



**Tecnologia da Informação e Comunicação**

**A importância da Integração Continuada em Ambientes de Desenvolvimento  
Ágil**

**Dissertação para obtenção do grau de:  
Mestre em Direção Estratégica em Engenharia de Software**

**Apresentado por:  
Cleudson Dias do Nascimento  
BRMDEISW2258608**

**Orientador:  
Prof. Dr. Roberto Fabiano Fernandes**

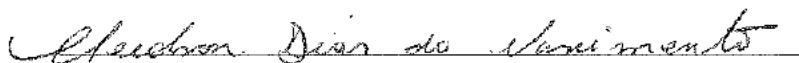
**Brasília, Brasil  
13 de junho de 2019**

**AGRADECIMENTOS:** À minha Núbia que me deu seu apoio e carinho nesta caminhada.

## COMPROMISSO DO AUTOR

Eu, **Cleidson Dias do Nascimento**, com identidade número **1.904.970 SSP/DF**, aluno do programa acadêmico **Mestrado em Direção Estratégica em Engenharia de Software** da **Universidad Europea del Atlantico**, declaro que o conteúdo do trabalho intitulado: **A importância da Integração Continuada em Ambientes de Desenvolvimento Ágil** é o reflexo de meu trabalho pessoal e manifesto que perante qualquer notificação de plágio, cópia ou falta em relação à fonte original, sou diretamente o responsável legal, econômica e administrativamente, isentando o Orientador, a Universidade e as instituições que colaboraram com o desenvolvimento deste trabalho, assumindo as consequências derivadas de tais práticas.

Assinatura:



**Brasília, 13 de junho de 2019.**

## [Autorização voluntária]


À

Fundação Universitária Iberoamericana – FUNIBER

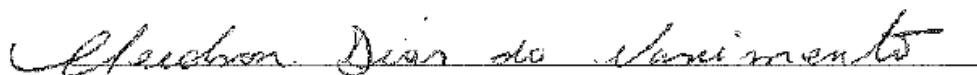
Att: Direção Acadêmica

Venho por meio deste, autorizar a publicação eletrônica da versão aprovada do meu Projeto Final com título: A importância da Integração Continuada em ambientes de desenvolvimento ágil, no Campus Virtual e em outras mídias de divulgação eletrônica desta Instituição.

Informo abaixo os dados para descrição do trabalho:

<b>Título:</b>	A importância da Integração Continua em Ambientes de Desenvolvimento Ágil	
<b>Autor:</b>	Cleudson Dias do Nascimento e Roberto Fabiano	
<b>Resumo</b>	O objetivo deste estudo e resolução de caso é demonstrar a importância da integração continuada como ferramenta fundamental para a adequação das equipes ágeis no alcance de maturidade do agilômetro é para isso iremos fazer uma pesquisa aplicada junto às equipes ágeis para identificação de processos que possam ser automatizados a fim de auxiliar de deixar mais eficaz a entrega de valor sem acarretar trabalhos extras para a equipe.	
<b>Programa</b>	Mestrado em Direção Estratégica em Engenharia de Software	
<b>Palavras-chave</b>	Equipes Ágeis; Integração Continuada; Agilômetro; Entrega Continuada de Valor.	
<b>Contato</b>	<a href="mailto:cleidsondias@hotmail.com">cleidsondias@hotmail.com</a>	

Atenciosamente,



## RESUMO

A partir do problema de pesquisa proposto: “É possível que implantação da integração continuada em ambientes de desenvolvimento ágil possibilite a entrega de produtos íntegros e dentro dos prazos acordados?”, a presente proposta de pesquisa fará uma análise aprofundada dos mecanismos que podem ser utilizados em ambientes de desenvolvimento de maneira a proporcionar maior produtividade e melhor desempenho na entrega do produto. Para tanto, far-se-á análise de extensa bibliografia em T.I e ferramentas com o fim de atender a Integração Contínua, bem como estudo de caso a partir da implantação de ferramentas que possam auxiliar a equipe ágil no cotidiano da produção em uma empresa desenvolvedora de software para o mercado de seguros. Deve-se entender a integração contínua, e os ambientes ágeis, como processos e procedimentos internos que possibilitam uma nova forma nas relações em ambientes que necessitem de uma relação mais dinâmica entre os colaboradores. As conclusões sugerem que a Integração Contínua aplicada em ambiente ágil consegue elevar a qualidade dos serviços, bem como do produto, dentro dos prazos acordados. Também consiste em uma prática de desenvolvimento de software que auxilia na execução de testes unitários, validação estática do código e a entrega automática em ambientes de sistemas integrados, garantindo uma velocidade na entrega de valor. A aliança dessa prática com os frameworks que se baseiam na metodologia ágil é um importante instrumento no apoio às equipes ágeis. A partir do objetivo geral, “Demonstrar a importância da integração contínua junto a equipes de desenvolvimento ágil”, e com base em estudo de caso aplicado, as conclusões demonstraram que a implantação de ambiente de integração em ambientes ágeis eleva a qualidade dos serviços internos e do produto entregue ao cliente. No entanto, pode-se incrementar ainda mais integração como, por exemplo, o uso de I.A e mesmos o uso de mecanismos de avaliação de qualidade como o uso do diagrama de ISHIKAWA.

**Palavras-chave:** Equipes Ágeis; Integração Continuada; Agilômetro; Entrega Continuada de Valor; ISHIKAWA.

## ABSTRACT

Based on the proposed research problem: "Is it possible that the implementation of continuous integration in agile development environments allows the delivery of products within the agreed timeframes?", This research proposal will make an in-depth analysis of the mechanisms that can be used in development environments to provide greater productivity and better performance in product delivery. In this way, an extensive IT bibliography and tools will be analyzed to participate in Continuous Integration, as well as a case study based on the implementation of tools that can help the agile team in the daily production in a company that develops software for to the insurance market. Continuous integration and agile environments should be understood as internal processes and procedures that enable a new form of relationship in environments that require a more dynamic relationship among employees. The conclusions suggest that the Continuous Integration applied in an agile environment can increase the quality of services, as well as the final product, within the agreed deadlines. It also consists of a software development practice that assists in performing unit testing, static code validation, and automatic delivery in integrated systems environments, ensuring speed of delivery of value. The alliance of this practice with the work tables that are based on the agile methodology is an important instrument in the support to the agile teams. From the general objective, "Demonstrating the importance of continuous integration with agile development teams", and based on an applied case study, the conclusions showed that the implementation of integration environment in agile environments raises the quality of internal services and of the final product delivered to the customer. However, further integration, such as the use of I.A and even the use of quality assessment mechanisms such as the use of the ISHIKAWA diagram, can be further enhanced.

**Keywords:** Agile Team; Continuous Integration; Agilometer; Continuous Value Delivery; ISHIKAWA.

## LISTA DE ILUSTRAÇÕES

Figura 1. Pontos Chaves do Agilômetro.....	29
Figura 2. SaaS, PaaS e IaaS e suas aplicações. ....	33
Figura 3. Ciclo de Integração Contínua.....	38
Figura 4. Arquitetura de um sistema de controle de versão distribuído. ....	41
Figura 5. Arquitetura de um sistema de controle de versão centralizado. ....	42
Figura 6. Ciclo de integração entre Desenvolvedores e Operadores. ....	43
Figura 8. Insurtech termo que oriunda de duas palavras Insurance e Technology.....	51
Figura 9. Desenvolvimento nos principais mercados de seguros em 2017.....	53
Figura 10. Visão Geral da Plataforma. ....	54
Figura 11. Site da Kakau, uma seguradora exclusiva.....	54
Figura 12. Site da Bidu. Plataforma com foco em corretores únicos. ....	55
Figura 13. Oceano azul pretendido.....	56
Figura 14. Cadeia de atendimento no mercado de seguros. ....	56
Figura 15. Ferramentas que não atendem a cadeia de atendimento completa.....	57
Figura 16. Plataformas incompletas. ....	57
Figura 17. Tecnologia da Plataforma. ....	58
Figura 18. Integração B.I.....	58
Figura 19. O Ambiente de Produção. ....	59
Figura 20. O Time. ....	60
Figura 21. <i>Learning Solutions Process</i> . ....	61
Figura 22. Processo fim a fim do Scrum. ....	61
Figura 22. Plataforma ZIM para os corretores de seguros. ....	64
Figura 23. Tela inicial do Jenkins.....	69
Figura 24. Tela inicial do Flyway.....	71
Figura 25. Tela informacional sobre a Qualidade do Projeto.....	73
Figura 26. Arquitetura do SonarQube. ....	73
Figura 27. Gerenciamento do Código Fonte com o Jenkins e o SonarQube.....	76
Figura 28. Gerenciamento dos repositórios.....	78
Figura 29. Lista de repositórios compatíveis do Artifactory. ....	79
Figura 30. Ciclo de desenvolvimento sem Integração Continua. ....	82
Figura 31. Ciclo de desenvolvimento utilizando Integração Continua.....	83
Figura 32. Integração Jenkins, Sonar e Artifactory. ....	84
Figura 33. Fluxo entre o Git, Jenkins, Maven, SonarQube e Artifactory.....	84
Figura 34. DevOps e a Nuvem computacional.....	85
Figura 35. Diagrama de Causa-e-Efeito. ....	87

Figura 36. As 5 principais esferas do desenvolvimento de software.....	96
--	----



## **LISTA DE QUADROS**

Quadro 1. Manifesto Ágil.....	27
Quadro 2. Parâmetros para avaliação dos Colaboradores .....	89
Quadro 3. Parâmetros para avaliação do Gerenciamento.....	90
Quadro 4. Demonstrativo de IC em ambiente de desenvolvimento. ....	91

## SUMÁRIO

<b>CAPÍTULO 1 - INTRODUÇÃO .....</b>	<b>14</b>
<b>1.1. Tema/Área de pesquisa e contextualização.....</b>	<b>14</b>
<b>1.2. Problema de pesquisa.....</b>	<b>15</b>
<i>1.2.1. Justificativa .....</i>	<i>15</i>
<b>1.3. Objetivo geral .....</b>	<b>16</b>
<b>1.4. Objetivos específicos .....</b>	<b>16</b>
<b>1.5. Resultados esperados .....</b>	<b>16</b>
<b>1.6. Estrutura da dissertação.....</b>	<b>16</b>
 <b>CAPÍTULO 2 - MARCO TEÓRICO .....</b>	 <b>18</b>
<b>2.1. Gestão da Qualidade .....</b>	<b>18</b>
<i>2.1.1. A Qualidade nos dias de hoje.....</i>	<i>18</i>
<i>2.1.2. Gestão da Qualidade nas Organizações .....</i>	<i>19</i>
<i>2.1.3. A gestão da Qualidade como Diferencial Competitivo .....</i>	<i>20</i>
<i>2.1.4. Desafios de implantar um Sistema de Gestão da Qualidade .....</i>	<i>20</i>
<b>2.2. As sete ferramentas da qualidade (ISHIKAWA) .....</b>	<b>21</b>
<i>2.2.1. Carta de Controle e gráficos .....</i>	<i>21</i>
<i>2.2.2. Diagrama de Causa-e-Efeito .....</i>	<i>21</i>
<i>2.2.3. Diagrama de Correlação.....</i>	<i>22</i>
<i>2.2.4. Estratificação .....</i>	<i>22</i>
<i>2.2.5. Folha de verificação.....</i>	<i>22</i>
<i>2.2.6. Gráfico de Pareto .....</i>	<i>22</i>
<i>2.2.7. Histograma.....</i>	<i>22</i>
<b>2.3. Código Aberto.....</b>	<b>22</b>
<i>2.3.1. Distribuição livre.....</i>	<i>23</i>
<i>2.3.2. Código fonte .....</i>	<i>23</i>
<i>2.3.3. Trabalhos Derivados .....</i>	<i>24</i>

2.3.4.	<i>Integridade do autor do código fonte</i>	24
2.3.5.	<i>Não discriminação contra pessoas ou grupos</i>	24
2.3.6.	<i>Distribuição da Licença</i>	24
2.3.7.	<i>Licença neutra em relação à tecnologia</i>	24
2.4.	<b>Licença Livre</b>	25
2.4.1.	<i>Licenças Permissivas</i>	25
2.4.2.	<i>Licenças Recíprocas Totais</i>	26
2.4.3.	<i>Licenças Recíprocas Parciais</i>	26
2.5.	<b>Manifesto Ágil</b>	27
2.6.	<b>Agilômetro</b>	28
2.6.1.	<i>Flexibilidade com relação às entregas</i>	29
2.6.2.	<i>Nível de colaboração</i>	29
2.6.3.	<i>Facilidade de comunicação</i>	29
2.6.4.	<i>Capacidade de trabalhar com entregas iterativas e incrementais</i>	30
2.6.5.	<i>Fatores ambientais</i>	30
2.6.6.	<i>Aceitação dos métodos ágeis</i>	30
2.7.	<b>Computação em nuvem</b>	31
2.7.1.	<i>Principais benefícios da computação em nuvem</i>	31
2.7.1.1.	<i>Custo</i>	31
2.7.1.2.	<i>Velocidade</i>	31
2.7.1.3.	<i>Escala global</i>	32
2.7.1.4.	<i>Produtividade</i>	32
2.7.1.5.	<i>Desempenho</i>	32
2.7.1.6.	<i>Confiabilidade</i>	32
2.7.2.	<i>Tipos de serviços de nuvem: IaaS, PaaS e SaaS</i>	33
2.7.2.1.	<i>IaaS (Infraestrutura como serviço)</i>	33
2.7.2.2.	<i>PaaS (plataforma como serviço)</i>	34
2.7.2.3.	<i>SaaS (software como serviço)</i>	34
2.7.3.	<i>Tipos de implantação em nuvem: pública, privada e híbrida</i>	34
2.7.3.1.	<i>Nuvem pública</i>	34
2.7.3.2.	<i>Nuvem privada</i>	34
2.7.3.3.	<i>Nuvem híbrida</i>	35
2.8.	<b>Gerenciamento de Dependências</b>	35

<b>2.9. Integração Contínua .....</b>	<b>35</b>
2.9.1. <i>Manter um repositório de código.....</i>	36
2.9.2. <i>Automatize a construção.....</i>	36
2.9.3. <i>Automatize teste antes da construção.....</i>	36
2.9.4. <i>A equipe se compromete com a linha de base todos os dias .....</i>	36
2.9.5. <i>Toda entrega para baseline deve ser construída.....</i>	37
2.9.6. <i>Mantenha a compilação rápida .....</i>	37
2.9.7. <i>Teste em um clone do ambiente de produção.....</i>	37
2.9.8. <i>Facilite a obtenção das entregas mais recentes .....</i>	37
2.9.9. <i>Automatizar a implantação.....</i>	38
2.9.10. <i>A necessidade da integração contínua .....</i>	38
2.9.11. <i>O Funcionamento da integração contínua .....</i>	38
2.9.12. <i>Custos e benefícios .....</i>	39
<b>2.10. Controle de Versão.....</b>	<b>40</b>
2.10.1. <i>Controle de versão distribuída .....</i>	41
2.10.2. <i>Controle de versão centralizado.....</i>	42
<b>2.11. DevOps .....</b>	<b>42</b>
2.11.1. <i>Integração entre áreas .....</i>	43
2.11.2. <i>Simplificação de processos .....</i>	43
2.11.3. <i>Automação de tarefas.....</i>	43
2.11.4. <i>Racionalização de processos.....</i>	43
2.11.5. <i>Modernização da TI da empresa .....</i>	44
2.11.6. <i>Estímulo à colaboração.....</i>	44
2.11.7. <i>Empoderamento dos times de TI .....</i>	44
2.11.8. <i>Elasticidade e escalabilidade .....</i>	44
<b>CAPÍTULO 3 - PROCEDIMENTOS METODOLÓGICOS .....</b>	<b>46</b>
<b>3.1. Tipo de pesquisa .....</b>	<b>46</b>
<b>3.2. Coleta de dados.....</b>	<b>46</b>
<b>3.3. Técnicas e ferramentas de análise.....</b>	<b>47</b>
<b>3.4. Análise e discussão dos resultados .....</b>	<b>47</b>
3.4.1. <i>Análise inicial do Agilômetro .....</i>	47
3.4.2. <i>Análise da Folha de Verificação .....</i>	48

3.4.3. <i>Análise da Avaliação pela Equipe</i> .....	48
3.4.4. <i>Resultado das Análises</i> .....	49
<b>CAPÍTULO 4 - ESTUDO DE CASO</b> .....	51
4.1. Contexto da organização .....	51
4.2. Análise do mercado de seguros .....	52
4.3. A legislação Brasileira.....	53
4.4. Mitigação de riscos externos.....	53
4.5. Tipos de produtos oferecidos aos corretores.....	54
4.5.1. <i>Seguradoras Exclusivas</i> .....	54
4.5.2. <i>Corretora Única</i> .....	54
4.6. Oceano Azul pretendido .....	55
4.7. Cadeia de atendimento no mercado de seguros.....	56
4.8. Tecnologia da Plataforma.....	57
4.9. Ambiente de Desenvolvimento .....	58
4.10. O Ambiente de Produção.....	59
4.11. Esquadras de trabalho ( <i>Squads</i> ) .....	59
4.12. O Time.....	60
4.13. Gerenciamento do projeto .....	60
4.14. Metodologia e dinâmica dos trabalhos na Empresa do Estudo de Caso .....	61
4.15. O produto .....	62
<b>CAPÍTULO 5 - FERRAMENTAS PROPOSTAS</b> .....	65
5.1. Maven .....	65
5.1.1. <i>Project Object Model</i> .....	66
5.1.2. <i>Plugins</i> .....	66

5.1.3. Ciclos de vida de Construção .....	66
5.1.3.1. Process-resources (processar recursos) .....	67
5.1.3.2. Compile (compilar) .....	67
5.1.3.3. Process-test-resources (processar recursos de teste).....	67
5.1.3.4. Test-compile (testar compilação).....	67
5.1.3.5. Test (testar).....	67
5.1.3.6. Package (empacotar).....	67
5.1.3.7. install (instalar) .....	68
5.1.3.8. Deploy (implantar) .....	68
5.1.4. Dependências .....	68
5.1.5. Profiles.....	68
5.1.6. Tipos de perfil de compilação .....	69
5.2. Jenkins.....	69
5.2.1. Unidade de trabalho (Job) .....	70
5.2.2. Fluxo de valor (Pipeline) .....	70
5.2.3. Plugins.....	70
5.2.4. Vantagens .....	70
5.3. Flyway .....	70
5.3.1. Migração.....	71
5.3.2. Chamadas de Retorno (Callbacks) .....	72
5.3.3. Manuseio de Erros (Error Handlers).....	72
5.3.4. Somente o necessário (Dry Runs).....	72
5.4. SonarQube .....	72
5.4.1. Analisadores .....	74
5.4.2. Métricas .....	74
5.4.3. SonarQube e integração contínua.....	76
5.5. Git .....	76
5.5.1. Comunicação.....	77
5.6. Gitlab.....	78
5.7. Artifactory.....	79
5.7.1. Principais vantagens .....	79
5.7.2. Por que utilizar o Artifactory?.....	80

<b>5.8. Microsoft Azure.....</b>	<b>80</b>
<b>5.8.1. <i>Quais são as vantagens do Microsoft Azure?</i> .....</b>	<b>80</b>
 <b>CAPÍTULO 6 - SOLUÇÃO PROPOSTA .....</b>	<b>82</b>
<b>6.1. Ciclo de Desenvolvimento de Sistema sem Integração Continua.....</b>	<b>82</b>
<b>6.2. Ciclo de Desenvolvimento de Sistema com Integração Continua .....</b>	<b>83</b>
 <b>CAPÍTULO 7 - CONCLUSÕES .....</b>	<b>86</b>
<b>7.1. Quando ao problema de pesquisa .....</b>	<b>86</b>
<b>7.2. Quanto ao objetivo geral.....</b>	<b>86</b>
<b>7.3. Quanto aos objetivos específicos.....</b>	<b>93</b>
<b>7.4. Sugestões .....</b>	<b>94</b>
 <b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>97</b>

# CAPÍTULO 1 - INTRODUÇÃO

## 1.1. Tema/Área de pesquisa e contextualização

Sommerville (2011), referência na engenharia de software, ensina que software não é apenas um programa, mas sim, um programa adjunto de um conjunto de documentos funcionais e não funcionais que nos permitem resolver um problema. A compreensão do que é um software nos ajuda a entender a importância da implantação de metodologias (cascata, incremental e espiral).

Ao longo dos últimos anos, vem despontando no mercado e no mundo acadêmico metodologias mais ágeis que as tradicionais que, alimentadas por bons resultados, tem se tornado cada vez mais utilizada por equipes que buscam a redução de desperdícios além de proporcionar uma maior eficiência na entrega valor ao usuário.

Empresas têm utilizado essas novas metodologias como diferencial estratégico sendo elas: empresas que estão passando por uma transformação digital, as empresas que já são digitais e as empresas com bases tecnológicas. Empresas como Google, Yahoo!, Microsoft, IBM, Cisco, Symantec e Siemens os têm utilizados (Gomes, 2014, p. 2).

As bases dessas novas metodologias são o chamado “Manifesto Ágil”, que surgiu mais precisamente no ano de 2001, constituído por um conjunto de valores e princípios, sendo que sua maior importância é dar base para criação de *frameworks*<sup>1</sup> que focam na entrega de valor ao usuário: “Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor” (Manifesto Ágil, 2001). Assim, o manifesto tem despontado como um diferencial no conceito de elaboração de software.

No passado, os membros das equipes trabalhavam de forma mais isolada e só juntavam a maioria de suas criações ou alterações quando o trabalho estava quase todo concluído. Deste modo, a junção das alterações e adições de novos comportamentos era difícil e demorada, além de resultar no acúmulo de comportamentos inesperados na aplicação e impossibilitando a correção pontual por longos períodos. Estes fatores impossibilitavam a distribuição de atualizações rápidas, afetando a entrega de valor para o usuário.

A integração contínua consiste em uma prática de desenvolvimento de software que auxilia na execução de testes unitários, validação estática do código e a entrega automática em ambientes de sistemas integrados, garantindo uma velocidade na entrega de valor. A aliança dessa prática com os frameworks que se baseiam na metodologia ágil é um importante instrumento no apoio as equipes ágeis.

---

<sup>1</sup> Framework: Trata-se de uma estrutura conceitual básica de procedimentos, técnicas e processos com o objetivo de predefinir entregáveis comuns para diversos tipos de projetos.



Este estudo tem como objetivos realizar uma pesquisa aplicada a fim de propor um conjunto de ferramentas que possam auxiliar a equipe na entrega continuada de valor, responder à mudança sem injetar erros e, não menos importante, demonstrar a importância da Integração Continua como um meio de auxiliar as metodologias ágeis na entrega continuada de valor para o usuário.

## **1.2. Problema de pesquisa**

Com o advento de metodologias de desenvolvimento de software mais ágeis as equipes conseguem produzir o dobro pela metade do tempo, com a outra fatia de tempo que sobra permite que as equipes possam experimentar novas ideias ou propor novas melhorias gerando um diferencial no produto ou no serviço, e assim, agregando bons resultados.

A cada iteração permite ver os projetos crescendo e, diferentemente das metodologias tradicionais, logo nas primeiras entregas já temos uma aplicação funcionando. Os *frameworks* que se baseiam em metodologias ágeis vieram para fazer com que as entregas agreguem valor ao negócio e consequentemente deixar o usuário cada vez mais satisfeito.

O problema começa a ocorrer em ambientes complexos, demonstrado em documentação que impossibilita a atualização e entrega de novas funcionalidades; a alta complexidade na construção de artefatos afeta nos prazos previamente acordados; a contínua dependência de outras equipes não ágeis impossibilita a entrega de valor para o usuário; a falta de teste funcional e teste de usabilidade automatizado permitem que erros obscuros cheguem à produção; a falta de análise do artefato produzido gera uma pouca visão de qualidade; a falta de rastreio de versões em homologação e produção, fatalmente fará sua equipe subir algo que não deveria. Tudo isso acarretará na demora da entrega de resultados gerando perda da credibilidade e falta de alinhamento com a equipe além de insatisfação com os patrocinadores do projeto.

A partir dessa problematização, o problema de pesquisa que se propõe situa-se da seguinte forma: “É possível que implantação da integração continuada em ambientes de desenvolvimento ágil possibilite a entrega de produtos íntegros e dentro dos prazos acordados?”

### **1.2.1. Justificativa**

Em minha experiência de pouco mais de dez (10) anos em projetos de *software*, no âmbito governamental e iniciativa privada, um após o outro, identifiquei a dificuldade existente no processo de entrega de valor ao produto, mesmo utilizando *frameworks* baseados nas metodologias ágeis. Geralmente, esses problemas começam a ocorrer na linha de produção, acarretando atrasos em toda a cadeia produtiva, gerando significantes atrasos.

Empresas que estão passando por transformações digitais, as empresas digitais e com bases tecnológicas podem implantar um processo de integração contínua junto com sua metodologia ágil

de desenvolvimento de software, utilizando este estudo como base técnica ou teórica, entretanto, esta pesquisa contribui, mas não esgota o assunto sobre o conjunto de ferramentas que possam automatizar o processo da entrega continuada de valor ao usuário aliado a metodologias ágeis.

### 1.3. Objetivo geral

Demonstrar a importância da integração contínua junto a equipes de desenvolvimento ágil.

### 1.4. Objetivos específicos

- Aprofundar e entender o nível de aceitação dos métodos ágeis pela organização.
- Indicar um conjunto básico de ferramentas que possam auxiliar a entrega de valor.

### 1.5. Resultados esperados

Já existe uma solução para problemas que envolvam entrega de valor, que é a Integração Contínua. Ela nos permite ter uma visão da saúde dos artefatos produzidos, auxilia na automatização da entrega dos artefatos, a visão da versão do artefato em produção, entretanto, apesar do conceito muitas empresas nem sempre tem a real visão de um conjunto de benefícios e muito menos do conjunto de possíveis ferramentas que possam utilizar como um aliado na entrega continuada de valor. Dessa forma tem-se como resultado esperados responder as seguintes perguntas:

Quais os conjuntos de ferramentas que podemos implantar para obtermos agilidade na entrega de valor pelas equipes ágeis?

Quais os reais benefícios que as organizações podem ter na adoção de tais ferramentas e como elas permitem o processo de entrega de valor ao cliente?

### 1.6. Estrutura da dissertação

A dissertação que aqui se apresenta será composta dos seguintes capítulos:

- **CAPÍTULO 1 – INTRODUÇÃO:** Neste capítulo serão abordados o objetivo geral e os objetivos específicos desta dissertação.
- **CAPÍTULO 2 – MARCO TEÓRICO:** Neste capítulo faremos a revisão conceitual a fim de dar apoio aos objetivos específicos de construir um conjunto básico de ferramentas que possa auxiliar a entrega de valor que foi apresentado.
- **CAPÍTULO 3 – PROCEDIMENTOS METODOLÓGICOS:** Neste capítulo será abordado o procedimento metodológico com o objetivo de aprofundar e entender o nível de aceitação dos métodos ágeis pela organização.

- **CAPÍTULO 4 – ESTUDO DE CASO:** No capítulo 4 será apresentado o estudo de caso em um ambiente real a fim de identificar a proposta de valor que será entregue ao usuário.
- **CAPÍTULO 5 – FERRAMENTAS PROPOSTAS:** Neste capítulo serão apresentadas as ferramentas que visam atender ao objetivo específico de construir um conjunto básico de mecanismos que possam auxiliar a entrega de valor.
- **CAPÍTULO 6 – SOLUÇÃO PROPOSTA:** Neste capítulo será apresentado um fluxo que permitirá a adequação do ambiente estudado e, não obstante, atender o objetivo específico de identificar os pontos que possam ser automatizados utilizando um conjunto de ferramentas.
- **CAPÍTULO 7 – CONCLUSÕES:** São apresentadas as conclusões do trabalho a fim de atender o objetivo geral, e responder ao problema de pesquisa, ou seja, demonstrar a importância da integração contínua junto a equipes de desenvolvimento ágil.

## CAPÍTULO 2 - MARCO TEÓRICO

Para responder qual o objetivo de demonstrar a importância da integração continuada junto a equipes de desenvolvimento ágil, quanto à qualidade do produto como um diferencial estratégico de mercado e que possa agregar valor à sociedade, a gestão da qualidade será abordada como um conceito maior para esse objetivo aliado ao “agilômetro” que é uma metodologia que tem como finalidade gerar uma métrica de maturidade da instituição quanto à implantação e uso de metodologias ágeis.

O marco teórico para atender aos objetivos específicos será a análise do próprio manifesto ágil e, na sequência, iremos nos aprofundar um pouco mais no conceito de código aberto aliado com a licença livre, peças fundamentais para nortear a busca de ferramentas. Consta da nossa abordagem, o conceito de nuvem computacional para instanciar as ferramentas, bem como será tratado do conceito de Integração continua, Gerenciamento de dependência, Controle de versão e DevOps<sup>2</sup>.

