCA EXAMINADORA

_____

Vinícius Cardoso Garcia

*Universidade Federal de Pernambuco*

_____

Kiev Santos da Gama

*Universidade Federal de Pernambuco*

Recife, Julho de 2017

# Agradecimentos

Primeiramente, agradeço a Deus, pois ele é a causa primeira e sem a sua misericórdia, eu jamais estaria escrevendo este texto. Deus colocou no meu caminho muitas pessoas especiais, que me ajudaram nessa árdua caminhada e quero agradecer a algumas delas.

Quero agradecer àqueles que me são muito caros e que sem eles eu não conseguiria fazer o curso de Sistemas de Informação. Agradeço a Carina Souza, minha amada esposa, que teve muita paciência comigo durante essa caminhada, pois sempre me incentivou e me deu forças para que eu não desistisse. Outra pessoa muito importante é a Aparecida Araújo, minha mãe, que me criou com muito zelo e amor, possibilitando que eu chegasse até aqui. Quero agradecer aos meus sogros Luiz Francisco da Mota e Caselinda Maria Silva da Mota, que me deram uma força muito grande, além de sempre acreditarem em mim. Com certeza, sem essas pessoas eu não teria chegado até aqui.

Quero a agradecer aos professores do Colégio Santa Bárbara, principalmente Jorge e Lêucio, que acreditaram muito em mim quando eu me julguei incapaz de passar no vestibular.

Quero agradecer aos professores do Centro de Informática da UFPE que me deram a oportunidade de conhecer essa área tão incrível que é a computação. Em especial quero citar os nomes de Vinícius Garcia, meu orientador, Kiev Gama e Carla Silva, estes professores se tornaram exemplos para mim.

Quero agradecer aos colegas do Cin, que durante a graduação fizeram parte da minha trajetória. Em particular Paulo Viana, que se tornou um grande amigo.

Quero também agradecer a todos os meus colegas de trabalho na Ustore, que sempre me impulsionam a ser um profissional melhor. Em particular, quero citar meu colega de trabalho e amigo Allan Monteiro, que me ajudou muito com este trabalho, revisando o texto.

Por fim, quero agradecer a Fernando Carvalho (Fish), que me ajudou na revisão do texto e no entendimento do trabalho. Também me ensinou muito no tempo em que trabalhamos juntos, na Ustore.

# Resumo

Após a criação do Manifesto Ágil e a popularização das metodologias ágeis no desenvolvimento de *software*, muitas formas de otimizar ainda mais os processos do desenvolvimento foram estudadas. Destes estudos, destacam-se as tentativas de se inserir a ideologia *Lean* no desenvolvimento de *software*. Em 2003, um estudo tornou possível esta adaptação, dando a Engenharia de *Software* mais uma meta importante, a saber, a eliminação do desperdício, a reação ainda mais rápida às mudanças nos requisitos e um maior aumento na qualidade, dando origem as práticas contínuas da Engenharia de *Software*. Nestas práticas, se destacam a integração, entrega e implantação contínuas. Baseando-se nestas práticas, uma tentativa de aproximar as equipes de desenvolvimento de *software* e as equipes de operações das empresas, surgiu o *DevOps*, que além de promover o alinhamento destas equipes ainda se utiliza de todas as práticas contínuas, de forma que é possível contemplar todos os setores da organização com esta cultura. Para que as práticas contínuas citadas sejam realizadas, é necessário que todas as etapas do *pipeline* sejam automatizadas. Com esse objetivo, muitas ferramentas são criadas, através de empresas que veem na automação de processos um mercado lucrativo ou da comunidade *DevOps*, que é muito ativa e desenvolve constantemente novas ferramentas. Com essa grande quantidade de ferramentas, surge a dúvida de qual é adequada, ao se criar um projeto novo ou na iniciativa de migração de um projeto existente para as práticas do *DevOps*.

Palavras-chave: *DevOps*, *Pipeline* automatizado, Integração Contínua, Entrega Contínua, Implantação Contínua, Ferramentas, Engenharia de *Software* Contínua, Ideologia *Lean*.

# Abstract

After the creation of the Agile Manifesto and the popularization of agile methodologies in software development, many ways to further optimize the processes of development have been studied. These studies, noteworthy are the attempts to insert the ideology of Lean in software development. In 2003, a study made possible this adaptation, giving the Software Engineer and more an important goal, namely, the elimination of waste, the reaction still faster to changes in requirements and a greater increase in the quality, giving rise to the practice of continuous Software Engineering. These practices include the integration, delivery and deployment, continuous. On the basis of these practices, an attempt of approaching the software development teams and the operations teams of the companies, it emerged in the DevOps, that in addition to promoting the alignment of these teams still use all the practices continued, so that it is possible to cover all sectors of the organization with this culture. For that the practices of continuous mentioned are carried out, it is necessary that all the stages of the pipeline are automated. With this goal, many tools are created through companies that you see on the automation of processes to a lucrative market or community DevOps, which is very active and constantly develops new tools. With such a large amount of tools, comes the question of what is appropriate, if you create a new project or on the initiative of the migration of an existing project to the practices of DevOps.

Keywords: DevOps, Pipeline, automated, Continuous Integration, Continuous Delivery, Continuous Deployment, Tools, Software Engineering Continuous, Lean Thinking.

# Lista de abreviaturas, siglas e símbolos

| | |
|---|---|
| ES | Engenharia de Software |
| XP | Extreme Programing |
| IC | Integração Contínua |
| EC | Entrega Contínua |
| DC | Deploy Contínuo |
| DevOps | Desenvolvimento e Operações |
| SLA | Acordo de Nível de Serviço |
| VCS | Sistema de Controle de Versão |
| UI | Interface do Usuário |

# Índice de imagens

# Sumário

# 1. Introdução

## 1.1 Contexto

Em 2017, a Engenharia de *Software* (ES) está completando 49 anos de existência. A busca por métodos de engenharia para o desenvolvimento de *software* foi proposta em 1968 na conferência da OTAN [13, 16]. A ES foi criada na tentativa de resolver os problemas que levaram a falta de eficiência generalizada na entrega de grandes sistemas, envolvendo atrasos, ausência de funcionalidades e custos que iam muito além do esperado [16]. Ainda seguindo o raciocínio de [16], como o *software* é intangível, logo, ele não tem limites para evoluir, entretanto, de todo esse potencial emerge a complexidade que envolve os sistemas de *software*. Ao se tornarem complexos, além dos problemas citados anteriormente, os sistemas se tornaram difíceis de entender e muito caros para serem modificados. Como consequência disso, os grandes projetos de *software* passaram a ser encarados com grande desconfiança.

A ES então passou a dar suporte a criação de produtos de *software* criados por equipes que eram responsáveis por alterar e manter o *software* durante todo o seu ciclo de vida. A ES trouxe consigo também, um conjunto de técnicas destinadas a apoiar a especificação, projeto e evolução dos sistemas de *software*. *Software*, a partir desse momento, não era mais apenas um programa de computador ou meramente um sistema, o termo passou a agregar a documentação associada e os dados de configurações, que eram necessários para tornar os sistemas operáveis. Os documentos e dados poderiam ser arquivos de configuração, documentação do sistema, documentação da estrutura, manuais, documentos de arquitetura e documentações do usuário [16].

Para dar suporte a emergente ES, foram criadas as Metodologias de Desenvolvimento (ou Processos), que são conjuntos de atividades com resultados associados que auxiliam a produção de *software, onde* resultado final é o produto [16, 17]. As primeiras metodologias foram criadas utilizando modelos tradicionais da Engenharia de Sistemas [16].

As Metodologias Tradicionais foram criadas para o contexto da época visando os *mainframes* e os terminais burros [18], entretanto com o avanço da tecnologia, surgiram alternativas de menor custo, o que ajudou a acelerar o surgimento da globalização, que por sua vez, abriu novos mercados, gerando uma concorrência a nível mundial. Nesse momento, a competitividade das empresas aumentou severamente, tornando os modelos tradicionais incompatíveis, devido a grande demanda por produtos de *software* [2, 16]. As Metodologias Tradicionais passaram a ser encaradas como pesadas e orientadas a documentação [2, 16, 17, 24] e na década de 1980, muitos pesquisadores e desenvolvedores publicaram artigos com o objetivo de divulgar dados que informavam sobre os problemas que ocorriam com o uso dessas metodologias no desenvolvimento de *software* [16, 17]. As mudanças constantes nos projetos e as incertezas nos requisitos converteram o desenvolvimento de *software* em uma atividade muito mais dinâmica, inviabilizando o levantamento dos requisitos de forma estável, como era exigido nos modelos tradicionais, atrasando o cronograma dos projetos ao postergar a entrega para muito além do prazo acordado inicialmente [16]. Esta falta de agilidade dos projetos utilizando métodos tradicionais se tornou séria, de modo que alguns projetos ao chegarem na sua fase final com o *software* já disponível para uso, o mesmo já havia se tornado inútil devido as mudanças radicais em requisitos, regras de negócio e etc [16].

Em 1995, o *Chaos Report* [19] divulgou um preocupante conjunto de dados tendo como base 8380 projetos. Estes dados mostravam que apenas 16,2% dos projetos foram entregues com tempo, custos e requisitos de acordo com o esperado. 31% destes projetos, foram cancelados mesmo antes de estarem finalizados e 52,7% deles foram entregues com algumas das variáveis citadas anteriormente fora do limite acordado, ou seja, foram entregues fora do prazo, com o custo acima do acordado ou as funcionalidades não correspondiam as expectativas dos clientes. Os dados ainda mostravam que entre os projetos que foram entregues com problemas, a média dos atrasos foi de 222% do tempo, a de custo foi de 189% e a média de funcionalidades entregues conforme o acordado foi de 61% [17, 19].

Na década de 1990, a insatisfação com os métodos tradicionais aumentou ainda mais. Isso porque esses projetos orientados a planos não atendiam a maioria dos projetos de *software*, pois se gastava mais tempo com análises e documen-

tações do que com desenvolvimento e testes [16]. Com isso os desenvolvedores propuseram a criação das Metodologias Ágeis de Desenvolvimento de *Software*. Segundo os desenvolvedores, essas metodologias atendiam melhor o contexto dos projetos de *software* e a grande demanda, pois são baseadas em entregas incrementais, sendo mais flexíveis no momento em que os requisitos mudam [16, 17]. Finalmente, em 2001 foi criada a Aliança Ágil, a partir do encontro de 17 especialistas em processos de *software* que representavam as metodologias Scrum [23], *Extreme Programing* (XP) [22] e outras. Neste encontro foram estabelecidos princípios comuns entre as metodologias, dando origem ao Manifesto Ágil [20, 21].

A partir desse momento, as metodologias ágeis se tornaram muito populares [1, 24], passando a ser largamente adotadas até os dias de hoje. Isso se deve às promessas na velocidade de entrega, na capacidade de se adaptar as mudanças, além do aumento na qualidade e maior satisfação dos clientes [2, 4, 9, 12, 16, 17]. Grandes empresas do ramo da tecnologia como Amazon, Facebook e Google utilizam as metodologias ágeis [2].

Apesar de atender os projetos de *software* modernos muito melhor que as metodologias tradicionais, as metodologias ágeis ainda desperdiçam recursos valiosos ao não incentivar uma sinergia entre os times de qualidade, operações e desenvolvimento. Estudos demonstram que uma colaboração mais aperfeiçoada entre estas equipes, possibilita melhorias nos processos de desenvolvimento e operações, tornando a organização mais competitiva [4, 13].

Nos últimos anos, estamos vivenciando mais uma revolução nos processos de desenvolvimento de *software*, que está sendo chamada de era Pós-Ágil [4]. A necessidade da equipe de desenvolvimento obter maior frequência nos *feedbacks* dos clientes e de entregar frequentemente o *software* no ambiente de produção, torna possível, dependendo do domínio da aplicação, a realização de várias implantações por dia, o que não é viável em uma abordagem ágil, pois apesar de curtas, as iterações duram um certo tempo para serem realizadas [4]. É nesse contexto que surge o conceito de Implantação Contínua (DC[1]) [4].

O DC tem suas raízes nos conceitos da ideologia *Lean* [2, 4, 6, 9], que é originária da fabricação, sendo inserida no contexto da ES pela primeira vez no *Lean*

[1] Para fins didáticos, este trabalho utilizará a sigla DC, misturando o os termos em inglês e em português: Deploy Contínuo.

*Software Development* [4]. Esta abordagem tem a Integração Contínua (IC) e a Entrega Contínua (EC) como pré-requisitos para ser alcançada, além de necessitar de um *pipeline* de desenvolvimento automatizado e uma ampla infraestrutura acompanhada de processos e práticas compatíveis [4, 6]. Uma outra abordagem que utiliza largamente os métodos contínuos de desenvolvimento é o *DevOps* [4, 5, 6, 7, 9, 11], que enfatiza a colaboração e a comunicação entre as equipes de desenvolvimento e as de operações, sob a premissa que a cooperação das duas áreas resulta em um rápido desenvolvimento contínuo [4, 5, 11, 18]. Já existem muitas empresas que utilizam práticas contínuas, incluindo o DC [2, 6, 10]. Trabalhos como [2, 4, 6, 11], mostram que a implantação frequente ajuda na criação e desenvolvimento de produtos melhores por serem mais reativos.

Para que tudo isso seja possível, uma mudança na cultura organizacional e uma quantidade considerável de ferramentas de apoio são requeridas [2, 4, 8, 11]. Para que o *pipeline* de desenvolvimento seja automatizado até que a implantação seja feita no ambiente de produção, a empresa deve configurar uma cadeia de ferramentas autoassociadas e estabelecer fluxos de trabalho redirecionados à estas ferramentas [2, 11].

## 1.2 Problema

No mundo *DevOps*, vários tipos de ferramentas são utilizados para garantir a entrega rápida e com qualidade do *software*. As ferramentas devem suportar uma boa parte do *pipeline* de desenvolvimento, podendo chegar até a sua totalidade, isso vai depender de vários fatores que envolvem cultura organizacional e limitações técnicas das equipes e ferramentas, além do tipo de natureza do produto [25].

Um estudo mostra várias observações feitas por usuários do *DevOps* [15]. Estas observações foram colhidas através de oficinas de *design* e entrevistas que posteriormente alimentaram um banco de dados, onde foi realizada uma análise de sentimentos. Um dos problemas mais evidentes identificados pelos participantes é a escolha de ferramentas para a automação dos processos.

## 1.3 Objetivos

Este trabalho tem como objetivo geral o mapeamento de ferramentas e a criação de uma ferramenta conceitual, para auxiliar na escolha das ferramentas que serão utilizadas em projetos *DevOps.* A ferramenta de escolha deve dar apoio a projetos que utilizem as práticas de IC, EC ou DC, informando as principais ferramentas, disponíveis nos melhores trabalhos da literatura atual, para todo o *pipeline* de desenvolvimento.

Há destaque também para alguns objetivos específicos importantes:

1. Informar quais as diferenças entre as práticas contínuas e as metodologias ágeis;
2. Verificar até que ponto as práticas contínuas são benéficas para os projetos de *software*;
3. Analisar as ferramentas mais utilizadas em projetos reais, através de estudos de caso disponíveis na literatura;
4. Compreender como as ferramentas são classificadas e organizadas para cada etapa do *pipeline* de desenvolvimento.

