

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA  
UNIDADE DE PÓS-GRADUAÇÃO, EXTENSÃO E PESQUISA  
MESTRADO PROFISSIONAL EM GESTÃO E TECNOLOGIA  
EM SISTEMAS PRODUTIVOS

RAQUEL BORTOLUCI

O USO DE TESTES NO DESENVOLVIMENTO DE SOFTWARE POR MÉTODOS  
ÁGEIS: UM ESTUDO A PARTIR DA VISÃO DOS PARTICIPANTES DO PROCESSO  
PRODUTIVO NO BRASIL

São Paulo

Abril/2016

RAQUEL BORTOLUCI

O USO DE TESTES NO DESENVOLVIMENTO DE SOFTWARE POR MÉTODOS  
ÁGEIS: UM ESTUDO A PARTIR DA VISÃO DOS PARTICIPANTES DO PROCESSO  
PRODUTIVO NO BRASIL

Dissertação apresentada como exigência parcial para a obtenção do título de Mestre em Gestão e Tecnologia em Sistemas Produtivos do Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos, sob a orientação do Prof. Dr. Marcelo Duduchi.

São Paulo

Abril/2016

Bortoluci, Raquel

B739u O uso de testes no desenvolvimento de software por métodos ágeis: um estudo a partir da visão dos participantes do processo produtivo no Brasil. / Raquel Bortoluci. – São Paulo : CEETEPS, 2016.

147 f. : il.

Orientador: Prof. Dr. Marcelo Duduchi

Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos) – Centro Estadual de Educação Tecnológica Paula Souza, 2016.

1. Processo produtivo de software . 2. Métodos ágeis. 3. Teste de software. I. Duduchi, Marcelo. II. Centro Estadual de Educação Tecnológica Paula Souza. III. Título.

RAQUEL BORTOLUCI

O USO DE TESTES NO DESENVOLVIMENTO DE SOFTWARE POR MÉTODOS  
ÁGEIS: UM ESTUDO A PARTIR DA VISÃO DOS PARTICIPANTES DO PROCESSO  
PRODUTIVO NO BRASIL

---

Prof. Dr. Marcelo Duduchi

---

Profa. Dra. Marília Macorin de Azevedo

---

Prof. Dr. Alfredo Goldman vel Lejbman

São Paulo, 05 de abril de 2016

À minha família que sempre me apoiou.

## AGRADECIMENTOS

Aos meus pais, Otávio e Vandeci, que são exemplos de vida e estiveram sempre ao meu lado. Obrigada pai por insistir para que eu agisse de forma correta, ética e resiliente em todos os momentos. Obrigada mãe por trazer conforto e carinho ao meu coração nos momentos em que mais precisei ser forte.

Ao meu orientador, Marcelo Duduchi, que mostrou caminhos para a pesquisa e conseguiu captar o verdadeiro objetivo do estudo, fazendo com que eu pensasse de forma mais ampla e estruturada, além de dedicar seu tempo e de se empenhar para a elaboração deste trabalho.

Aos membros da banca, Profa. Dra. Marília Macorin de Azevedo e Prof. Dr. Alfredo Goldman vel Lejbman, pela disponibilidade e conselhos dados em prol da melhoria deste trabalho.

Ao Prof. Dr. Carlos Vital Giordano pela disposição em ajudar com os conceitos estatísticos.

À minha irmã, Débora, que sempre foi minha confidente e pessoa com a qual sempre pude contar. Obrigada, inclusive, à sua família linda, Gustavo e Gabriela, que sempre estiveram presentes, apoiando-me em todos os momentos.

Obrigada ao meu irmão, Mateus, que sempre foi parceiro. Obrigada, também, à Camila, minha cunhada, que desde o primeiro contato foi minha amiga e tornou-se uma pessoa muito especial em minha vida.

Obrigada ao meu namorado, Marcio, que chegou em um momento tumultuado e soube ser compreensivo e amoroso, e esteve ao meu lado mesmo nos momentos mais difíceis.

À Nanci, minha melhor amiga desde a infância, por ser um exemplo para mim e me mostrar que a vida pode ser leve, mesmo nos momentos mais difíceis.

Ao meu gerente, Julio Madeira, por acreditar em minha capacidade e tornar possível a flexibilidade entre o trabalho e o tempo dedicado ao estudo nas aulas de mestrado.

Aos colegas e amigos do mestrado, que compartilharam experiências de artigos, dissertações, aulas e algumas cervejas e, dessa forma, tornaram-se grandes amigos durante esta jornada.

Aos colegas e amigos da IBM, que não apenas contribuíram com a pesquisa, mas também ajudaram com o trabalho quando não pude estar presente por causa das aulas.

E aos professores e funcionários do mestrado, que sempre estiveram dispostos a ajudar.

"Uma coisa boa dos tempos em que vivemos, a  
despeito de todas as suas confusões, é que as  
pessoas descobriram que é possível mudar a  
direção do vôo"  
(Rubem Alves)

## RESUMO

BORTOLUCI, R. **O uso de testes no desenvolvimento de software por métodos ágeis: um estudo a partir da visão dos participantes do processo produtivo no Brasil.** 147 f. Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2016.