### 2.1. Gestão da Qualidade

Gestão da Qualidade e sua evolução são amplas e essa preocupação vem de tempos primórdios. Por volta de 2150 a.c. O código de Hamurabi, já demonstrava uma preocupação com a durabilidade e funcionalidade das habitações produzidas na época, tanto que, se fosse construído um imóvel que, não fosse o suficientemente seguro para atender a sua finalidade e desabasse, o construtor seria imolado.

Na época dos fenícios, os fabricantes de determinados produtos que não estivessem dentro das especificações governamentais com perfeição, teriam suas mãos amputadas.

Já na sua época, os romanos usavam técnicas de pesquisas altamente sofisticadas, buscando controlar as terras rurais incorporadas ao império, desenvolveram padrões de Qualidade, métodos de medição e ferramentas específicas para execução desses serviços.

Pode-se também falar dos avançados procedimentos adotados pela França no reinado de Luís XIV, que detalhavam os critérios para escolher os fornecedores e instruções para supervisão do processo de fabricação das embarcações usadas na época. Assim, podemos perceber que, a preocupação com a Qualidade, vem trilhando um longo caminho ao longo da história da humanidade.

#### 2.1.1. A Qualidade nos dias de hoje

---

<sup>2</sup> é uma metodologia de desenvolvimento de software que utiliza a comunicação para integrar desenvolvedores de software e profissionais de infraestrutura de TI. Ver tópicos 2.11 a 2.11.8

Na atualidade, a qualidade já é um critério adotado por grande parte das organizações preocupadas com a eficiência e resultados de seus processos internos, a palavra qualidade abrange a visão geral no sentido de amplitude e integração a diversas áreas do conhecimento humano, porém, só recentemente ela surgiu como função de gerência. Na ideia original, tal função era relativa e voltada para a inspeção; atualmente as atividades relacionadas com a Qualidade se ampliaram e são consideradas essenciais para o sucesso organizacional. O que vai ao encontro de da percepção mais consciente quanto às mudanças no modo que se oferta um produto ou serviço. Ressalta-se que o cliente é a figura principal de todo processo organizacional, com todas as ações e decisões empresariais voltadas para o consumidor; para suas necessidades e expectativas, a fim de satisfazê-los completamente.

Nos últimos anos a palavra qualidade é vista como um propulsor de sucesso para as organizações que em busca do aperfeiçoamento, procuram adequar-se as demandas do mercado com o objetivo de melhorar o máximo suas ações, atrair novos clientes, retendo os antigos e somando-se com esse processo um diferencial para alcançar seu cliente.

### ***2.1.2. Gestão da Qualidade nas Organizações***

Existem muitas definições para o termo qualidade, e dentre tantas definições e conceitos, estão as definições com foco no consumidor, com suas percepções e seus diversos valores, como: cultura, produto, serviço prestado, as suas necessidades e expectativas. Em relação a produtos e serviços também há uma diversidade de definições, com o escopo em comum de que o produto deve estar em conformidade com as exigências dos clientes e seu valor agregado.

Qualidade não é uma ideia ou coisa concreta, mas uma terceira entidade independente das duas e embora não se possa defini-la, sabe-se o que ela é. Segundo Marshall Jr, Cierco, Rocha, Mota, & Leusin (2008) “existem cinco abordagens principais para a definição de Qualidade: transcendental, baseada no produto, baseada no usuário, baseada na produção e baseada no valor”.

- **Transcendental**

É buscar um alto nível de excelência nos produtos ou serviços que se quer adquirir, uma condição que implica ótima qualidade, distinta da má qualidade, ou seja, atingir ou buscar o padrão mais alto em vez de se contentar com o malfeito ou fraudulento.

- **Baseada no produto**

A qualidade é constituída de variáveis e atributos que podem ser medidos e controlados, ou seja, os produtos de uma mesma família têm que apresentar as mesmas características, não podendo haver diferenciação entre eles (Blog da Qualidade, 2012).

- **Baseada no usuário**

A qualidade está baseada nos desejos do consumidor, nas suas expectativas, ou seja, consiste na capacidade de satisfazer desejos.

- **Baseada na produção**

A linha de produção do produto deve estar nas especificações e normas de Qualidade e o grau em que o produto específico está de acordo com o projeto ou especificação.

- **Baseada no valor**

São as razões pelas quais o cliente quer comprar um produto de Qualidade, e que realmente se adéque as suas reais necessidades, ou seja, o melhor para certas condições do cliente. Pode-se concluir que mais seguro para se definir Qualidade em uma empresa é através de sua política de Gestão, podendo incluir mais de uma das abordagens indicadas.

### ***2.1.3. A gestão da Qualidade como Diferencial Competitivo***

Nas abordagens literárias sobre Gestão da Qualidade, enfatiza-se muito qual a importância da qualidade para as organizações, como diferencial competitivo. As ideias propostas sobre Gestão da Qualidade evidenciam que é na administração da organização que reside a responsabilidade primária pela qualidade, devendo criar em todos colaboradores o orgulho e a satisfação de obter o conhecimento e o prazer de aplicar esses atributos. De acordo com essa fundamentação, enfatiza-se a importância de mudança no foco administrativo para a melhoria da Qualidade, definir o que deve ser mudado e como deve ser mudado.

O controle da Qualidade foi aprimorado por meio da utilização de técnicas estatísticas para coletar dados sobre o processo, e só depois seria possível observar a reação do sistema sobre as mudanças na empresa. Foi o norteador do conhecimento a respeito da qualidade e teve como principal ideia, servindo de agente libertador do poder de motivação, “a constância de propósito”.

Da mesma forma, tem uma maior preocupação com a estrutura organizacional e se faz necessário melhorar a Qualidade e as formas de desenvolvimento, da cultura e da história de política da empresa, sendo fundamental que o comprometimento ocorra a partir da alta administração. Introduziu o conceito de que o controle da Qualidade deveria ser responsabilidade de toda a organização, foi um dos formuladores do conceito de controle da Qualidade total (*total quality control* – TQC), onde sua premissa básica é que a Qualidade está ligada a todas as funções e atividades da organização, e não apenas à fabricação ou à engenharia, e por ser um instrumento estratégico todos os trabalhadores devem ser responsáveis.

### ***2.1.4. Desafios de implantar um Sistema de Gestão da Qualidade***

São requisitos gerais para programar e implantar um sistema de qualidade, com modelo de processo e de integração horizontal e vertical, portanto, identificar os processos necessários para programar o sistema e para a sua aplicação e os principais elementos e requisitos necessários para a elaboração do manual da Qualidade são: Controle de documentos e controle de registros. “A organização deve identificar os processos necessários ao sistema de gestão da Qualidade e a sua

aplicação, determinando sua sequência e interação”. (Marshall Jr, Cierco, Rocha, Mota, & Leusin, 2008, p. 76). Devem incluir declarações documentadas da política e dos objetivos da Qualidade, contendo o manual de Qualidade.

## **2.2. As sete ferramentas da qualidade (ISHIKAWA)**

As sete ferramentas do controle de qualidade também conhecida como as ferramentas administrativas tradicionais é um conjunto de metodologias que primeiro foi reunido por Ishikawa (1993) que é amplamente difundido, sua utilização permite a melhoras dos processos das empresas. Estas ferramentas ensinam o significado de variabilidade que se encontra em todos os processos e que a qualidade exige que as pessoas compreendam as causas dos problemas, ou seja, a variação não controlada.

São ditas como as ferramentas da qualidade as seguintes:

- Carta de Controle e gráficos
- Diagrama de Causa-e-Efeito
- Diagrama de Correlação
- Estratificação
- Folha de verificação
- Gráfico de Pareto
- Histograma

### **2.2.1. Carta de Controle e gráficos**

Este gráfico permite que sejam observadas as tendências dos itens mensurados em um período. Podemos utilizá-lo para acompanhar o processo e determinar a frequência dos fatos ocorridos e definindo limite na linha superior e inferior contribuindo para uma estatística mais precisa.

### **2.2.2. Diagrama de Causa-e-Efeito**

A finalidade deste diagrama é a expressar de forma gráfica um conjunto de possíveis causas para um problema específico. É um importante instrumento quando um problema pode ser decorrente de várias causas distintas. De acordo com (Schwarzer, 2014) “os processos devem ser encarados de forma ampla, e se constituírem sempre no fluxo do objeto, tempo e espaço”. Dessa forma:

A percepção anterior tem o propósito de proporcionar o entendimento de processos como forma de coordenação de trabalho. Derivando disso os ciclos de melhoria de processos oportunizam associar a gestão de processos ao aprendizado organizacional e são essas melhorias que possibilitam que a gestão de processos integre diretamente no dia a dia da

organização, não sendo uma parte da melhoria dos processos fora do tempo de trabalho (Paim, Cardoso, Caulliraux, & Clemente, 2009).

### ***2.2.3. Diagrama de Correlação***

Este gráfico pode ser um aliado quando surge a necessidade de avaliar a relação entre duas variáveis, geralmente utilizando quando a correlação entre o resultado de um processo “A”, com outros processos que não tenham correlação, ou seja, que não seja insumo e nem produto de um processo.

### ***2.2.4. Estratificação***

Esta ferramenta tem como objetivo agrupar dados semelhantes, do mesmo tipo, mesmo que vindo de outros processos, com a finalidade de possibilitar que seja feita uma avaliação dos processos, em conjunto ou não, por aspectos comuns.

### ***2.2.5. Folha de verificação***

Esta ferramenta tem a semelhança com um *checklist* e sua diferença é que permite que itens pré-estabelecidos adquiram comportamento esperado, sua utilização mais comum é a certificação que partes do processo tenham sido executadas de forma esperada além de permitir em momento posterior avaliações quanto à eficiência dos mesmos.

### ***2.2.6. Gráfico de Pareto***

Ferramenta que apresenta de forma gráfica, barras ordenadas da maior para a menor, sua real finalidade é apresentar um conjunto de possíveis problemas que um ou mais processos tenham executados permitindo assim uma melhor priorização dos mesmos.

### ***2.2.7. Histograma***

Este diagrama apresenta-nos um conjunto de barras com o objetivo de representar a distribuição de uma frequência de eventos, geralmente relacionado ao mesmo processo.

## **FUNDAMENTOS TEÓRICOS DE DESENVOLVEDORES E TI**

A seguir serão tratados o referencial acerca de TI e Código aberto que fundamentará o desenvolvimento dos objetivos propostos para a presente pesquisa.

## **2.3. Código Aberto**



O código aberto é uma linguagem que está disponível, ou seja, “aberta” à consulta, pesquisa e aprimoramentos pelo público em geral. Desenvolvedores possuem acesso livre para realizar qualquer tipo de “experiência” com o código matriz, de maneira a permitir atualizações.

De acordo com Percília (2018) para ser considerado um código aberto, faz-se necessário que o software apresente as seguintes características:

- Redistribuição livre, a licença não deverá em hipótese nenhuma cobrar pela sua distribuição a qualquer tempo.
- O código fonte, a distribuição do mesmo deve vir ou informar de forma fácil e factível onde se encontra o código origem sendo ou não compilado.
- Nunca a licença deve discriminar qualquer tipo de pessoa e deve prezar pela democracia do acesso ao mesmo.

O *software* livre, apesar dos benefícios, tem despertado controvérsia no mercado e isso se dá em grande parte pela alegação da concorrência desleal, pois, através dos softwares livres pode se criar uma variedade de aplicações, plataformas e pacotes semelhantes aos disponíveis no mercado a preços mais acessíveis.

Entretanto, cada vez mais, empresas como Google, IBM, Dell entre outras vêm adicionando recursos à iniciativa “*Open Source*” e isso se dá pelo novo modelo de negócio que é a consultoria a produtos e serviços que elas podem oferecer.

De acordo com o Software Livre Brasil (2009) o termo código aberto foi definido seguindo dos preceitos do *Open Source Initiative* (OSI). São características de um software de código aberto: i) Distribuição livre; ii) Código fonte; iii) Trabalhos Derivados; iv) Integridade do autor do código fonte; v) Não discriminação contra pessoas ou grupos; vi) Distribuição da Licença; vii) Licença não específica e sem restrições a um ou mais programas; viii) Licença neutra em relação à tecnologia.

### ***2.3.1. Distribuição livre***

A distribuição pode ser vendida ou distribuída de forma gratuita como peça de outro programa. A distribuição se dá através de plataformas que são mantidas por entidades de programadores que aderem ao conceito de software livre. Como contrapartida, solicitam doações para a manutenção do projeto, bem como apoio técnico de outros profissionais da área no desenvolvimento e aprimoramento do projeto.

### ***2.3.2. Código fonte***

Isso quer dizer que o código fonte tem que estar disponível na mesma plataforma ou de forma fácil, mesmo se a distribuição for uma compilação. Este código fonte tem que ser inteligível e que possa permitir alterações. Também não poder ser uma caixa preta a forma de como foi feito,

e os passos para sua reconstrução devem ser claros de forma que a sua construção possa ser feita por qualquer profissional da área.

### ***2.3.3. Trabalhos Derivados***

Uma das características do código aberto é a democratização do acesso ao programa distribuído sob essa licença, caso surjam derivados do código fonte, esses têm que herdar a licença do “pai”, isso quer dizer, os trabalhos derivados devem garantir que a licença siga os preceitos da licença do produto original. Ainda deve ser informado que os trabalhos derivados não podem em hipótese nenhuma serem cobrados, ainda que sejam melhorias ou correções de qualquer natureza e isso se dá pelo fato de que melhorias e correções devem e podem ser disponibilizados pela comunidade.

### ***2.3.4. Integridade do autor do código fonte***

A integridade do autor do código fonte diz que o produto derivado desse código fonte tenha um nome e número de versão a fim de distinguir e garantir a integridade do propósito original do que foi proposto pelo autor original do programa. Para isso o autor deve informar na licença que as cópias modificadas não podem ter o mesmo nome do original, isso acontece geralmente em produtos que tiveram ramificações como o caso do banco de dados MySQL e o MarinaDB.

### ***2.3.5. Não discriminação contra pessoas ou grupos***

A licença deve garantir a democracia ao acesso ao programa. Classe, credo, cor, posicionamento político ou qualquer discriminação de forma ou espécie não pode restringir o acesso ao programa. Essa licença permite que diversos grupos em diversos países criem grupo de trabalhos com objetivos de melhorar a tecnologia em si, um dos grupos que se valem desse tipo de licença são os JUG (*Java User Group*) e outras diversas comunidades de pesquisa científica de diversos fins pelo globo.

### ***2.3.6. Distribuição da Licença***

Conforme nos orienta o *Software Livre Brasil* (2009), Direitos que se encontram vinculados aos programas não obrigam que outros programas tenham uma licença adicional para poder utilizar estes programas como parte dos mesmos, ou seja, o direito vinculado ao programa não pode ser vinculado à distribuição de outro programa. Um exemplo disso são as licenças que estão disponíveis no Android que não estão sobre a mesma licença.

### ***2.3.7. Licença neutra em relação à tecnologia***

Para que a licença seja neutra, ela deve ser neutra em relação à tecnologia. Isso se deve ao fato de que a licença não determina que o programa seja distribuído na linguagem “A” ou “B”. A licença tem que favorecer a democracia em relação à tecnologia, interfaces ou qualquer outra especialização tecnológica que possa vir a ter.

## **2.4. Licença Livre**

É uma licença que tem como objetivo garantir ao usuário liberdades para cópias, estudar, modificar e distribuir essas modificações sem ter que pagar pelos direitos autorais para seus criadores.

Conforme GNU (2019), um programa sobre essa licença possui as quatro liberdades essenciais:

- A liberdade de executar o aplicativo ou programa de maneira livre, para qualquer propósito e finalidade (liberdade zero).
- A permissão para que se possa estudar a forma que o programa funciona, em todos os seus aspectos, tanto quanto a frameworks, bibliotecas, interfaces e qualquer outro aspecto funcional ou não (liberdade 1), e para isso, a disponibilização do código fonte e sua documentação e de vital importância.
- Distribuições de programas com a finalidade de contribuir com a democratização do projeto (liberdade 2).
- Liberdade para redistribuir suas próprias versões modificadas de distribuições de programas a comunidade em geral (liberdade 3), com a finalidade de democratizar o software livre ao compartilhar com outros as inovações e atualizações realizadas e para isso, a disponibilização do código fonte e sua documentação e de vital importância.

Somente é um *software* livre se o mesmo segue a todas as liberdades de forma adequada, garantindo que o usuário tenha todas as liberdades presentes em seus requisitos, por óbvio, se não há esses princípios não existe *software* livre. A liberdade de alterar o código fonte de maneira a aprimorá-lo e torná-lo usual a qualquer usuário, mesmo para uso comercial, é que garante que a licença livre permaneça como uma alternativa barata e viável.

Licenças são separadas em três categorias, elas podem impor restrições nas distribuições do licenciamento, na criação de trabalhos derivados através de redistribuições baseada em distribuições para diversos objetivos ou fins e assim, tem-se: (i) licenças permissivas; (ii) licenças recíprocas (parciais ou totais).

### **2.4.1. Licenças Permissivas**

As licenças acadêmicas tais como as da Universidade da Califórnia em Berkeley (BSD) e do *Massachusetts Institute of Technology* (MIT), têm como foco popularizar um programa e para isso se vale que os derivados destes, possam explorar ainda que comercialmente.

Suas restrições mínimas permitem que diversas empresas procurem programas nesta licença por permitir uma exploração comercial sem a necessidade que pagar pelos seus direitos autorais, exemplos de licenças permissivas são as BSD, MIT e Apache.

#### **2.4.2. Licenças Recíprocas Totais**

Este tipo de licença preza por garantir que a distribuição, parcial ou total, de qualquer parte do software seja redistribuída nos mesmos termos da licença original. Estas licenças deram base à filosofia conhecida atualmente como *copyleft*. A ideia do *copyleft* é que a cópia, modificações, redistribuição e execução, como partes ou não de novos produtos não possam sofrer restrições. Programas nesta licença não podem ser incorporados a programas que tenham licenças mais restritivas quanto aos seus direitos autorais. A GPL (*GNU General Public Licence*) da Free Software Foundation é um bom exemplo de licença recíproca.

#### **2.4.3. Licenças Recíprocas Parciais**

Diferentemente das licenças recíprocas totais, as licenças recíprocas parciais têm o princípio de que se as modificações oriundas por outros sejam disponibilizadas sobre os mesmos termos da licença original, tem-se, então, o *copyleft* fraco. A contrapartida é que programas que possam utilizar programas sobre essa licença como um componente e que estes podem restringir a sua própria licença.

Vale ressaltar que tanto licenças recíprocas totais e as licenças recíprocas parciais são licenças voltadas aos softwares livres e isso quer dizer que elas garantem o acesso ao código fonte a fim de democratizar e popularizar a tecnologia.

As licenças EPL (*Eclipse Public Licence*), MPL (*Mozilla Public Licence*) e LGPL (*GNU Lesser General Public Licence*) são exemplos e logo abaixo damos uma dimensão de cada uma delas:

- A Licença LGPL foi redigida com o propósito de dar mais flexibilidade mediante a GPL, sua alteração permite, entre outros, que outros softwares possam agregar como componente sem impedir que outros programas restrinjam suas licenças, a diferença é que se o programa principal for LGPL seus componentes também devem ter o nível de reciprocidade fidedigno.
- A Licença pública Mozilla (*Mozilla Public Licence*). Esta licença tem muito em comum com a LGPL, as mesmas determinam que se o programa foi escrito sob essa licença seus componentes devem ter um nível de reciprocidade fidedigno, entretanto, esta

licença permite que caso seja utilizado como componente, estes podem estar sob outras licenças inclusive licenças comerciais.

- A Licença Pública Eclipse foi criada pela fundação *Eclipse Foundation*. Sem muitas diferenças das outras duas acima, esta licença é uma resposta da IBM a época com a finalidade de substituir duas outras licenças *Public Licence* de 1999 e a *Common Public Licence*.

## 2.5. Manifesto Ágil

O manifesto ágil (2001) foi um marco inédito na história do software, isso se dá pelo fato de que grandes profissionais se dedicaram um tempo a criar um documento que visa responder aos anseios do usuário no cerne da construção de software. Baseada em um conjunto de princípios e não regras permitem que melhores formas de trabalho, avaliação, coordenação, reavaliação, processos internos e externos sofram uma melhoria contínua.

### Quadro 1. Manifesto Ágil.

#### O MANIFESTO ÁGIL

Estamos descobrindo maneiras melhores de desenvolver softwares, fazendo-o nós mesmos e ajudando outros a fazê-lo.

Através desse trabalho, passamos a valorizar:

- Indivíduos e a interação entre eles mais do que processos e ferramentas;
- Software em funcionamento mais do que documentação abrangente;
- Colaboração com o cliente mais do que negociação contratual;
- Responder a mudanças mais do que seguir um plano.

Mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Fonte: Manifesto Ágil (2001).

Além de ser composto por apenas quatro princípios, o manifesto ágil (2001) apresenta-nos doze (12) princípios:

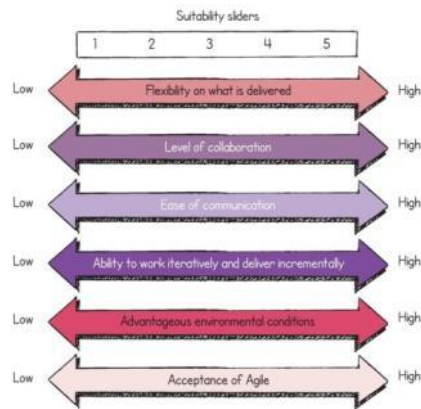
- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.

- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário, e confie neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informações entre uma equipe de desenvolvimento é através de conversa cara a cara.
- Software funcionando é a medida primária de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- Contínua atenção a excelência técnica e bom design aumentam a agilidade.
- Simplicidade — a arte de maximizar a quantidade de trabalho não realizado é essencial.
- As melhores arquiteturas, requisitos e designs emergem de equipes que são auto-organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, e então refina e ajusta seu comportamento de acordo (Manifesto Ágil, 2001).

Como de se esperado, o manifesto dá à base ideológica para diversas metodologias e frameworks existentes no mercado sendo algumas delas o Scrum, Extreme Programming (XP), *Crystal Clear* e *Featrure Driven Development*.

## **2.6. Agilômetro**

Surgido como uma ferramenta utilizada na metodologia britânica PRINCE2 da Axelos (2015), mas que pode ser utilizado em qualquer iniciativa que utiliza métodos ágeis. A ideia é obter uma visão da aceitação que, por conseguinte, a maturidade dos métodos ágeis pela organização e para isso ela se baseia em seis pontos chaves sendo eles: (i) Flexibilidade com relação às entregas; (ii) Nível de colaboração; (iii) Facilidade de Comunicação; (iv) Capacidade de trabalhar com entregas iterativas e incrementais; (v) Fatores ambientais e (vi) Aceitação dos métodos ágeis.



**Figura 1.** Pontos Chaves do Agilômetro.

Fonte: Axelos (2015)

Em consonância com a Axelos (2015) o agilômetro define seis pontos-chaves possibilitam que se possa mensurar o nível de maturidade da instituição quanto ao nível de maturidade em relação à adoção da metodologia ágil.

#### ***2.6.1. Flexibilidade com relação às entregas***

Alguns questionamentos são necessários para compreender a aplicação do agilômetro, de acordo com (Axelos, 2015): (i) A empresa compreende que o escopo-tempo-custo define o sucesso de um projeto? (ii) A empresa sabe o real escopo, priorizando adequadamente, utilizando protótipos que realmente atendam a real necessidade, incorporando mudanças progressivamente, remove itens não necessários e permite que todos participem do processo.

#### ***2.6.2. Nível de colaboração***

O nível de colaboração se refere exatamente à maturidade da instituição como um todo. A colaboração é medida horizontalmente e como um dos princípios do ágil e a valorização de relação entre pessoas, o nível de colaboração pretendido é metrificado tanto horizontalmente quanto verticalmente. Horizontalmente, é o nível de colaboração que se é desprendido para os membros da equipe, ou seja, se os membros entendem que o sucesso não depende apenas de suas entregas se o colega ao lado não está conseguindo se desenvolver.

O nível de colaboração verticalmente se refere à colaboração dos colegas numa cadeia hierárquica mais alta, ou seja, os gerentes, diretores mantêm uma política de portas abertas ou algo do gênero.

#### ***2.6.3. Facilidade de comunicação***

Difere-se do nível de colaboração porque neste nível o que é mensurado são as ferramentas e processos existentes que possam facilitar a comunicação da equipe tanto em nível hierárquico

horizontal, quanto vertical. O grande desafio aqui é conseguir obter acesso aos diretores e aos colegas. Outra coisa importante neste nível é se a empresa preza por manter uma documentação atualizada que permita ao próximo membro da equipe continuar o trabalho ou começar uma nova mudança e se a empresa consegue fazer funcionalidades corretivas ou adaptativas sem a necessidade de se obter um conjunto extensivo de documentos pretéritos.

#### ***2.6.4. Capacidade de trabalhar com entregas iterativas e incrementais***

Nesse nível procura-se atestar o nível da empresa quanto à capacidade de produzir soluções que sejam adequadas, no tempo certo, e adaptativo. Este nível trabalha muito com comparações utilizando-se como base outras metodologias não ágeis a fim de obter uma forma de combater, por exemplo, o processo preditivo de construção de software. A grande finalidade também é garantir que entregas adaptativas, geralmente utilizando-se do conceito adaptativo e incremental, sejam utilizadas.

Um dos conceitos ao se escolher o que vai ser feito, muito utilizado ultimamente, chama-se MVP<sup>3</sup>, que consiste em entregar aquilo que atenda ao usuário em detrimento ao tudo ou nada que se era utilizado.

#### ***2.6.5. Fatores ambientais***

Os fatores ambientais se referem à motivação e desempenho das equipes, para isso, alguns itens são classificados como sendo satisfatório ou não, apesar dessa lista poder ir do cafezinho ao maquinário utilizado pela equipe, os mais comuns são: (i) O ambiente favorece a agilidade, isto é, existem equipes ágeis convivendo com equipes tradicionais; (ii) A empresa favorece uma cultura “beta” e entende que todos estão em processo de melhoria continua; (iii) A empresa favorece a melhoria continua no ambiente, nas pessoas, nos processos?

#### ***2.6.6. Aceitação dos métodos ágeis***

A aceitação dos métodos ágeis se refere mais precisamente ao nível de aceitação das equipes quanto à adoção e uso de metodologias ágeis em detrimento as metodologias tradicionais. Algumas possíveis perguntas para poder se mensurar corretamente neste ponto são: (i) As equipes entendem o por que dos métodos ágeis?; (ii) A alta gerência realmente entrega o poder para equipe mesmo no fracasso?; (iii) as pessoas conseguem compreender que o nível hierárquico é o que menos importa, sendo mais relevante a entrega do valor ao produto e serviço?

---

<sup>3</sup> MVP e a sigla de *Minimum Viable Product* que em português significa produto mínimo viável, metodologia que preza por entregas menores, entretanto, procurando entregar maior valor ao cliente.



Com a avaliação destes seis pontos chaves, pode-se traçar a real usabilidade da introdução das ferramentas que irão dar suporte à integração contínua e assim obter agilidade na entrega de valor para o usuário.

Para o objetivo específico de aprofundar e entender o nível de aceitação dos métodos ágeis pela organização este trabalho utilizará o agilômetro como parâmetro para identificar pontos de melhorias e que permitirá ter a base teórica para a busca de metodologias e ferramentas.

## **2.7. Computação em nuvem**

Computação em nuvem é a abstração de serviço de tecnologia voltado à simplificação do ambiente computacional para as empresas utilizando-se a internet como fonte de acesso a esta malha computacional. Empresas que fornecem esse tipo de solução cobram pelo uso de seu poder computacional por demanda tornando simplificado a complexidade para empresas que procuram esses serviços, ainda é válido citar que a cobrança por esses serviços se dá pelo uso o que garante certa assertividade intrínseca quanto ao tamanho da malha que seu aplicativo venha a usar.

### ***2.7.1. Principais benefícios da computação em nuvem***

A computação em nuvem traz significantes benefícios para as empresas que procuram ter certa economicidade em sua malha computacional. Nos dias de hoje, a computação em nuvem se torna cada vez mais atrativa para as empresas graças a seis motivos sendo eles:

- a) Custo;
- b) Velocidade;
- c) Escala global;
- d) Produtividade;
- e) Desempenho e
- f) Produtividade.

#### ***2.7.1.1. Custo***

De acordo com a Microsoft (Microsoft, 2018), ao eliminar o custo inicial na compra de maquinário e licenças softwares, também e não menos importante, habilita instituições de menores portes a utilizar um ambiente sem a necessidade de uma sala cofre, custo com energia ou pessoal capacitado para dar manutenção neste parque, e isso permite que muitas empresas iniciantes já comecem operando com um ambiente profissional a custo bem menos significativo que de outra forma não seria possível.