## 1.4 Escopo

Este trabalho se limita à criação de uma ferramenta conceitual que informa quais são as principais ferramentas que envolvam as práticas contínuas descritas na seção 1.3, cobrindo todas as áreas do *pipeline* exibido em [10]. O *pipeline* será explicado na seção 2.2.2.3.1. O estudo e mapeamento das ferramentas foi suportado por dois trabalhos com um viés prático sobre o assunto, um deles [2] demonstra empiricamente quais as ferramentas mais usadas pelas equipes *DevOps* através de entrevistas realizadas em 18 empresas finlandesas que estão comprometidas em realizar entregas rápidas com maior valor para o cliente. O outro trabalho [26], fornece uma revisão sistemática da literatura contendo 69 estudos primários publicados de 2004 à 2012, onde a segunda pergunta que motivou a revisão está relacionada ao uso de ferramentas que automatizam o *pipeline* de implantação.

Nestes trabalhos, as ferramentas mapeadas são suficientes para automatizar os *pipelines* das diversas empresas e projetos que foram estudados [2, 10], no entanto, vale ressaltar que as ferramentas não cobrem a totalidade da grande maioria dos casos e o mapeamento contido neste trabalho não é exaustivo.

O foco deste trabalho não é criar um meio para resolver o problema da escolha de ferramentas, mas sim levantar questionamentos que incentivem a construção de:

1. Ferramentas que suportem completamente o *pipeline* dos diversos tipos de projetos de *software*;
2. Sistemas que auxiliem as empresas, informando as boas práticas e façam o mapeamento das diversas ferramentas que automatizam o *pipeline* dos diversos tipos de projetos de *software*.

## 1.5 Metodologia

O método adotado para chegar ao objetivo é composto por uma pesquisa exploratória, se utilizando dos principais repositórios em busca de artigos, publicações, periódicos, revistas, dissertações, estudos de caso e revisões sistemáticas que tragam informações das principais pesquisas sobre práticas contínuas e cadeias de ferramentas que suportem estas práticas, a fim de trazer embasamento para a redação deste trabalho e possibilitar que os objetivos sejam alcançados.

Os repositórios de busca foram:

- IEEEXplore Digital Library [27];
- Science Direct [28];
- ACM [29];
- Scopus [30];
- Google Scholar [31].

Outra técnica utilizada na obtenção de trabalhos foi a leitura das referências dos trabalhos mais relevantes.

Cada trabalho relevante encontrado foi inserido na ferramenta Mendeley[1], um gerenciador e compartilhador de documentos de pesquisa. Para capturar o trabalho, um *plugin* desta ferramenta para o Google Chrome[2] foi utilizado.

Na leitura de cada trabalho, o parágrafo lido foi grifado e resumido para ajudar no mapeamento da informação.

Para modelar as ferramentas, foi utilizada a ferramenta Shapes[3]. Para criar a ferramenta, foi utilizada a ferramenta Visual Studio Code.

## 1.6 Organização do trabalho

Este trabalho está organizado da seguinte forma: O capítulo 1 dá uma visão histórica da ES até a chegada do *DevOps*, informando as reais necessidades das criações de novas formas de se construir *software*. O capítulo 2 dá um embasamento teórico para todos os conceitos apresentados. O capítulo 3 exibe o mapeamento das ferramentas e informa os dados coletados. O capítulo 4 fala da ferramenta criada e mostra um estudo de caso feito em um projeto real. Por fim, o capítulo 5 conclui o trabalho, dissertando sobre trabalhos futuros e as limitações deste.

[1] https://www.mendeley.com

[2] https://www.google.com/chrome/browser/desktop/index.html

[3] http://shapesapp.com/

# 2. Referencial Teórico

Este capítulo servirá de arcabouço teórico do trabalho, descrevendo os principais termos que são utilizados.

## 2.1 A ideologia *Lean*[1]

No fim da década de 1940, uma pequena empresa chamada Toyota decidiu fabricar carros no Japão [35]. Porém, verificou-se que os carros tinham que ser baratos, pois os estragos causados na segunda grande guerra devastou a economia do país e, consequentemente, as pessoas não tinham muito dinheiro [36].

Com o conhecimento que se tinha na época, a maneira de haver fabricação barata de automóveis era através de uma linha de produção em massa. O problema é que este tipo de produção não atendia o mercado japonês, pois era um mercado relativamente pequeno. Para resolver o problema de fabricar carros em pequena quantidade tão baratos quanto os produzidos em massa, surgiu o Sistema Toyota de Produção, Tendo como a cabeça por trás deste sistema, Taiichi Ohno[2][35].

Esta nova forma de desenvolver produtos tinha uma meta: A eliminação de desperdícios [1, 35]. A forma de encarar o desperdício foi modificada com a nova ideologia. Desperdício, a partir desse viés teve o seu significado ampliado, entre os vários tipos, é possível destacar:

- Superprodução (Recursos não desejados);
- Atividades que não são necessárias;
- Produto esperando que outras atividades terminem;
- Passos extras no processo de desenvolvimento;
- Defeitos.

O foco da ideologia era criar e entregar o produto imediatamente após o pedido, ou seja, o desenvolvimento deveria começar após a solicitação do cliente, porém a meta é entregar o produto rapidamente. Segundo a ideologia, quando um projeto começa, ele deve ser finalizado o mais rápido possível, isso porque o produ-

[1] Termo criado por Krafcik em seu trabalho intitulado “Triumph of the lean production system” no ano de 1988 [2].

[2] O Homem que ficou conhecido como o “pai” do Sistema Toyota de Produção e do Sistema Kanban [37].

to que está em desenvolvimento não está agregando valor até que seja entregue. Produtos em desenvolvimento são encarados como um estoque de fábrica [35].

O primeiro passo para o desenvolvimento enxuto (do inglês, *lean*) é aprender a detectar o desperdício. O segundo passo é descobrir as maiores fontes de desperdício e eliminá-las. Estes dois passos devem ser repetidos sempre. Depois de um tempo, até as coisas que pareciam mais essenciais serão gradualmente eliminadas [35].

### 2.1.1 A ideologia *Lean* na Engenharia de Software

As origens desta metodologia vem da fabricação de carros, mas os seus princípios podem ser aplicados em outras áreas. Entretanto, elas não podem ser aplicadas diretamente vindas de uma linha de produção de fábrica na ES.

Houveram muitas tentativas de inserir no desenvolvimento de *software* as práticas da ideologia *Lean*. Segundo [35] isso acontece porque a criação de *software* não é uma prática de produção, e sim uma prática de desenvolvimento.

O conceito de desenvolvimento de *software Lean* foi introduzido pelo trabalho publicado por Poppendieck M. e Poppendieck T. [35]. Neste trabalho, o desenvolvimento de *software* deve seguir sete princípios fundamentais [2, 24, 35]:

1. Eliminação de desperdício;
2. Amplificação do aprendizado;
3. Adiamento das decisões;
4. Entregar o mais rápido possível;
5. Fortalecimento da equipe;
6. Construção de qualidade;
7. Otimização do todo.

Ao incorporar estes princípios à ES, surge a abordagem contínua no desenvolvimento de *software*, uma tendência emergente que adota uma abordagem holística no desenvolvimento contínuo de *software* [24].

#### 2.1.1.1 1º Princípio - Elimine o desperdício

O desperdício é qualquer coisa que não agrega valor a um produto. O valor considerado aqui é como o produto é percebido pelo cliente. Na ideologia *Lean*, o conceito de desperdício é um grande obstáculo. Se o documento de requisitos ficou somente no papel, isso é um desperdício. Se algo foi feito além do necessário, isso é um desperdício. Se houver código além do necessário, isso é um desperdício. Na fabricação, mover o produto é um desperdício. No desenvolvimento de produtos, transferir o desenvolvimento de um grupo para outro é o desperdício. O ideal é descobrir o que o cliente quer, e depois criá-lo ou desenvolvê-lo e entregar exatamente o que ele quer imediatamente. Qualquer coisa que não satisfaça rapidamente a necessidade do cliente é desperdício [35].

#### 2.1.1.2 2º Princípio - Amplifique o aprendizado

O desenvolvimento de *software* é um trabalho de aprendizado, diferente da produção que é um exercício de práticas rígidas e, por esse motivo, a ideologia *Lean* no desenvolvimento resulta em práticas diferentes das que são realizadas na produção. Como o desenvolvimento produz variações no processo de aprendizagem, a equipe deve buscar meios de transferência de conhecimentos entre os seus membros. Muito importante para este princípio é manter a comunicação entre a equipe, bem como o *feedback* contínuo entre as diferentes equipes e usuários durante o desenvolvimento do *software* [35].

#### 2.1.1.3 3º Princípio - Adie uma decisão o máximo possível

Como as práticas de desenvolvimento envolvem muitas incertezas, principalmente em suas fases iniciais, o adiamento das tomadas de decisões são mais eficazes. Postergar as decisões é importante porque as decisões podem ser tomadas de forma mais eficiente quando são baseadas em fatos, não em especulações como ocorre diante das incertezas. Em um projeto em evolução, manter as opções disponíveis é mais valioso do que fazer uma escolha logo no início [35].

#### 2.1.1.4 4º Princípio - Entregue o mais rápido possível

O desenvolvimento rápido tem muitas vantagens. Sem rapidez na entrega, você não consegue atrasar as decisões. Sem velocidade, você não tem *feedback* confiável. No desenvolvimento, a descoberta é fundamental para a aprendizagem, *design*, implementação, *feedback*, melhoria. Quanto mais curtos forem estes ciclos, maior a possibilidade de se aprender. Velocidade assegura que os clientes estão recebendo o que precisam agora, e não o que eles precisavam ontem. Também permite adiar a ideia final do que eles realmente querem até que eles realmente saibam. Aumentar o fluxo de valor, tanto quanto for possível, é uma estratégia fundamental para a eliminação de desperdício [35].

#### 2.1.1.5 5º Princípio - Fortaleça a equipe

Envolver os desenvolvedores nas decisões técnicas é fundamental para alcançar a excelência. Quando equipados com conhecimentos necessários e guiados por um líder, os desenvolvedores tomarão decisões melhores em grupo, tanto na área técnica como na de processos. Com as decisões sendo tomadas tardiamente e com velocidade na entrega, não é viável apenas uma pessoa comandando as atividades. Assim, as práticas *Lean* usam técnicas de produção puxada para agendar o trabalho através de mecanismos de sinalização de tarefas para que os trabalhadores saibam o que precisa ser feito. No desenvolvimento de *software Lean*, a técnica de produção puxada[1] é acordada para entregar versões cada vez mais refinadas do *software* em intervalos regulares. A sinalização é feita através de gráficos visíveis, reuniões diárias, integração frequente e testes abrangentes [35].

#### 2.1.1.6 6º Princípio - Construa qualidade

Um sistema é de qualidade quando um usuário pensa: "Sim! Isso é exatamente o que eu quero. Alguém leu meus pensamentos!". O *software* precisa de um

[1] Um bom texto sobre a técnica pode ser encontrado aqui: http://www.lean.org.br/conceitos/102/definicao-de-producao-puxada-e-sistemas-puxados.aspx

nível adicional qualidade: deve manter sua utilidade ao longo do tempo. Espera-se que o *software* evolua à medida que se adapta às mudanças. O *software* com qualidade possui uma arquitetura coerente, alto grau de usabilidade, eficácia, é sustentável, adaptável e extensível. Qualidade vem de um projeto liderado sabiamente, com uma equipe com conhecimentos relevantes, comunicação efetiva, disciplina saudável [35].

#### 2.1.1.7 7º Princípio - Otimize o todo

Obter qualidade em sistemas complexos requer uma experiência profunda em diversas áreas. Um dos problemas mais difíceis de resolver no desenvolvimento de um produto é que os especialistas em alguma área (por exemplo, banco de dados ou GUI) tendem a maximizar o desempenho na parte do produto que representa sua própria especialidade em vez de se concentrar no desempenho geral do sistema. Muitas vezes, o bem comum sofre se as pessoas atendem primeiro a seus próprios interesses [35].

## 2.2 Engenharia de Software Contínua

No início da ES, o desenvolvimento de *software* foi marcado pela falta de flexibilidade nos processos, o que prejudicava atividades importantes como planejamento, análise, *design* e construção. Isso pode ser exemplificado no modelo de processo em cascata para desenvolvimento de *software*, descrito por [18]. Ultimamente, surgiram abordagens que buscam aumentar a freqüência de certas atividades críticas no intuito de superar os desafios. Práticas como a entrega antecipada e entrega frequente estão bem estabelecidas no desenvolvimento de *software* aberto [2]. Podemos ver claramente esta afirmação concretizada ao observarmos a crescente adoção da prática de IC. A popularidade desta prática se deve a sua recomendação explícita no XP [22].

Porém o que se nota nas tendências recentes é que as práticas contínuas vão muito mais além do que apenas a prática da IC. Um exemplo é que as práticas ágeis não suportam o alinhamento de práticas com as áreas de negócios ou recur-

sos humanos[1]. Outro exemplo é o DevOps, que reconhece a necessidade das práticas contínuas [2]. Em vez de uma seqüência de atividades que são realizadas por equipes ou departamentos claramente distintos, a engenharia de *software* contínua estabelece práticas contínuas, com suas origens nos conceitos encontrados na ideologia *Lean*, no entanto, há alguns estudos que erroneamente ligam estas práticas apenas as práticas ágeis [9, 24]. Enquanto o desenvolvimento de *software* ágil foca principalmente no desenvolvimento do *software*, o desenvolvimento contínuo tem um foco muito explícito no processo de ponta a ponta: do cliente para a entrega. Com base em uma comparação de princípios e práticas de desenvolvimento ágil e *Lean*, estudos concluem que este foco de ponta a ponta é a principal diferença entre as abordagens [24].

Há uma grande quantidade de termos e conceitos envolvidos na engenharia de software contínua. Por ser uma área de pesquisa relativamente nova em muitos aspectos, uma discordância entre os trabalhos sobre o que é uma prática ou outra e em certos momentos, algumas práticas se confundem em seus significados. Por exemplo, o significado de IC em [24] é igual ao significado de DC em [2]. Já em [4], o significado de DC se confunde com o de EC apresentado por [2]. Para os fins deste trabalho, a distinção entre IC, EC e DC é crucial. Portanto vamos adotar a seguinte definição:

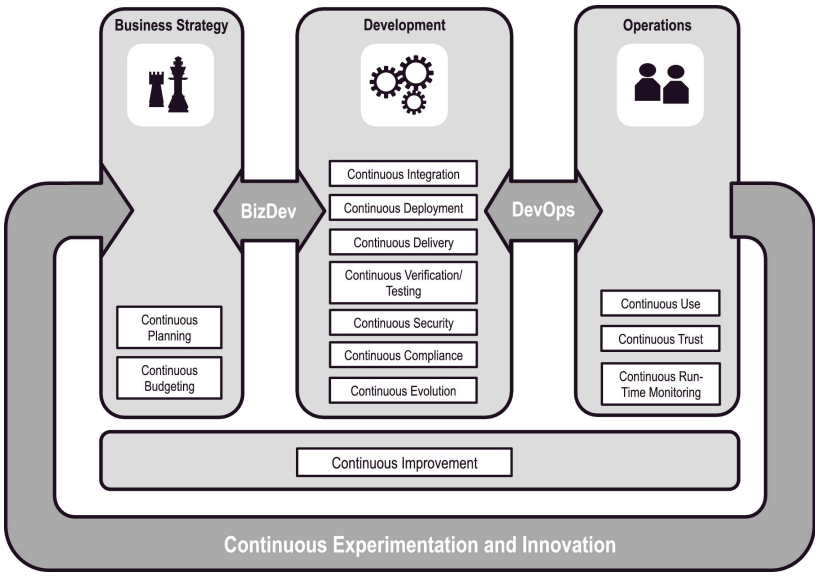
- **Integração Contínua:** os desenvolvedores enviam suas mudanças com frequência para repositórios de controle de versão, que desencadeiam testes em servidores de IC;
- **Entrega Contínua:** as etapas da EC incluem passos automatizados, adicionais à IC, para preparar uma versão do *software* pronto para a implantação. Normalmente, a implantação para o ambiente de produção não é totalmente automatizada e requer intervenção manual na EC;
- **Deploy Contínuo:** a implantação no ambiente de produção é automatizada.

Esta definição foi a mais comum encontrada nos trabalhos pesquisados.

[1] Ultimamente têm surgido algumas tentativas do alinhamento das metodologias ágeis com a área de negócios, como podemos ver em [45]

A figura 1 mostra todas as práticas contínuas e suas relações com a organização. Uma descrição sucinta de cada uma será feita, porém as mais importantes para este trabalho serão discutidas mais detalhadamente.

Figura 1. Continuous * - O conjunto de todas as práticas contínuas




Fonte: [2]

### 2.2.1 Estratégia e Planejamento

Aqui serão citadas as práticas contínuas relativas ao setor estratégico da organização, conhecidas como BizDev[1].

#### 2.2.1.1 Planejamento Contínuo

Envolve os vários *stakeholders* das áreas de negócios e de desenvolvimento de *software*. Esta prática gera planos e artefatos dinâmicos que vão evoluindo de acordo com as mudanças no ambiente dos negócios e, portanto, envolvem uma maior integração entre planejamento e execução [9, 24].

#### 2.2.1.2 Orçamento Contínuo

[1] BizDev não será detalhado aqui, pois não faz parte do escopo deste trabalho.

O orçamento é tradicionalmente um evento anual, informando as previsões de investimentos, receitas e despesas de uma organização para o próximo ano. Esta prática sugere que o orçamento também se torne uma atividade contínua, para facilitar as mudanças durante o ano [9, 24].

### 2.2.2 Desenvolvimento de *software*

Aqui serão citadas as práticas contínuas relativas ao setor de desenvolvimento de *software* da organização, que faz parte do DevOps.

#### 2.2.2.1 Integração contínua

De todas as praticas contínuas, a IC é a mais popular. Esta popularidade é possível porque o XP faz uma recomendação explícita da prática. De fato, a IC é altamente compatível com as metodologias ágeis e por isso muitas ferramentas de código aberto estão disponíveis gratuitamente para automatizar o processo de IC [9].

Como vimos, um dos princípios do desenvolvimento *Lean* é "entregar o mais rápido possível". Seguindo esta regra, o processo de desenvolvimento ágil incorpora o conceito de IC. Para permitir a IC, os desenvolvedores da equipe integram seu trabalho com frequência no servidor de controle de versão. A IC também inclui implicitamente a prática de Teste Contínuo para obter *feedback* rápido e frequente para a equipe de desenvolvimento[1] [2]. Por exemplo, há ferramentas que sinalizam assim que problemas são encontrados na construção [9].

Ståhl e Bosch estudaram quais os efeitos da IC [38], apresentando as diferenças entre as implementações desta prática. Mais tarde, eles desenvolveram um meta-modelo para a implementação da IC nas organizações, visando as diferentes implementações encontradas em seus estudos [39]. Eles descobriram que a prática da IC varia entre as empresas e até mesmo entre projetos da mesma empresa. Também descobriram que apesar de as empresas chamarem o seu processo de IC, nem sempre os princípios das práticas contínuas são seguidos [2].

Um fluxo de etapas da IC está descrito na seção 2.2.2.3.1.

[1] A figura 3 ilustra este fluxo.

#### 2.2.2.2 Entrega contínua

A EC é a prática que vem logo após as práticas de IC, implantando o pacote criado na construção em um ambiente de testes, que não deve ser confundido com o ambiente de produção. No pacote implantado, há toda a alteração de código de cada desenvolvedor já integrada, testada e compilada. É possível que existam vários estágios de teste neste ambiente antes da implantação na produção. Alguns testes nesta etapa podem ser manuais, o que a torna um pouco mais demorada. Com os testes realizados e aprovados, o software é aprovado para ser implantado na produção. A Diferença da EC para o DC, é que no DC, o envio para a produção acontece automaticamente, sem aprovação explícita [2, 45].

A EC permite que os desenvolvedores e o time de qualidade automatizem testes que vão além dos testes de unidade, de forma que seja possível verificar as novas versões em várias dimensões antes de implantá-las na produção. Os testes realizados na EC, podem incluir testes de interface gráfica, carga, integração, confiabilidade, API, etc. Esta prática auxilia a equipe a validar com melhor precisão novas versões, tonando mais fácil a prevenção de erros [2, 45].

#### 2.2.2.3 *Deploy* contínuo

A implantação contínua é a prática de implantação de novos softwares e funcionalidades no ambiente de produção à medida em que se desenvolve, ao invés de fazer implantações agendadas e menos frequentes. Na teoria, para que a prática de DC se concretize, é obrigatório o uso de um *pipeline* de desenvolvimento ininterrupto desde o repositório até a implantação. Para que isso seja possível, é necessária uma cadeia de ferramentas para automatizar o processo, e estabelecer fluxos de trabalho relacionados à cadeia de ferramentas. Uma cadeia de ferramentas devidamente documentada e de fácil configuração, permite que uma empresa adquira o desenvolvimento rápido e facilite a implantação rápida de novos softwares para os clientes [9, 24].

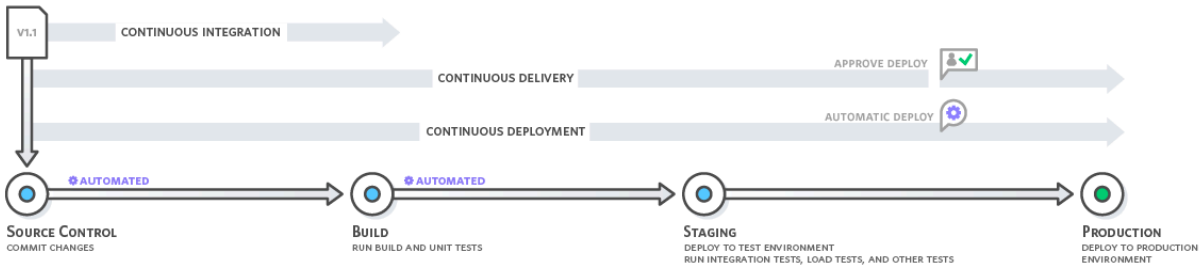
#### 2.2.2.3.1 *Pipeline* de implantação

O DC Só pode ser alcançado se a organização tiver um *pipeline* de implantação definido e automatizado. Os desenvolvedores fazem alterações no código. Após isso, o sistema de controle de versão desencadeia uma fase de confirmação no *pipeline*. Nesta fase, o *software* é normalmente compilado, testado, e um artefato de construção é criado. Muitas vezes, a análise de código estático e a coleta de métricas de código também são feitas. O estágio de confirmação deve ser rápido e capaz de capturar os erros mais graves [2, 10]. Até aqui, as etapas compreendem a prática de IC.

Para iniciar as etapas de EC no *pipeline*, a etapa de testes de aceitação é executada. Esta etapa é automática, mas estes testes serão executados em um ambiente de testes parecido com o de produção. Estes testes podem demorar mais tempo para serem concluídos. O objetivo desta fase é validar que o *software* atende aos seus critérios de aceitação e pode ser enviado para o ambiente de produção. Depois de passar no estágio de teste de aceitação, o *software* pode passar por outros estágios de teste que podem ser manuais, como testes de aceitação do usuário ou testes de capacidade. A etapa de testes manuais deve, no entanto, ser reduzida ao mínimo, porque ela trava o *pipeline* de implantação. Aqui, a EC é finalizada.

Depois de passar por todas as etapas de IC e EC, o software pode ser implantado na produção, configurando a prática do DC [2].

Figura 2. Áreas de cobertura da IC, EC e DC no *pipeline* de implantação

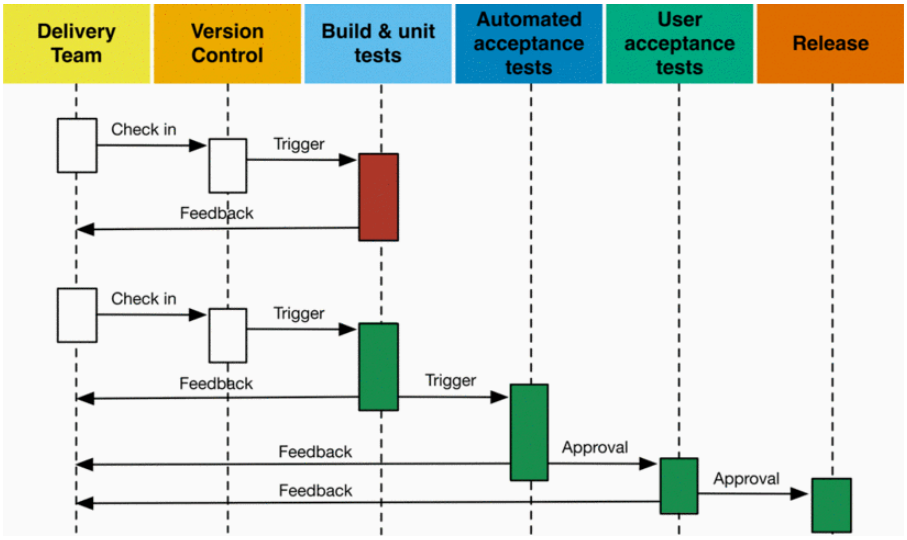



Fonte: [44]

Com base nos trabalhos encontrados, o *pipeline* de implantação deve ser automatizado, tanto quanto possível, ou no mínimo, semi-automatizado. É recomendado que todo o trabalho manual seja reduzido ao mínimo possível, portanto, as cadeias de ferramentas adequadas são essenciais para o DC [2].

Se durante o processo houverem falhas, será criada uma série de artefatos e relatórios gráficos para ajudar a garantir que os problemas que levaram a essas falhas sejam priorizados, sendo resolvidos o mais rápido possível por quem for considerado responsável [9].

Figura 3. *Pipeline* de implantação




Fonte: [10]

#### 2.2.2.3.2 Benefícios e desafios do Deploy contínuo

Em um dos estudos encontrados, o foco está nos potenciais benefícios e desafios da implantação contínua, através de um estudo de caso em 15 empresas [25]. As entrevistas indicam que, embora que nem todas as empresas precisem de um DC totalmente automatizado, implantações frequentes podem ajudar as empresas a criar produtos melhores e mais reativos no desenvolvimento. A cultura organizacional e fatores externos à empresa, podem prejudicar a implementação de práticas de implantação contínua. As questões técnicas, como o tamanho e a complexi-

dade dos projetos, e a necessidade de realizar estágios de testes manuais também são vistos como impedimento para a prática efetiva do DC [2].

O DC permite a implantação (e remoção, se necessário) de novos recursos e modificações com a menor sobrecarga possível. Isso permite a organização determinar o valor dos componentes do software mais rapidamente, com usuários reais [25].

Um estudo de caso pode detalhar melhor os benefícios e desafios do DC [25]. Este estudo mostra os benefícios e desafios percebidos por funcionários após a adoção do DC em várias empresas finlandesas que fazem parte do N4S[1], através de entrevistas.

#### 2.2.2.4 Verificação contínua

Adoção de atividades de verificação, incluindo métodos e inspeções formais ao longo do processo de desenvolvimento. Se opõe às práticas de execução dos testes apenas no fim das atividades de desenvolvimento [9, 24].

#### 2.2.2.5 Teste contínuo

Prática que envolve a automação dos testes e a priorização de casos de teste, para ajudar a reduzir o tempo entre a introdução dos erros e a detecção destes, com o objetivo de eliminá-los de uma forma mais eficaz [9, 24].

#### 2.2.2.6 Conformidade contínua

Nesta prática, o desenvolvimento de *software* busca satisfazer os padrões de conformidade de forma contínua, ao invés de operar uma abordagem de "*big-bang*" para garantir globalmente a conformidade imediatamente antes da liberação do produto [9, 24].

[1] Um programa finlandês de pesquisa em empresas da área de tecnologia da informação e comunicação que utilizam as práticas da ES Contínua [40].

#### 2.2.2.7 Segurança contínua

Esta prática muda a forma como se vê a segurança do *software*. Antes vista como mais um requisito não funcional, agora é encarada como uma preocupação fundamental em todas as fases do ciclo de vida do desenvolvimento e até mesmo depois da implantação, apoiada por uma abordagem inteligente para identificar vulnerabilidades de segurança [9, 24].

#### 2.2.2.8 Evolução contínua

A maioria dos sistemas de *software* evoluem durante o seu ciclo de vida. No entanto, a arquitetura de um sistema se baseia em um conjunto de decisões de projeto que foram feitas durante a criação do sistema. Com o decorrer do projeto, algumas destas decisões podem não ser mais válidas e a arquitetura poderá dificultar algumas mudanças. Quando uma arquitetura não adequada dificulta a inserção de novos requisitos e certos "atalhos" são feitos para contornar o problema, a dívida técnica aumenta. Esta atividade foca em sempre haver revisões destas decisões de projeto ao longo do ciclo de vida [9, 14, 24].

### 2.2.3 Operações

Aqui serão citadas as práticas contínuas relativas ao setor de operações da organização, que faz parte do DevOps.

#### 2.2.3.1 Uso contínuo

Esta prática visa a retenção de clientes ao invés de atrair novos [9, 24]. Isso só é possível através da prática de manutenção contínua, que é melhor explorada em [3].

#### 2.2.3.2 Confiança contínua

Nesta prática, a confiança é desenvolvida ao longo do tempo como resultado de interações, baseadas na crença de que um fornecedor atuará de forma coopera-

tiva para atender às expectativas dos clientes sem explorar suas vulnerabilidades [9, 24].

#### 2.2.3.3 Monitoramento contínuo

Nesta prática, os comportamentos de execução devem ser monitorados para permitir a detecção precoce de problemas na qualidade do serviço, como a degradação do desempenho, e também o cumprimento de acordos de nível de serviço (SLAs) [9, 24].

### 2.2.4 Melhoria e Inovação

Aqui serão citadas as práticas contínuas relativas a melhoria e inovação no desenvolvimento de *software*.