#### ***2.7.1.2. Velocidade***

Quando se usa um parque no modelo tradicional, muita das vezes, logo este fica obsoleto além de que para garantir o atendimento a todos os usuários nem sempre é possível escalonar o parque de forma a economizar os recursos, o que provoca, na maioria dos casos, lentidão no uso do mesmo.

#### 2.7.1.3. *Escala global*

Como o parque se encontra na nuvem, através da internet os aplicativos estão disponíveis a qualquer lugar do globo. O dimensionamento elástico ainda permite que você disponibilize seus aplicativos em localidades mais próximos da gama de usuários de seu aplicativo. O poder da computação em nuvem também permite que o escalonamento se dê por região do globo. Um exemplo disso é que se seus usuários estiverem no outro lado do hemisfério com alguns cliques é possível tornar o acesso mais rápido apenas para aquela região.

#### 2.7.1.4. *Produtividade*

Analista de produção, gerente de banco de dados, telefonista e a equipe de rede não se fazem mais necessários, tendo em vista que, a gestão da infraestrutura fica agora a cargo do provedor de serviço. Como o esvaziamento de perfis as demandas para o Analista DevOps<sup>4</sup> dentro da equipe, a gestão do ambiente se torna muito mais eficiente, ideal para equipes ágeis que procuram velocidade na entrega de valor ao usuário do produto.

#### 2.7.1.5. *Desempenho*

Quanto ao desempenho, a computação em nuvem abstrai as configurações de máquinas o que permite que o provedor do serviço sempre tenha o maquinário mais atual, com os *softwares* sempre atualizados e com o nível de segurança sempre protegido quando ao acesso indevido às informações. A latência de rede é reduzida ao passo de simples configurações utilizando a escalabilidade global, a compra dentro de seu produto não pode ficar apresentando lentidão o que com certeza poderá gerar perda de receita.

#### 2.7.1.6. *Confiabilidade*

A computação em nuvem por padrão já conta com diversos mecanismos de *backup*, tanto de máquina quanto de bases de dados. O agendamento fica a cargo da ferramenta disponibilizado pelo provedor que fornecem vários locais de armazenamento espalhado em diversos sites na rede do provedor.

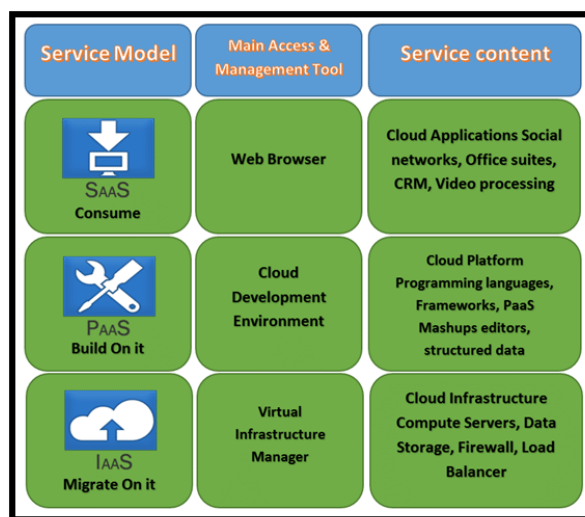
---

<sup>4</sup> Ver tópico 2.11 a 2.11.8

### 2.7.2. Tipos de serviços de nuvem: IaaS, PaaS e SaaS

Atualmente, os serviços ofertados pelas provedoras de serviços em nuvem se dividem em três categorias IaaS (infraestrutura como serviço), PaaS (plataforma como serviço) e SaaS (software como serviço). Fato notório é que elas se organizam como pilha de serviços, ou seja, o IaaS é a base para os outros dois tipos de serviço por fornecer soluções mais no nível de máquina, mesmo essas máquinas sendo acessadas via protocolo RDP<sup>5</sup>. Já o segundo é o PaaS serviço que se baseia em plataformas para atendimento a equipe de TI que não dispõe tempo e recurso para montar a infraestrutura do zero.

A última do topo da cadeia de serviços está o SaaS que fornece soluções já feitas na rede, ou seja, nestes constam apenas aplicativos disponibilizado na internet para fins comerciais apenas.



**Figura 2.** SaaS, PaaS e IaaS e suas aplicações.

Fonte: Elgazzar, A Saleh, & El-Bakry (2017)

Abaixo, aborda-se de forma sucinta qual é a finalidade de cada serviço:

#### 2.7.2.1. IaaS (Infraestrutura como serviço)

Como citado anteriormente, trata-se do serviço mais básico dentro da pilha da computação em nuvem. Essa categoria permite que a corporação crie e administre a infraestrutura para a solução ou produto com o uso de VMs (máquinas virtuais), redes, armazenamento e tudo que um parque tradicional pode oferecer.

---

<sup>5</sup> RDP é um protocolo para acesso remoto utilizado por diversos programas a fim de permitir uma máquina possa ser acessada através da internet.

#### 2.7.2.2. *PaaS (plataforma como serviço)*

Com a finalidade de abstrair diversas configurações, geralmente repetitivas, esta modalidade de serviço apresenta uma possibilidade para aquelas equipes ou instituições que desejam apenas especializar o ambiente. O PaaS foi projetado para que algumas configurações sejam disponibilizadas para que a equipe possa especializar, mas não permite que se tenha um controle da infra. Para muitos serviços PaaS a equipe pode utilizar o recurso de *hardware* compartilhado entre outras corporações a separação disso se dá de forma lógica e não física como é o caso do IaaS.

#### 2.7.2.3. *SaaS (software como serviço)*

O SaaS é atualmente a última pilha de serviços oferecido na nuvem, com ela é possível fornecer produtos ou serviços onde o usuário e nem a empresa contratante tem noção onde se encontra o *hardware* físico. Provedores desse tipo de serviço os fornecem através de assinatura, um bom exemplo disso é a *Salesforce* que a instituição apenas usa o produto ou especializa através de meios que a própria provedora disponibiliza, mas todas as outras configurações ficam a cargo da provedora.

O SaaS (*software* como serviço) é um método para fornecer aplicativos de software pela Internet, de acordo com a demanda e, normalmente, em uma base de assinaturas. Como SaaS, os provedores de nuvem hospedam e gerenciam o aplicativo de software e a infraestrutura subjacente e fazem manutenções, como atualizações de software e aplicação de patch de segurança. Os usuários conectam o aplicativo pela Internet, normalmente com um navegador da Web em seu telefone, tablet ou PC (Microsoft, 2018, p. 1).

### 2.7.3. *Tipos de implantação em nuvem: pública, privada e híbrida.*

Existem vários tipos de nuvem computacionais no mercado, entretanto, atualmente, também temos três maneiras para implantar uma nuvem em uma corporação. As três mais comumente usadas são: nuvem privada, nuvem pública e nuvem híbrida.

#### 2.7.3.1. *Nuvem pública*

De acordo com a Microsoft (2018), nuvens públicas, apesar de que seu fim seja atender as corporações, a propriedade de todo o maquinário e a operação deste fica a cargo do provedor de serviço. A Microsoft Azure é um exemplo de nuvem pública. Com uma nuvem pública, todo o hardware, software e outras infraestruturas de suporte são de propriedade e gerenciadas pelo provedor de nuvem (Microsoft, 2018).

#### 2.7.3.2. *Nuvem privada*

Para exemplificar bem, uma nuvem privada, tende a operar em um datacenter da própria organização ou empresa. Isso se dá pelo fato de que algumas organizações precisam ter um controle maior dessa nuvem. Ainda que algumas empresas paguem por datacenter de terceiros toda a gestão da nuvem é gerida pela própria organização.

#### 2.7.3.3. Nuvem híbrida

Nuvem híbrida combina as nuvens privadas e públicas. Essa necessidade surge em algumas corporações que procuram novas possibilidades mais rentáveis ao seu negócio ou procuram fazer uma migração para uma nuvem pública em definitivo. Fato é que a nuvem do tipo híbrida consiste de uma nuvem pública gerida por um provedor de serviço e uma nuvem privada que é gerida pela própria corporação, essa ligação se dá através de protocolos de comunicações como VPN.

### 2.8. Gerenciamento de Dependências

Projetos que tem como predominância o paradigma de desenvolvimento orientado a objetos são, hoje em dia, dominantes em sistemas corporativos, pois apresentam diversos benefícios como o baixo acoplamento e a divisão de responsabilidades, a criação e manutenção de projetos se torna uma tarefa extremamente produtiva ainda mais alinhada com metodologias ágeis no desenvolvimento de *software* o que torna extremamente atrativa para qualquer indústria que trilhe o caminho para se tornar digital.

Esta produtividade característica de projetos orientados a objetos deve-se principalmente pela capacidade de reuso de componentes criados que podem ser facilmente acoplados em qualquer tipo de projetos. Com base nisso um projeto pode ser dividido em diversos componentes menores que na maioria das vezes são reaproveitáveis ou já fabricados para uso de propósito geral.

Entretanto, a manutenção desta lista de componentes se torna complexa, além de custosa para toda a equipe, o que pode gerar um oneroso trabalho na busca e identificação desta lista, além de, tirar o foco dessa equipe no que mais importa que seja a entrega de valor ao final da *Sprint*.

Visando resolver esse problema, no mercado existem ferramentas chamadas de Gerenciadores de dependências que focam na resolução deste problema eminente permitindo as equipes ter uma forma de gerir mais facilmente a dependências destes projetos.

### 2.9. Integração Contínua

Integração Contínua é uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho com frequência. Para Fowler (2006) “por integração contínua de desenvolvedores deve-se entender a reformulação e compilação atualizada do código fonte de um

software, em determinados ciclos pré-estabelecidos”. A integração contínua é geralmente iniciada por um agendamento com a finalidade de detectar erros, executar testes, entregar a construção em ambiente de produção, testes, homologação. Muitas equipes acreditam que essa metodologia adianta o serviço e permite que a aplicação fique mais coesa.

Os principais objetivos da integração contínua são encontrar e investigar *bugs* mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de *software* (AWS, 2016).

Para atingir esses objetivos, a integração contínua se baseia nos princípios descritos na sequência abaixo.

### ***2.9.1. Manter um repositório de código***

Essa prática defende o uso de um sistema de controle de versão para o código-fonte do projeto. Todos os artefatos necessários para construir o projeto devem ser colocados no repositório. Nesta prática e na comunidade de controle de versão, a convenção é que o sistema deve ser construído a partir de um novo processo de construção e não requerer dependências adicionais. Para Fowler (2006) onde a ramificação é suportada por ferramentas, seu uso deve ser minimizado. Em vez disso, é preferível que as mudanças sejam integradas ao invés de múltiplas versões do software ser mantidas simultaneamente.

### ***2.9.2. Automatize a construção***

Um único comando deve ter a capacidade de construir o sistema. Muitas ferramentas de construção, como o *Jenkins*, existem há muitos anos. Outras ferramentas mais recentes são frequentemente usadas em ambientes de integração contínua. A automação da compilação deve incluir a automação da integração, que geralmente inclui a implantação em um ambiente de produção. Em muitos casos, o *script* de compilação não apenas compila binários, mas também gera documentação, páginas de sites, estatísticas e mídias de distribuição (como arquivos *Debian DEB*, *Red Hat RPM* ou *Windows MSI*).

### ***2.9.3. Automatize teste antes da construção***

Depois que o código é criado, todos os testes devem ser executados para confirmar se ele se comporta como os desenvolvedores esperam que ele se comporte. Isso consiste em dizer que em cada automação da construção testes serão executados com a finalidade de se obter sucesso e caso haja falhas a equipe faça a correção o mais rápido possível.

### ***2.9.4. A equipe se compromete com a linha de base todos os dias***

Ao se comprometerem diariamente, os membros da equipe reduzem o número de alterações conflitantes. Ao fazer o *check-in* de uma semana de trabalho corre-se o risco de entrar em conflito com outros recursos, o que pode ser muito difícil de resolver. No início, pequenos conflitos em uma área do sistema fazem com que os membros da equipe se comuniquem sobre a mudança que estão fazendo. Confirmar todas as alterações pelo menos uma vez por dia (uma vez por recurso criado) é geralmente considerado parte da definição de Integração Contínua. Além disso, geralmente é recomendável executar uma compilação noturna. Estes são os padrões mínimos e se espera que a frequência da entrega de recursos alterados seja muito maior.

#### ***2.9.5. Toda entrega para baseline deve ser construída***

O sistema deve criar confirmações para a versão de trabalho atual para verificar se elas se integram corretamente. Uma prática comum é usar a Integração Contínua Automatizada, embora isso possa ser feito manualmente. Para muitos, a integração contínua é sinônimo de uso da Integração Contínua Automatizada (ICA), em que um servidor de integração contínua ou uma agente monitora o sistema de controle de revisão em busca de alterações e, em seguida, executa automaticamente o processo de criação.

#### ***2.9.6. Mantenha a compilação rápida***

A construção precisa ser concluída rapidamente, de modo que, se houver um problema com a integração, ela será rapidamente identificada.

#### ***2.9.7. Teste em um clone do ambiente de produção***

Ter um ambiente de teste pode levar a falhas nos sistemas testados quando eles são implantados no ambiente de produção porque o ambiente de produção pode diferir significativamente do ambiente de teste. No entanto, criar uma réplica de um ambiente de produção é custo proibitivo. Em vez disso, o ambiente de teste ou um ambiente de pré-produção separado (*staging*) deve ser construído para ser uma versão escalável do ambiente de produção real para reduzir os custos e, ao mesmo tempo, manter a composição e as nuances da pilha de tecnologia. Dentro desses ambientes de teste, a virtualização de serviços é comumente usada para obter acesso por demanda a dependências (por exemplo, *API's*, aplicativos de terceiros, serviços, mainframes, etc.) que estão além do controle da equipe, ainda em desenvolvimento ou muito complexo para serem configurados em um laboratório de testes virtual.

#### ***2.9.8. Facilite a obtenção das entregas mais recentes***

Tornar as construções prontamente disponíveis para as partes interessadas e os testadores pode reduzir a quantidade de retrabalho necessário ao reconstruir um recurso que não atende aos requisitos. Além disso, os testes iniciais reduzem as chances de que os defeitos sobrevivam até a implantação. Todos os programadores devem começar o dia atualizando o projeto do repositório, dessa forma, todos eles ficarão atualizados. Todos podem ver os resultados da última versão e deve ser fácil descobrir se a construção quebra e, em caso afirmativo, quem fez a alteração relevante e qual foi essa alteração.

### 2.9.9. Automatizar a implantação

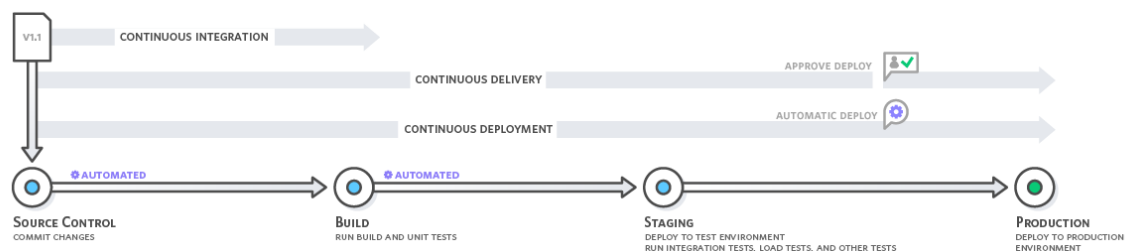
A maioria dos sistemas de IC permite a execução de *scripts* após a conclusão de uma compilação. Na maioria das situações, é possível escrever um script para implantar o aplicativo em um servidor de teste ao vivo que todos possam ver. Um avanço adicional nessa forma de pensar é a implantação contínua, que exige que o *software* seja implantado diretamente na produção, geralmente com automação adicional para evitar defeitos ou regressões.

### 2.9.10. A necessidade da integração contínua

A integração contínua se faz necessário nos dias atuais devido ao fato que atualmente com o advento de metodologias mais ágeis aliado a um poder computacional acentuado, equipes passaram a ter a necessidade de fazer a entrega de valor de forma sistemática, com metodologias tradicionais essa entrega se dava de forma mais morosa, o que nem sempre atendiam a necessidade do negócio.

### 2.9.11. O Funcionamento da integração contínua

A integração contínua permite que os desenvolvedores tenham uma entrega fácil para os ambientes de teste, homologação ou até mesmo produção. Existe um fluxo básico, mas que não impede de ser personalizado de acordo com a equipe. Abaixo vemos um fluxo conforme apresentado na imagem abaixo.



**Figura 3.** Ciclo de Integração Contínua.  
Fonte: AWS (2016)



### 2.9.12. Custos e benefícios

A integração contínua destina-se a produzir benefícios como:

- *Bugs* de integração são detectados precocemente e são fáceis de rastrear devido a pequenos conjuntos de mudanças. Isso economiza tempo e dinheiro ao longo da vida útil de um projeto.
- Evita o caos de última hora nas datas de lançamento, quando todos tentam verificar suas versões ligeiramente incompatíveis.
- Quando os testes de unidade falham ou um *bug* aparece, se os desenvolvedores precisarem reverter à base de código para um estado livre de erros sem depuração, apenas um pequeno número de alterações será perdido (porque a integração acontece com frequência).
- Disponibilidade constante de uma versão "atual" para fins de teste, demonstração ou lançamento.
- O *check-in* de código frequente faz com que os desenvolvedores criem códigos modulares e menos complexos.
- Reforça a disciplina de testes automatizados frequentes.
- *Feedback* imediato sobre o impacto de mudanças locais em todo o sistema.
- Métricas de software geradas a partir de testes automatizados e CI (como métricas para cobertura de código, complexidade de código e integridade de recursos) concentram os desenvolvedores no desenvolvimento de códigos funcionais e de qualidade e ajudam a desenvolver o momento em uma equipe.

Nota-se no quesito dos benefícios uma relação cíclica, e sistemática, de avaliação e reavaliação de processos e procedimentos. Tal qual o que interessa ao presente trabalho de pesquisa.

Algumas desvantagens da integração contínua podem incluir:

- A construção de um conjunto de testes automatizado requer uma quantidade considerável de trabalho, incluindo o esforço contínuo para cobrir novos recursos e seguir as modificações intencionais do código.
- Há algum trabalho envolvido na configuração de um sistema de construção e pode se tornar complexo, dificultando sua modificação de forma flexível, no entanto, existem vários projetos de software de integração contínua, proprietários e de código aberto, que podem ser usados.
- A Integração Contínua não é necessariamente valiosa se o escopo do projeto for pequeno ou contiver código legado não testável.

- O valor agregado depende da qualidade dos testes e de quão testável o código realmente é.
- Equipes maiores significam que o novo código é constantemente adicionado à fila de integração, portanto, o rastreamento de entregas (embora preservando a qualidade) é difícil e a criação de filas de espera pode atrasar todos.
- Com várias confirmações e uniões de códigos feitos pela equipe por dia, o código parcial de um recurso pode ser facilmente enviado e, portanto, os testes de integração falharão até que o recurso seja concluído.
- Segurança e garantia de desenvolvimento de missão crítica (por exemplo, DO-178C, ISO 26262) exigem documentação rigorosa e revisão em processo que são difíceis de alcançar usando Integração Contínua. Esse tipo de ciclo de vida geralmente exige que etapas adicionais sejam concluídas antes da liberação do produto quando a aprovação regulamentar do produto é necessária.

Este trabalho irá utilizar-se do conceito de integração contínua a fim de identificar uma ferramenta de código aberto para ser utilizada na solução do problema proposto.

## **2.10. Controle de Versão**

Um sistema de controle de versão (SCV), também conhecido como controle de revisão ou sistema de controle de origem, é um utilitário de *software* que rastreia e gerencia mudanças em um sistema de arquivos. Um SCV também oferece utilitários colaborativos para compartilhar e integrar essas alterações no sistema de arquivos a outros usuários do SCV. Ao operar no nível do sistema de arquivos, um SCV rastreará as ações de adição, exclusão e modificação aplicadas a arquivos e diretórios. Um repositório é um termo do VCS que descreve quando o SCV está rastreando um sistema de arquivos. No escopo de arquivos de código-fonte individuais, um SCV rastreará adições, exclusões, modificações das linhas de texto dentro desse arquivo. As opções populares do setor de software SCV incluem Git, Mercurial, SVN e *preforce*.

O controle de versão é uma maneira de acompanhar as alterações no código, de modo que, se algo der errado, seja possível fazer comparações em versões de código diferentes e reverter para qualquer versão anterior que se queira. O SCV é necessário em ambientes onde vários desenvolvedores estão trabalhando continuamente com o código-fonte.

Um sistema de controle de versão, de acordo com a Atlassian (2019), atende aos seguintes objetivos, entre outros.

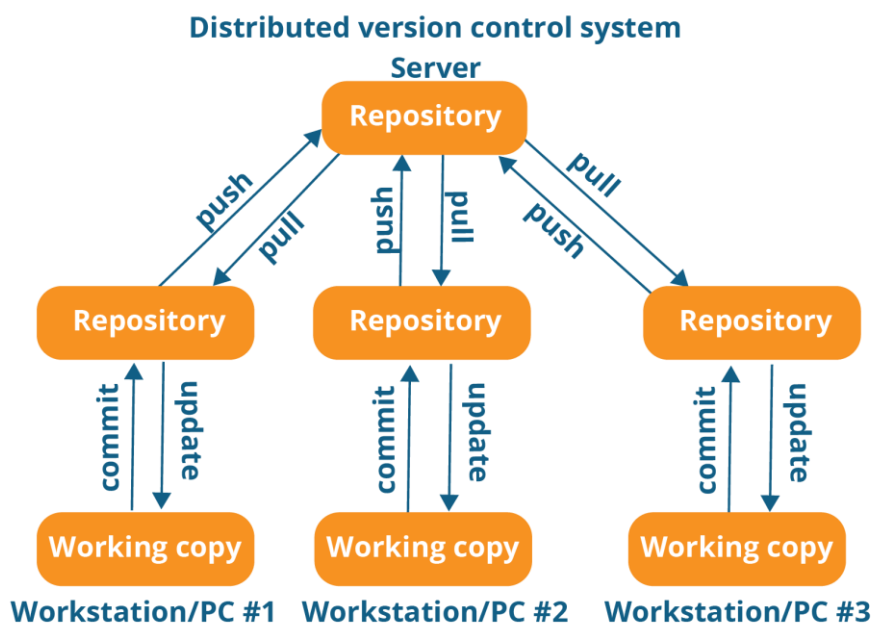
- O controle de versão permite que várias pessoas trabalhem simultaneamente em um único projeto. Cada pessoa edita sua própria cópia dos arquivos e escolhe quando compartilhar

essas alterações com o restante da equipe. Assim, edições temporárias ou parciais de uma pessoa não interferem no trabalho de outra pessoa.

- O controle de versão também permite que uma pessoa utilize vários computadores para trabalhar em um projeto, por isso é valioso mesmo que você esteja trabalhando sozinho.
- O controle de versão integra o trabalho realizado simultaneamente por diferentes membros da equipe. Na maioria dos casos, as edições em arquivos diferentes ou até mesmo no mesmo arquivo podem ser combinadas sem perder nenhum trabalho. Em casos raros, quando duas pessoas fazem edições conflitantes na mesma linha de um arquivo, o sistema de controle de versão solicita assistência humana para decidir o que fazer.
- O controle de versão dá acesso a versões históricas do seu projeto. Isso é seguro contra falhas no computador ou perda de dados. Se você cometer um erro, poderá reverter para uma versão anterior. Você pode reproduzir e entender um relatório de bug em uma versão anterior do seu software. Você também pode desfazer edições específicas sem perder todo o trabalho que foi feito nesse meio tempo. Para qualquer parte de um arquivo, você pode determinar quando, por que e por quem foi editado (Atlassian, 2019).

#### 2.10.1. Controle de versão distribuída

O sistema de controle de versão distribuída tem a como fundamento o trabalho distribuído onde cada colaborador tem um repositório, e a cada iteração deve fazer a entrega em uma ferramenta de controle de mudança centralizado onde os outros colaboradores devem buscar as mudanças e atualiza-las em seus repositórios.



**Figura 4.** Arquitetura de um sistema de controle de versão distribuído.

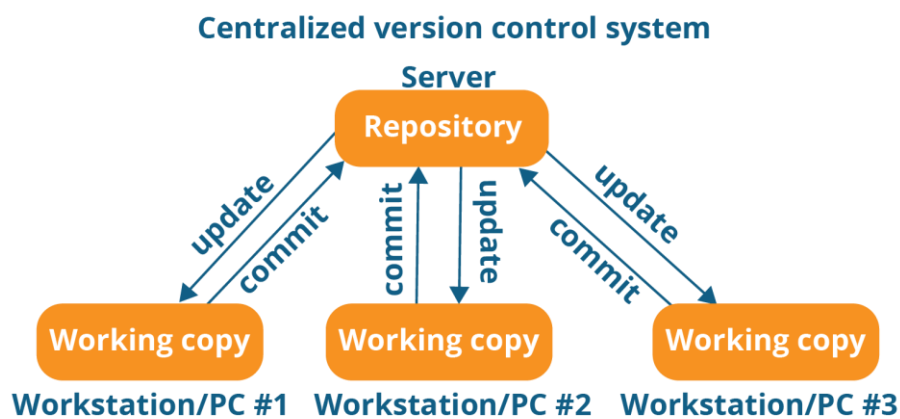
Fonte: Ahmed (2019)

Neste caso é importante salientar que o controle de mudança deve ficar em um servidor onde todos os colaboradores consigam acessar para criar ou atualizar seus repositórios de acordo com a necessidade.

A vantagem do controle de versão distribuído é possibilitar mais autonomia ao colaborador do projeto e permite que o mesmo tenha um controle de suas alterações sem implicar numa atualização do servidor de mudanças. Outra vantagem também é que a atualização ocorre no final da iteração o que pode significar que problemas que ocorrerem por indisponibilidade do servidor de mudanças não interrompe o trabalho do colaborador.

### 2.10.2. Controle de versão centralizado

O sistema de controle de versão centralizado tem como fundamento a centralização do controle de mudança e o repositório permitindo apenas que os colaboradores tenham uma versão local.



**Figura 5.** Arquitetura de um sistema de controle de versão centralizado.  
Fonte: Ahmed (2019)

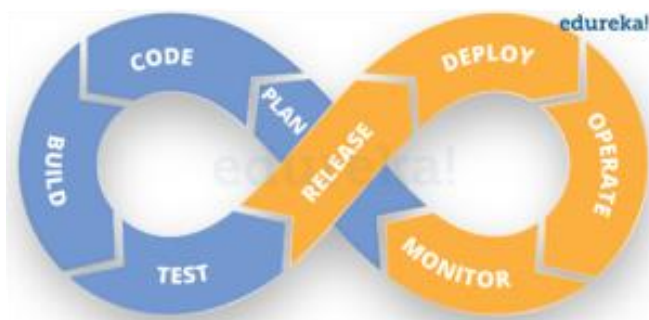
A vantagem desta modalidade é a simplicidade, colaboradores com menos experiência já conseguem utilizar sem maiores problemas, outra vantagem é que como o servidor de mudança e o repositório ficam no mesmo lugar e mais fácil de ter uma gestão, mesmo significando menos autonomia para a equipe de colaboradores.

## 2.11. DevOps

Conceitualmente, *DevOps* é uma metodologia de desenvolvimento de *software* que utiliza a comunicação para integrar desenvolvedores de *software* e profissionais de infraestrutura de TI (GAEA, 2018).

Tem se despontando cada vez mais esse tipo de profissional nas corporações, verdade seja dita, com o advento da computação em nuvem cada dia mais os desenvolvedores seniores dos projetos têm assumido essa qualidade como parte de suas qualidades como profissional.

Para reduzir a incidência de problemas e aumentar a flexibilidade e a automação, foi definido sempre que possível os recursos não operacionais e em ambientes que não sejam o de produção devem ficar a cargo da equipe, permitindo que a equipe tenha sempre um controle do que de fato está ocorrendo.



**Figura 6.** Ciclo de integração entre Desenvolvedores e Operadores.  
Fonte: Atlassian (2019)

#### ***2.11.1. Integração entre áreas***

A ideia inicial do analista de DevOps é preencher a área cinzenta entre o desenvolvimento e a produção, ela visa não apenas unir os times como também cria uma ruptura nas barreiras que existem com o negócio, gestores e com os “donos” dos produtos e serviços.

Promovendo uma sinergia com o profissional que desenvolve e com o que coloca a produção no ambiente de produção, o Analista DevOps ou a pessoa com a competência de DevOps promove uma maior interação com o solicitante da funcionalidade, e permite que toda a equipe tenha conhecimento do processo de entrega fim a fim e assim entregar o real valor ao cliente.

#### ***2.11.2. Simplificação de processos***

Simplicidade de processo permite que fluxos de trabalhos sejam menos onerosos e burocráticos, isso quer dizer que os aplicativos e soluções devem chegar à produção sem uma documentação extensiva devido à natureza dinâmica do aplicativo.

#### ***2.11.3. Automação de tarefas***

Os analistas DevOps pregam uma mudança cultural onde entregas manuais cedam espaço aos processos e rotinas automatizadas.

#### ***2.11.4. Racionalização de processos***

Com a simplificação e automação o próximo passo é o trabalho voltado à racionalização dos processos a fim de torná-los economicamente viáveis. Anteriormente, o custo de um aplicativo ir à produção era proibitivo, ou seja, muitas soluções não conheceram a luz do dia devido ao custo proibitivo principalmente o custo na manutenção, algo que, com a computação em nuvem vem provendo melhorias significativas.

#### ***2.11.5. Modernização da TI da empresa***

Como mencionado, na racionalização de processos fica a cargo do DevOps a mineração por melhorias tecnológicas que venham permitir que a instituição possa redirecionar os recursos no ativo mais importante que são as pessoas. Atualmente, muito se fala em computação em nuvem, que proporciona a liberdade de uso sem desperdícios desnecessários com recursos excedentes para atender demandas sancionais.

#### ***2.11.6. Estímulo à colaboração***

A colaboração na cultura DevOps é sempre bem-vinda, permitindo que as equipes tenham acesso facilitado ao ambiente de uma forma em geral e, com isso, aumentar a iteração dos diversos perfis dos membros da equipe. Cada membro passa a ter a possibilidade, desde que tenha algum conhecimento, de aperfeiçoar rotinas e processos com a finalidade de afinar os processos para garantir uma melhor economicidade.

#### ***2.11.7. Empoderamento dos times de TI***

Com o perfil de DevOps na equipe, a mudança de paradigma se torna óbvia e o que competia a outras equipes agora faz parte do cotidiano da equipe. A equipe passa a ter mais gestão sobre seus recursos inclusive no quesito orçamento, no que fomenta a busca constante por soluções mais economicamente viáveis. Neste contexto não é difícil a mudança de soluções legadas a fim de liberar recursos para novas soluções geralmente mais rentáveis e com uma base tecnológica mais atual.

#### ***2.11.8. Elasticidade e escalabilidade***

Elasticidade e escalabilidade é um princípio de racionalização na disponibilidade de informações e dados. Assim, essa responsabilidade passa a cargo dos provedores de serviços e mensalmente informa uma real utilização dos recursos utilizados por usuários. A instituição passa a se preocupar apenas com seu produto e serviço, e aliado com a equipe, às soluções passam a ter um viés mais economicamente factível.

A identificação do conceito de DevOps dar-se ao fato de identificarmos no estudo de caso a falta de um membro da equipe com a expertise para atuar na resolução de problemas e identificação de solução quando a operação das ferramentas propostas.

## CAPÍTULO 3 - PROCEDIMENTOS METODOLÓGICOS

### 3.1. Tipo de pesquisa

A presente pesquisa quanto aos objetivos será qualitativa, ou seja, terá como foco o aprofundamento da compreensão de um grupo social, de uma organização, de seus métodos e procedimentos. De acordo com Gerhardt & Silveira (2009) os métodos qualitativos buscam explicar o porquê das coisas, entretanto, não quantificam valores e não submetem à prova de fatos, haja vista, a possibilidade de hierarquizar problemas e propor soluções estanques ou genéricas ao conjunto de dados estudados.

As características da pesquisa qualitativa são: objetivação do fenômeno; hierarquização das ações de *descrever*, *compreender*, *explicar*, precisão das relações entre o global e o local em determinado fenômeno; observância das diferenças entre o mundo social e o mundo natural; respeito ao caráter interativo entre os objetivos buscados pelos investigadores, suas orientações teóricas e seus dados empíricos; busca de resultados os mais fidedignos possíveis; oposição ao pressuposto que defende um modelo único de pesquisa para todas as ciências (Gerhardt & Silveira, 2009, p. 32).

### 3.2. Coleta de dados

Para a coleta de dados e informações, foi realizado um Estudo de Caso em uma Empresa de Seguros no Distrito Federal, WIZ Soluções e Corretagem de Seguros S/A, detentora de ampla carteira de clientes, que consta com uma iniciativa voltada ao mercado de aplicativos no ramo de seguros. Com o objetivo de mensurar a real importância da integração continuada em ambiente de desenvolvimento ágil, o nível de maturidade foi metrificado utilizando o agilômetro, que auxiliou na extração dos dados com o objetivo de responder o objetivo geral desta pesquisa.

Os dados foram extraídos através de intervenção realizada com a aplicação das ferramentas que apoiam a integração continua para mensurar o acréscimo da qualidade e velocidade de entrega de valor ao usuário. As percepções e resultados foram obtidos através de relatório de execução do Jenkins e questionário a equipe.

As análises dos dados coletado pelo Jenkins será avaliado com base na folha de verificação, que permite ao pesquisador uma rápida percepção da realidade e uma interpretação da situação, auxiliando na redução de erros além de evitar que o mesmo volte a ocorrer, já a percepção da equipe quanto à importância de se ter um ambiente ágil e suas percepções e resultados foram coletados através dos membros da equipe e seus gestores. Por fim, os processos e procedimentos que foram indicados como passivo de melhoria pelo agilômetro serão avaliados com base no diagrama de Causa-e-Efeito.



### **3.3. Técnicas e ferramentas de análise**

Quanto aos procedimentos técnicos, far-se-á uso de estudo de caso. Para tanto, será proposto à implantação das ferramentas que apoiarão a integração contínua, que com o auxílio da folha de verificação identificaremos a distribuição do processo de produção, verificação de itens defeituosos, possíveis causas de defeitos, coletar dados sobre a frequência de defeitos e suas causas, verificar a execução com foco na entrega de valor ao usuário e delegar tarefas para que sejam automatizadas e assim garantir um produto mais coeso.

Quanto à aceitação dos métodos ágeis será utilizada a ferramenta “Agilômetro”, após a identificação dos pontos que apresentaram a necessidade de utilização de ferramenta para a automatização do processo de entrega, a fim de auxiliar na entrega de valor do produto ou serviço.

O controle de processos consiste em três ações fundamentais: planejamento, manutenção dos níveis de controle e melhorias (Schwarzer, 2014). Para avaliar a eficiência da integração contínua sobre o processo de entrega de valor ao usuário será utilizado o diagrama de Causa-e-Efeito que permite corroborar as informações obtidas pela folha de frequência.

### **3.4. Análise e discussão dos resultados**

#### ***3.4.1. Análise inicial do Agilômetro***

A organização já trabalha com o desenvolvimento ágil e para mensurar o nível de maturidade, foi realizada uma avaliação da organização utilizando o Agilômetro nos 6 (seis) aspectos essenciais, a fim de obter o nível geral e específico da aceitação da metodologia ágil, os seis aspectos são: (i) Flexibilidade com relação às entregas; (ii) Nível de colaboração; (iii) Facilidade de Comunicação; (iv) Capacidade de trabalhar com entregas iterativas e incrementais; (v) Fatores ambientais e; (vi) Aceitação dos métodos ágeis.

Para o aspecto de flexibilidade com relação às entregas, foi obtido à pontuação 5 com a justificativa que a empresa utiliza corretamente o conceito de MVP, incorpora mudanças ao produto e remove itens desnecessários sempre que possível.

Para o aspecto de nível de colaboração, obteve nota 4, devido a organização já se fomenta uma cultura ágil, pela alta gestão, o nível de colaboração é bom, mas ainda existem equipes que exigem certa formalidade na colaboração.

No aspecto de facilidade de comunicação, obteve a nota 5, devido à facilidade de comunicação tanto com o imediato superior quanto a alta gestão, também importante reportar que as equipes ágeis trabalham de forma integralizada.

No aspecto da capacidade de trabalhar com entregas iterativas e incrementais, a nota obtida foi 5, devido à empresa apresentar maturidade de planejar iterativamente seus projetos, visando à entrega de valor ao usuário.

No aspecto de fatores ambientais a nota obtida foi 3, devido à empresa favorecer o uso de metodologia ágil, entretanto, a entrega de valor ao usuário é impactada devido à falta de integração contínua, tanto os processos de correção de funcionalidades quanto os processos de melhorias geralmente se veem impactados pela falta de um processo mais automatizado.

No aspecto de aceitação dos métodos ágeis, a nota foi 5 (cinco), pois a empresa já apresenta maturidade para perceber que tem um ganho no uso da metodologia e como diferencial a empresa já está implantando a metodologia fora da área da TIC.

### ***3.4.2. Análise da Folha de Verificação***

Com o objetivo de analisar a efetividade da solução proposta foi elaborada uma folha de frequência com as seguintes indagações: Quantas vezes houve entrega de valor? Quantas vezes houve algum impedimento técnico? e Quantas vezes houve problema de qualidade?

Em 65% das vezes que o Jenkins foi acionado para executar o processo descrito na solução proposta, houve a entrega de valor ao usuário. Em 15% das vezes que o Jenkins auxiliou na construção e entrega do artefato o mesmo apresentou problemas de ordem técnica. Já 19,5% das vezes o Jenkins, aliado ao SonarQube, impediu que débitos técnicos chegassem à produção.

### ***3.4.3. Análise da Avaliação pela Equipe***

Com o intuito de se aprofundar sobre a importância da integração continuada após a implantação das ferramentas propostas, foi elaborado um questionário com dez questões fechadas e aplicado à equipe. Para deixar que as formalidades não obstruíssem a pesquisa, o pronome de tratamento “você” foi utilizado.

A primeira questão levantada foi questionando se o colaborador já conhecia o método ágil, 36,4% responderam que tinha algum conhecimento, e 63,6% responderam que conhecem plenamente.

A segunda questão levantada foi sobre qual a percepção sobre o método ágil, como resposta, 36,4% responderam como “é impressionante e devem ser adotados pela empresa”, outros 36,4% responderam como funcional, e 18,2% informaram que “é interessante”, e 9,1% responderam como “impressionante”.

A terceira questão levantada foi se o membro da equipe já conhecia o Jenkins, 63,6% responderam que tinha algum conhecimento, e 36,4% responderam que conhecia plenamente.

A quarta questão levantada foi “se o membro da equipe já conhecia o SonarQube”, 45,5% respondeu que tinha algum conhecimento, para as respostas “conheço plenamente”, “não conheço, mas sei que existe” e “não conheço” tiveram 18,2% cada.

A quinta questão levantada foi questionando aos membros da equipe se já conheciam o Scrum como metodologia ágil, 45,5% responderam que tinha algum conhecimento, e já 54,5% responderam que conheciam plenamente.

A sexta questão teve como questionamento se o membro da equipe já conhecia o Maven, 36,4% responderam que sim, outros 36,4% responderam que lembrava vagamente de algo, e 18,2% responderam que não, mas que sabia que existe, e apenas 9,1% respondeu como não tinha ideia do que seria.

A sétima questão teve como foco o questionamento se os métodos e procedimentos internos se tornaram mais ágeis e menos complexo, 72,7% responderam que sim, contra 27,3% de não.

A oitava pergunta questionou sobre se o membro da equipe preferia trabalhar em ambientes hierarquizados ou integrados, e como resposta teve 81,8% responderam “achei interessante e mais dinâmico trabalhar em ambientes integrados”, contra 18,2% que responderam que “a política motivacional existe e é amplamente eficaz no cotidiano corporativo contribuindo para um clima organizacional”.

A nona pergunta, qual é a percepção sobre a nova metodologia de trabalho, 72,7% responderam que teve uma melhora significativa e 27,3% responderam que teve melhorias.

A décima pergunta teve como objetivo questionar ao membro da equipe se a integração contínua melhorou a gestão na entrega de valor ao usuário, 90,9% responderam que “Sim”, contra 9,1% de “Não”.

A décima primeira questão foi deixada em aberto com a finalidade de que os membros da equipe registrassem suas opiniões sobre o assunto, sendo respondida por um participante: *“Gosto mais do método ágil Lean Kanban. Além disso, sem teste codificado não existe integração continua. Sem testes só existe uma melhora na infraestrutura. Qualidade não pode ser opcional”*.

#### **3.4.4. Resultado das Análises**

A avaliação inicial do agilômetro demonstrou que a empresa já se encontra em um estado de maturidade bem avançado quanto à aceitação de metodologias ágeis, inclusive um dos seus dilemas internos é “BeAgile”, entretanto, a mesma não se saiu bem quanto ao aspecto de fatores ambientais que de acordo com agilômetro a nota deve ser superior a 3 que não foi o caso. Esta análise inicial permitiu a identificação da necessidade da integração contínua como uma forma de aumentar a entrega de valor ao usuário.

Quanto à avaliação da folha de frequência, em apenas para uma API o Jenkins aliado ao sonar impediu que débitos técnicos chegassem à produção na ordem de 66,6%, o que podemos concluir que o Jenkins corrobora com a qualidade almejada pelo projeto.

Diante dos resultados do questionário, pode-se perceber que os membros da equipe que responderam, mostraram-se satisfeito com o resultado obtido com a solução proposta e com a percepção que a integração continua melhora a gestão na entrega de valor ao usuário em 90,9%, ou seja, que a integração continuada é de fundamental importância em ambiente de desenvolvimento ágil.

Ainda podemos concluir que foram aplicados testes de integração no momento da construção dos artefatos e que no período estudado houve 46 construções sendo 30 com sucesso de um universo de 11 API's.

## CAPÍTULO 4 - ESTUDO DE CASO

### 4.1. Contexto da organização

O estudo foi feito utilizando uma organização que está inserida no ramo de seguros, WIZ Soluções e corretagem de Seguros S/A. A empresa conta com uma iniciativa voltada para a *insurtech*.

De acordo com (Prado, 2017) “Burocracia, falta de personalização, inacessibilidade financeira e dificuldade para entender os planos são alguns dos problemas que os usuários enfrentam quando está à procura de contratar uma seguradora, a organização que este estudo se baseia em uma *insurtech*”.

O Termo *insurtech* é o resultado da junção das palavras *insurance* (seguro) e *technology* (tecnologia). Há algum tempo a tecnologia já permeia o mercado de seguros, mais precisamente na década de 90, muitos processos foram automatizados com a finalidade de remover os gastos excessivos nas centrais de atendimento, entretanto, o corretor de seguros em detrimento as seguradoras em sua grande maioria ainda não disponibiliza recursos para melhorar seu atendimento no seu processo de prospecção e venda.

As *insurtechs* surgiram com o propósito de revolucionar o setor de seguros, ela visa justamente, melhorar de forma significativa tanto a procura pelo seguro, quanto atender ao corretor, ou seja, adicionar uma forma mais rápida, fácil, confiável na utilização, compra, venda de seguros em geral.



**INSURTECH = INSURANCE + TECHNOLOGY**

**Figura 8.** Insurtech termo que oriunda de duas palavras Insurance e Technology.

Fonte: Prado, José (2017)

A organização analisada no estudo de caso também se encontra utilizando esquadras por produtos e cada produto atende a um tipo de plano de seguros onde as equipes já utilizam metodologias ágeis, desde a concepção do produto ou funcionalidades, até sua manutenção.

É importante frisar que a organização já é uma empresa digital e busca novos horizontes e modelos que atendam a necessidade do negócio de *insurtechs* e apesar da utilização de metodologias ágeis, sua base de ferramentas se encontra incompletas, gerando atrasos e proporcionando uma visão turva das funcionalidades que se encontram em desenvolvimento, teste, homologação e em produção.

## 4.2. Análise do mercado de seguros

A ideia inicial para a concepção do produto se deu por uma pesquisa de mercado utilizando o conceito de oceano azul.

Na concepção de Kim (2005) “os oceanos vermelhos são todos os setores existentes no mercado”. O autor faz uma comparação do mercado com águas em um oceano, e este espaço disputados, e à medida que mais competidores lutam por uma fatia desse mercado o lucro vai ficando cada vez menor e os produtos se tornando commodities. Essa briga cada vez mais desenfreada vai tornando as águas ensanguentadas. Não que navegar por águas vermelhas seja necessariamente ruim, o ganho sempre é a melhoria continua nos processos, aumento significativo da tecnologia, mas também existe o risco de seu produto e serviço saturar o mercado pela oferta em demasia do mesmo.

Os oceanos azuis representam os mercados que ainda não foram explorados, geralmente sempre que surge uma nova tecnologia esses mercados ficam pungentes de serem descobertos. Fato significativo é que com a concorrência desenfreada dos oceanos vermelhos logo surge novas tecnologias que quase sempre criam oceanos azuis propícios para obter uma boa navegabilidade.

Também e não menos importante, o mercado de seguros no Brasil está com penetração inferior à média mundial, e se aliado à tecnologia, os prêmios poderão ser significantes. Para isso contamos com uma pesquisa do *Swiss Re Institute* (2018), nos mercados emergentes, as premissas de vida e não vida aumentou respectivamente, 14% e 6,1% em 2017. No setor de não vida o aumento desacelerou, todavia, se manteve bastante robusto. A pesquisa mostra que a desaceleração foi em grande parte impulsionada pela China, tendo ficado com 10% apenas. Os mercados de seguros em países emergentes como o Brasil as rendas, rendimentos e bens tem tido um aumento o que acaba impulsionando a procura por seguros.

Jérôme Haegeli, economista-chefe do Swiss Re Group declara:

Naquela época, a Ásia avançada e emergente contabilizava 5% dos prêmios de seguro globais, contra 22% em 2017. Na próxima década, é provável que esse deslocamento para a China continue. Dado o número impressionante de iniciativas de infraestrutura em andamento na China, a contribuição chinesa para os prêmios de seguro mundiais pode novamente exceder as expectativas. Nas décadas seguintes, outros mercados como Índia, Indonésia, Brasil, México, Paquistão, Nigéria e Quênia podem se tornar mais importantes (sindsegs, 2018).

Classificação por volume de prêmios	Prêmios de vida			Prêmios de não vida			Prêmios totais		
	US\$ bilhões	Alteração* vs 2016	US\$ bilhões	Alteração* vs 2016	US\$ bilhões	Alteração* vs 2016	Densidade de seguros (US\$)	Penetração de seguros	
	2017		2017		2017		2017	2017	
<b>Mercados avançados</b>	<b>2'059</b>	<b>-2.7%</b>	<b>1'760</b>	<b>1.9%</b>	<b>3'820</b>	<b>-0.6%</b>	<b>3'517</b>	<b>7.8%</b>	
Estados Unidos	1	547	-4.0%	830	2.6%	1'377	-0.1%	4'216	7.1%
Japão	3	307	-6.1%	115	0.0%	422	-4.5%	3'312	8.6%
Reino Unido	4	190	-0.7%	93	0.5%	283	-0.3%	3'810	9.6%
França	5	154	-2.7%	88	1.1%	242	-1.3%	3'446	8.9%
Alemanha	6	97	-1.8%	126	1.3%	223	-0.1%	2'687	6.0%
Coreia do Sul	7	103	-6.5%	78	2.3%	181	-2.9%	3'522	11.6%
Itália	8	114	-7.5%	42	-0.5%	156	-5.7%	2'660	8.3%
<b>Mercados emergentes</b>	<b>598</b>	<b>13.8%</b>	<b>474</b>	<b>6.1%</b>	<b>1'072</b>	<b>10.3%</b>	<b>166</b>	<b>3.3%</b>	
América Latina e Caribe		78	1.1%	90	-0.9%	168	0.1%	262	3.1%
Brasil	12	47	1.2%	36	1.6%	83	1.4%	398	4.1%
México	25	12	1.0%	13	0.9%	25	1.0%	196	2.2%
Europa Central e Oriental		19	12.2%	44	3.3%	63	5.8%	198	1.9%
Rússia	28	6	48.2%	16	-5.4%	22	4.4%	152	1.4%
Ásia Emergente		448	17.7%	272	10.1%	720	14.7%	188	4.1%
China	2	318	21.1%	224	10.2%	541	16.4%	384	4.6%
Índia	11	73	8.0%	25	16.7%	98	10.1%	73	3.7%
Oriento Médio e Ásia Central		15	7.0%	45	4.1%	60	5.0%	163	2.1%
Emirados Árabes Unidos	35	3	3.3%	10	13.5%	14	11.0%	1'436	3.7%
África		45	0.3%	22	1.0%	67	0.5%	54	3.0%
África do Sul		38	-0.3%	10	1.3%	48	0.1%	842	13.8%
<b>Mundo</b>	<b>2'657</b>	<b>0.5%</b>	<b>2'234</b>	<b>2.8%</b>	<b>4'892</b>	<b>1.5%</b>	<b>650</b>	<b>6.1%</b>	

Notas: \* em termos reais, ou seja, ajustados à inflação.

Penetração de seguro = prêmios como uma % do PIB; Densidade dos seguros = prêmios per capita

Fontes: números finais e provisórios liberados por autoridades supervisoras e associações de seguros, e algumas estimativas.

**Figura 9.** Desenvolvimento nos principais mercados de seguros em 2017.

Fonte: Swiss Re Institute (2018).

### 4.3. A legislação Brasileira

O Decreto-Lei n. 296, de 28 de fevereiro de 1967, disciplina a venda conforme seu art.22, *in verbis* “O Corretor de seguros, pessoa física ou jurídica, é o intermediário legalmente autorizado a angariar e promover contratos de seguro entre as Sociedades Seguradas e as pessoas físicas ou jurídicas de Direito Privado” (BRASIL, 1966).

### 4.4. Mitigação de riscos externos

No Brasil, conforme a legislação vigente, apenas o corretor de seguro é a entidade autorizada a comercializar produtos de seguros, com isso em mente nosso produto foca na persona do corretor de seguros, ao qual será apresentada uma plataforma onde o corretor possa ter mais agilidade no atendimento ao segurado.

## Visão Geral da Plataforma



**Figura 10.** Visão Geral da Plataforma.

Fonte: Autor (2019).

### 4.5. Tipos de produtos oferecidos aos corretores

Foi feito uma avaliação inicial dos produtos oferecidos no mercado. Atualmente, segue a seguinte lógica: seguradoras exclusivas, corretora exclusiva.

#### 4.5.1. Seguradoras Exclusivas

Essas plataformas têm a finalidade de atender a corretores e segurados através de uma única seguradora, atualmente podemos encontrar as seguintes: kakau e a youse. Entretanto, hoje no Brasil só temos esse tipo de plataforma atendendo com corretoras exclusivas que, neste caso, não atende os corretores, apenas os segurados.



**Figura 11.** Site da Kakau, uma seguradora exclusiva.

Fonte: Kakau (2019)

#### 4.5.2. Corretora Única



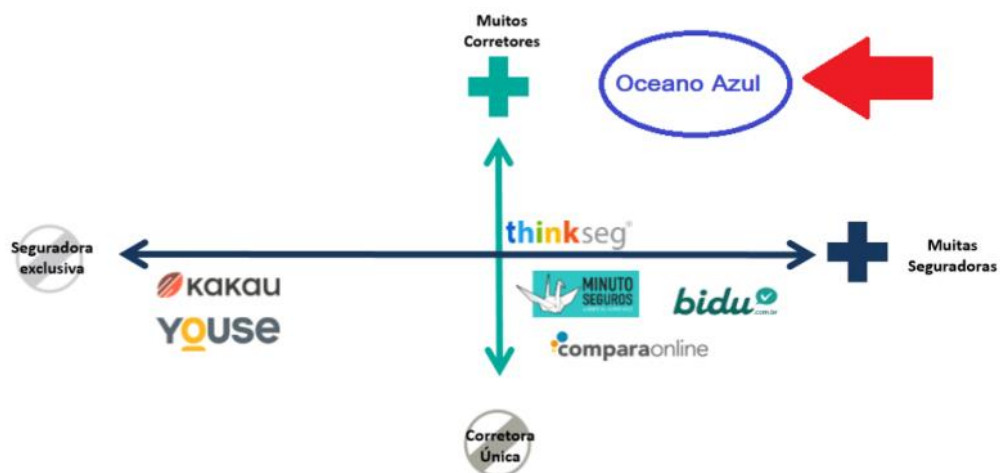
Essas plataformas têm a finalidade de atender apenas aos segurados, dando-os a possibilidade de escolha entre diversas seguradoras. Geralmente, eles fornecem um serviço de multicálculo que permite o usuário decidir pelo melhor preço para o segurado.



**Figura 12.** Site da Baidu. Plataforma com foco em corretores únicos.  
Fonte: Baidu (2019).

#### 4.6. Oceano Azul pretendido

Após uma pesquisa de mercado, foi identificado que os produtos oferecidos se enquadravam como seguradoras exclusivas ou de corretora única, dentre este universo estudado não identificamos uma plataforma de serviços que atendia aos conceitos de múltiplas seguradoras e múltiplas corretoras de seguros.



**Figura 13.** Oceano azul pretendido.

Fonte: Autor (2019)

#### 4.7. Cadeia de atendimento no mercado de seguros

Na cadeia no mercado de seguros basicamente existe três perfis distintos e harmoniosos entre si, de um lado temos as seguradoras que de fato irão segurar o produto ou serviço por um dado período, de acordo com um valor estipulado de prêmio.

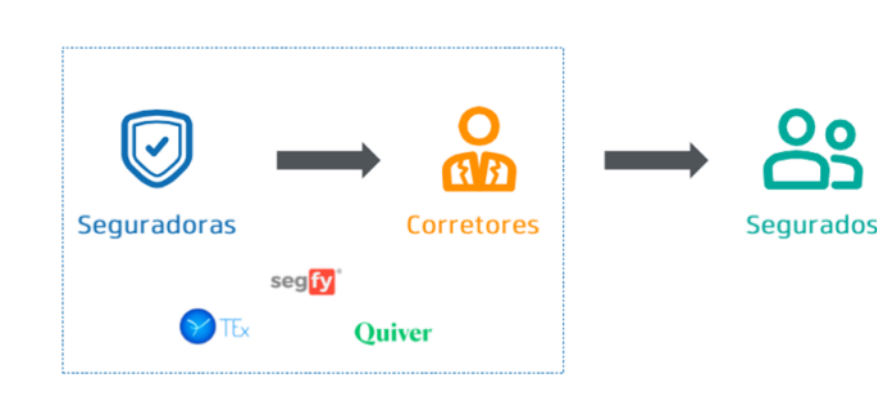
O Segurado é a pessoa que espera que receba o valor estipulado na apólice, caso venha ocorrer algum tipo de sinistro.



**Figura 14.** Cadeia de atendimento no mercado de seguros.

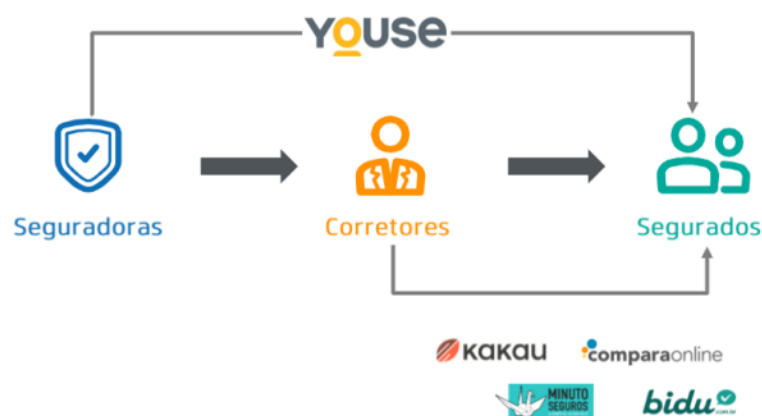
Fonte: Autor (2019)

O produto da empresa do estudo de caso tem como foco principal atender à necessidade do corretor de seguros, apresentando os produtos oferecidos no mercado atualmente que atendem a cadeia completa, de maneira que ele possa atender a expectativa de seus clientes.