#### 2.2.4.1 Melhoria contínua

Tem base nos princípios da ideologia *Lean*, onde a tomada de decisão está pautada na eliminação de desperdício, trazendo melhorias incrementais de qualidade. Aumentando a vantagem competitiva do produto [9, 24].

#### 2.2.4.2 Inovação contínua

Esta prática provê um processo sustentável que responde as constantes evoluções do mercado, com base em métricas apropriadas durante todo o ciclo de vida das fases de planejamento, desenvolvimento e execução [9, 24].

#### 2.2.4.3 Experimentação contínua

Uma prática baseada em experimentação com *stakeholders*, consistindo em ciclos de construção-medição-aprendizado [9, 24].

## 2.3 DevOps

Não há ainda uma definição rígida nos trabalhos do que é o DevOps. Em alguns trabalhos é definido como uma metodologia, em outros, uma técnica, e em alguns, um processo. Os efeitos causados pela sua adoção nas organizações vão além disso. O DevOps pode ser classificado como uma ideologia que evoluiu em resposta à falta de esforços na sinergia entre as equipes de desenvolvimento e operações. A cultura DevOps atua como um facilitador para oferecer mais funcionalidades continuamente, mantendo a estabilidade [11, 12, 15]. Se baseia nas práticas da ES Contínua, que ajudam a reduzir o tempo entre o desenvolvimento do *software* e a sua implantação no ambiente de produção, mantendo alta qualidade de software [7]. Para obter esta qualidade, a equipe de desenvolvimento deve reagir imediatamente a qualquer demanda do cliente com o objetivo de entregar o mais rápido possível. Para isso, tanto a equipe de desenvolvimento como a de operações precisam chegar a um acordo de quais práticas da ES Contínua serão utilizadas para alcançar este objetivo. Isso inclui o canal de comunicação entre as equipes [13].

### 2.3.1 DevOps e Agile

*Agile* é o conjunto de práticas baseadas no Manifesto Ágil relevantes para o desenvolvimento de *software* [45]. O *Agile* pode ser utilizado em conjunto com o DevOps. Nesse caso, o DevOps incentiva a colaboração entre os membros da equipe, a automação da construção, implantação e teste, medições e métricas de custo, compartilhamento de conhecimento e ferramentas. DevOps adiciona valor ao *Agile* ao fornecer uma extensão pragmática para as atividades ágeis. Por exemplo, como o *DevOps* enfatiza mais a comunicação e a colaboração entre as equipes de desenvolvimento e operações, em vez de ferramentas e processos, é possível atingir metas ágeis para reduzir a carga de trabalho em equipe e estender princípios ágeis em todo o *pipeline* de implantação [5].

No entanto, há um momento em que o DevOps rompe com o *Agile*. Ao utilizar as práticas contínuas, o DevOps entra em conflito com os princípios propostos pelo manifesto ágil, em especial com o primeiro princípio. Além disso, à medida que as

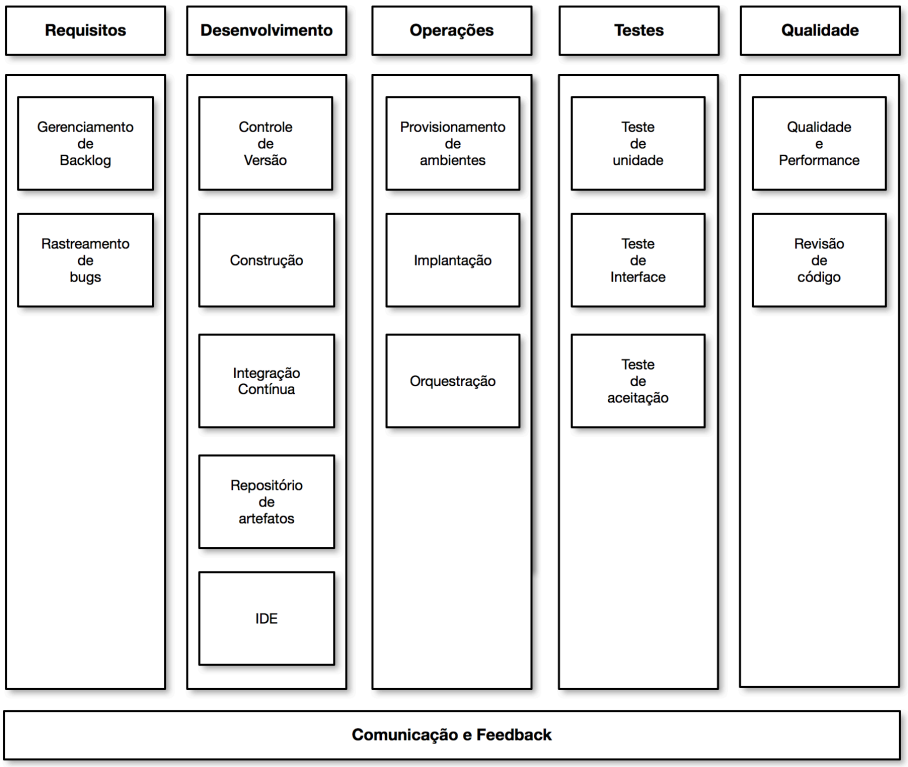
funcionalidades vão sendo desenvolvidas e implantadas, gradualmente vai diminuindo a necessidade das iterações [13].

# 3. Mapeamento das Ferramentas

Neste capítulo, serão apresentadas as ferramentas utilizadas por equipes cujos projetos possuem a cultura *DevOps*. As ferramentas são citadas em uma série de estudos de caso, artigos e revisões sistemáticas da literatura. Os estudos forneceram as informações que tornaram possível a realização do mapeamento das ferramentas. Os trabalhos utilizados são: [2, 3, 5, 7, 8, 11, 12, 26, 39, 49, 50].

O desenvolvimento de *software* contínuo deve ser suportado por uma cadeia de ferramentas que automatizam o *pipeline*, desde a criação do código até a implantação. O papel das ferramentas é garantir que seja possível desenvolver, testar e implantar o *software* enquanto ao mesmo tempo são produzidos comentários adequados para todas as etapas do processo de desenvolvimento. A figura 4 ilustra as áreas do *DevOps* que podem se beneficiar do suporte de ferramentas. A figura ilustra um modelo baseado em outro, apresentado por [2], com algumas modificações, por influência dos modelos apresentados nos estudos restantes.

Figura 4. Áreas de cobertura das ferramentas




Fonte: o Autor, adaptado de [2]

## 3.1 Áreas de atuação das ferramentas

As áreas de atuação das ferramentas são: requisitos, desenvolvimento, operações, Testes, Qualidade e Comunicação e *Feedback*, da mesma forma que é feito em [2]. Nos tópicos seguintes, serão justificadas a importância dos tipos de ferramentas que foram apresentadas na figura 4.

### 3.1.1 Requisitos

Mesmo que o DC seja realizado várias vezes por dia, os requisitos devem ser claros. Para que isso seja possível, manter um registro de todos os requisitos, bem como de todos os dados associados à eles é essencial no desenvolvimento de *software*. Para os requisitos, as ferramentas que gerenciam os requisitos e o *backlog* são consideradas. Os requisitos e *feedbacks* do usuário podem ser obtidos não somente através da coleta direta com usuários, também é possível realizar essa aproximação através de ferramentas de solicitação de melhorias e de monitoramento de uso [2].

O rastreamento de erros também está relacionado ao gerenciamento de *backlog*, pois a correção de um *bug* também é uma tarefa de desenvolvimento, logo, também é uma tarefa do *backlog* como qualquer outra. Em alguns casos, os *bugs* também são sinalizados por clientes ou usuários finais. Nesse caso, uma ferramenta para a comunicação pode ser necessária [2].

### 3.1.2 Desenvolvimento

Sistemas de controle de versão (VCS) existem para gerenciar mudanças em vários documentos, como o código-fonte e páginas da Web, e outras informações que podem ser alteradas durante o desenvolvimento do *software* [2].

Os sistemas de construção são fundamentais para o desenvolvimento de *software*. Criam os pacotes de instalação baseados em arquivos de especificação. Estes arquivos de especificação podem incluir instruções para compilação de código, execução de casos de teste e criação de pacotes de implantação. Ferramentas de construção interpretam estes arquivos, ajudando os desen-

volvedores a gerenciar as dependências internas e externas de *software*, como acontece com as bibliotecas externas utilizadas [2].

As construções automatizadas concretizam a prática da IC, que por sua vez é um pré-requisito para a EC e para o DC. Isto torna a IC uma espinha dorsal para qualquer *pipeline* de DC. As ferramentas de IC monitoram VCSs para verificar mudanças que, consequentemente, desencadeiam a execução das várias etapas configuradas no servidor de IC. As etapas podem incluir a compilação e o empacotamento do código para criar uma versão implantável do *software* [2].

Os repositórios de artefatos oferecem uma solução para os desenvolvedores gerenciarem dependências e versões de diferentes bibliotecas [2].

### 3.1.3 Operações

Os diferentes ambientes utilizados nas etapas do *pipeline* possibilitam a criação de instâncias do ambiente de execução do *software*. Os diferentes ambientes em que o *software* será executado, são usados em paralelo durante o desenvolvimento, no entanto, devem ser idênticos, de forma que a mesma construção possa ser implantada em qualquer um deles. Quando se trata do DC, é necessário que a criação de instâncias dos ambientes seja rápida e de fácil execução, para que a construção possa passar do desenvolvimento para a produção, sem muitos obstáculos ou problemas originados pelas diferenças nos ambientes [2].

As ferramentas de orquestração automatizam, organizam, coordenam e gerenciam a infraestrutura e a criação dos ambientes, fornecendo recursos necessários, como criação de máquinas virtuais, alocação de capacidade de armazenamento, e o gerenciamento de recursos de rede [2].

Humble e Farley apresentam anti-padrões comuns relacionados à entrega de *software* [48]:

1. Implantação manual do *software*;
2. Implantação em produção apenas no fim de todo o desenvolvimento;
3. Configuração manual dos ambientes.

O objetivo do DC deve ser a implantação do *software* da forma mais automática possível [2].

### 3.1.4 Testes

As ferramentas de IC com testes de unidade automatizados devem garantir que as mudanças feitas no desenvolvimento não introduzam novos erros.

O teste da interface do usuário (UI) é uma tarefa onde a interface gráfica do usuário é testada. Algumas áreas de testes de UI, como usabilidade ou teste de experiência do usuário, se aproximam do domínio da garantia de qualidade, pois não são testes puramente funcionais e podem ser difíceis de automatizar [2].

O teste de aceitação é um caso especial de teste, onde é determinado se o sistema de *software* está em conformidade com os requisitos, funcionais ou não. Do ponto de vista da IC, os testes de aceitação devem ser executados em um ambiente idêntico ao de produção [48].

Geralmente, os testes de aceitação exigem que o cliente esteja presente em sua execução, porém isso nem sempre é possível, principalmente se houver uma entrega frequente do *software*. Para que estes testes sejam automatizados, a presença do cliente só é necessária ao se construir a automação do caso de teste [2].

### 3.1.5 Qualidade

Os testes funcionais não são suficientes para garantir a qualidade do *software*. No entanto, não apenas a IC, mas as práticas ágeis em geral, incluem na definição de alta qualidade de *software* a forma em que o código foi escrito e a abrangência de requisitos não funcionais [2].

Uma maneira aparente de garantir a qualidade é a criação de métricas. Por exemplo, garantir desempenho sem monitorar a execução do *software* é muito difícil. Para isso, ferramentas de teste de carga e estresse podem ser usadas para garantir que o *software* funcionará mesmo em circunstâncias ainda mais desfavoráveis [2].

Alguns dos parâmetros de qualidade podem ser verificados mesmo que o *software* não esteja em execução. Um exemplo são as ferramentas de análise de

código estático, que verifica o padrão de escrita do código, prevendo possíveis bugs, vulnerabilidades e código suspeito [2].

### 3.1.6 Comunicação e *Feedback*

Comunicação e *feedback* são elementos-chave no desenvolvimento de *software*. Os desenvolvedores devem compartilhar o conhecimento através de vários mecanismos, ajudando a equipe a construir uma identidade. Sistemas colaborativos envolvem também os demais *stakeholders*, permitindo que eles participem do projeto nas etapas de desenvolvimento, criando uma prática de *feedback* contínuo. O *pipeline* de DC pode até ser útil em um sentido técnico, mas sem os *feedbacks* dos *stakeholders*, a equipe de desenvolvimento perderá o rumo, sem saber em que direção seguir [2].

## 3.2 Separação e classificação das ferramentas

Ao fim, cento e dez ferramentas foram catalogadas e divididas conforme as áreas e subáreas descritas. Todos os dados coletados estão no APÊNDICE A.

Os dados estão divididos em onze áreas diferentes, ajudando na criação da ferramenta de escolha. As áreas são: Nome, Subárea, *Website*, Artigos, Tipo de projeto, Mantenedores, Tipo de licença, Integração, Plataforma e Paradigma.

### 3.2.1 Nome

Indica o **nome** da ferramenta, Não deve ser confundido com o nome da empresa ou grupo que a criou.

### 3.2.2 Subárea

São as **áreas internas** de cada área apresentada na figura 4 e já explanadas na seção 3.1.

### 3.2.3 *Website*

Informa em qual **site** estão as informações oficiais da ferramenta. Quando as ferramentas provêm de projetos *open source* ou têm acesso grátis, o website também é onde o *download* da ferramenta deve ser realizado. As informações das ferramentas foram retiradas dos seus websites.

### 3.2.4 Artigos

Informa onde cada ferramenta foi encontrada na lista de **estudos** que foram utilizados na pesquisa. Os estudos estão descritos no Início deste capítulo.

### 3.2.5 Tipo de Projeto

Indica se o projeto da ferramenta é realizado por grupos **privados**, ou se o projeto é ***open source***.

### 3.2.6 Linguagem de programação

Informa qual a linguagem de programação que a ferramenta suporta. Há ferramentas que não envolvem programação. Estas ferramentas recebem a classificação "**indiferente**". Há ferramentas que se acoplam a qualquer linguagem de programação, recebendo portanto, a classificação "**qualquer**".

As demais ferramentas, se acoplam a linguagens de programação específicas, recebendo como classificação o **nome dessas linguagens**.

### 3.2.7 Mantenedores

Informa o **nome das empresas** que mantêm as ferramentas, no caso de projetos privados. Em todos os projetos *open source*, os mantenedores foram classificados como "**comunidade**".

### 3.2.8 Tipo de licença

Informa se a ferramenta é paga. A classificação dada para as ferramentas não pagas é **grátis**. Se a ferramenta não for grátis ela recebe a classificação **paga**.

### 3.2.9 Integração

Informa com quais ferramentas esta se integra. O **nome** da ferramenta é utilizado aqui como classificação. Para as ferramentas que se integram com todas as outras, é dada a classificação "**diversos, quase todos os tipos**". As ferramentas que as informações de integração eram inexistentes em seus *websites*, é dada a classificação "**não informado**".