**Figura 15.** Ferramentas que não atendem a cadeia de atendimento completa.

Fonte: Autor (2019)

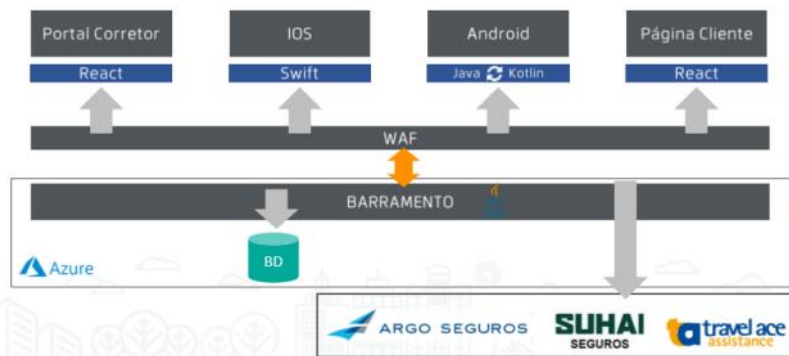


**Figura 16.** Plataformas incompletas.

Fonte: Autor (2019)

#### 4.8. Tecnologia da Plataforma

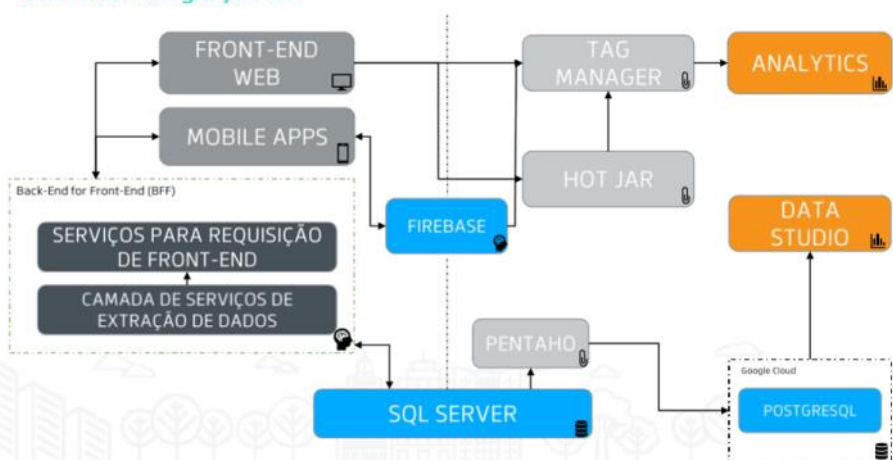
A plataforma fora pensada para atender a cadeia completa (do mercado de seguros), propondo-se a ser um diferencial no mercado, visando se integrar com o máximo de seguradoras e não menos importante atender ao corretor na árdua tarefa de efetivar o produto aos seus clientes além de ser um ponto onde o segurado possa obter suas apólices e acionar seus sinistros.



**Figura 17.** Tecnologia da Plataforma.

Fonte: Autor (2019)

#### Overview Integração B.I.



**Figura 18.** Integração B.I

Fonte: Autor (2019)

## 4.9. Ambiente de Desenvolvimento

O ambiente de desenvolvimento da plataforma utiliza o conceito de equipes baseado no *Spotify Engineering Culture*, buscando empregar um modelo interno de código aberto, uma cultura mais compartilhada do que proprietária. Baseado no respeito mútuo e pouco “ego”<sup>6</sup>. Procura-se ter uma revisão de código por pares, onde qualquer um pode adicionar qualquer código a qualquer momento. Em seguida, um par<sup>7</sup> pode rever o código e fazer ajustes. Todo mundo colabora e espalha o conhecimento. Também procuramos fomentar uma cultura que se concentra na motivação, a fim de construir ótimo local de trabalho.

<sup>6</sup> Em um ambiente tradicional de trabalho, cada profissional traz consigo um perfil egoísta, ou seja, de auto capacidade de produção, natural de todo ser humano e típico dos modelos hierarquizados de decisão. No ambiente ágil, esse egoísmo é deixado de lado em prol de um trabalho colaborativo.

<sup>7</sup> Por “par” deve-se entender dois membros da equipe ágil que estão imersas em determinada solução.

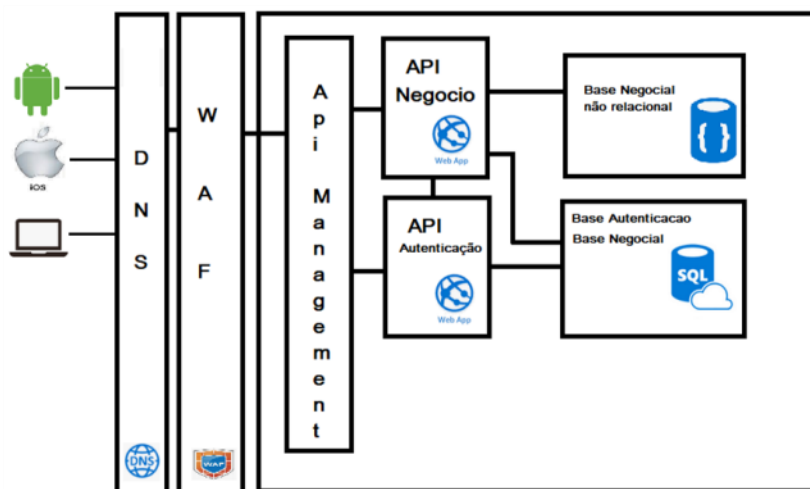
Em outras palavras o ambiente de desenvolvimento da plataforma se baseia no modelo de trabalho do *spotify*, buscando fomentar uma cultura onde todos os membros da equipe agem de forma a melhorar o código. Esta cultura permite que toda a equipe conheça o que a plataforma faz, proporciona a equipe ter uma visão das limitações negociais e tecnologias, além de obter mais sinergia da equipe.

#### 4.10. O Ambiente de Produção

Para o ambiente de produção foi disponibilizado uma nuvem computacional e essa escolha se deve ao fato de permitir o escalonamento e dimensionamento elástico de acordo com o uso, além de proporcionar uma melhor gestão do ambiente pela equipe como um todo.

Para o ambiente de produção foi selecionado o tipo de serviço *PaaS* que fornecem um ambiente de acordo com a demanda para desenvolvimento, teste, fornecimento e gerenciamento de aplicativos de software.

Para a segurança do ambiente também foi utilizado um detector de intrusão baseado em um serviço SaaS por assinatura. Para a integração da plataforma com as seguradoras utilizamos um serviço IaaS, porque neste caso foi necessário devido à necessidade de criação de um mecanismo de busca das apólices automático.



**Figura 19.** O Ambiente de Produção.  
Fonte: Autor (2019)

#### 4.11. Esquadras de trabalho (*Squads*)

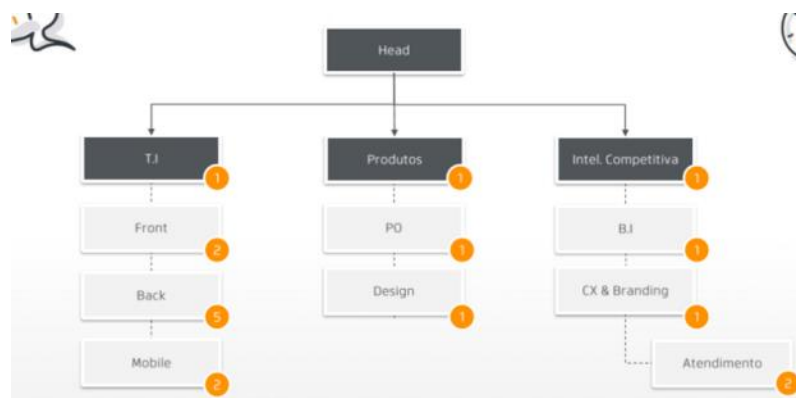
Esquadra de trabalho trata-se de uma equipe Cross-funcional onde os membros da equipe têm responsabilidade de ponta a ponta, e isso é possível juntando diversos perfis e disciplinas para entregar valor ao negócio que tenha relevância no mercado. Na esquadra de trabalho o fator principal é a autonomia. A utilização deste modelo de trabalho se deve ao fato de permitir uma

melhor sinergia entre membros que normalmente não interagiriam entre si, evitando problemas de comunicação, menos burocracia além de aumentar a latência das entregas no processo de construção da plataforma.

Diferentemente de outras formas de trabalho, mesmo com especialidades diferentes de cada membro da equipe, nada impede que outro membro não possa opinar ou sugerir algo que no modelo tradicional de trabalho estaria além de suas atribuições.

#### 4.12. O Time

Para a plataforma ser construída foi utilizada uma equipe de 30 (trinta) funcionários, sendo: (1) um *Head* de Operações; (2) um Gerente de TI; (3) um Gerente de Produtos; (4) um Gerente de Inteligência Competitiva; (5) dois Analistas Front-End; (6) cinco analistas Back-End; (7) dois Analistas Mobile; (8) um Product Owner; (9) um Designer; (10) um Analista de B.I; (11) um Analista de CX & Branding; (12) dois Analistas de atendimento e; (13) os demais estão ligados aos serviços administrativos e de vendas. Mesmo a cultura procurar um ambiente horizontal onde cada membro possa colaborar, ainda existe um nível hierárquico com a finalidade de obter que tipo de profissional possa compor a equipe multidisciplinar na construção da plataforma.



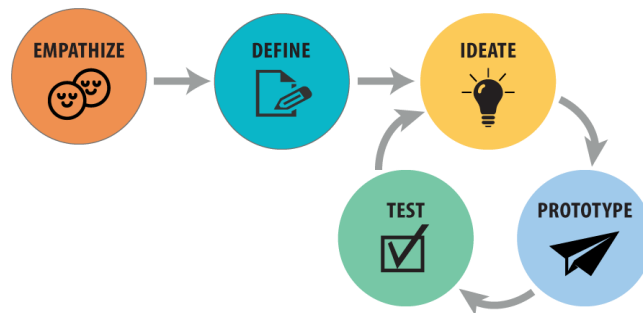
**Figura 20.** O Time.

Fonte: Autor (2019)

#### 4.13. Gerenciamento do projeto

A estratégia de gerenciamento de projetos na empresa do ramo de seguros, objeto do nosso estudo de caso, é baseada em metodologia ágil se valendo do uso do Scrum para a entrega de valor. O principal fator de escolha para utilização de métodos ágeis em detrimento aos métodos tradicionais tais como os descritos no PMBOOK foi que existia a necessidade de se adaptar ao mercado de forma rápida, ágil e dinâmica.

Para esta análise prévia dos projetos, foi utilizado à ferramenta *Design Thinking*:



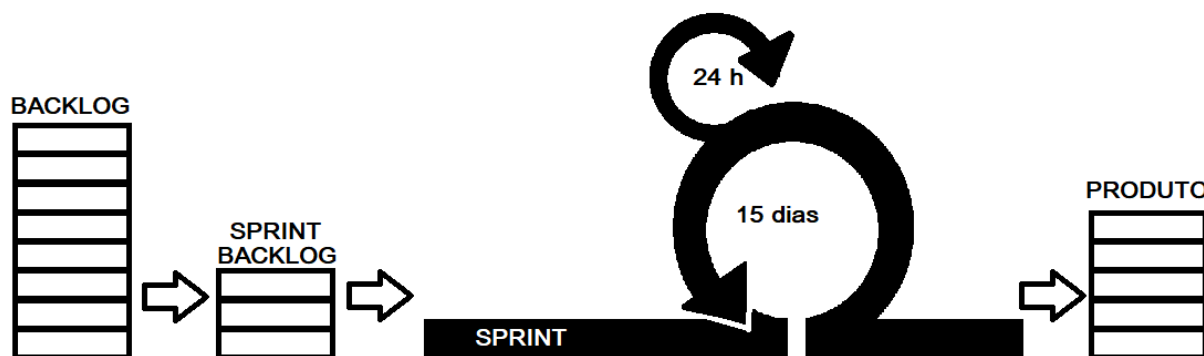
**Figura 21.** *Learning Solutions Process.*

Fonte: Learning Solutions (2018)

De acordo com a descrição da Learning Solutions:

O Design Thinking é eficaz porque requer equipes multidisciplinares. Essa mistura de perspectivas únicas e variadas cria a faísca que pode levar ao pensamento inovador. O Design Thinking exige que você entenda o público e o usuário em um nível profundo, observando-os e conhecendo-os (a fase de empatia). Isso leva a entender a causa e não o aspecto superficial de um problema (a fase de definição). Suporta pensamento divergente, para que as equipes possam gerar uma ampla gama de soluções. (Learning Solutions, 2018, p. 1)

Para auxiliar a equipe multifuncional o a metodologia utilizada para a construção da solução é o *Scrum*. Trata-se de uma metodologia utilizada para auxiliar equipes ágeis no desenvolvimento do software, entretanto, pode ser utilizado por qualquer equipe ágil que queira produzir outro tipo de produto além da construção de software.



**Figura 22.** Processo fim a fim do Scrum.

Fonte: Autor (2019)

Diferentemente de outras metodologias ágeis, o *Scrum* permite iterações com menores tempos, geralmente cada *Sprint* é de 4 (quatro) semanas, mas para a construção da plataforma fora decidido sprints de 15 dias tendo a vantagem de mudança de estratégia a tempo caso haja a necessidade.

#### 4.14. Metodologia e dinâmica dos trabalhos na Empresa do Estudo de Caso

Ao término do *Sprint*, o *Product Owner*, junto com o Head de Operação, decide quais *features* irão entrar em produção, então utiliza o processo de Integração Contínua para fazer a entrega em produção. O monitoramento do funcionamento da aplicação se dá através de *Dashboard* contabilizando usuários, acesso às funcionalidades e afins, utilizando Data Science.

A equipe segue o modelo ágil e se baseia no *Scrum* que se divide basicamente em três perfis: o *Product Owner*; o *Scrum Master* e; a *Equip* e na utilização do *Kanban* como um agregador de atividades diárias. O *Scrum* foi o mais utilizado para montar o *sprint-backlog* as *sprints* e cerimônias de entregas (*sprint review*).

Deve-se considerar que os únicos processos ágeis conhecidos, preteritamente pelas equipes, eram o *Scrum* e *Kanban*.

O trabalho das equipes, a partir dos processos e procedimentos ágeis, foi de forma colaborativa, sendo detectada uma lacuna quando houve a necessidade de solicitar a entrega do produto em ambiente de produção, sendo sanado em dado momento da integração contínua.

Cabe ressaltar que na empresa em estudo, além de ser uma empresa altamente baseada em processos ágeis, suas equipes utilizavam o modelo de esquadras multidisciplinares, o que reduzia a quase zero o ruído de comunicação, também não menos importante, um “*agile coach*”, que orienta as equipes quanto ao melhor método de seu utilizar os processos ágeis. Apesar do foco na entrega de valor ao cliente, no entanto, não há um processo que avalie a satisfação do cliente.

Por fim, os desenvolvedores também são foco de avaliação, pois existe um *feedback* anualmente, com base em sua produtividade, a análise técnica dos códigos e do próprio produto entregue, além de um *feedback* que em cada *sprint review*.

#### **4.15. O produto**

O produto é uma plataforma de serviços voltada ao corretor de seguros denominado ZIM que após estudo de mercado e visando navegar em um oceano azul, onde a concorrência é irrelevante.

A plataforma ZIM consta de um APP e um CRM, e não outro produto, como, por exemplo, um site, blog ou software para aplicação de mesa, é que esta plataforma permite mobilidade a persona estudada (Corretor de Seguro) que se vale do advento da tecnologia para alavancar seu negócio. Além do mais, foi estudada a forma de fornecer o nosso aplicativo utilizando as lojas (IOS, Android) como forma do APP alcançar mais personas além do que as estudadas, considerando que no Brasil há quase um smartphone por habitante, são razoáveis que aplicativos para celulares sejam mais requisitados do que um microcomputador de mesa ou notebook.

A Plataforma foi elaborada por uma equipe, que se valeu do processo de *Design Thinking* para concepção do produto com as seguintes tarefas:

- 1) Empatia;



- 2) Definição;
- 3) Idealização;
- 4) Protótipos;
- 5) Testes.

O processo de construção do software se valeu do uso da metodologia Scrum com as seguintes tarefas;

- 1) Product Backlog;
- 2) Sprint Backlog;
- 3) Sprint;
- 4) Sprint review.

O *Software* foi desenvolvido em linguagem JAVA, para o *Back-End* da plataforma, posto que aliado com o framework da *Spring*, tendo em vista que se trata do quadro de trabalho de propósito geral mais completo do mercado além de ser código aberto, bem como fácil delineamento dos serviços baseados em *REST* com validação de acesso utilizando-se da JWT (*Jason Web Toolkit*), com o objetivo de escalonar por demanda as funcionalidades foi utilizando arquitetura de micro serviços e, por fim, verificação simples e fácil de atualizar a qualquer tempo.

Foi contratado um serviço de nuvem para hospedarmos os serviços, a equipe se reuniu e foram definidos os requisitos do software, de acordo com o estudo da persona definido no processo de concepção do produto:

- 1) Multicálculo;
- 2) Cadastro de apólices;
- 3) Cadastro de clientes;
- 4) Aviso de vencimento de apólices;
- 5) Obtenção de apólices do segurado nas seguradoras.

. E os requisitos não funcionais:

- 1) Escalabilidade
- 2) Disponibilidade
- 3) Latência

Foi considerando ainda, na etapa prévia aos trabalhos, uma análise de cenário acerca do público-alvo do projeto, bem como da concorrência e o tamanho do mercado.

O projeto final, feitas todas as verificações, foi apresentado ao cliente pela equipe com uma lista de todas as suas funcionalidades, bem como suas limitações:



**Figura 22.** Plataforma ZIM para os corretores de seguros.

Fonte: Autor (2019)

Deve-se ressaltar que nessa primeira etapa o aplicativo foi formatado para um público-alvo, inicial, de aproximadamente 50 mil pessoas, ou seja, espera-se que num período de 12 meses, que cerca de 50 mil downloads sejam efetivados.

O suporte ao cliente será feito de forma remota, com monitoramento de bugs, e armazenamento de informações e dados, colhidos pelos APPs, em “nuvem”.

Dessa forma, a partir dessa primeira etapa, será apresentada uma segunda versão, considerando todas as informações colhidas no período anterior, atualizada e que possa atender às demandas do cliente e seus usuários. Assim, serão considerados nessa segunda etapa pela equipe: 1) o código utilizado foi o ideal; 2) houve lacunas nas verificações; 3) O suporte remoto pode ser aprimorado; 4) A configuração do APP atende às expectativas do cliente e seus usuários; 5) O que pode ser melhorado?

## CAPÍTULO 5 - FERRAMENTAS PROPOSTAS

Para este trabalho, antes de informar as ferramentas propostas que terão por finalidade a automatização e permitirão a implantação da integração contínua como um diferencial estratégico na entrega de valor, far-se-á uso de ferramentas de código aberto que além de se alinhar com o propósito estabelecido, fornecem acesso aos códigos fontes, e também podem utilizar ferramentas elaboradas para uso livre, o que implica na ausência de gastos com licenças. Os únicos custos serão os custos necessários na utilização de qualquer ferramenta de software tais como rede, energia, sala-cofre, etc...

Uma das principais ferramentas de integração contínua do mercado é o *Jenkins*. A escolha do mesmo foi baseada nos custos, sendo que não há necessidade de se pagar licença tendo em vista ser uma ferramenta *open source* e, junto a isso, foi apresentada também algumas ferramentas que aliada ao *Jenkins* permitirão guardar o que foi produzido, o que permite contar com uma ferramenta de análise de código estático para se obter uma métrica dos códigos que forem construídos e alterados pela equipe.

### 5.1. Maven

Um dos principais problemas ao se desenvolver um sistema, e fazer com que a equipe faça a entrega de valor o mais cedo possível, e para que isso ocorra o reuso de componentes é fundamental. Copiar esses componentes para dentro do projeto torna o trabalho oneroso além de caso ocorra alguma mudança na versão deste componente pode implicar em horas de retrabalho para identificar problemas de versões utilizadas pelos membros da equipe.

O Apache Maven, ou simplesmente Maven foi criada com o propósito de ser uma ferramenta que auxilia a automação e compilação primeiramente para atender os anseios da comunidade JAVA, contudo, com a sua arquitetura baseada em plugins logo começou a ser utilizada para outras plataformas como PHP, .NET entre outros.

O Maven é uma ferramenta de automação de compilação que também se vale do conceito de gerenciamento de dependências, e com isso se torna possível gerenciar os componentes que estão sendo compartilhado pela equipe, além de adicionar informações de como o projeto será construído de acordo com o *profile* definido em seus arquivos de configuração. O maior poder do Maven e em sua grande parte à pela sua extensibilidade através de uso de *plugins* que podem automatizar tarefas na construção do projeto.

Além de permitir a extensibilidade através de plugins ele é construído utilizando uma arquitetura baseada em plugins, e teoricamente, isso permite uma personalização apurada dessa ferramenta o que torna seu uso altamente recomendado.

### **5.1.1. Project Object Model**

Podemos dizer que o *Project Object Model* (POM) fornece as configurações para o projeto, sua configuração cobre diversos aspectos como nome de projeto a proprietário além de permitir a extensibilidade das funcionalidades através de *plugins*, além permitir comportamentos de acordo com o ciclo de vida de construção, e especificar novas ou alterar dependências existentes no projeto.

### **5.1.2. Plugins**

Todas as funcionalidades do Maven estão em plugins, isso se dá pelo fato de sua arquitetura baseada em plugins, cada plugin fornece um comportamento específico ao Maven, como por exemplo, um projeto pode ser compilado através do plugin de compilador e ele pode ser testado usando um plugin de teste.

Inicialmente, o Maven já vem equipado com plugins de construção, teste, geração de projetos, criação de arquétipos (templates para projetos), por óbvio também se pode adicionar plugins disponível na comunidade ou se criar plugins personalizados para utilização pela equipe.

Os plugins são introduzidos dentro do Maven através da sessão de plugins definidos dentro do *Project Object Model* (POM), os plugins são o modo primário de se seja feita a extensão do Maven.

### **5.1.3. Ciclos de vida de Construção**

Ciclo de vida de construção é um conjunto de fases pré-definidas como intuito de serem usadas para a construção de um projeto, e como tudo no Maven também pode ser estendido e personalizado. Os comportamentos fornecidos pelos plugins podem ser associados a diversas fases do ciclo de vida do projeto.

O Maven já vem com um ciclo de vida padrão, e este é denominado *default lifecycle*, este ciclo de vida padrão permite que o projeto seja construído, empacotado entre outros. O ciclo de vida padrão consta com as principais fases, exatamente nesta ordem:

- *process-resources* (processar recursos)
- *compile* (compilar)
- *process-test-resources* (processar recursos de teste)
- *test-compile* (testar compilação)
- *test* (testar)
- *package* (empacotar)
- *install* (instalar)
- *deploy* (implantar)

Mais abaixo iremos verificar sucintamente o que cada fase do ciclo padrão faz.

#### 5.1.3.1. *Process-resources (processar recursos)*

A fase de processar recursos, faz-se uma cópia dos recursos para o diretório destino, deixando-o disponível para outras fases, em outras palavras, nesta fase o Maven irá copiar os códigos fontes dentro da pasta *SRC* e irá disponibilizá-la na pasta *target*. É nesta fase que alguns plugins adicionam recursos como arquivos de configurações, ou, ainda poderão adicionar funcionalidades padrões para o projeto.

#### 5.1.3.2. *Compile (compilar)*

Esta fase irá compilar o projeto, ou seja, transformar o código fonte em linguagem de máquina. Nesta fase também que pode ser utilizado compiladores para diversas linguagens o que torna o Maven extremamente poderoso sendo possível ser utilizado tanto para compilação de módulos de *front-end* como de *back-end*.

#### 5.1.3.3. *Process-test-resources (processar recursos de teste)*

Esta fase irá gerar os arquivos de teste e copiá-los para o diretório destino de testes, em outras palavras, nesta fase os arquivos dentro da pasta *test* irá ser disponibilizado para a próxima fase que compilará os arquivos de testes e subsequentemente executar os testes.

#### 5.1.3.4. *Test-compile (testar compilação)*

Esta irá fazer a mescla do código fonte disponibilizada no diretório *target* pela fase de *process-resources* com o código escrito com a finalidade de testar o aplicativo disponibilizado pela fase de *process-test-resources*, e nesta fase que alguns *plugins* são utilizados para mudar o comportamento da aplicação através de programação orientada a aspecto.

#### 5.1.3.5. *Test (testar)*

Essa fase o Maven, procurar fazer os testes da aplicação utilizando alguma suíte de testes, um ponto importante e esta fase é pré-requisito para as fases de *package* e *deploy*. Nessa fase também alguns plugins irão coletar dados com a finalidade de obter parâmetros de qualidade, segurança, entre outros. Um dos plugins geralmente utilizados é o de integração com o SonarQube.

#### 5.1.3.6. *Package (empacotar)*

Vários tipos de projetos escrito em diversas linguagens necessitam de uma estrutura de arquivos e diretórios geralmente diferente da estrutura de arquivos e diretórios que são construídos pelo desenvolvedor, também em algumas linguagens a estrutura de arquivos e diretórios sofre um adendo sendo a conversão destes arquivos claros em arquivos binários entendidos apenas por máquinas. Alguns plugins que utilizam essa fase são os que empacotam projetos com as dependências como parte do código fonte.

#### *5.1.3.7. install (instalar)*

Esta fase tem como função a entrega do pacote produzido para o repositório local do Maven, não diferente das outras fases seu comportamento pode ser expandido através da criação de plugins, como podemos perceber o Maven permite que seus comportamentos padrões sejam especializados em qualquer fase do ciclo de vida de construção.

#### *5.1.3.8. Deploy (implantar)*

Esta fase tem como função a fazer a entrega no repositório remoto geralmente NEXUS ou Artifactory, de onde alguma ferramenta de integração continua irá buscar para fazer a implantação em algum ambiente.

#### *5.1.4. Dependências*

O Maven também funciona muito bem como um gerenciador de dependências para o projeto, ao informar qual dependência seu projeto irá utilizar o Maven através das configurações informadas na lista de repositórios, se incumbirá de localizá-la e como a busca se dá através do protocolo HTTP, muitas fabricantes de soluções para TI disponibilizam suas soluções na internet. Essas dependências ao serem baixados serão disponibilizadas no diretório *repository*, dentro da pasta informada na variável de ambiente *M2\_HOME*, além dos empacotamentos que foi gerado pelo ciclo de vida de construção, o uso de uma pasta comum visa um repositório único de dependências com a finalidade de facilitar a gestão das dependências locais.

#### *5.1.5. Profiles*

De acordo com as informações do Tutorials Point (2019), “um perfil de compilação é um conjunto de valores de configuração que pode ser usado para definir ou substituir valores padrão da compilação do Maven”.

Ainda de acordo com o referido tutorial, um perfil permite a customização de acordo que se é possível construir o pacote com diversas configurações de acordo com o ambiente que se pretende fazer a entrega como produção ou homologação.

Os perfis são definidos no arquivo de configuração, eles indicam na execução do ciclo de construção do projeto parâmetros que de acordo com os profiles e assim é possível se ter uma construção e entrega de acordo com o especificado.

#### 5.1.6. Tipos de perfil de compilação

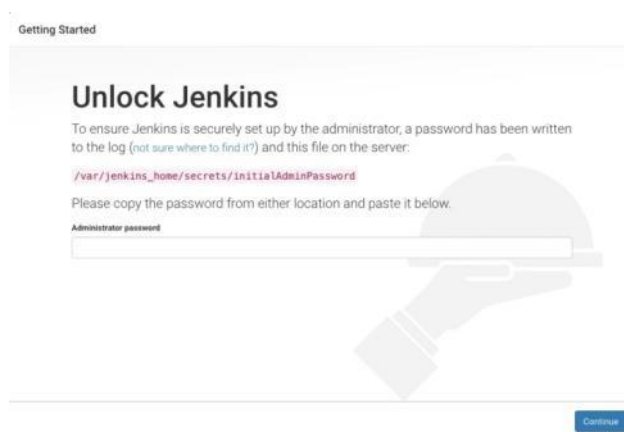
Existem ainda os perfis de criação para se construir um *profile*: (1) *profiles* que define configurações para projeto; (2) *profiles* para configurações de usuário e; (3) *profiles* que visam definir configurações de escopo global no projeto. Estes tipos definiram uma configuração específica para os profiles, que permite uma customização enorme para o projeto em si.

### 5.2. Jenkins

De acordo com Boaglio (2016), “o *Jenkins* é um servidor de integração contínua open source e é feito em Java, que disponibiliza mais de 1.000 (mil) *plugins* para suportar construção (*build*) e testes de virtualmente qualquer tipo de projeto”.

Com ele poderemos automatizar os processos do ciclo de desenvolvimento dos projetos das equipes ágeis permitindo uma agilidade na entrega do valor ao usuário. O *Jenkins* irá auxiliar na execução de testes, rodar *scripts*, atualizar as aplicações em seus ambientes.

O *Jenkins* existe no mercado há aproximadamente 10 (dez) anos e vem demonstrando ser uma ferramenta robusta e capaz de atender as equipes ágeis de diversos tamanhos e modelos, equipes funcionais e Cross<sup>8</sup>-funcionais além de ser uma ferramenta de código aberto. Por estes atributos apresentados é que foi escolhido para uso na empresa do presente Estudo de Caso.



**Figura 23.** Tela inicial do Jenkins.

Fonte: Jenkins (2019)

---

<sup>8</sup> É preciso que a equipe seja formada por pessoas com todas as habilidades necessárias para desenvolver o produto (*por exemplo: desenvolvedores, designers, UX designers, testadores, analistas de negócio*), todo mundo trabalhando na mesma equipe e colaborando para atingir um **objetivo comum**. (Faria, 2016).

Também, mas não menos importante, o *Jenkins* trabalha com projetos feitos com uma gama imensa de linguagens de programação o que nos permite que tenhamos equipes multidisciplinares.