### 3.2.10 Plataforma

Informa a plataforma que a ferramenta suporta. As plataformas que as ferramentas suportam, são classificadas como: *Desktop*, *Desktop*(*Mac*), *Mobile*, *Mobile*(*Android*), *Mobile*(*IOS*), *Web*, Microsserviços.

***Desktop*:** são as ferramentas que suportam projetos em que o *software* criado será usado apenas no ambiente em que será executado.

***Desktop*(*Mac*):** são as ferramentas que suportam projetos em que o *software* criado será executado apenas no sistema Mac OS.

***Mobile*:** são as ferramentas que suportam projetos em que o *software* que será criado executará em dispositivos móveis. Se limita aos dispositivos com os sistemas operacionais Android e IOS.

***Mobile*(*Android*):** são as ferramentas que suportam projetos em que o *software* que será criado executará em dispositivos móveis com o sistema Android.

***Mobile*(IOS):** são as ferramentas que suportam projetos em que o *software* que será criado executará em dispositivos móveis com o sistema IOS.

***Web*:** são as ferramentas que suportam projetos em que o *software* que será criado executará tecnologias para a *web*, como sistemas que são acessados por *browsers* ou *websites*.

**Microsserviços:** são as ferramentas que suportam projetos de *software* com arquitetura distribuída que serão acessados através da *web*.

Quando a ferramenta não envolve programação, ela é classificada como "**indiferente**". Para as ferramentas que suportam projetos para todas as plataformas, a classificação é **todas**.

### 3.2.11 Paradigma

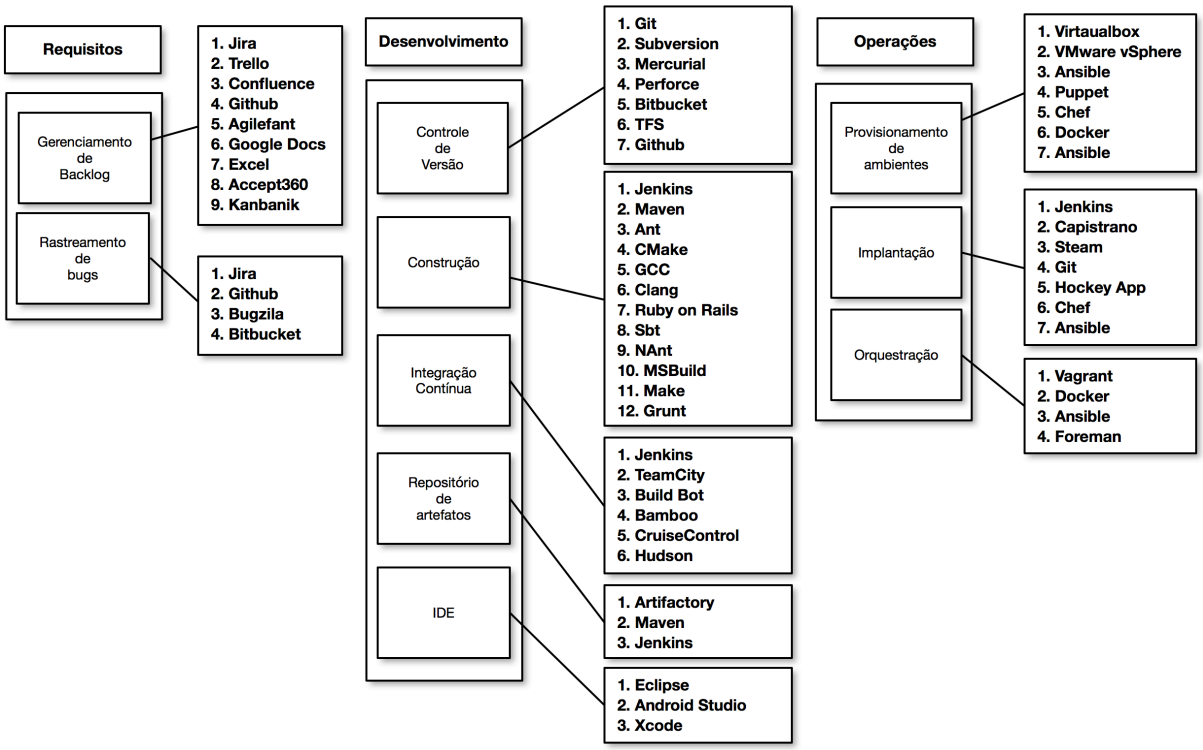
Informa o paradigma de programação[1] que a ferramenta dá suporte. Os paradigmas de programação que as ferramentas mapeadas dão suporte são: o **funcional**, **o Orientado a Objetos** e o **Imperativo.**

## 3.3 Ferramentas mapeadas

As figuras 5, 6 mostram as ferramentas de acordo com as áreas e subáreas relatadas na figura 4. Todas as ferramentas são citadas nos artigos já informados no início do capítulo.As demais informações sobre ferramentas se encontram no APÊNDICE A.

### 3.3.1 Ferramentas catalogadas

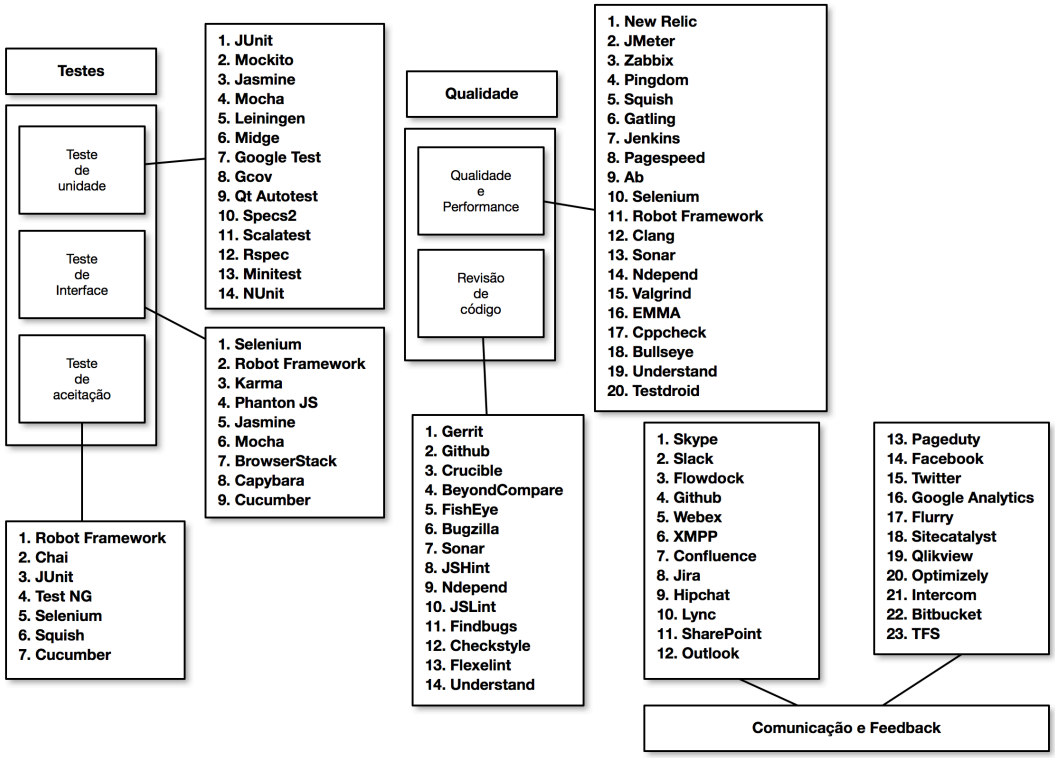
Figura 5. Ferramentas de requisitos, desenvolvimento e operações




Fonte: o Autor

[1] Um estudo sobre paradigmas de programação pode ser encontrado aqui: http://www.petry.pro.br/sistemas/programacao1/materiais/artigo_paradigmas_de_programacao.pdf

Figura 6. Ferramentas de testes, qualidade e comunicação




Fonte: o Autor

### 3.3.2 Grafo de integração de ferramentas

Um dos pontos mais importantes para a classificação das ferramentas é a integração entre elas. Ferramentas interoperáveis facilitam o trabalho, portanto na ferramenta criada, a quantidade de integrações determina a preferência por uma ou outra ferramenta. Para exibir a integração entre as ferramentas, um grafo foi criado, entretanto, a quantidade de nós e arestas do grafo não permitem a criação de uma imagem legível. Para resolver esse problema, uma ferramenta[1] foi criada para exibir o grafo de forma interativa, permitindo que seja possível visualizar as integrações[2].

[1] Acesse o grafo em: https://projetotcc-urbdvuqhng.now.sh//integracaoferrementas.html

[2] Coloque o mouse em cima do nó e as ligações serão destacadas. Clique duas vezes no nó e as arestas ficarão transparentes.

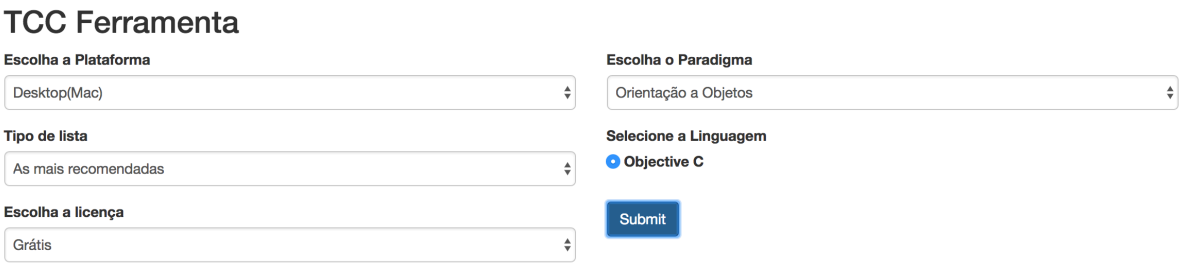
# 4. Automatizando a escolha das ferramentas

A automação será feita da forma mais simples possível. Uma ferramenta baseada na web foi criada para servir como conceito, na criação de ferramentas que efetivamente automatizem o processo de escolha de ferramentas. A ferramenta pode ser encontrada no seguinte endereço: https://projetotcc-urbdvuqhng.now.sh/

## 4.1 Ferramenta

A ferramenta foi desenvolvida utilizando a linguagem Javascript em conjunto com os frameworks bootstrap e D3. Inicialmente, a ferramenta exibe um menu principal conforme a figura, abaixo:

Figura 7. Menu principal da ferramenta

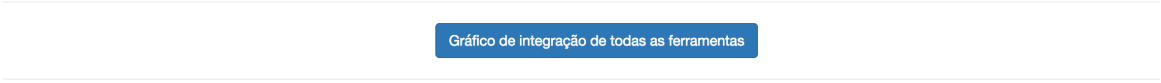



Fonte: O Autor

Inicialmente, o usuário insere os dados conforme exibido no capítulo anterior. O único dado que deve ser fornecido e não foi abordado antes é a escolha do tipo de lista que o usuário quer visualizar. Com essa opção, o usuário decide se quer ver todas as ferramentas recomendadas ou se quer apenas as mais recomendadas, que no caso, exibe apenas a ferramenta que mais se integra com as demais da lista. O paradigma de programação interfere na linguagem que o usuário poderá selecionar.

No centro da tela, um botão pode ser selecionado para direcionar o usuário para a página que contém o grafo de integração de todas as ferramentas.

Figura 8. Botão de redirecionamento de página da ferramenta




Fonte: O Autor

As imagens referentes ao grafo e a ferramenta podem ser encontradas no APÊNDICE B.

Após selecionar todas as opções, um resultado é exibido, como no exemplo abaixo:

Figura 9. Ferramentas gratuitas mais recomendadas para desenvolvimento Desktop(Mac) pela ferramenta de escolha


Fonte: O Autor

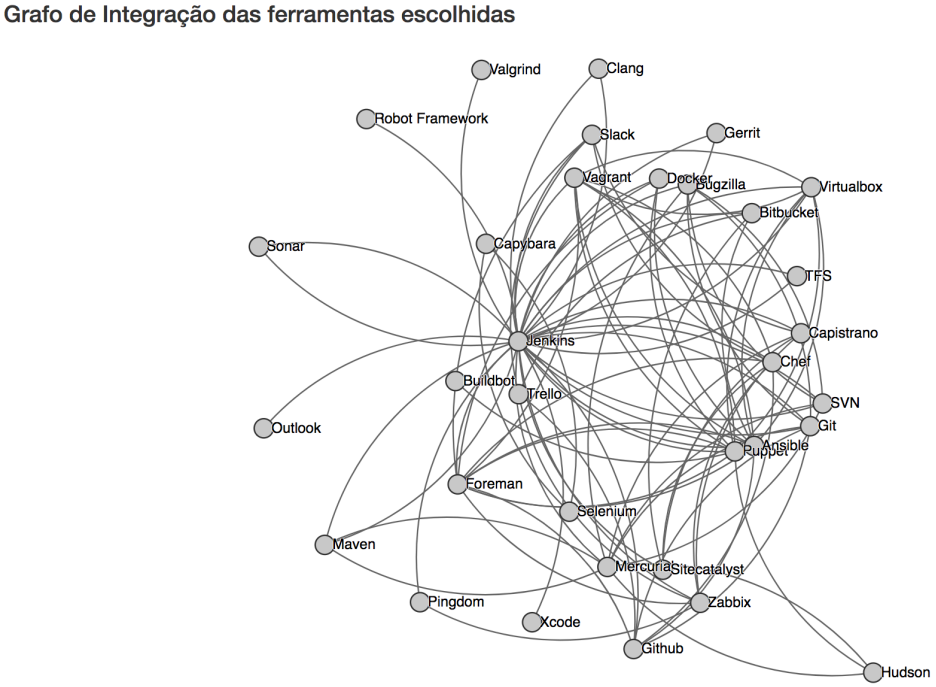
Podemos ver, que o resultado das ferramentas mais recomendadas é eficiente para uma busca rápida, indicando as ferramentas que mais se integram de acordo com os parâmetros inseridos. Caso seja necessária uma busca mais abrangente, a opção de todas as ferramentas recomendadas, pode ser selecionada para indicar todas as ferramentas catalogadas, atendendo aos parâmetros selecionados.

Quando a opção "Todas as ferramentas recomendadas" for a escolhida, As ferramentas indicadas devem ser observadas de cima para baixo, onde as ferramentas ao topo da lista são as mais indicadas para o projeto. A escolha das ferramentas mais indicadas está baseada na quantidade de integrações com as demais ferramentas exibidas.

Figura 10. Todas as ferramentas gratuitas recomendadas para desenvolvimento Desktop(Mac) pela ferramenta de escolha


Fonte: O Autor

Figura 11. Grafo representando as integrações entre as ferramentas recomendadas




Fonte: O Autor

Por fim, para ajudar na visualização das integrações, como possível ver na figura 11, um grafo interativo é exibido.

## 4.2 Estudo de caso

A ferramenta foi aplicada em um projeto real, o projeto MidiaCenter da empresa Ustore. Esta seção informa a experiência vivenciada no estudo de caso.