#### **5.2.1. Unidade de trabalho (Job)**

A unidade de trabalho permite se construir unidade para execução de diversas tarefas, essas tarefas podem ser integradas ao fluxo de valor a fim de se reaproveitar tarefas em fases diversas do fluxo.

#### **5.2.2. Fluxo de valor (Pipeline)**

Os *pipelines* modelam o seu processo de entrega (fluxo de valor), ao invés de torná-lo dependente apenas de opiniões e decisões sem embasamento em dados concretos.

O fluxo típico de valor de entrega dentro das organizações tem uma semelhança: atipicidade. A maioria dos processos de entrega é diferente entre si, de acordo com o produto que está sendo desenvolvido. Ou seja, não há uma fórmula que possa ser usada em qualquer situação: é preciso realizar adaptações para atender às necessidades e peculiaridades de cada organização.

#### **5.2.3. Plugins**

O *Jenkins* também permite que seus comportamentos sejam estendidos e para isso conta com a possibilidade de criar componentes para isso.

#### **5.2.4. Vantagens**

- *Builds* periódicos;
- Testes automatizados;
- Possibilita análise de código;
- Identificar erros mais cedo;
- Fácil de operar e configurar;
- Comunidade ativa;
- O *Jenkins* integra outras ferramentas através de *plugins* existentes na própria aplicação.

### **5.3. Flyway**

O banco de dados é uma ferramenta digital que ajuda na organização interna, e pode definir os rumos de uma organização.



Além de tarefas operacionais, essa ferramenta preciosa ajuda uma organização a conhecer o seu público, manter contato com clientes e construir um bom relacionamento com eles, vender e avisar sobre novos produtos ou serviços, alertar sobre ofertas e promoções especiais, enviar informações a um grupo seletivo de pessoas, traçar estratégias sólidas de crescimento entre outras coisas.



**Figura 24.** Tela inicial do Flyway.  
Fonte: Flywaydb (2018)

É neste cenário de extrema importância que nos deparamos com o *Flyway*, que nos auxilia na automatização de migração de bases de dados de um ambiente a outro no passo de cliques com o auxílio do *Jenkins*. O *Flyway* trabalha com um conjunto de conceitos sendo eles: migração, chamadas de retorno, manuseio de erros e somente o necessário.

### 5.3.1. Migração

Com o *Flyway*, todas as alterações no banco de dados são chamadas de migrações. As migrações podem se criar versões ou repetidas. As migrações com versão vêm em duas formas: regular e desfazer. As migrações com versão têm uma versão, uma descrição e uma soma de verificação. A versão deve ser exclusiva. A descrição é puramente informativa para que você possa lembrar o que cada migração faz. A soma de verificação está lá para detectar alterações acidentais. As migrações com versão são o tipo mais comum de migração. Eles são aplicados na ordem exatamente uma vez. Opcionalmente, seu efeito pode ser desfeito fornecendo uma migração desfazer com a mesma versão.

Migrações repetíveis têm uma descrição e uma soma de verificação, mas nenhuma versão. Em vez de serem executados apenas uma vez, eles são reaplicados toda vez que sua soma de verificação muda. Em uma única execução de migração, as migrações repetíveis são sempre aplicadas por último, depois que todas as migrações com versão pendentes foram executadas. Migrações repetíveis são aplicadas na ordem de sua descrição.

Por padrão, as migrações com controle de versão e repetíveis podem ser gravadas em SQL ou em Java e podem consistir em várias instruções. O *Flyway* descobre automaticamente as migrações no sistema de arquivos. Para acompanhar quais migrações já foram aplicadas quando e por quem, o *Flyway* adiciona uma tabela de histórico de esquemas ao seu esquema.

### **5.3.2. Chamadas de Retorno (Callbacks)**

Embora as migrações sejam suficientes para a maioria das necessidades, há certas situações que exigem que você execute a mesma ação repetidamente. Isso poderia ser recompilar procedimentos, atualizar visões materializadas e muitos outros tipos de tarefas domésticas. Por esse motivo, o *Flyway* oferece a você a possibilidade de conectar-se ao seu ciclo de vida usando *Callbacks*.

### **5.3.3. Manuseio de Erros (Error Handlers)**

Quando o *Flyway* executa instruções SQL, ele reporta todos os avisos retornados pelo banco de dados. No caso de um erro ser retornado, o *Flyway* o exibe com todos os detalhes necessários, marca a migração como falha e volta automaticamente para o estado de volta, se possível.

### **5.3.4. Somente o necessário (Dry Runs)**

Quando o *Flyway* migra um banco de dados, ele procura por migrações que precisam ser aplicadas, classifica-as e as aplica em ordem direta ao banco de dados.

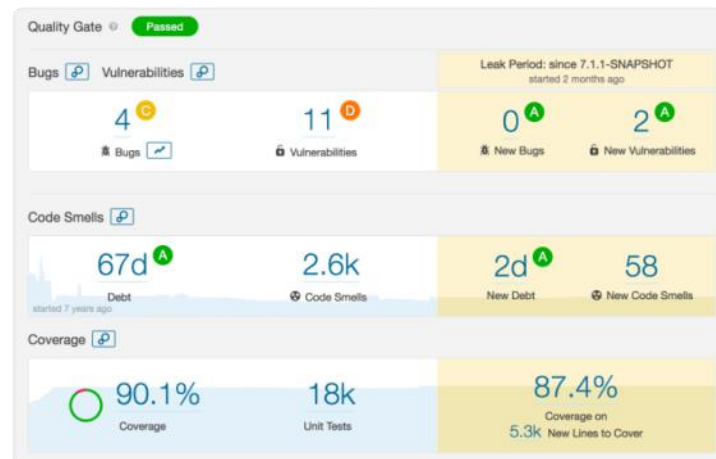
- Esse comportamento padrão é ótimo para a grande maioria dos casos.
- Existem, no entanto situações em que você pode querer.
- Visualize as mudanças que o *Flyway* fará no banco de dados

## **5.4. SonarQube**

“Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente” (Manifesto Ágil, 2001). As mudanças sempre serão bem-vindas em equipes ágeis. Isso se dá devido à procura para se obter um diferencial estratégico diante dos concorrentes. Não é muito

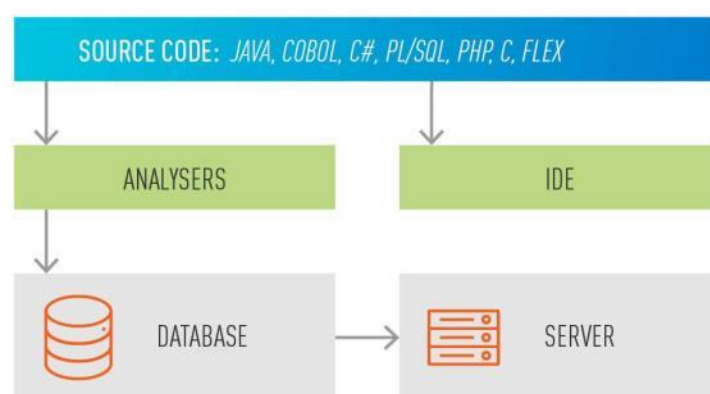
difícil que o código comece a se degradar com as mudanças e isso invariavelmente em algum momento irá causar transtornos.

É muito importante constantemente analisarmos o código de nossos sistemas, essa tarefa é necessária para cada vez mais termos certeza de que o que foi criado realmente atende ao negócio e que o desenvolvedor não terá mais problemas com o que foi escrito.



**Figura 25.** Tela informacional sobre a Qualidade do Projeto.  
Fonte: SonarQube (2019)

O SonarQube possui um numeroso e extenso conjunto de regras para várias linguagens de programação. Ele também permite vários *plugins*, assim, você pode escrever seus próprios *plugins*, caso queira escrever suas próprias regras. A plataforma do SonarQube é composta por 4 (quatro) componentes: 1) analisadores; 2) servidor; 3) *plugins* instalados e; 4) base de dados.



**Figura 26.** Arquitetura do SonarQube.  
Fonte: Menkovic (2016)

### 5.4.1. Analisadores

Os analisadores são responsáveis pela execução da análise do código linha por linha. Eles podem fornecer informações sobre a dívida técnica, cobertura do código, complexidade do código, problemas detectados e etc. Os problemas detectados no código podem ser *bugs*, *bugs* potenciais, tópicos que podem levar a erros no futuro. Quando a análise é concluída, os resultados podem ser visualizados em uma página hospedada no servidor do SonarQube. Há diversos problemas que podem ser encontrados em um código.

As regras são diferentes e podem ser classificados em cinco grupos, com base no tipo e na sua gravidade: Bloqueador, Crítico, Principal, Pequeno e Info. Assim temos como principais benefícios para a utilização de um verificador estático, os seguintes pontos:

- Erros e códigos de risco são encontrados mais facilmente;
- Programadores passam a ter uma visão analítica objetiva, que possibilita e ajuda no reconhecimento de onde os mesmos foram ou não precisos e desatentos;
- Líder de projeto adquire uma oportunidade de estudar o código, seu projeto e sua equipe, passando a possuir uma perspectiva diferenciada;
- Com a exclusão de determinadas classes de defeitos, a equipe se encontra concentrada em outros tipos de eficiência do projeto.

### 5.4.2. Métricas

E quanto às métricas onde cada qual mede um aspecto, sendo elas:

#### a) Complexidade

Refere-se a contagem dos caminhos possíveis que o fluxo de controle que um método pode tomar. Pode ser calculado a partir da representação do programa como um grafo, representando instruções como nós e o fluxo de controle como arestas, um bloco de decisão como um comando *if* ou um bloco *while* divide o fluxo de controle do método em dois.

A complexidade é um dos principais fatores na avaliação da manutenção de qualquer artefato de software e tende a ser concentrada fortemente nas camadas de negócio das aplicações. A presença de valores altos de complexidade na camada de persistência e apresentação é indicativa de uma arquitetura mal planejada. Limitações da tecnologia utilizada também podem afetar a distribuição da complexidade no projeto.

O aumento da complexidade requer um esforço significativo da equipe de testes para que estados possíveis no programa não fiquem sem cobertura. A complexidade possui uma ligação direta com a cobertura de testes, criando a necessidade de considerar a cobertura dos possíveis caminhos de controle no programa separada da cobertura geral, garantindo uma melhor distribuição dos testes. Essa necessidade é suprida pela a adição da medida de cobertura por caminho.

b) Índice de Emaranhamento de Pacote

Métrica composta que mede de 0 a 100 o esforço necessário para remover ciclos de dependências entre pacote. Aplica-se especialmente bem a arquiteturas em camadas com regras restritivas de acesso. Um valor baixo indica que não existem dependências cíclicas entre os pacotes do projeto, ou que a dependência é pontual e o esforço necessário para eliminá-la é baixo. Um valor alto indica um grande esforço de reescrita necessário para retirar os ciclos de dependência. A métrica afeta diretamente o débito técnico do projeto.

c) Linhas Duplicadas

Importante para evitar a propagação de erros por cópia de código antigo. Um valor alto é indicador de uma solução mal planejada, o que dificulta a reutilização do código e a manutenção. Outra causa possível é uma limitação da tecnologia utilizada. A métrica contribui para o débito técnico e aumenta as chances de violações à arquitetura se propagarem do código legado para novos projetos.

d) Cobertura de Testes de Unidade e de Integração

Importante para a manutenção e confiabilidade do sistema. Além de assegurar o bom funcionamento da aplicação, a cobertura de testes é importante para verificar os requisitos e facilitar a manutenção e evolução do projeto. A métrica sofre uma limitação primordial na medição da distribuição da cobertura. Se a cobertura de testes for concentrada em um conjunto de artefatos que não realizam operações de negócios, o valor da métrica não se traduz em qualidade no projeto final.

É preciso estabelecer regras para a cobertura baseadas na arquitetura do sistema, focando o esforço de testes na camada de negócios e nos artefatos mais importantes. Um controle rígido da nomenclatura dos artefatos pode facilitar aplicação desta métrica.

e) LCOM4

Define-se a Métrica LCOM 4 como:

uma variação da métrica LCOM descrita na suíte de Chidamber-Kemerer. É utilizada para checar se a classe analisada obedece ao princípio da única responsabilidade. A métrica é calculada a partir do uso das variáveis locais por bloco de código. Um bloco é definido a partir dos delimitadores da linguagem e as chaves são utilizadas para delimitar os blocos no caso do Java. O valor da métrica é incrementado em um para cada grupo isolado de blocos utilizando um conjunto de variáveis locais. (Moreira, 2015, p. 15).

f) Débito Técnico

Quanto ao débito técnico, Lucas Neto Moreira, da UnB, faz uma excelente análise das suas causas e consequências:

Métrica fundamental para a visualização da saúde do projeto pela gerência faz uma conversão do esforço necessário para corrigir as falhas arquiteturais do projeto em um custo baseado no preço do homem-hora praticado na organização. A metáfora da dívida inclui o pagamento de juros, na forma de homem/horas gastas no desenvolvimento devido ao peso das falhas arquiteturais, falhas como alto acoplamento, que dificultam a evolução

do projeto e fazem com que novas funcionalidades levem mais tempo para ser construídas. As fontes de débito técnico podem ser de natureza intencional ou não intencional. Fontes intencionais incluem todas as práticas adotadas com o intuito de diminuir os custos ou aumentar a produtividade em detrimento das boas práticas de qualidade de software. Fontes não intencionais incluem diversos fatores, como decisões de projeto propensas a erros, adoção de novos componentes com débito técnico acumulado e a descoberta de vulnerabilidades em componentes utilizados, entre outros aspectos fora do controle ou do conhecimento da equipe. A medida do débito técnico deve ser adaptada para a organização interessada em controlá-lo, tanto no valor do homem-hora como nos fatores que aumentam o débito, como violações pontuais à arquitetura, valores altos em métricas de acoplamento, erros de projeto que dificultam a construção de novas funcionalidades, entre outros. (Moreira, 2015, pp. 15-16)

#### 5.4.3. SonarQube e integração contínua

Aliado ao nosso servidor de integração contínua, permite a automatização com ganho de tempo, criando regras que impeçam de a integração continuar, caso o código produzido venha a adicionar algum débito técnico.



**Figura 27.** Gerenciamento do Código Fonte com o Jenkins e o SonarQube.

Fonte: Menkovic (2016)

Isso permite rastrear o controle de qualidade do que está sendo produzido a cada entrega. Além disso, tem-se o histórico sobre a qualidade do código, para observarmos o comportamento dessa qualidade durante o projeto.

O uso do SonarQube facilita o controle de qualidade do código e diminui o número de bugs reais e potenciais. A equipe fica mais focada na lógica e podem dedicar seu tempo para os requisitos de análise do negócio e para encontrar a solução ideal para casos concretos. Com isso os gerentes começaram a monitorar métricas, pois, com base nos seus resultados, é possível ter uma visão melhor do produto desenvolvido.

## 5.5. Git

De longe, o sistema de controle de versão moderno mais utilizado no mundo hoje é o Git. O Git é um projeto livre maduro, mantido ativamente, originalmente desenvolvido em 2005 por Linus Torvalds, o famoso criador do *kernel* do sistema operacional Linux. Um número

impressionante de projetos de *software* depende do Git para controle de versão, incluindo projetos comerciais e código aberto. Os desenvolvedores que trabalharam com o Git estão bem representados no conjunto de talentos de desenvolvimento de *software* disponíveis e funcionam bem em uma ampla gama de sistemas operacionais.

Tendo uma arquitetura distribuída, o Git permite que cada colaborador tenha um repositório em sua máquina, apesar de cada colaborador ter também uma versão do histórico de entregas de outros colaboradores, o mesmo conta com um servidor de atualizações que opera como um semáforo, e a cada entrega feita por um colaborador ele atualiza e monta uma árvore de atualizações baseada em suas entregas.

É um exemplo de um DVCS (daí o *Distributed Version Control System*). Em vez de ter apenas um único local para o histórico da versão completa do software, como é comum em sistemas de controle de versões populares como o CVS ou o *Subversion* (também conhecido como SVN), no Git, a cópia de trabalho de cada desenvolvedor é também um repositório que pode conter o histórico completo de todas as alterações.

#### **5.5.1. Comunicação**

O Git se comunica com os repositórios remotos através da rede interna ou pela própria internet. Três protocolos para esse tipo de comunicação são fornecidos pelo Git sendo eles: SSH, Git, HTTP/HTTPS.

- Protocolo SSH

O Git trabalha com o protocolo SSH com leves alterações em sua URL de comunicação, essa opção além de ter um custo computacional barato permite que os arquivos sejam transmitidos de uma forma rápida e garantindo um bom nível de segurança, ideal para a utilização com equipes remotas que utilize o protocolo TCP/IP para trafegar as alterações em seus arquivos.

- Protocolo GIT

O Git também possui um protocolo próprio, este protocolo de comunicação foi desenvolvido para ter mais velocidade que outros protocolos. Apesar de ser uma boa opção para equipes que queiram trabalhar com um protocolo veloz, o mesmo pode apresentar um problema tendo em vista que permite acesso anônimo aos recursos mesmo tendo um controle de acesso na persistência dos arquivos.

- Protocolo HTTP/HTTPS

Para finalizar o Git trabalha com o protocolo HTTP/HTTPS que é utilizado como o melhor recurso quando a equipe não quer mexer nas configurações de rede interna, além de ter um controle de acesso que garante a leitura somente se o usuário tiver o acesso a determinado recurso esta opção é um pouco mais demorada devido ao fato de esta no topo da pilha de comunicação.

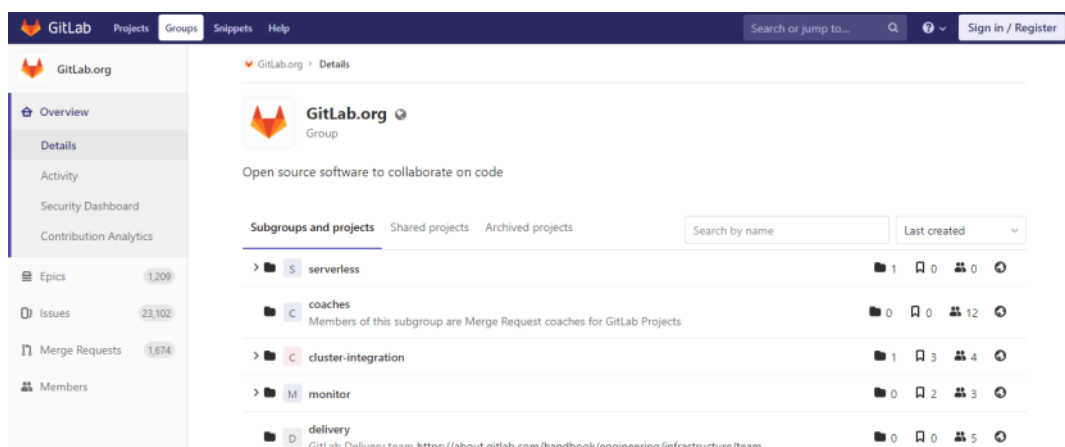
A comunicação entre a estação de trabalho do colaborador com o Git se dá através de seus protocolos. O protocolo Git para este trabalho não foi utilizado tendo em vista que o mesmo permite que o acesso ao conteúdo dentro do controle de versão se de forma anônima, sobrando o SSH e o HTTPS/HTTP que permitem um controle de acesso do colaborador, tanto na leitura quanto na escrita dos repositórios.

## 5.6. Gitlab

Apesar de o Git permitir trabalhar com um controle de versão e histórico localmente, este trabalho irá abordar a utilização do conceito de controle de versão distribuída é para isso a ferramenta selecionada foi o Gitlab. O controle de versão distribuída permite que cada colaborador tenha certo nível de autonomia quando a produção de conteúdo para o projeto.

O Gitlab é uma plataforma de código aberto com a finalidade de gerenciar vários repositórios permitindo que a instituição possa ter em um só local vários repositórios. O acesso à área gerencial dos repositórios também é uma facilidade tendo em vista que o gestor apenas precisar acessar o sistema com um navegador, entretanto, ainda podemos contar com:

- Controle de acesso ao repositório
- Gerenciamento de usuários
- Gerenciamento de chaves públicas
- Procurar o código-fonte
- Procurar uma entrega
- Visualização de arquivos com destaque
- Histórico do arquivo
- 



**Figura 28.** Gerenciamento dos repositórios.  
Fonte: Gitlab (2019)



A escolha do Gitlab para este trabalho dá-se ao fato de permitir vários repositórios além de garantir para a toda a equipe uma visibilidade maior do que foi entregue.

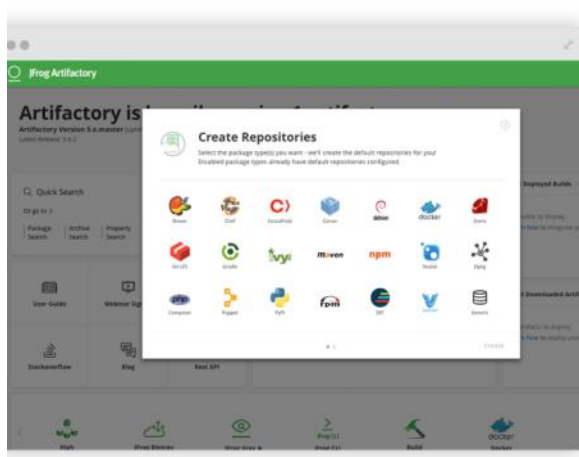
## 5.7. Artifactory

Com o aumento constante de demandas por parte dos usuários e da concorrência por parte do mercado, fatores como a confiabilidade dos sistemas e o tempo de recuperação após falhas podem ser decisivos para o sucesso de um produto. Por isso, o mercado de TI tem inovado ao criar soluções que visam à melhoria do trabalho das equipes e da qualidade dos serviços criados.

Não é raro que equipes visando o reaproveitamento de componentes criem soluções compartilhadas que possam ser utilizadas entre diversos projetos, por isso surge a necessidade de ter um local onde possam ser armazenados esses códigos para um uso futuro.

O *Artifactory* é um sistema open source que é usado como repositório de arquivos binários. Existe uma versão paga dele, com suporte a outros tipos de repositório, como por exemplo, pacotes RPM de algumas distribuições Linux (Boaglio, 2016). Além de funcionar também como um gerenciador de dependência externa de nossos projetos, também irá guardar os pacotes de cada versão que for para a produção que nossas equipes ágeis irão entregar.

Ele também permite que o desenvolvimento e a inovação possam ser realizados de maneira mais rápida, centralizar componentes comuns e servir de repositório para diversas versões de componentes para serem utilizados por diversos projetos.



**Figura 29.** Lista de repositórios compatíveis do Artifactory.

Fonte: Jfrog (2019)

### 5.7.1. Principais vantagens

- Economia de tempo

Uma única fonte confiável para a gestão dos componentes utilizados no desenvolvimento resulta em menos tempo gasto na busca ou na espera de downloads lentos.

- Maior produtividade

A alta disponibilidade nativa garante que suas equipes tenham o componente certo e a versão correta para entregar aplicativos sem demora.

- Controle de qualidade

Você terá um local centralizado para controle de qualidade, operações e provisionamento, melhorando a análise e o monitoramento de seus processos.

- Confiabilidade

A comunicação criptografada (SSL) mantém o seu trabalho de desenvolvimento a salvo de problemas de segurança, como ataques, bem como outros problemas relacionados à privacidade.

Além disso, o *Artifactory* oferece a possibilidade de realizar a criação de repositórios compatíveis com o gerenciador de componentes da sua linguagem preferida, ou então se pode criar um simples repositório HTTP. Para o trabalho em equipe, a ferramenta permite o compartilhamento integrado entre grupos de desenvolvedores, como a publicação de coleções de artefatos que podem ser testados, promovidos ou descartados de maneira fácil e rápida.

#### **5.7.2. Por que utilizar o Artifactory?**

- Integração entre times

Com o *Artifactory* há uma fonte única e confiável para realizar o armazenamento e o gerenciamento dos artefatos produzidos.

### **5.8. Microsoft Azure**

Apesar de não pagarmos por licenças quanto ao uso das ferramentas selecionadas, temos a necessidade de utilizar um servidor para que as mesmas sejam hospedadas e a escolha desta nuvem se dá ao fato de que a Microsoft Azure se comporta bem entre nuvem híbridas e privadas.

A Microsoft Azure é um conjunto cada vez maior de serviços de nuvem para ajudar sua organização a enfrentar seus desafios de negócios. É a liberdade de criar, gerenciar e implantar aplicativos em uma enorme rede global usando suas ferramentas e estruturas favoritas (Microsoft, 2018).

#### **5.8.1. Quais são as vantagens do Microsoft Azure?**

O Azure é indicado para vários ambientes: hospedagem de websites e *e-commerces*, armazenamento de dados e *backup*, recuperação em caso de desastres, serviços de mídia para transmissão de vídeos, análise de dados ou Big Data e plataformas open source.

O empreendedor pode escolher os serviços que quer usar, na medida exata de sua necessidade, acrescentando ou removendo-os sempre que precisar. O controle e a configuração podem ser feitos online, do computador ou por dispositivos móveis.

- Serviço gabaritado

São vários os motivos para confiar na Microsoft com o Azure. A companhia é o primeiro provedor de nuvem reconhecido pelas autoridades de proteção de dados da União Europeia por respeitar as rigorosas leis de privacidade da região e foi o primeiro serviço da área a adotar o novo padrão de privacidade de nuvem internacional, ISO 27018.

- Disponibilidade

O Azure está disponível em 21 regiões e 90 mercados espalhados pelo mundo e dobra sua capacidade de armazenamento e computação a cada seis meses. O investimento da empresa ultrapassa 1 bilhão de dólares e, no Brasil, inclui uma data center já em operação dedicado aos serviços em nuvem.

- Contratos acessíveis

O Azure oferece tecnologia e alta capacidade de recursos das grandes corporações a um preço acessível e com modelos de compra que se adequam à realidade dos empreendedores. Um exemplo? Você paga apenas os serviços do Azure que efetivamente utilizar, sem mensalidade fixa ou compromisso mínimo. Se já tiver ideia do tamanho de infraestrutura que precisa, pode optar por um plano pré-pago e obter descontos adicionais.

## CAPÍTULO 6 - SOLUÇÃO PROPOSTA

O capítulo do Marco Teórico teve como objetivo apresentar e revisar os conceitos a fim de dar apoio a este trabalho que visa responder o objetivo de construir um conjunto de ferramentas que possa dar o devido aporte na entrega de valor e neste sentido apresentou de forma sucinta, conceitos que deram apoio ao capítulo de Ferramentas Propostas que apresentou um conjunto de ferramentas que possam ser utilizadas em ambiente ágil.

O fundamento do presente capítulo, Estudo de Caso, apresenta-nos um ambiente real com fins a apresentar a proposta de valor que será entregue para o usuário, visando identificar os pontos passíveis de serem automatizados, utilizando uma ou mais ferramentas para este fim.

Para responder ao objetivo específico deste capítulo utilizaremos a Integração Contínua, mas, antes de apresentar a solução utilizando a Integração Contínua, é necessária a apresentação do ciclo de desenvolvimento de um sistema.

### 6.1. Ciclo de Desenvolvimento de Sistema sem Integração Contínua

Como observado quando o desenvolvedor começa seu trabalho criando um código, a sua próxima tarefa é fazer a entrega no repositório de código fontes, subsequentemente executa o teste da aplicação, e no caso de sucesso vem a publicação deste sistema gerado a partir desse código, oriundo de um repositório de código fontes, e, finalmente, é feito o repasse para o usuário usar ou homologar.

Posteriormente, vêm as alterações no sistema, que podem ser melhorias ou correções de erro, onde o desenvolvedor retorna ao código. E assim o ciclo se repete iniciando na primeira etapa do ciclo de desenvolvimento. Pode-se resumir esse processo de desenvolvimento conforme a figura abaixo:



**Figura 30.** Ciclo de desenvolvimento sem Integração Contínua.  
Fonte: Boaglio (2016)

## 6.2. Ciclo de Desenvolvimento de Sistema com Integração Contínua

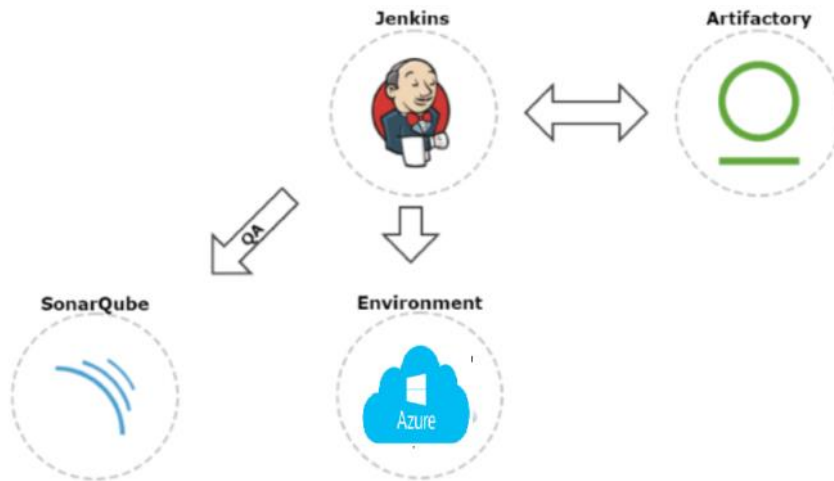
Ao verificar o ciclo de desenvolvimento sem a integração contínua pode-se perceber várias atividades que não agregam valor ao produto, entretanto, para proposta do ciclo de desenvolvimento utilizando integração contínua, faz-se uso do *Jenkins* como o orquestrador que irá interagir com as demais ferramentas que este trabalho apresentou.

O ciclo de desenvolvimento com integração contínua, o membro da equipe cria um código ou solicita um código para fazer alguma mudança e com o trabalho concluído o mesmo faz a entrega em um repositório e logo após esta tarefa o *Jenkins* identificará a mudança e se encarregar de fazer os testes, a publicação do sistema e avisará ao Scrum Master ou o PO de que uma nova versão foi publicada no ambiente. Deste modo a tarefa de codificar um sistema se torna menos onerosa, e com ganho de tempo.



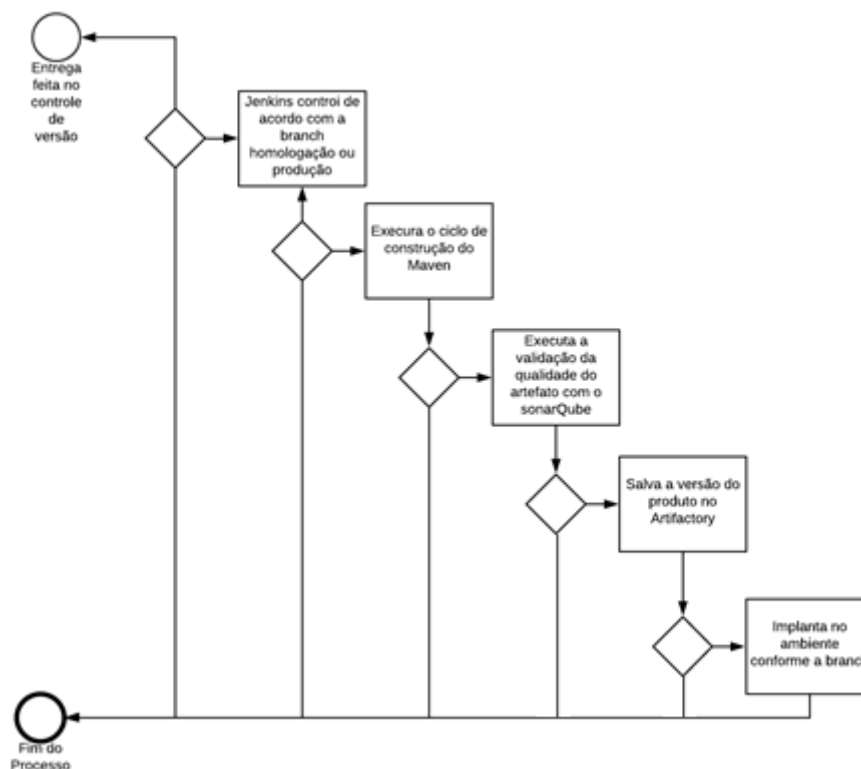
**Figura 31.** Ciclo de desenvolvimento utilizando Integração Contínua.  
Fonte: Boaglio (2016)

Também não menos importante a cada ciclo de desenvolvimento o *Jenkins* age como um orquestrador entre outras ferramentas como o *SonarQube* o *Artifactory*. Antes de fazer a publicação do sistema o *Jenkins* executa o *SonarQube* a fim de evitar que débitos técnicos sejam adicionados a solução, caso o *SonarQube* não encontre débitos técnicos o *Jenkins* irá publicar uma versão do código no *Artifactory* e na sequência irá fazer a publicação do artefato no ambiente conforme esquema demonstrado na figura 32:



**Figura 32.** Integração Jenkins, Sonar e Artifactory.  
Fonte: Autor (2019)

A integração entre as ferramentas com a finalidade de implantar a integração continua no ambiente foi utilizado o diagrama de atividade, nele se encontra as iterações possíveis entre o *Jenkins*, o controle de versão que no nosso caso estamos utilizando o *GIT* e não menos importante o *Maven* e o *Flyway*, conforme figura abaixo:



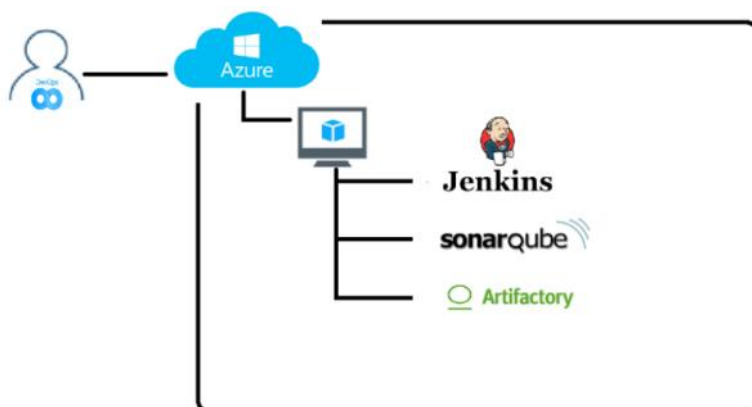
**Figura 33.** Fluxo entre o Git, Jenkins, Maven, SonarQube e Artifactory.  
Fonte: Autor (2019)

O fluxo de atividades entre as ferramentas acontece da seguinte forma:

- 1) Após o desenvolvedor fazer a entrega do código fonte no controle de versão, o GIT através de um *Webhook* ativa a atividade pré-agendada no *Jenkins* dando início da construção;
- 2) O *Jenkins* começa a construção do projeto através do Maven que inicia o ciclo de construção do projeto;
- 3) Nas fases do Maven, ele irá atualizar o banco de dados de homologação ou produção de acordo com a tarefa iniciada através do *Webhook* previamente configurado no GIT e também irá fazer a entrega do artefato no Artifactory permitindo uma rastreabilidade pós-toma além de disponibilizar o artefato para outras ferramentas como o Sonar que tem a finalidade de auferir a qualidade e o próprio *Jenkins* que irá fazer a implantação no ambiente previamente selecionado.

Para atualizar o banco de dados, foi utilizado o Flyway que é uma ferramenta que permite uma integração com o Maven e Jenkins, entretanto, devido a critérios de segurança o processo da atualização automática do banco de dados foi implantado apenas no ambiente de teste integrado e homologação.

Apesar de o *Jenkins* trabalhar de forma automatizada para a manutenção do ambiente precisamos de um analista DevOps, este profissional tem como objetivo garantir que as ferramentas estejam funcionando. Abaixo temos uma visão de integração onde apresentamos o analista de DevOps atuando junto à plataforma em nuvem e as ferramentas de que auxiliaram a integração continua.



**Figura 34.** DevOps e a Nuvem computacional.

Fonte: Autor (2019)

## CAPÍTULO 7 - CONCLUSÕES

Este trabalho apresentou vários fatores limitantes como a ausência de mais informações sobre a entrega continuada no ambiente de desenvolvimento ágil, assim como falta de uma visão holística do ambiente como um todo, apesar da cultura ágil já está sendo utilizado na instituição, houve dificuldade inicial na implantação das ferramentas que deram apoio à integração contínua além da falta de documentação de cunho acadêmico sobre esse assunto.

Por se tratar de uma pesquisa que visa atender ao estudo de caso e que venha a ter uma real utilidade pela sociedade, as ferramentas escolhidas foram com base no código aberto que permite a implantação por qualquer organização com custo quase zero em licenças e afins.

O fato da instituição estudada, WIZ Soluções e corretagem de Seguros S/A, já estão trabalhando com metodologias ágeis além do departamento de TIC, foi uma aliada na chegada de tais níveis. Conforme fora corroborado com o uso do Agilômetro.

O que não foi auferido foi à integração no ambiente de Data Science, e também não foi feito nenhuma pesquisa utilizando-se algum modelo de business process management para entrega de valor ao usuário, a aplicação basicamente consistia em um crud e estudos posteriores poderiam identificar formas de integração contínua onde a aplicação se utiliza de BPM<sup>9</sup> dinâmico ou estático dependendo do contexto.

### 7.1. Quando ao problema de pesquisa

É possível que implantação da integração continuada em ambientes de desenvolvimento ágil possibilite a entrega de produtos íntegros e dentro dos prazos acordados?”

Foi respondido de acordo com o disposto no capítulo 2, nos tópicos 2.6 a 2.6.4, e 2.6.6 e de 2.9 a 2.9.12, bem como no capítulo 3, tópico 3.4 e subsequentes.

### 7.2. Quanto ao objetivo geral

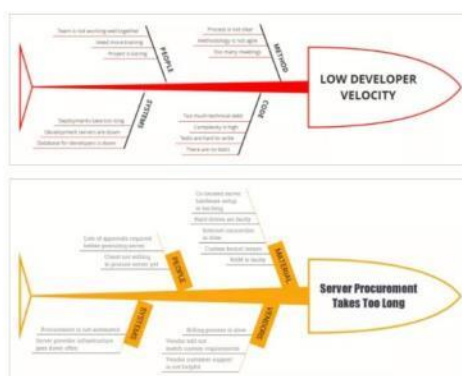
Com o uso do Agilômetro foi possível ter uma visão real do nível de maturidade quanto à adoção de metodologias ágeis, podendo assim identificar melhor pontos de melhoria e agir de forma mais assertiva em relação à solução proposta.

---

<sup>9</sup> É a abreviação de **Business Process Management**, que traduzido para o português **significa** Gerenciamento de Processos de Negócio. **BPM** é uma abordagem gerencial adaptável, desenvolvida com a finalidade de sistematizar e facilitar processos organizacionais individuais complexos, dentro e fora das empresas (Oliveira, 2014)



Para corroborar ao objetivo geral desta dissertação, foi utilizado o diagrama de causa e efeito é desenhado começando pelo efeito que está sendo produzido. Isso pode ser um erro, ou algum outro problema com o software ou o projeto de TI, ressalta-se que mecanismos de avaliação como ISHIKAWA são preponderantes para a memória laboral das equipes, de maneira a ter sempre soluções hábeis e oportunas para suplantar barreiras que se apresentem.



Fonte: Karasek (2018)

Método (inclui ciclo de vida de desenvolvimento de software, processo de controle de qualidade, metodologia ágil, metodologia SCRUM, *Waterfall*).

Pessoas (membros de sua equipe, colegas de trabalho e outras partes interessadas).

Assim, fez-se análise das causas comuns MVPS (Material, Fornecedores, Pessoas, as) e MCPS (Método, Código, Pessoas, Sistemas) como ponto de partida nesse estudo de

87

Deve-se ressaltar que em alguns casos as causas podem ser ignoradas porque são de baixo risco para o projeto. Todo projeto de desenvolvimento de software é diferente e cada problema pode ter uma ou várias causas. Digamos que “o cliente não está disposto a adquirir o servidor ainda” é uma causa extremamente provável do nosso problema. Assim como outros pontos como destacados a seguir:

O cliente pode não ter orçamento para o servidor até uma determinada data ou marco de projeto;

O cliente não tem certeza do sucesso do projeto e se será lançado e pode não precisar do servidor;

O cliente não é técnico e não vê a necessidade de um servidor;

O cliente já tem seus próprios servidores;

O cliente não quer usar nossa infraestrutura de nuvem privada;

O cliente não deseja usar a infraestrutura de nuvem pública;

O cliente quer construir sua própria infraestrutura de servidores;

O cliente quer determinar todas as especificações de hardware / software do servidor, o que leva tempo;

Enfim, uma análise de cenário adequada, anterior ao desenvolvimento do projeto, minimizar esses entraves, de maneira que a equipe consiga cumprir os cronogramas.

Acima de sete ferramentas básicas de qualidade ajudam você a abordar diferentes preocupações em uma organização. Assim sendo, o uso de tais ferramentas deve ser uma prática básica na organização para aumentar a eficiência. Realizando o diagrama de Cause-e-Efeito de uma forma mais detalhada, a fim de determinar as possíveis causas de um defeito encontrado tem a vantagem de oferecer a possibilidade de identificar e analisar todos os fatores, que se relacionam com o problema estudado. Esta ferramenta é excelente para capturar a saída de brainstorming da equipe e para preencher a partir da 'imagem ampla'. Ajuda a organizar e relacionar fatores, fornecendo uma visão sequencial.

Este diagrama trata da direção do tempo, mas não da quantidade. Pode tornar-se muito complexo e pode ser difícil identificar ou demonstrar inter-relações.

As análises de avaliação do produto foram divididas da seguinte maneira, a partir da avaliação feita pelo diagrama de Causa-e-Efeito: homem, gerenciamento, qualidade.

### **1) Homem (A equipe):**

As equipes ágeis que trabalharam na plataforma são formadas por profissionais desenvolvedores com uma boa experiência em desenvolvimento em múltiplas plataformas.

Os colaboradores passaram por uma capacitação em Scrum com carga horária de 80 horas com a participação de um *agile coach*. Também passaram por avaliação de desempenho anualmente, bem como são avaliados após a conclusão de cada projeto.

As avaliações são objetivas, realizadas por seus superiores, com base sempre em dados anteriores para facilitar a recomposição dos índices avaliados.

Os parâmetros para avaliação da equipe foram: (i) Projeto; (ii) código; (iii) tempo e (iv) trabalho em equipe; conforme abaixo:

**Quadro 2.** Parâmetros para avaliação dos Colaboradores

PROJETO	CÓDIGO	TEMPO	TRABALHO EM EQUIPE
<b>Tipo de projeto, avaliado em uma escala de complexidade em relação ao projeto anterior (de 1 a 5, sendo 5 considerado e alta complexidade e 1 de baixíssima complexidade)</b>	Habilidades em desenvolver parâmetros relacionáveis e objetivos que tenham funcionalidade	Entrega do projeto, e suas partes, dentro do tempo estimado.	A capacidade do colaborador de trabalhar em equipe, principalmente dentro da abordagem ágil.

Fonte: o autor (2019)

Os resultados foram apresentados individualmente e para as equipes, de maneira que cada equipe pudesse ter uma avaliação geral dos trabalhos realizados em equipe. Para evitar choques de desmotivação, não foram feitas às equipes comparações com resultados anteriores, muito menos entre os resultados entre equipes. Dessa forma, cabe ao *agile coach* identificar as lacunas e tentar saná-las.

## 2) Gerenciamento

O gerenciamento das equipes ágil se deu através de um *Scrum Master* sendo orientado por um *agile coach* e de um gerente de TI. A eles cabiam as seguintes funções: (i) administrar o escopo do projeto e apresentá-lo às equipes; (ii) administrar a rotina de trabalho in loco ou remotamente; (iii) motivar as equipes; (iv) trabalhar em conjunto com as equipes na construção de soluções para problemas críticos; (v) fazer com que as equipes incorporassem o “espírito” ágil de trabalho.

As reuniões em equipe eram diárias e não ultrapassando os 15 minutos conforme os ritos do *Scrum*, sempre com o objetivo de avaliar os avanços do projeto, dificuldades das equipes e seus membros, a *Sprint Review* era um momento onde a equipe se reunia para se ajustar e ter alguns momentos de descontração para aliviar a pressão inerente do projeto;

Foi adotada para as equipes a possibilidade de trabalho remoto, em vários casos, problemas de trânsito e afins, como forma de compensação para as horas extrapoladas e serviços que

precisaram desenvolver fora do horário estipulado. Lembrando que o trabalho remoto não excetuava a presença no local.

Também foi estabelecido um informativo interno com comentários e editoriais sobre o projeto desenvolvido. Assim, os colaboradores poderiam escrever sobre seus projetos de maneira crítica e criativa. Também podem explanar sobre assuntos novos, dentro da temática de desenvolvimento de software, administração, gerenciamento, rotinas e tecnologias.

Dentro do processo, o gerenciamento pode ser inserido nas causas de sucesso ou insucesso. Além do mais, o gerenciamento também passou por processo avaliativo, tanto da empresa contratada, quanto dos colaboradores. Para tanto, foram enviados os seguintes parâmetros para avaliação do gerenciamento: (i) Habilidade de liderança; (ii) Conhecimento técnico; (iii) capacidade de gerenciar equipes; (iv) capacidade de lidar com situações críticas.

Os *Scrum Master* tinham a obrigação de emitir relatórios semanais sobre os trabalhos dos membros da equipe, bem como a saúde do projeto. Também deveriam tratar com o *Head* de Operação e mantê-lo informado sobre o desenvolvimento do produto e tirar dúvidas, bem como alimentar as equipes de informações relevantes e necessárias para consecução de suas funções.

**Quadro 3. Parâmetros para avaliação do Gerenciamento.**

HABILIDADE DE LIDERANÇA	CONHECIMENTO TÉCNICO	CAPACIDADE DE GERENCIAR EQUIPES	CAPACIDADE DE LIDAR COM SITUAÇÕES CRÍTICAS
Habilidade de Liderança, avaliado em uma escala de complexidade em relação à capacidade de ser seguida, de erquer a bandeira em detrimento de delegar tarefas. (de 1 a 5, sendo 5, considerado e alta habilidade e 1 de baixíssima habilidade).	Habilidades em distinguir problemas tecnicos de negociais alem da capacidade de agir como um itermmediador entre negocio e sistema. (de 1 a 5, sendo 5, considerado alto conhecimento e 1 de baixissima conhecimento).	Capacidade de gerencias as tarefas da equipe, alem de atuar como um facilitador para a conclusão da sprint. (de 1 a 5, sendo 5, considerado e alta capacidade e 1 de baixissima capacidade).	Capacidade de lidar com situações critica, tais como prazos, absouver as presões sem transparecer a equipe, entre outros. . (de 1 a 5, sendo 5, considerado e alta capacidade e 1 de baixissima capacidade).

Fonte: o autor (2019)

### 3) Qualidade

Nesse tópico, as propriedades de design estrutural e comportamental de classes, objetos e seus relacionamentos são avaliados usando o SonarQube que com um conjunto de métricas de design e análise do código. O mesmo avaliar o modelo e relaciona as propriedades de design, como encapsulamento, modularidade, acoplamento e coesão, a atributos de qualidade de alto nível, como reusabilidade, flexibilidade e complexidade, usando parâmetros informados tais como o *Checkstyle*, *Findbugs* e *PMD* que ao findo do processo informa o *Jenkins* um relatório de erros. As propriedades de design para atributos de qualidade são ponderadas de acordo com sua influência e importância.

O processo é iniciado pelo *Jenkins* que após a primeira compilação envia os parâmetros para avaliação no SonarQube que compara os resultados obtidos com os padrões preestabelecidos. Foi constatado que quanto mais o modelo ágil adotado é trabalhado de uma forma adaptativa o débito técnico pode aumentar e para isso uma ferramenta adicional foi adicionado no ambiente de desenvolvimento como o Sonarlint.

A análise de classificação de qualidade do software se baseou em métricas de atualização do produto e a taxa de dinamismo em relação ao que se tinha antes, em outras palavras, como foi implantado *a posteriori* o processo de integração contínua, os débitos técnicos foram contatados após a primeira execução dos processos não contabilizando débitos técnicos já existentes.

Tais procedimentos ajudam a direcionar os esforços de melhoria da qualidade para os módulos em operação, o que determina certa economicidade, haja vista que os recursos de teste e aprimoramento de qualidade de *software* não esperam um evento crítico acontecer, pelo contrário, antecipam-se a ele.

Dessa forma, são feitas comparações para avaliar o desempenho relativo dos modelos propostos, colocando-os em hierarquia de desempenho e custo. Deve-se considerar ainda que o desempenho do *software* é classificado também a partir de suas linhas de código.

Essa sistemática foi adotada reiteradamente pela equipe ágil até que ser apresentada a versão final do software ao avaliador e ao cliente.

**Quadro 4.** Demonstrativo de IC em ambiente de desenvolvimento.

ANTES DA INTEGRAÇÃO CONTÍNUA	APÓS INTEGRAÇÃO CONTÍNUA
Integração da equipe se dava de maneira compartimentada	A integração da equipe ficou mais integrada e todos obtiveram mais visão sobre “o onde está” e “para onde vai”
Comunicação precária e com intermediação de chefias	Comunicação mais transparente sem intermediação da chefia, tendo em vista que com o Jenkins a equipe pode ter visão do projeto, além de conseguir mensurar a qualidade apenas em uma olhada no sonar.

Fluxos de trabalhos com óbices intermitentes	O Fluxo de trabalho fica claro, sem óbices, permitindo ao <i>Scrum Máster</i> junto com o <i>Product Owner</i> decidir de forma assertiva quais as funcionalidades estariam em produção.
Sem avaliação de processos e procedimentos internos	Com a visão do processo como um todo. A equipe começou a aperfeiçoar o processo de entrega de valor.
Sem avaliação contínua do objeto a ser entregue	Com uma visão de 360 quanto à qualidade do produto a ser entregue e exata visão da entrega de valor ao usuário
Inexistência de uma cultura de trabalho em equipe	Com uma visão integrada através da integração continua a equipe, o <i>Scrum Master</i> e o <i>Product Owner</i> , conseguiram trabalhar com uma visão voltada a entrega de valor o que culminou numa equipe mais voltada a estratégia de ganha-ganha quando se tratava da entrega de valor.
Responsabilização individual, o que gerava excesso de cobrança sobre um determinado <b>indivíduo</b> .	Responsabilidade para a equipe como um todo, o resultado compartilhado tornou a equipe mais madura quanto ao que estava sendo entregue como valor.
Avaliação do código não participativa	Avaliação do código passou para a equipe, e forneceu insumos de melhorias e mais atenção de não inserir debito técnico no produto valorado.
Prazos não assertivos	Os prazos de entrega em algum ambiente se tornaram factível, ainda ocorreram atrasos, mas poucos e com motivos mais justificáveis.
Segurança do código não compartilhada	Com o controle de versão alinhado com a integração continua, permitiu um rastreio de quem e quando ocorreu a publicação, dando uma segurança maior ao produto entregue em ambiente de produção.
Trabalhos compartimentados, desde a	Trabalho compartilhado, dando

apresentação do projeto, ou segmentados, e não <b>interativos</b> .	maleabilidade ao <i>Scrum Master</i> , definir a equipe as funcionalidades a serem entregue passando de uma visão incremental e iterativa preditiva, para uma visão adaptativa incremental.
---	---

Fonte: o autor (2019)

Além das conclusões acima o objetivo definido como “Demonstrar a importância da integração contínua junto a equipes de desenvolvimento ágil” foi integralmente demonstrado conforme disposto nos tópicos 2.9 a 2.9.12 e no quadro de nº 4 (Demonstrativo de Integração Contínua em ambiente de desenvolvimento), no capítulo 4, tópico 4.1.4 e no capítulo 6, tópicos 6.1 e 6.2.

### 7.3. Quantos aos objetivos específicos

Como resultado das ferramentas propostas, descritas no marco teórico, foi possível responder de forma sucinta quais são os conjuntos de ferramentas que se podem implantar para obter agilidade na entrega de valor pelas equipes de desenvolvimento ágil.

Em relação ao conjunto de ferramentas que foram implantadas para auxiliar na entrega de valor ao usuário, implantou-se o Servidor de Integração *Jenkins*, que alinhado ao *SonarQube* e *Artifactory* e deram ganho de agilidade na entrega, rastreamento dos artefatos e, não menos importante, conseguiu-se fornecer para a gerência a saúde dos artefatos produzidos.

Em relação a ferramenta *Flyway*, a mesma está sendo utilizada apenas na integração contínua quanto a entrega de valor nos ambientes de homologação e teste integrado, esta medida foi tomada tendo em vista um critério de segurança para o ambiente de produção.

Após a implantação de integração contínua, foi possível ter uma visão de tempo de entrega de artefatos em produção, permitindo também dar ao *Product Owner* mais autonomia do que de fato iria entrar em produção além de permitir o *Head* de Operação um acompanhamento mais visual.

Foi realizada uma apresentação após as mesmas já estarem instaladas e configuradas, apresentando também o novo fluxo de iteração e entrega. Deve-se destacar que a equipe da empresa em estudo já utilizava metodologia ágil, mas, sem utilização de integração contínua o que impactava muito a entrega de valor ao usuário sendo gasto em média 2 (dois) a 3 (três) dias. Anteriormente ao processo de construção, as ações eram quase artesanais, após a implantação das ferramentas a parte de inspeção do código ficou no sonar, onde é possível avaliar a qualidade do código produzido quanto à aceitação foi boa quanto à implantação da ferramenta, mas, mesmo assim levou duas *sprints* para rodar o processo sem precisar de intervenção manual.

Após a implantação das ferramentas, conseguiu-se minimizar a geração de mais débitos técnicos no código produzido pelas equipes, tanto na correção de funcionalidades quanto na criação de novas funcionalidades, dessa forma, após a implantação das ferramentas do fluxo ficou mais dinâmico.

Foram mitigados alguns riscos de disponibilidade que foram resolvidos utilizando *cloud computer*, para os de segurança foi implantado um *WebFirewall*, e com a integração contínua resolveram-se os erros humanos na hora de gerar e entregar em produção as aplicações. Dessa forma, a instituição foi madura o suficiente para alinhar as necessidades do projeto com as necessidades das equipes.

Foi possível também, através da solução adotada, fornecer à equipe, à área gestora e a todos os colaboradores a visão da saúde do projeto. Também foi possível passar às equipes a visão de métrica do que está sendo produzido utilizando o *SonarQube*. Obteve-se, assim ganho significativo no tempo de publicação nos diversos ambientes, além de obter, através do *Artifactory*, o rastreamento das versões em seus diversos ambientes.

Identificou-se também no decorrer da implantação que gargalos ocorriam de forma constante na entrega de valor, ora na construção, ora na entrega do agregado de valor ao usuário, todavia, a percepção de ganho se dissipou muito rápido fornecendo a impressão de ubiquidade dos processos automatizados na entrega de valor ao usuário.

Para corroborar a eficácia e eficiência das ferramentas adotadas foi metrificado os resultados através da folha de frequência que foi constatado que houve débitos técnicos evitado aumentando assim a qualidade do produto quanto sua entrega de valor.

Além das conclusões acima o para os objetivos específicos propostos, (i) aprofundar e entender o nível de aceitação dos métodos ágeis pela organização; e (ii) Indicar um conjunto básico de ferramentas que possam auxiliar a entrega de valor, foram plenamente alcançados através do estudo de caso que permitiu compreender a aceitação dos métodos ágeis a partir do agilômetro (cap.3, tópico 3.4.1) e dos questionários aplicados : (i) Parâmetros de avaliação de colaboradores, quadro 2; (ii) Parâmetros para avaliação de gerenciamento, quadro 3; e (iii) Demonstrativo de Integração Contínua em ambiente de desenvolvimento, quadro 4.

#### **7.4. Sugestões**

Como ponto de melhorias, pode-se citar um caso em específico encontrado, que é o ambiente de desenvolvimento em nuvem, dessa forma o ideal é que o ambiente de desenvolvimento fique no ambiente local do profissional de criação de software.

Por fim, outra possibilidade aventada para o presente estudo de caso, e já tem se tornado realidade, é uso de Inteligência Artificial para o desenvolvimento de software, principalmente aqueles que possuem algum padrão, como em uma linha de montagem, e mesmo pequenas



atividades que não dependem de uma montagem de código complexa como, por exemplo, gerenciamento de rotinas de trabalho, atendimento ao usuário, APPs também podem entrar nesse cesto.

Deve-se considerar que a programação em grande escala é feita por grupos muito grandes de pessoas e os programas têm várias centenas de milhares ou milhões de linhas, com vidas úteis muito longas. Mesmo APPs e outros softwares de rotinas administrativas exigem equipes que precisam estar sempre conectadas e relacionadas entre si, em um processo complexo como o descrito em nosso estudo de caso.