### 4.2.1 Empresa

A Ustore[1], é uma empresa brasileira de *software* para infraestrutura de computação em nuvem. Nasceu no Porto Digital[2] em Recife-PE, mantendo em suas equipes de desenvolvimento e executivos, profissionais doutores em Ciência da Computação e Segurança de Sistemas com grande experiência no mercado de tecnologia da informação e comunicação. As soluções da Ustore tornam a infraestrutura das corporações totalmente programável de forma segura e distribuída, viabilizando a implementação de um datacenter 100% virtual, ofertando serviços abrangentes de acordo com a análise de necessidades dos clientes, por meio de consultoria e profissionais especializados.

### 4.2.2 Projeto

O MidiaCenter[3] é o projeto educacional da Ustore. Hoje, se encontra em fase de desenvolvimento, sendo desenvolvido com tecnologias atuais e baseadas nas plataformas *web* e *mobile.* O projeto *web* é desenvolvido em JavaEE[4], com o *framework*[5] *Play*[6], utilizando uma arquitetura monolítica. Consiste em um sistema que promove a troca de conhecimentos entre professores e alunos, permitindo o compartilhamento e a exibição de conteúdos educacionais entre a comunidade es-

[1] http://ustore.com.br

2 http://www.portodigital.org/parque/o-que-e-o-porto-digital

[3] http://midiacenter.usto.re

[4] Padrão Java para aplicações corporativas (ver: http://www.oracle.com/technetwork/java/javaee/overview/index.html).

[5] Abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica

[6] https://www.playframework.com/

colar. Para fazer o armazenamento dos conteúdos, o sistema utiliza outros produtos da Ustore.

Atualmente, o projeto MidiaCenter utiliza a prática de IC. As demais etapas são realizadas manualmente. Pelos motivos já citados neste trabalho, as etapas do desenvolvimento devem ser automatizadas máximo possível para que a equipe possa obter os benefícios fornecidos pela prática do DC. Futuramente o projeto passará por uma modificação nas práticas de desenvolvimento, adotando o *DevOps* e automatizando completamente o *pipeline*.

### 4.2.3 Utilizando a ferramenta

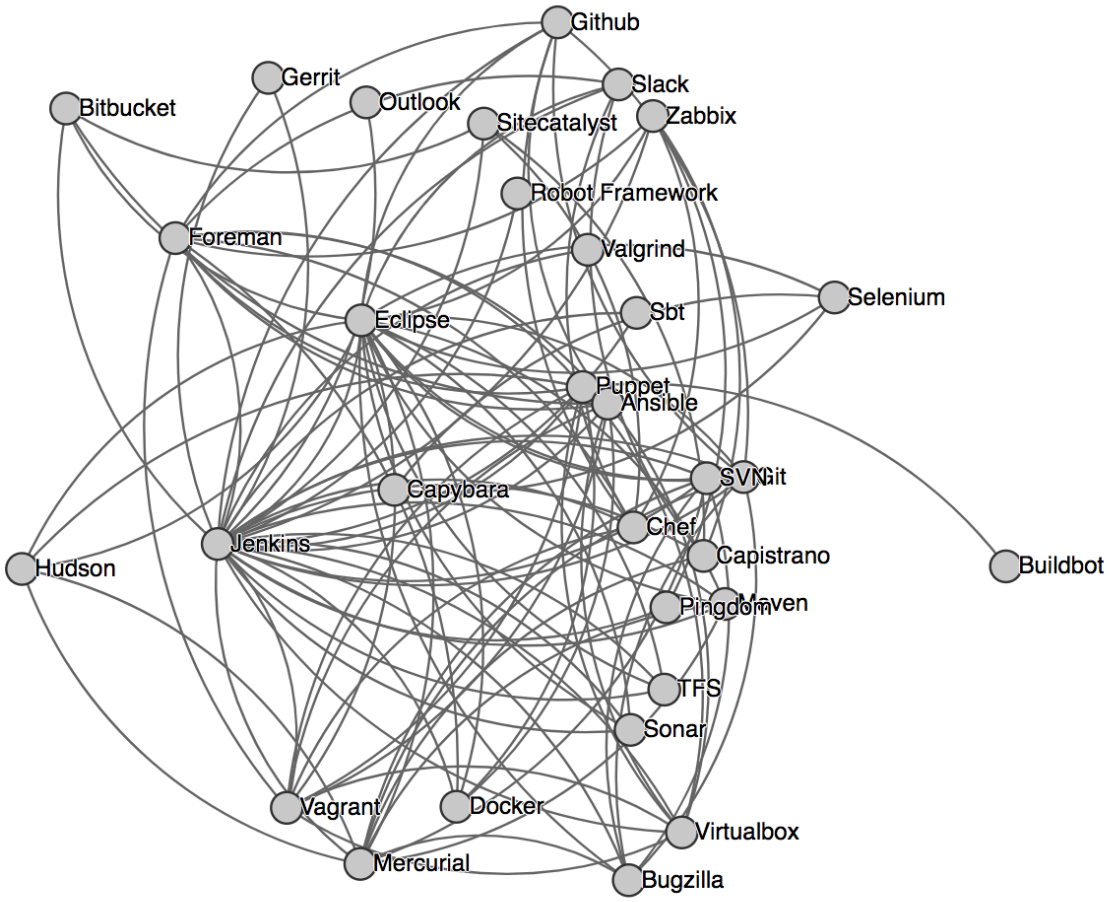
Figura 12. Todas as ferramentas gratuitas catalogadas para o MidiaCenter


Fonte: O Autor

Inicialmente, o menu principal foi exibido, como na figura 7. A plataforma do MidiaCenter é *web* não distribuída, portanto a opção "*Web* Monolítica" foi selecionada. Com a plataforma selecionada, a escolha do paradigma de programação foi o Orientado a Objetos. Após escolher o paradigma, a escolha da linguagem então foi realizada, sendo o caso do sistema da Ustore, o Java com *Play Framework*.

No caso do MidiaCenter, que fará experimentos antes de migrar definitivamente para as práticas contínuas, as ferramentas grátis escolhidas.

Figura 13. Todas as ferramentas gratuitas catalogadas para o MidiaCenter




Fonte: O Autor

# 5. Considerações finais

Este capítulo conclui o trabalho com uma pequena discussão sobre as limitações na criação deste trabalho. Uma discussão sobre trabalhos futuros também é iniciada. No final do capítulo uma conclusão é feita.

## 5.1 Limitações

As limitações deste trabalho estão relacionadas principalmente, a ausência de informações sobre boas práticas no uso das ferramentas. A falta destas informações tornam o modelo criado limitado apenas às áreas de aplicação das ferramentas, deixando o resultado das sugestões em alguns casos bastante genérico. Este trabalho se limita apenas ao estudo de ferramentas, logo, não está no seu escopo a discussão de boas práticas no *DevOps*.

Também pode ser considerada uma limitação, a quantidade de ferramentas mapeadas. Apesar da quantidade de ferramentas ser adequada para este estudo, existem muitas ferramentas em uso por equipes que utilizam práticas contínuas e o *DevOps*.

Outra limitação identificada é ferramenta criada, que serve apenas como um modelo conceitual para a construção de uma ferramenta mais elaborada. Devido a limitações por ser apenas uma página, além do tempo, melhores formas de se escolher as ferramentas não foram criadas.

## 5.2 Trabalhos futuros

Para complementar este trabalho, uma revisão sistemática da literatura é necessária, buscando verificar o estado da arte de estudos que tratam de melhores práticas em projetos *DevOps*, e como essas práticas são realizadas na indústria.

Outro trabalho muito importante que deve ser considerado, é a criação de uma ferramenta que torne possível, a real automação na escolha destas ferramentas,

através das melhores práticas mapeadas no trabalho anterior, possibilitando a escolha, instalação e configuração automática das ferramentas.

## 5.3 Conclusão

Este trabalho fez um panorama do início da ES até o *DevOps*. Apresentou o conceito da Ideologia *Lean* e informou quais os benefícios de inserir essa ideologia na ES. Também foram informados os conceitos das práticas contínuas e do *DevOps*. Com os conceitos aprendidos é possível diferenciar o *DevOps* do *Agile*.

O *DevOps* só pode ser aplicado efetivamente em uma organização, se esta optar por práticas contínuas, principalmente a integração contínua, a entrega contínua e o *deploy* contínuo. A entrega e deploy contínuos dependem de um pipeline automatizado, que por sua vez necessita de ferramentas que concretizem essa automação. Quando falamos em ferramentas, olhar somente para elas não basta e isso fica evidente ao se observar a quantidade de ferramentas encontradas. Temos que entender como elas são aplicadas nas organizações e quais as melhores práticas nesse uso. Por exemplo, Ståhl e Bosch indicam que as práticas contínuas são realizadas de várias formas diferentes nas organizações, as vezes há diferenças de como estas práticas são realizadas até entre equipes da mesma empresa, mostrando que não existe apenas uma forma de se aplicar a ES contínua [38]. O que existe, no entanto, é uma quantidade finita destas variações e, portanto, é possível estudá-las buscando a melhor forma de aplicar todos estes conceitos e ferramentas da forma mais otimizada possível.

Com base em todas as informações estudadas na pesquisa e na criação deste trabalho, conclui-se que é possível criar ferramentas robustas e úteis para automatizar o processo de escolha de ferramentas, no entanto, é importante que estas informem como as ferramentas podem ser aproveitadas da melhor forma possível em todo o pipeline de implantação.

# Referências Bibliográficas


[1] LEHTONEN, T. *et al.* **Defining metrics for continuous delivery and deployment pipeline**. CEUR Workshop Proceedings. **Anais...** 2015.

[2] MÄKINEN, S. *et al.* Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. **Information and Software Technology**, v. 80, p. 1339–1351, 2016.

[3] PANG, C.; HINDLE, A. **Continuous Maintenance.** 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). **Anais**...IEEE, out. 2016Disponível em: <http://ieeexplore.ieee.org/document/7816494/>. Acesso em: 7 maio. 2017.

[4] LEPPANEN, M.; KILAMO, T.; MIKKONEN, T. **Towards Post-Agile Development Practices through Productized Development Infrastructure.** Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015. **Anais**...IEEE, maio 2015. Disponível em: <http://ieeexplore.ieee.org/document/7167171/>. Acesso em: 7 maio. 2017.

[5] JABBARI, R. *et al.* **What is DevOps?** Proceedings of the Scientific Workshop Proceedings of XP2016 on - XP '16 Workshops. **Anais**... New York, New York, USA: ACM Press, 2016Disponível em: <http://dl.acm.org/citation.cfm?doid=2962695.2962707>. Acesso em: 7 maio. 2017.

[6] KARVONEN, T. *et al.* Systematic Literature Review on the Impacts of Agile Release Engineering Practices. **Information and Software Technology**, v. 86, n. C, p. 87–100, jun. 2017.

[7] WETTINGER, J. *et al.* Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. **Future Generation Computer Systems**, v. 56, p. 317–332, 2016.

[8] OHTSUKI, M.; OHTA, K.; KAKESHITA, T. **Software engineer education support system ALECSS utilizing DevOps tools.** Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services - iiWAS '16. **Anais**...New York, New York, USA: ACM Press, 2016. Disponível em: <http://dl.acm.org/citation.cfm?doid=3011141.3011200>. Acesso em: 7 maio. 2017.

[9] FITZGERALD, B.; STOL, K.-J. **Continuous software engineering and beyond: trends and challenges.** Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering - RCoSE 2014. **Anais**... New York, New York, USA: ACM Press, 2014Disponível em: <http://dl.acm.org/citation.cfm?doid=2593812.2593813>. Acesso em: 7 maio. 2017.

[10] GEBERT, S. *et al.* **Continuously delivering your network.** Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015. **Anais**...IEEE, maio 2015. Disponível em: <http://ieeexplore.ieee.org/document/7140371/>. Acesso em: 7 maio. 2017.

[11] WETTINGER, J.; BREITENBUCHER, U.; LEYMANN, F. **Standards-based DevOps automation and integration using TOSCA.** Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014. **Anais**...IEEE, dez. 2014. Disponível em: <http://ieeexplore.ieee.org/document/7027481/>. Acesso em: 7 maio. 2017.

[12] OLSZEWSKA, M.; WALDÉN, M. DevOps meets formal modelling in high-criticality complex systems. **Proceedings of the 1st International Workshop on Quality-Aware DevOps - QUDOS 2015**, p. 7–12, 2015.

[13] KILAMO, T.; LEPPÄNEN, M.; MIKKONEN, T. The social developer: now, then, and tomorrow. **Proceedings of the 7th International Workshop on Social Software Engineering - SSE 2015,** p. 41–48, 2015.

[14] SHAHIN, M.; BABAR, M. A.; ZHU, L. **The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives.** Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. **Anais**...New York, New York, USA: ACM Press, 2016. Disponível em: <http://dl.acm.org/citation.cfm?doid=2961111.2962587>. Acesso em: 7 maio. 2017.

[15] RAJKUMAR, M. *et al.* **DevOps culture and its impact on cloud delivery and software development.** 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Spring). **Anais**...IEEE, abr. 2016. Disponível em: <http://ieeexplore.ieee.org/document/7578902/>. Acesso em: 7 maio. 2017.

[16] SOMMERVILLE, I. **Engenharia de Software, 9a Edição**. Pearson Ed ed. São Paulo: Pearson Education, 2011.

[17] SOARES, M. D. S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. **INFOCOMP Journal of Computer Science**, v. 27, n. 2, p. 6, 2003.

[18] ROYCE, D. W. W. Managing the Development of large Software Systems. **Ieee Wescon**, n. August, p. 1–9, 1970.

[19] **The Chaos Report**. . Dennis, MA 02638, USA. 1995. Disponível em: <https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf>.

[20] BERNARDO, K. **Manifesto ágil, como tudo começou**. Disponível em: <https://www.culturaagil.com.br/manifesto-agil-como-tudo-comecou/>. Acesso em: 4 junho. 2017.

[21] BECK, K. *et al.* **Manifesto for Agile Software Development**. Disponível em: <http://agilemanifesto.org/>. Acesso em: 4 jun. 2017.

[22] BECK, K. **Extreme programming eXplained: embrace change**. [s.l.] Addison-Wesley, 2000.

[23] SCHWABER, K.; BEEDLE, M. **Agile software development with Scrum.** [s.l.] Prentice Hall, 2002.

[24] FITZGERALD, B.; STOL, K. J. Continuous software engineering: A roadmap and agenda. **Journal of Systems and Software**, v. 123, p. 176–189, 2017.

[25] LEPPANEN, M. *et al.* **The highways and country roads to continuous deploymentIEEE Software**, mar. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7057604/>. Acesso em: 22 maio. 2017.

[26] SHAHIN, M.; ALI BABAR, M.; ZHU, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. **IEEE Access**, v. 5, p. 1–1, 2017.

[27] **IEEE Xplore Digital Library**. Disponível em: <http://ieeexplore.ieee.org/Xplore/home.jsp>. Acesso em: 27 jun. 2017.

[28] **ScienceDirect.com | Science, health and medical journals, full text articles and books**. Disponível em: <http://www.sciencedirect.com/>. Acesso em: 27 jun. 2017.

[29] ASSOCIATION FOR COMPUTING MACHINERY. **ACM Digital Library**. Disponível em: <http://dl.acm.org/>. Acesso em: 27 jun. 2017.