A IBM tem um produto desses, é chamado de WHATSON. De acordo com a empresa:

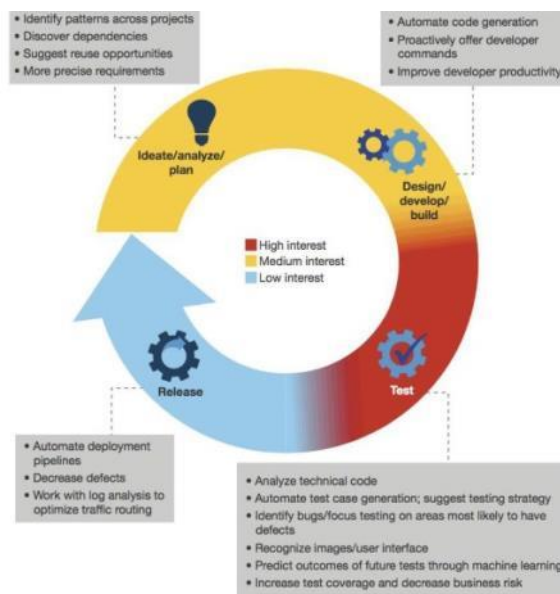
Levando a promessa da medicina de precisão para mais pacientes com câncer, Watson pode interpretar os resultados dos testes genéticos com mais rapidez e com maior precisão do que os métodos manuais. Nossa parceria com a Quest Diagnostics significa que todos os fornecedores podem se beneficiar potencialmente, independente do acesso ao sequenciamento interno (IBM, 2019, p. 1).

A IA na elaboração de software pode, por exemplo, sanar alguns problemas de comunicação que criam lacunas, toma tempo e dificultam a identificação de erros, bugs. Deve-se compreender que a elaboração de um código não envolve subjetividade, mas uma relação entre o design e a operacionalidade do produto a ser entregue. Por exemplo, preciso do cadastro de um cliente que atenda determinadas condições, forma de pagamento, tipo de cartão de crédito e local de entrega. Assim, não há aqui uma “discrecionalidade” do desenvolvedor para em estabelecer essa ou aquela condição, pelo contrário, deverá escrever um código que atenda especificamente, e exatamente, o que se pede para o produto.

Assim, a IA na engenharia de Software não pretende dispensar a figura do desenvolvedor apenas auxilia-lo na elaboração de códigos de maneira muito rápida, a partir de determinados parâmetros e condições pré-estabelecidas para o produto a ser entregue.

Essas disposições da IA são possíveis, e já se avizinham no horizonte da engenharia de software. Conforme explicado abaixo por (Dsouza, 2018):

A maior parte do interesse em aplicar a IA ao desenvolvimento de software já é visto em ferramentas de testes automatizados e detecção de bugs. Em seguida, estão os preceitos de design de software, as estratégias de tomada de decisão e, finalmente, a automatização de pipelines de implantação de software De acordo com (Dsouza, 2018).



**Figura 36.** As 5 principais esferas do desenvolvimento de software.  
Fonte: Dsouza (2018)

Basicamente, a engenharia de software tende a ser mais precisa do que o trabalho de equipes de desenvolvedores, pois percebe mais rápido os padrões de desvios do que a mente humana. Assim, bugs e falhas diversas podem simplesmente não existir, diferente do código produzido por humanos.

Dentro dessa perspectiva de IA é possível sua aplicação em ambientes ágeis, considerando que o nível de rotinas, processos e procedimentos internos, bem como a relação de tomada de decisões.

É uma possibilidade real e factível, que tende a ser incorporada em diversos segmentos comerciais que demandem uma compreensão precisa do cenário mercadológico e uma avaliação mais detalhada do perfil de seus clientes.

A informação está disponível, mas é necessário saber trabalhar essa informação. É um processo metodológico que envolve etapas, análises de diversas variáveis, depuração e, por fim, um esquema do que será feito com essa informação. Assim, o ambiente ágil, incorporando a IA tende a potencializar e a eficácia no uso das diversas informações e dados compilados para desempenho das atividades em que estejam inseridos os procedimentos ágeis.

## REFERÊNCIAS BIBLIOGRÁFICAS

- Ahmed, R. (19 de fevereiro de 2019). *What is Git? - Explorer a Distributed Version Control Tool*. Acesso em 09 de maio de 2019, disponível em Edureka: <https://www.edureka.co/blog/what-is-git>
- Atlassian. (2019). *DevOps: Breaking the Development-Operations barrier*. Acesso em 01 de fevereiro de 2019, disponível em atlassian.com: <https://www.atlassian.com/devops>
- AWS. (12 de abril de 2016). *O que significa integração contínua?* Acesso em 20 de novembro de 2018, disponível em aws.amazon.com: <https://aws.amazon.com/pt/devops/continuous-integration>
- Axelos. (2015). *Prince2 Agile*. Acesso em 07 de maio de 2019, disponível em Axelos Global Best Practice: [https://publications.axelos.com/Prince2Agile2015/content.aspx?page=cros\\_148&showNav=true&expandNav=true](https://publications.axelos.com/Prince2Agile2015/content.aspx?page=cros_148&showNav=true&expandNav=true)
- Bidu. (2019). *Cotação online de seguro e crédito*. Acesso em 18 de maio de 2019, disponível em bidu.com.br: [https://www.bidu.com.br/?utm\\_source=google&utm\\_medium=cpc&utm\\_campaign=265401184&utm\\_term=bidu&utm\\_adgroup=21708668344&utm\\_adposition=1t2&utm\\_device=c&gclid=Cj0KCQJw2v7mBRC1ARIsAAiw34-duJrFl4G6Pwlvz4U9pwbnuVmkZxpnA6Fo9xC9\\_DQvWjuYy958h2gaApeaEALw\\_wcB](https://www.bidu.com.br/?utm_source=google&utm_medium=cpc&utm_campaign=265401184&utm_term=bidu&utm_adgroup=21708668344&utm_adposition=1t2&utm_device=c&gclid=Cj0KCQJw2v7mBRC1ARIsAAiw34-duJrFl4G6Pwlvz4U9pwbnuVmkZxpnA6Fo9xC9_DQvWjuYy958h2gaApeaEALw_wcB)
- Blog da Qualidade. Gestão de processos. *Definições de qualidade e qualidade de produto*. Publicado em 18 de outubro de 2012. Acesso em junho de 2019. Disponível em: <<https://blogdaqualidade.com.br/definicoes-de-qualidade-e-qualidade-de-produto/>>.
- Boaglio, F. (2016). *Jenkins: automatize tudo sem complicações*. São Paulo, Brasil: Casa do Código.
- BRASIL. (1966). *Decreto-Lei nº296 de fevereiro de 1967. Altera dispositivos do Decreto-Lei nº 73, de 21 de novembro de 1966*.
- Dsouza, M. (02 de setembro de 2018). *5 ways artificial intelligence is upgrading software engineering*. Acesso em 10 de março de 2019, disponível em packtpub: <https://hub.packtpub.com/5-ways-artificial-intelligence-is-upgrading-software-engineering>

- Elgazzar, S., A Saleh, A., & El-Bakry, H. (2017). Overview of Using Private Cloud Model with GIS. *International Journal of Electronics and Information Engineering*.(7), pp. 68-78.
- Faria, A. (19 de março de 2016). *Equipes Cross-Funcionais e Multifuncionais*. Acesso em abril de 2019, disponível em blog.andrefaria.com: <https://blog.andrefaria.com/equipes-funcionais-cross-funcionais-e-multi-funcionais>
- Flywaydb. (2018). *Flyway is a registered trademark of Boxfuse GmbH*. Acesso em 18 de dezembro de 2018, disponível em Flywaydb.org: [www.flywaydb.org](http://www.flywaydb.org)
- Fowler, M. (1 de maio de 2006). *Continuous Integration*. Acesso em 20 de novembro de 2018, disponível em [martinfowler.com](http://martinfowler.com): <https://martinfowler.com/articles/continuousIntegration.html>
- GAEA. (dezembro de 2018). *O que é DevOps e Como Iniciar sua Jornada DevOps*. Acesso em dezembro de 2018, disponível em GAEA Consulting: <https://gaea.com.br/o-que-e-devops-conceito>
- Gerhardt, T. E., & Silveira, D. T. (2009). *Planejamento e Gestão para o Desenvolvimento Rural da SEAD/UFRGS*. Porto Alegre: Editora da UFRGS.
- Gitlab. (2019). *A full DevOps Too*. Acesso em 01 de abril de 2019, disponível em gitlab.com: <https://about.gitlab.com>
- GNU. (abril de 2019). *O que é o software livre?* Acesso em 07 de maio de 2019, disponível em O Sistema Operacional GNU: <https://www.gnu.org/philosophy/free-sw.pt-br.html>
- Gomes, A. F. (2014). *Agile: Desenvolvimento de software com entregas frequentes e foco no valor do negócio*. Casa Código.
- IBM. (2019). *Resolvendo os desafios da saúde*. Acesso em 08 de 05 de 2019, disponível em IBM Watson: <https://www.ibm.com/watson/br-pt/health>
- ISHIKAWA, K. (1993). *Controle de qualidade total: à maneira japonesa*. Rio de Janeiro: Campos.
- jenkins.io. (2019). *Installing Jenkins*. Acesso em 11 de maio de 2019, disponível em Jenkins User Documentation Home: <https://jenkins.io/doc/book/installing>
- Jfrog. (2019). *Artifactory*. Acesso em 01 de abril de 2019, disponível em <https://jfrog.com/artifactory>

- Kakau. (2019). *Kakau - Seu Futuro Seguro*. Acesso em 18 de maio de 2019, disponível em <https://www.kakau.co>
- Karasek, N. (06 de maio de 2018). *How to do a Ishikawa Diagram in Software Development*. Acesso em 08 de maio de 2019, disponível em [nathaliekarasek.com: http://www.nathaliekarasek.com/ishikawa-diagram-software-development](http://www.nathaliekarasek.com/ishikawa-diagram-software-development)
- Kim, W. C. (2005). *A estratégia do oceano azul: como criar novos mercados e tornar a concorrência irrelevante*. Rio de Janeiro, Brasil: Elsevier.
- Learning Solutions. (16 de março de 2018). *A Designer Addresses Criticism of Design Thinking*. Acesso em 27 de fevereiro de 2019, disponível em Learning Solutions: <https://www.learningsolutionsmag.com/articles/a-designer-addresses-criticism-of-design-thinking>
- Manifesto Ágil. (fevereiro de 2001). *Manifesto para o desenvolvimento ágil de software*. Acesso em janeiro de 2019, disponível em Manifesto Ágil: <https://www.manifestoagil.com.br>
- Marshall Jr, I., Cierco, A. A., Rocha, A. V., Mota, E. B., & Leusin, S. (2008). *Gestão da qualidade* (9º ed.). Rio de Janeiro: Editora FGV.
- Menkovic, N. (janeiro de 2016). *Melhorando a qualidade do código com SonarQube*. Acesso em 10 de abril de 2019, disponível em [infobip.com: https://www.infobip.com/pt/desenvolvedor/melhorando-a-qualidade-do-codigo-com-sonarqube](https://www.infobip.com/pt/desenvolvedor/melhorando-a-qualidade-do-codigo-com-sonarqube)
- Microsoft. (03 de novembro de 2018). *O que é computação em nuvem?* Acesso em 07 de maio de 2019, disponível em Microsoft Azure: <https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing>
- Microsoft. (03 de outubro de 2018). *O que é o Azure?* Acesso em 07 de maio de 2019, disponível em Microsoft Azure: <https://azure.microsoft.com/pt-br/overview/what-is-azure/>
- Moreira, L. N. (2015). *Avaliação de Qualidade de Software Baseada em Métricas Estáticas - Um Estudo de Caso no Tribunal de Contas da União*. Brasília: Universidade de Brasília.
- Oliveira, W. (14 de 11 de 2014). *O que significa BPM – Benefícios, aplicações e estudos*. Acesso em 18 de 05 de 2019, disponível em [venki.com.br: https://www.venki.com.br/blog/o-que-significa-bpm/](https://www.venki.com.br/blog/o-que-significa-bpm/)

- Paim, R., Cardoso, V., Caulliraux, H., & Clemente, R. (2009). *Gestão de processos*. Porto Alegre: Bookman.
- PERCÍLIA, E. (2018). *Código Aberto*. Acesso em 07 de maio de 2019, disponível em Brasil Escola: <https://brasilecola.uol.com.br/informatica/codigo-aberto.htm>
- Prado, J. (02 de 02 de 2017). *O que é Insurtech?* Acesso em 01 de fevereiro de 2019, disponível em Conexao Fintech: <https://conexaofintech.com.br/guia/o-que-e-insurtech>
- Schwarzer, M. M. (junho de 2014). *Análise e Sugestões de Melhorias nos Processos Internos de uma Empresa de Software*. (Univates, Editor) Acesso em 28 de janeiro de 2019, disponível em <https://www.univates.br/bdu/bitstream/10737/734/1/2014MatiasMacielSchwarzer.pdf>
- sindsegs. (06 de setembro de 2018). *Mercado mundial de seguros chega a US\$ 5 trilhões em 2017*. Acesso em 01 de abril de 2019, disponível em Sindseg SP: <http://sindsegs.org.br/site/noticia-texto.aspx?id=30223>
- Software Livre Brasil. (27 de maio de 2009). *Página inicial da Comunidade Open Source*. Fonte: Software Livre Brasil: <http://softwarelivre.org/open-source-codigo-aberto>
- Sommerville, I. (2011). *Engenharia de Software*. São Paulo: Pearson.
- sonarque. (2019). *Continuous Inspection / SonarQube*. Acesso em 11 de maio de 2019, disponível em sonarque.org: <https://www.sonarque.org>
- Swiss Re Institute. (março de 2018). *sigma 3/2018: World insurance in 2017: solid, but mature life markets weigh on growth*. Acesso em 07 de abril de 2019, disponível em Swiss Re Institute: <https://www.swissre.com/institute/research/sigma-research/sigma-2018-03.html>
- Tutorials Point. (2019). *Maven Tutorial*. Acesso em 11 de maio de 2019, disponível em Tutorials Point: <https://www.tutorialspoint.com/maven/index.htm>

## ANEXOS

### MODELO DE QUESTIONÁRIO APLICADO À EQUIPE

Também foi aplicado questionário à equipe de desenvolvedores com perguntas objetivas e diretas. O questionário foi elaborado na plataforma docs do Google, com e-mail cadastrado como [ambientedesenvagil@gmail.com](mailto:ambientedesenvagil@gmail.com).

Os formulários foram enviados a um grupo cadastrado de 20 e-mails, todos desenvolvedores, que trabalharam ou trabalham na empresa objeto do presente estudo de caso, ou, que de alguma forma colaboram durante o período de análise do presente estudo de caso.

Os questionamentos, numerados de 1 a 10, de forma objetiva e direta. Usou-se o pronome de tratamento “você” para que as formalidades não obstruíssem a pesquisa, deixando o respondente mais à vontade:

1. Você já conhecia o método ágil?
  - a) Sim, conheço plenamente.
  - b) Sim, tenho algum conhecimento.
  - c) Não conheço, mas sei que existe.
  - d) Não conheço
  
2. Qual sua percepção sobre o método ágil
  - a) É interessante
  - b) É funcional
  - c) É impressionante
  - d) É impressionante e deve ser adotado pela empresa
  
3. Você já conhecia o Jenkins
  - a) Sim, conheço plenamente.
  - b) Sim, tenho algum conhecimento.
  - c) Não conheço, mas sei que existe.
  - d) Não conheço
  
4. Você já conhecia o SonarQube?
  - a) Sim, conheço plenamente.
  - b) Sim, tenho algum conhecimento.
  - c) Não conheço, mas sei que existe.
  - d) Não conheço

5. Você já conhecia o SCRUM
- a) Sim, perfeitamente
  - b) Sim, lembro vagamente de algo.
  - c) Não, mas sei que existe.
  - d) Não tenho ideia do que seja
6. Você já conhecia o Maven
- a) Sim, perfeitamente
  - b) Sim, lembro vagamente de algo.
  - c) Não, mas sei que existe.
  - d) Não tenho ideia do que seja
7. Os métodos e procedimentos internos se tornaram mais ágeis e menos complexos?
- a) Sim
  - b) Não
  - c) Não sei informar
8. Você prefere trabalhar em ambientes hierarquizados ou integrados (nivelados horizontalmente)?
- a) Prefiro trabalhar em ambientes hierarquizados
  - b) Achei interessante e mais dinâmico trabalhar em ambientes integrados
  - c) A política motivacional existe e é amplamente divulgada e aplicada de forma eficaz no cotidiano corporativo contribuindo, assim, para um bom Clima Organizacional.
9. Comparando o “Antes” e o “Depois”, qual sua percepção sobre a nova metodologia de trabalho?
- a) Não mudou nada.
  - b) Teve poucas melhoras.
  - c) Teve melhoras.
  - d) Melhorou significativamente.
10. Em sua opinião, a integração continua melhorou a Gestão na entrega de valor?
- a) Sim
  - b) Não
  - c) Não sei opinar



## 11. Sugestões ou comentários

A resposta dos questionamentos segue abaixo:

Cari mbo de data /hor a	Endereço de e-mail	1) Você já conhe cia o métod o ágil?	2) Qual sua percep ção sobre o métod o ágil?	3) Você já conh ecia o Jenki ns?	4) Você já conh ecia o Sonar Qube ?	5) Você já conh ecia o Scru m?	6) Você já conh ecia o Mave n?	7) Os métod os e proce dimen tos intern os se tornar am mais ágeis e meno s compl exos?	8) Você prefere trabalh ar em ambien tes hierarq uizados ou integra dos (nivela dos horizon talment e)?	9) Compa rando o “antes” e o “depois” , qual sua percep ção sobre a nova metodo logia de trabalh o?	10) Em sua opin ião, a inte graç ão cont ínua mel horo u a Ges tão na entr ega de valo r?	Suge stões ou come ntário s
25/0 5/20 19 13:2 6:58	cleidsondias @hotmail.c om	Sim, conhe ço plena mente ;	É impres sionan te e deve ser adotad o pela empre sa	Sim, conh eço plena ment e	Sim, conh eço plena ment e	Sim, conh eço plena ment e	) SIM, perfei tame nte	SIM	A política motiva cional existe e é ampla mente divulga da e aplicad a de forma eficaz no cotidia no corpora tivo contrib uindo, assim, para um bom Clima Organi zacion al.	(d) Melhor ou signific ativam ente	SIM	
25/0	gpedro.lui	Sim,	É	Sim,	Sim,	Sim,	)	NÃO	Achei	(d)	NÃ	Gost

5/20 19 13:3 8:17	@gmail.co m	conhe ço plena mente ;	impres sionan te e deve ser adotad o pela empre sa	conh eço plena ment e	conh eço plena ment e	conh eço plena ment e	SIM, perfei tame nte		interes sante e mais dinâmi co trabalh ar em ambien tes integra dos	Melhor ou signific ativam ente	O	o mais do méto do ágil Lean Kanb an. Além disso, sem teste codifi cado não existe integr ação conti nua. Sem teste s só existe uma melh ora na infrae strutu ra. Quali dade não pode ser opcio nal.
25/0 5/20 19 13:3 9:20	fredentino@ gmail.com	Sim, conhe ço plena mente ;	É impres sionan te e deve ser adotad o pela empre sa	Sim, conh eço plena ment e	Não conh eço	Sim, conh eço plena ment e	Não tenho ideia do que seja	SIM	Achei interes sante e mais dinâmi co trabalh ar em ambien tes integra dos	(d) Melhor ou signific ativam ente	SIM	
25/0 5/20 19 13:5 2:10	lucas@min dello.com.br	Sim, tenho algum conhe cimen to;	É Impres sionan te;	Sim, tenho algu m conh ecime nto	Sim, tenho algu m conh ecime nto	Sim, tenho algu m conh ecime nto	Não, mas sei que existe m	SIM	Achei interes sante e mais dinâmi co trabalh ar em	(d) Melhor ou signific ativam ente	SIM	

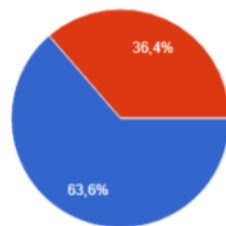
									ambientes integrados			
25/05/2019 14:54:19	thalesvinkler@gmail.com	Sim, conheço plenamente;	É funcional;	Sim, tenho algum conhecimento	Não conheço, mas sei que existe	Sim, conheço plenamente	SIM, lembro vagamente de algo	SIM	Achei interessante e mais dinâmico trabalhar em ambientes integrados	(c) Teve melhorias	SIM	
25/05/2019 15:12:05	windjsantos@gmail.com	Sim, tenho algum conhecimento;	É interessante;	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	) SIM, perfeitamente	SIM	Achei interessante e mais dinâmico trabalhar em ambientes integrados	(c) Teve melhorias	SIM	
25/05/2019 16:38:25	feemasson@gmail.com	Sim, conheço plenamente;	É interessante;	Sim, tenho algum conhecimento	Não conheço, mas sei que existe	Sim, conheço plenamente	Não, mas sei que existe	NÃO	Achei interessante e mais dinâmico trabalhar em ambientes integrados	(c) Teve melhorias	SIM	
25/05/2019 21:25:44	michaelpimentel@gmail.com	Sim, tenho algum conhecimento;	É funcional;	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	SIM, lembro vagamente de algo	NÃO	Achei interessante e mais dinâmico trabalhar em ambientes integrados	(d) Melhor ou significativamente	SIM	
26/05/2019 10:22:47	renanlq@gmail.com	Sim, conheço plenamente;	É funcional;	Sim, conheço plenamente	Sim, tenho algum conhecimento	Sim, conheço plenamente	SIM, lembro vagamente de	SIM	Achei interessante e mais dinâmico	(d) Melhor ou significativamente	SIM	

					nto		algo		trabalhar em ambientes integrados			
26/05/2019 10:44:43	ronergama@gmail.com	Sim, tenho algum conhecimento;	É impressionante e deve ser adotado pela empresa	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	Sim, tenho algum conhecimento	) SIM, perfeitamente	SIM	Achei interessante e mais dinâmico trabalhar em ambientes integrados	(d) Melhor ou significativamente	SIM	QNE 28 CAS A 24
27/05/2019 15:50:14	edussj3@gmail.com	Sim, conheço plenamente;	É funcional;	Sim, tenho algum conhecimento	Não conheço	Sim, tenho algum conhecimento	SIM, lembro vagamente de algo	SIM	A política motivacional existe e é amplamente divulgada e aplicada de forma eficaz no cotidiano corporativo contribuindo, assim, para um bom Clima Organizacional.	(d) Melhor ou significativamente	SIM	89 Brunswick Park Road

Abaixo, os gráficos por resposta.

### 1) Você já conhecia o método ágil ?

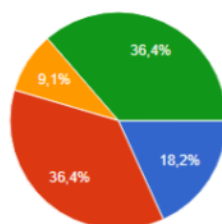
11 respostas



- Sim, conheço plenamente;
- Sim, tenho algum conhecimento;
- Não conheço, mas sei que existe
- Não conheço

### 2) Qual sua percepção sobre o método ágil?

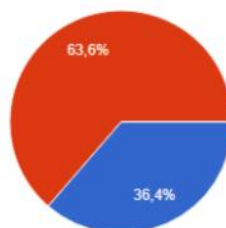
11 respostas



- É interessante;
- É funcional;
- É impressionante;
- É impressionante e deve ser adotado pela empresa

### 3) Você já conhecia o Jenkins?

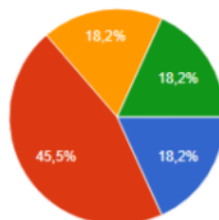
11 respostas



- Sim, conheço plenamente
- Sim, tenho algum conhecimento
- Não conheço, mas sei que existe
- Não conheço

### 4) Você já conhecia o SonarQube?

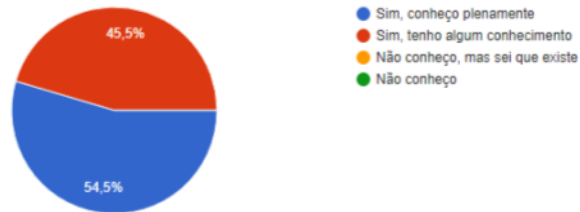
11 respostas



- Sim, conheço plenamente
- Sim, tenho algum conhecimento
- Não conheço, mas sei que existe
- Não conheço

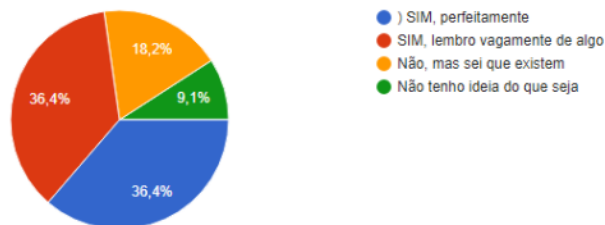
### 5) Você já conhecia o Scrum?

11 respostas



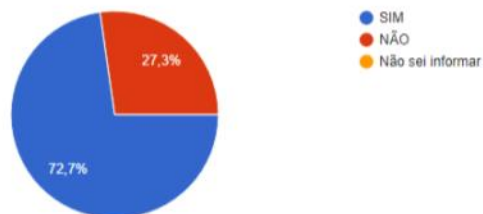
### 6) Você já conhecia o Maven?

11 respostas



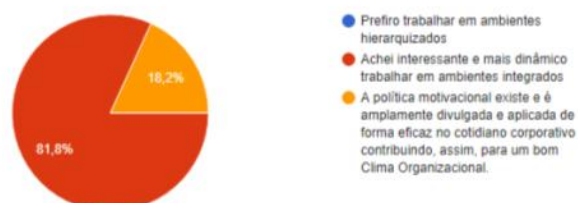
### 7) Os métodos e procedimentos internos se tornaram mais ágeis e menos complexos?

11 respostas



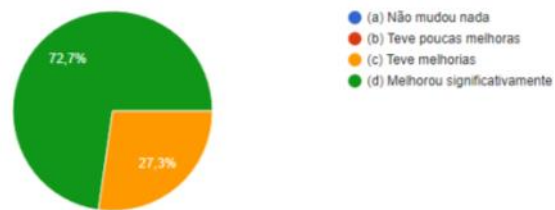
### 8) Você prefere trabalhar em ambientes hierarquizados ou integrados (nivelados horizontalmente)?

11 respostas



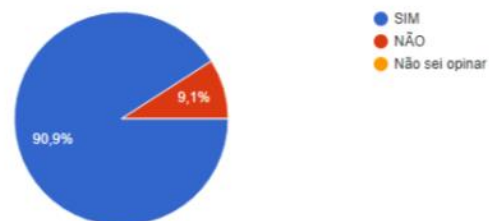
9) Comparando o "antes" e o "depois", qual sua percepção sobre a nova metodologia de trabalho?

11 respostas



10) Em sua opinião, a integração contínua melhorou a Gestão na entrega de valor?

11 respostas



### **RESULTADO DA FOLHA DE FREQUÊNCIA**

- a) Quantas vezes houve entrega de valor? 30
- b) Quantas vezes houve impedimento técnico? 7
- c) Quantas vezes houve problemas de qualidade? 9



**TERMO DE CONCORDÂNCIA E RESPONSABILIDADE COM AS NORMAS E CONDIÇÕES PARA UTILIZAÇÃO DE CASES DA WIZ EM TESES DE GRADUAÇÃO, PÓS GRADUAÇÃO, MESTRADO E DOUTORADO.**



**TERMO DE CONCORDÂNCIA E RESPONSABILIDADE COM AS NORMAS E CONDIÇÕES PARA UTILIZAÇÃO/CITAÇÃO DE CASES DA WIZ EM TESES DE GRADUAÇÃO, PÓS GRADUAÇÃO, MESTRADO E DOUTORADO**

Eu, CLAYSON DINS DO NASCIMENTO, inscrito(a) no CPF sob o nº 835280181-99, residente e domiciliado(a) em RUA 412 CJA FLOZ ILR APT 1001 - Sampaio, colaborador(a)/ex-colaborador(a) da Wiz Soluções e Corretagem de Seguros S.A. ("Wiz" ou "Companhia"), declaro, como condição para utilização de cases práticos da Wiz e citação do nome da Companhia em trabalhos acadêmicos, o que se segue:

I. Assumo, neste ato, total responsabilidade por tudo que escrever sobre a Wiz e seus negócios, inclusive no que tange a eventuais distorções relacionadas a fatos ou acontecimentos internos à Companhia;

II. Me comprometo a não realizar qualquer tipo de declaração ofensiva ou que possa causar qualquer tipo de dano à imagem/reputação da Wiz, e a tratar apenas sobre assuntos pertinentes ao tema do meu trabalho acadêmico;

III. Declaro que preservarei a confidencialidade quanto a informações e conteúdos ainda não divulgados ao público em geral;

IV. Estou ciente de que os conteúdos compartilhados por mim no meu trabalho acadêmico:

i) Não poderão infringir o Código de Conduta Ética da Companhia;

ii) Passarão por uma análise prévia da Diretoria da Wiz;

V. Estou ciente de que a violação do presente Termo e o uso inadequado do nome da Wiz será passível de avaliação pela Diretoria de Gente & Gestão e pela Diretoria de Compliance da Companhia, e serão adotadas as medidas judiciais cabíveis.

Isto posto, assino o presente Termo em 2 (duas) vias, na presença das testemunhas abaixo relacionadas.

Clayson Dins do Nascimento

Testemunhas:

Nome: Damien Valente  
CPF: 601.935.591-39

Nome: JOSE ADELINO DA SILVA  
CPF: 046.354.201-49