[30] **Scopus**. Disponível em: <https://www.scopus.com/home.uri?zone=header&origin=searchbasic>. Acesso em: 27 jun. 2017.

[31] **Google Acadêmico**. Disponível em: <https://scholar.google.com.br/>. Acesso em: 27 jun. 2017.

[32] IEEE; BOURQUE, P.; FAIRLEY, R. E. **SWEBOK v.3**. [s.l: s.n.].

[33] **IEEE Computer Society**. Disponível em: <https://www.computer.org/web/swebok>. Acesso em: 30 jun. 2017.

[34] ALLEN, J. H. **Software security engineering: a guide for project managers.** [s.l.] Addison-Wesley, 2008.

[35] POPPENDIECK, M.; POPPENDIECK, T. **Lean software development: an agile toolkit.** [s.l.] Addison-Wesley, 2003.

[36] SERAFINO, N. M.; TARNOFF, C.; NANTO, D. K. **U.S. Occupation Assistance: Iraq, Germany, and Japan Compared.** 23 mar. 2006.

[37] OHNO, T. **O sistema Toyota de produção: além da produção em larga escala.** [s.l.] Bookman, 1997.

[38] STÅHL, D.; BOSCH, J. **Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study.** 2013. Disponível em: <http://www.actapress.com/PaperInfo.aspx?paperId=455452>

[39] STÅHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. **Journal of Systems and Software**, v. 87, n. 1, p. 48–59, 2014.

[40] **DIMECC N4S-Program: Finnish Software Companies Speeding Digital Economy - Digile N4S.** Disponível em: <http://www.n4s.fi/en/>. Acesso em: 5 jul. 2017.

[41] **IEEE SA - 730-1980 - IEEE Trial Use Standard for Software Quality Assurance Plans**. Disponível em: <https://standards.ieee.org/findstds/standard/730-1980.html>. Acesso em: 5 jul. 2017.

[42] **International Organization for Standardization**. Disponível em: <https://www.iso.org/home.html>. Acesso em: 5 jul. 2017.

[43] **ISO/IEC TR 19759:2005 - Software Engineering -- Guide to the Software Engineering Body of Knowledge (SWEBOK).** Disponível em: <https://www.iso.org/standard/33897.html>. Acesso em: 5 jul. 2017.

[44] **O que significa distribuição contínua? – Amazon Web Services**. Disponível em: <https://aws.amazon.com/pt/devops/continuous-delivery/>. Acesso em: 5 jul. 2017.

[45] **What Is Agile?** Disponível em: <https://www.forbes.com/sites/stevedenning/2016/08/13/what-is-agile/#4b412ea726e3>. Acesso em: 7 jul. 2017.

[46] SOARES, M. DOS S. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. **Revista Eletrônica de Sistemas de Informação**, v. 3, n. 1, p. 1–8, 2004.

[47] **Scrum: metodologia ágil para gestão e planejamento de projetos**. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso em: 7 jul. 2017.

[48] HUMBLE, J.; FARLEY, D. **Continuous delivery**. [s.l.] Addison-Wesley, 2011.

[49] MEYER, M. Continuous integration and its tools. **IEEE Software**, v. 31, n. 3, p. 14–16, maio 2014.

[50] BECK, A.; UEBERNICKEL, F.; BRENNER, W. **Fit for continuous integration: How organizations assimilate an agile practice.** 20th Americas Conference on Information Systems, AMCIS 2014. **Anais**… 2014. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84905986447&partnerID=tZOtx3y1>

# Apêndice A - Tabela de ferramentas

Esta tabela é uma adaptação da tabela de ferramentas encontrada em [2].

| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| Jira | Gerenciamento de backlog, Rastreamento de bugs, Comunicação e feedback | https://www.atlassian.com/software/jira | [2, 5] | Privado | Indiferente | Atlassian | Paga | Bitbucket, Git, bamboo, Teamcity | indiferente |
| Trello | Geranciamento de backlog | https://trello.com/ | [2, 5] | Privado | Indiferente | Atlassian | Gratis | Slack, Github, Jira, Twitter, Bitbucket | indiferente |
| Confluence | Gerenciamento de backlog, Comunicação e feedback | https://www.atlassian.com/software/confluence | [2] | Privado | Indiferente | Atlassian | Paga | Jira, Teamcity | indiferente |
| Github | Geranciamento de Backlog, Rastreamento de bugs, Controle de versão, Revisão de código, Comunicação e feedback | https://github.com/ | [2, 4, 50] | Privado | Indiferente | Github | Gratis, para projetos públicos | Travis CI, Git | indiferente |
| Agilefant | Geranciamento de backlog | http://www.agilefant.com/ | [2] | Privado | Indiferente | Agilefant LTDA | Paga | Trello, Jira | indiferente |
| Google Docs | Geranciamento de backlog | https://www.google.com/docs/about/ | [2] | Privado | Indiferente | Google | Gratis | Não informado | indiferente |
| Excel | Geranciamento de backlog | https://products.office.com/en-us/excel | [2] | Privado | Indiferente | Microsoft | Paga | Não informado | indiferente |
| Accept360 | Geranciamento de backlog | http://www.accept360.com/ | [2] | Privado | Indiferente | Accept Software | Paga | Confluence | Indiferente |
| Kanbanik | Geranciamento de backlog | http://kanbanik.github.io/kanbanik/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Não informado | indiferente |
| Xcode | IDE | https://developer.apple.com/xcode/ | | Privado | Swift, Objective C | Apple | Gratis | Diversos, quase todos os tipos. | Desktop(Mac), Mobile(IOS) |
| Android Studio | IDE | https://developer.android.com/studio/index.html?hl=pt-br | | Privado | Android | Jetbrains | Gratis | Diversos, quase todos os tipos. | Mobile(Andr oid) |
| Eclipse | IDE | http://www.eclipse.org/ | [8, 12, 50] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Bugzilla | Rastreamento de bugs, Revisão de código | https://www.bugzilla.org/ | [2] | Open Source | Qualquer | Comunidade | Gratis | Eclipse, SVN, Git. | Todos |
| Git | Controle de versão, Implantação | https://git-scm.com/ | [2, 3, 5, 7, 8, 26, 49, 50] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Todos |
| Subversion | Controle de versão | https://subversion.apache.org/ | [2, 26, 50] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Todos |

| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| Bitbucket | Rastreamento de bugs, Controle de versão, Comunicação e feedback | https://bitbucket.org/ | [26] | Privado | Qualquer | Atlassian | Gratis | Jira, Bamboo, Hipchat | Todos |
| TFS | Controle de versão, Comunicação e feedback | https://www.visualstudio.com/tfs/ | [3] | Privado | Qualquer | Microsoft | Gratis | Diversos | Todos |
| Mercurial | Controle de versão | https://www.mercurial-scm.org/ | [2] | Open Source | Qualquer | Comunidade | Gratis | Bugzilla, Maven, Capistrano, Jenkins, Hudson, Git, SVN, Ant, MSBuild. | Todos |
| Perforce | Controle de versão | https://www.perforce.com/ | [2, 3] | Privado | Qualquer | Perforce | Paga | Jira, Jenkins, Eclipse. | Todos |
| Jenkins | Construção, Integração Contínua, Repositório de artefatos, Implantação, Qualidade e performance | https://jenkins.io/ | [2, 3, 5, 8, 26, 39, 49, 50] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Todos |
| Maven | Construção, Repositório de artefatos | https://maven.apache.org/ | [2, 26] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Todos |
| Nant | Construção | http://nant.sourceforge.net/ | [26] | Open Source | .Net | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| MSBuild | Construção | https://github.com/Microsoft/msbuild | [26] | Open Source | .Net | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Ant | Construção | http://ant.apache.org/ | [2, 3, 8, 26] | Open Source | Java | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| CMake | Construção | https://cmake.org/ | [2] | Open Source | C, C++ | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop |
| Make | Construção | https://www.gnu.org/software/make/ | [2, 26] | Open Source | C, C++ | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop |
| GCC | Construção | https://gcc.gnu.org/ | [2] | Open Source | C, C++, Objective C | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Mobile(IOS) |
| Clang | Construção, Qualidade de performance | http://clang.llvm.org/ | [2] | Open Source | C, C++, Objective C | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Mobile(IOS) |
| Ruby on Rails | Construção | http://rubyonrails.org/ | [2] | Open Source | Ruby on Raills | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| Sbt | Construção | http://www.scala-sbt.org/ | [2] | Open Source | Scala, Java Play | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Grunt | Construção | http://gruntjs.com/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| TeamCity | Integração Contínua | https://www.jetbrains.com/teamcity/ | [2, 26, 49] | Privado | Qualquer | Jetbrains | Paga | Diversos, quase todos os tipos. | Todos |
| Bamboo | Integração Contínua | https://br.atlassian.com/software/bamboo | [26, 49] | Privado | Qualquer | Atlassian | Paga | Diversos, quase todos os tipos. | Todos |
| CruiseControl | Integração Contínua | http://cruisecontrol.sourceforge.net/ | [26] | Open Source | Qualquer | Comunidade | Gratis | Não informado | Todos |
| Hudson | Integração Contínua | https://eclipse.org/hudson/ | [26] | Open Source | Qualquer | Comunidade | Gratis | Não informado | Todos |
| Buildbot | Integração Contínua | http://buildbot.net/ | [2] | Open Source | Qualquer | Comunidade | Gratis | Não informado | Todos |
| Artifactory | Repositório de artefatos | https://www.jfrog.com/artifactory/ | [2] | Privado | Qualquer | Jfrog | Paga | Controle de versão, Integração contínua, Construção | Todos |
| Vagrant | Orquestração | https://www.vagrantup.com/ | [2] | Open Source | Indiferente | Comunidade | Gratis | VMware vSphere, Virtualbox, Rackspace, Ansible, Chef, Puppet, Foreman, Steam | Todos |

| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| Docker | Provisionamento de ambientes, Orquestração | https://www.docker.com/ | [3, 7, 11] | Open Source | Indiferente | Comunidade | Gratis | Docker | Todos |
| Virtualbox | Provisionamento de ambientes | https://www.virtualbox.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Mobile(Android), Web, Microserviços |
| VMware vSphere | Provisionamento de ambientes | https://www.vmware.com/products/vsphere | [2] | Privado | Indiferente | Vmware | Paga | Diversos, quase todos os tipos. | Desktop, Mobile(Android), Web, Microserviços |
| Ansible | Provisionamento de Ambientes, Orquestração, implantação | https://www.ansible.com/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Docker, Github, Bitbucket, Vagrant, Jenkins, Teamcity, Virtualbox, Vmware, Foreman.. | Desktop, Mobile(Android), Web, Microserviços |
| Chef | Provisionamento de ambientes, Implantação | https://www.chef.io/ | [2, 5, 7, 11] | Open Source | Indiferente | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Mobile(Android), Web, Microserviços |
| Puppet | Provisionamento de ambientes | https://puppet.com/ | [2, 12, 26] | Open Source | Indiferente | Comunidade | Gratis | Jenkins, Teamcity, Bamboo, hudson, Buildbot, Virtualbox, vSphere, Docker, Vagrant, Foreman. | Desktop, Mobile(Android), Web, Microserviços |
| Foreman | Orquestração | http://theforeman.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Slack, Docker, Ansible, Chef, Puppet, Jenkins, Slack, vSphere, Github, | Desktop, Mobile(Android), Web, Microserviços |
| Capistrano | Implantação | http://capistranorb.com/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Slack, Hipchat | Todos |
| HockeyApp | Implantação | https://www.hockeyapp.net/features/ | [2] | Privado | Indiferente | Microsoft | Paga | Jenkins | Mobile |
| Nunit | Teste de Unidade | https://nunit.org/ | [26] | Open Source | .Net | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| JUnit | Teste de Unidade, Teste de aceitação | http://junit.org/ | [2, 3, 8] | Open Source | Java | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Mockito | Teste de Unidade | http://mockito.org/ | [2] | Open Source | Java | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Jasmine | Teste de Unidade, Teste de interface do usuário | http://jasmine.github.io/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| Mocha | Teste de Unidade | https://mochajs.org/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| Leiningen | Teste de Unidade | http://leiningen.org/ | [2] | Open Source | Clojure | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Midje | Teste de Unidade | https://github.com/marick/Midje | [2] | Open Source | Clojure | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Google Test | Teste de Unidade | https://github.com/google/googletest | [2] | Open Source | C++ | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop |
| Gcov | Teste de Unidade | https://gcc.gnu.org/onlinedocs/gcc/Gcov.html | [2, 26] | Open Source | C, C++, Objective C | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Mobile(IOS) |
| Qt Autotest | Teste de Unidade | https://wiki.qt.io/Qt_Autotest_Environment | [2] | Open Source | Qualquer | Comunidade | Gratis | Diversos, quase todos os tipos. | Todos |
| Specs2 | Teste de Unidade | https://etorreborre.github.io/specs2/ | [2] | Open Source | Scala, Java Play | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Scalatest | Teste de Unidade | http://www.scalatest.org/ | [2] | Open Source | Scala | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |

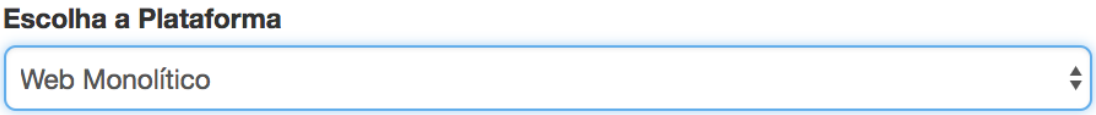
| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| Rspec | Teste de Unidade | http://rspec.info/ | [2] | Open Source | Ruby | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Minitest | Teste de Unidade | https://github.com/seattlerb/minitest | [2] | Open Source | Ruby | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Selenium | Teste de interface do usuário, Teste de aceitação, Qualidade e performance | http://www.seleniumhq.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| Robot Framework | Teste de interface do usuário, Teste de aceitação, Qualidade e performance | http://robotframework.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Jenkins | Web |
| Karma | Teste de Interface do usuário | https://karma-runner.github.io/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| PhantomJS | Teste de Interface do usuário | http://phantomjs.org/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| BrowserStack | Teste de Interface do usuário | https://www.browserstack.com/ | [2] | Privado | Indiferente | BrowserStack | Paga | Não informado | Web |
| Capybara | Teste de Interface do usuário | https://github.com/teamcapybara/capybara | [2] | Open Source | Qualquer | Comunidade | Gratis | Cucumber, Rspec, Minitest, Selenium | Web |
| Cucumber | Teste de interface do usuário, Teste de aceitação | https://cucumber.io/ | [2] | Open Source | .Net, Java, Ruby, Groovy, Javascript, Clojure, Gosu, PHP, Python | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Chai | Teste de aceitação | http://chaijs.com/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| TestNG | Teste de aceitação | http://testng.org/ | [2] | Open Source | Java | Comunidade | Gratis | Diversos, quase todos os tipos. | Desktop, Web, Microserviços |
| Squish | Teste de aceitação, Qualidade e performance | https://www.froglogic.com/squish/gui-testing/ | [2] | Privado | Qualquer | FrogLogic | Paga | TFS, Cruisecontrol, Jenkins, Hudson, Bamboo, Teamcity, Ant, Maven | Web |
| Sonar | Qualidade e performance, Revisão de código | http://www.sonarqube.org/ | [2, 26] | Privado | Qualquer | Sonarqube | Gratis | Diversos, quase todos os tipos. | Todos |
| JsHint | Revisão de código | http://jshint.com/ | [2] | Open Source | Javascript | Comunidade | Gratis | Eclipse | Web |
| Ndepend | Qualidade e performance, Revisão de código | http://www.ndepend.com/ | [26] | Privado | .Net | ZEN PROGRAM LTD | Paga | TFS, Cruisecontrol, Sonar, Jenkins, | Desktop, Web, Microserviços |
| JsLint | Revisão de código | http://www.jslint.com/ | [2] | Open Source | Javascript | Comunidade | Gratis | Diversos, quase todos os tipos. | Web |
| Valgrind | Qualidade e performance | http://valgrind.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Indiferente | Todos |
| FindBugs | Revisão de código | http://findbugs.sourceforge.net/ | [3, 8] | Open Source | Java | Comunidade | Gratis | Eclipse, Maven | Desktop, Web, Microserviços |
| EMMA | Qualidade e performance | http://emma.sourceforge.net/ | [3] | Open Source | Java | Comunidade | Gratis | Eclipse, Maven | Desktop, Web, Microserviços |
| Checkstyle | Revisão de código | http://checkstyle.sourceforge.net/ | [3, 8] | Open Source | Java | Comunidade | Gratis | Bitbucket, Eclipse, Gradle, Jenkins, Sonar, Sbt | Desktop, Android, Web, Microserviços |
| Flexelint | Revisão de código | http://www.gimpel.com/html/lintfaq.htm | [26] | Privado | C, C++ | Gimpel Software | Paga | Eclipse | Desktop |
| Cppcheck | Qualidade e performance | http://cppcheck.sourceforge.net/ | [2] | Open Source | C, C++ | Comunidade | Gratis | Hudson, Jenkins, Mercurial, SVN, Git | Desktop |
| Bullseye | Qualidade e performance | http://www.bullseye.com/ | [2] | Privado | C, C++ | Bullseye | Paga | Não informado | Desktop |
| Testdroid | Qualidade e performance | http://testdroid.com/ | [2] | Privado | Android, IOS | bitbar | Paga | Jenkins, Jira | Mobile |

| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| New Relic | Qualidade e performance | https://newrelic.com/ | [2, 5] | Privado | Java, Ruby, Python, Node | KingHost | Paga | Não informado | Web, Microserviços |
| JMeter | Qualidade e performance | http://jmeter.apache.org/ | [2] | Open Source | Java | Comunidade | Gratis | Não informado | Web, Microserviços |
| Zabbix | Qualidade e performance | http://www.zabbix.com/ | [2, 5] | Open Source | Qualquer | Comunidade | Gratis | Não informado | Web, Microserviços |
| Pingdom | Qualidade e performance | https://www.pingdom.com/ | [2, 5] | Privado | Qualquer | Royal Pingdom | Paga | Slack | Web, Microserviços |
| Gatling | Qualidade e performance | http://gatling.io/ | [2] | Privado | Qualquer | Gatling | Paga | Não informado | Web, Microserviços |
| Google PageSpeed | Qualidade e performance | https://developers.google.com/speed/pagespeed/ | [2] | Privado | Qualquer | Google | Paga | Nginx | Web |
| Ab | Qualidade e performance | http://httpd.apache.org/docs/2.2/programs/ab.html | [2] | Open Source | Qualquer | Comunidade | Gratis | Servidores apache | Web, Microserviços |
| Gerrit | Revisão de código | https://www.gerritcodereview.com/ | [2] | Open Source | Qualquer | Comunidade | Gratis | Git | Todos |
| Crucible | Revisão de código | https://www.atlassian.com/software/crucible | [2] | Privado | Qualquer | Atlassian | Paga | SVN, Git, Mercurial, Perforce, Bitbucket | Todos |
| FishEye | Revisão de código | https://www.atlassian.com/software/fisheye | [2] | Privado | Qualquer | Atlassian | Paga | Jira, Bamboo, Hipchat, Bitbucket | Todos |
| Skype | Comunicação e Feedback | https://www.skype.com/ | [2] | Privado | Indiferente | Microsoft | Gratis | Não informado | Indiferente |
| Slack | Comunicação e Feedback | https://slack.com/ | [2] | Privado | Indiferente | Slack | Gratis | Não informado | Indiferente |
| Flowdock | Comunicação e Feedback | https://www.flowdock.com/ | [2] | Privado | Indiferente | CA Tecnologies | Paga | Não informado | Indiferente |
| Webex | Comunicação e Feedback | https://www.webex.com/ | [2] | Privado | Indiferente | Cisco | Paga | Não informado | Indiferente |
| XMPP | Comunicação e Feedback | https://xmpp.org/ | [2] | Open Source | Indiferente | Comunidade | Gratis | Não informado | Indiferente |
| Hipchat | Comunicação e Feedback | https://hipchat.com/ | [2] | Privado | Indiferente | Atlassian | Gratis | Não informado | Indiferente |
| Lync | Comunicação e Feedback | https://products.office.com/en-us/skype-for-business/ | [2] | Privado | Indiferente | Microsoft | Paga | Não informado | Indiferente |
| SharePoint | Comunicação e Feedback | https://products.office.com/en-us/sharepoint/ | [2] | Privado | Indiferente | Microsoft | Paga | Não informado | Indiferente |
| Outlook | Comunicação e Feedback | https://products.office.com/en-us/outlook/ | [2] | Privado | Indiferente | Microsoft | Gratis | Não informado | Indiferente |
| Google Analytics | Comunicação e Feedback | https://analytics.google.com/ | [2] | Privado | Indiferente | Google | Gratis | Não informado | Indiferente |
| Pagerduty | Comunicação e Feedback | https://www.pagerduty.com/ | [5] | Privado | Indiferente | Pagerduty | Paga | Não informado | Indiferente |
| Flurry | Comunicação e Feedback | https://developer.yahoo.com/ | [2] | Privado | Indiferente | Yahoo | Paga | Não informado | Indiferente |
| Intercom | Comunicação e Feedback | https://www.intercom.io/ | [2] | Privado | Indiferente | Intecom | Paga | Não informado | Indiferente |
| Sitecatalyst | Comunicação e Feedback | http://www.adobe.com/uk/marketing-cloud.html | [2] | Privado | Indiferente | Adobe | Paga | Não informado | Indiferente |
| Qlikview | Comunicação e Feedback | http://www.qlik.com/ | [2] | Privado | Indiferente | Qilk | Paga | Não informado | Indiferente |
| Optimizely | Comunicação e Feedback | https://www.optimizely.com/ | [2] | Privado | Indiferente | Optimizely | Paga | Não informado | Indiferente |

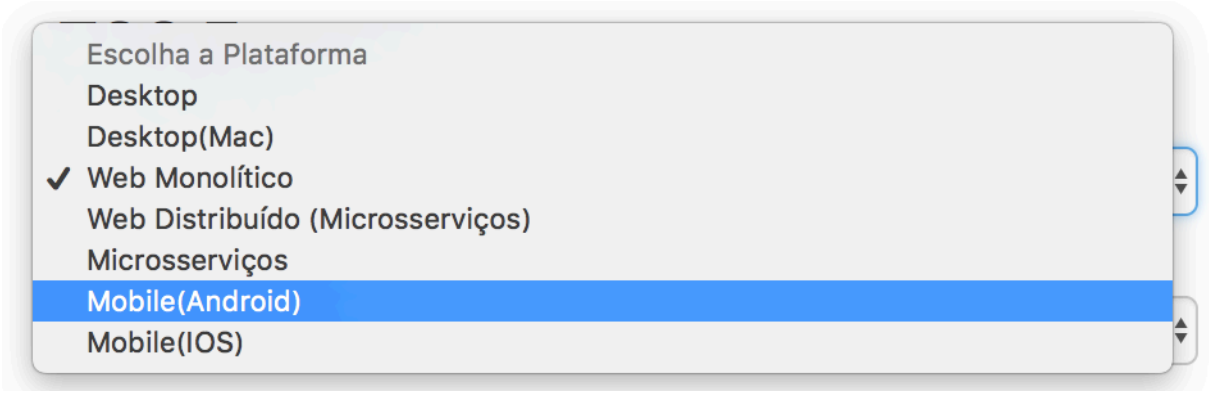
| Nome | Área | Website | Arigos | Tipo de projeto | Linguagem de programação | Mantenedores | Tipo de licença | Integração | Plataforma |
|---|---|---|---|---|---|---|---|---|---|
| Facebook | Comunicação e Feedback | https://www.facebook.com/ | [2] | Privado | Indiferente | Facebook | Gratis | Não informado | Indiferente |
| Twitter | Comunicação e Feedback | https://twitter.com/ | [2] | Privado | Indiferente | Twitter | Gratis | Não informado | Indiferente |

# Apêndice B - Imagens da ferramenta

Seletor de escolha de plataforma da ferramenta

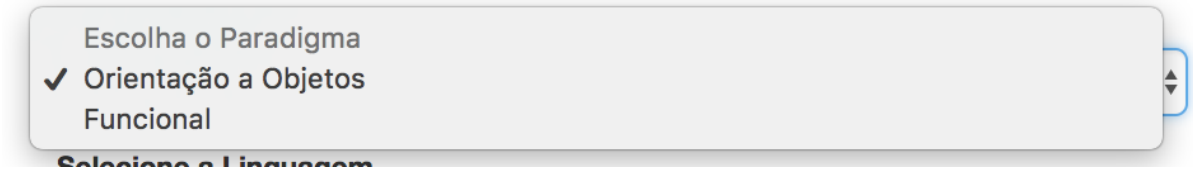



Opções de plataformas na ferramenta



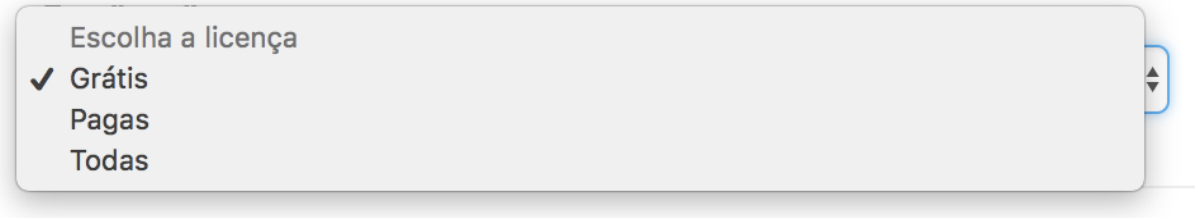

Seletor de escolha de paradigma da ferramenta


Opções de escolha de paradigma na ferramenta



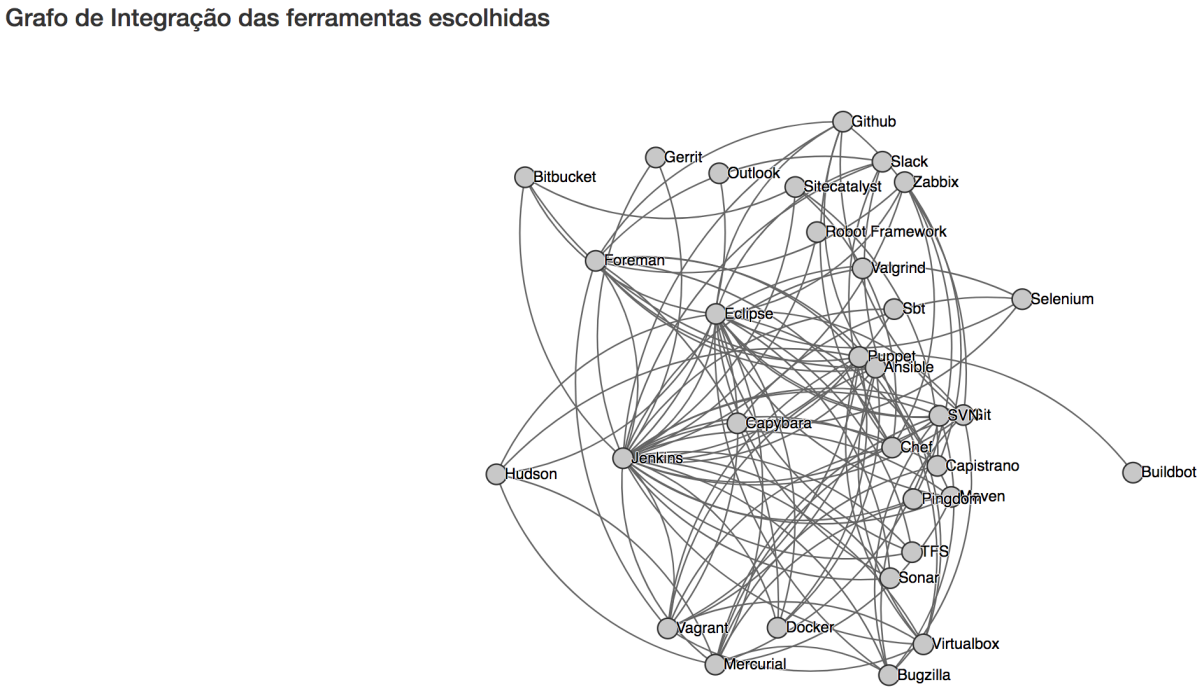
Seletor de escolha de licença da ferramenta


Opções de escolha de licença na ferramenta




Opções de escolha de linguagem na ferramenta


Grafo de integração das ferramentas selecionadas



Resultado de ferramentas selecionadas

# Resultado

## Requisitos

**Gerenciamento de Backlog**

- Google Docs
- Kanbanik

**Rastreamento de bugs**

- Bitbucket
- Bugzilla
- Github

## Desenvolvimento

**IDE**

- Eclipse

**Controle de versão**

- Mercurial
- Git
- SVN
- TFS
- Bitbucket
- Github

**Construção**

- Jenkins
- Maven
- Sbt

**Repositório de artefatos**

- Jenkins
- Maven

**Integração Contínua**

- Jenkins
- Hudson
- CruiseControl
- Buildbot

## Operações

**Provisionamento de ambientes**

- Puppet
- Ansible
- Chef
- Virtualbox
- Docker

**Implantação**

- Jenkins
- Ansible
- Git
- Chef
- Capistrano

**Orquestração**

- Foreman
- Ansible
- Vagrant
- Docker

## Testes

**Testes de unidade**

- Qt Autotest

**Teste de interface do usuário**

- Selenium
- Robot Framework

**Teste de aceitação**

- Selenium
- Robot Framework

## Qualidade

**Qualidade e performance**

- Jenkins
- Zabbix
- Sonar
- Selenium
- Robot Framework
- Valgrind
- Pingdom
- Ab

**Revisão de código**

- Sonar
- Bugzilla
- Github
- Gerrit

## Comunicação

- Sitecatalyst
- TFS
- Bitbucket
- Github
- Slack
- Facebook
- Outlook
- SharePoint
- XMPP

Grafo de integração de todas as ferramentas

**GRAFO - Integração das ferramentas**