Muitas organizações têm adotado o uso de métodos ágeis como solução para a atual demanda de velocidade, qualidade no desenvolvimento, gestão de mudanças e alinhamento entre o negócio e o software desenvolvido. Considerando que as atividades de teste de software são importantes no processo produtivo de software, o presente trabalho tem por objetivo verificar como essas atividades têm sido implementadas em equipes de desenvolvimento, que utilizam métodos ágeis de desenvolvimento para a produção a partir do ponto de vista dos participantes do processo produtivo. Com base na revisão bibliográfica sobre teste de software, métodos ágeis e uma pesquisa realizada com desenvolvedores, identificam-se as principais atividades utilizadas nas estratégias de testes de software e como atuam as equipes de desenvolvimento nos diversos portes de empresas, destacando tendências e práticas.

**Palavras-chave:** processo produtivo de software, métodos ágeis, teste de software.



## **ABSTRACT**

BORTOLUCI, R. **The testing usage in software development with agile methods: a study from the perspective of the production process participants in Brazil**. 147 f. Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2016.

Several organizations have been adopting agile methods as a solution for the current demand of speed, quality in software development, change management and business alignment with software development. Considering that software testing is important in software production process, this work aims at verifying how the testing activities have been implemented by development teams when using agile methods for software production from the point of view of the production process participants. Using as reference a bibliographic research for software testing, agile methods and then a survey with the developers, the research identifies the main activities of testing strategies and how the development teams act according to the size of companies, highlighting trends and practices.

**Keywords:** software production process, agile methods, software testing.

## LISTA DE QUADROS

Quadro 1:	Estratégia de teste em métodos ágeis .....	43
-----------	--	----

## LISTA DE TABELAS

Tabela 1:	Desafios do teste de software em métodos ágeis.....	38
Tabela 2:	Princípios de teste vs. práticas ágeis.....	40
Tabela 3:	Correlação de fatores planejados em uma iteração vs. características das empresas e software.....	87
Tabela 4:	Correlação de fatores planejados em uma iteração vs. características das empresas .....	88
Tabela 5:	Correlação de tempo de iteração vs. tempo de experiência.....	91
Tabela 6:	Correlação de realização de testes durante toda a iteração vs. tempo de iteração, utilização de testes automatizados e tamanho da equipe .....	93
Tabela 7:	Correlação do envolvimento com os <i>Stakeholders</i> vs. porte da empresa e natureza do software.....	95
Tabela 8:	Correlação do envolvimento com os <i>Stakeholders</i> vs. atividade da empresa e localização dos membros.....	96
Tabela 9:	Correlação de práticas de teste vs. tempo de experiência dos respondentes .....	99
Tabela 10:	Práticas de programação vs. desenvolvedor e outras funções .....	100
Tabela 11:	Testes de aceitação com os stakeholders vs. gerente de projeto e outras funções .....	101
Tabela 12:	Execução de testes exploratórios vs. testador e outras funções.....	101
Tabela 13:	Execução de testes de unidade vs. testador e outras funções .....	102
Tabela 14:	Correlação da utilização de testes manuais vs. características da empresa .....	104
Tabela 15:	Testes exploratórios sem planejamento vs. desenvolvedor .....	105
Tabela 16:	Testes exploratórios gerenciados vs. testador.....	105
Tabela 17:	Correlação de níveis do software automatizados vs. testador .....	108
Tabela 18:	Automação de nível de Interface vs. testador .....	109
Tabela 19:	Não automação de testes vs. desenvolvedor .....	109

Tabela 20:	Estratégias de teste vs. capacidade de testar durante toda a iteração.....	114
Tabela 21:	Correlação de ambiente de teste vs. características do ambiente .....	115
Tabela 22:	Organização das equipes vs. capacidade de testar durante toda a iteração.....	120

## LISTA DE FIGURAS

Figura 1:	Métodos de desenvolvimento de software .....	22
Figura 2:	Modelo em “V” .....	30
Figura 3:	Pirâmide do teste de automação .....	34
Figura 4:	Exemplo do ciclo de vida de um <i>sprint</i> em ágil.....	42
Figura 5:	Ciclo de TDD e BDD .....	46
Figura 6:	Modelo em “V” para métodos ágeis .....	47
Figura 7:	Quadrante de teste ágil.....	49
Figura 8:	Exemplo de mapa mental em teste de software .....	51
Figura 9:	Abordagem manual utilizada para teste – número de indicações .....	103
Figura 10:	Utilização das camadas do software na automação de testes – representação por número de indicações .....	107

## LISTA DE GRÁFICOS

Gráfico 1: Porte da empresa onde atuam os respondentes da pesquisa.....	78
Gráfico 2: Extensão de atuação da empresa .....	78
Gráfico 3: Classificação do software como atividade meio ou fim da empresa.....	79
Gráfico 4: Classificação do software em relação ao risco.....	80
Gráfico 5: Tamanho das equipes: número de membros .....	80
Gráfico 6: Distribuição das equipes .....	81
Gráfico 7: Função dos respondentes da pesquisa .....	81
Gráfico 8: Tempo de experiência em desenvolvimento de software com métodos ágeis.....	82
Gráfico 9: Certificação em métodos ágeis.....	83
Gráfico 10: Métodos ágeis utilizados pelos participantes da pesquisa.....	83
Gráfico 11: Métodos ágeis utilizados de forma conjunta pelos participantes da pesquisa ....	84
Gráfico 12: Fatores considerados no âmbito de testes no planejamento de uma iteração.....	86
Gráfico 13: Fatores considerados no âmbito das atividades de testes combinados.....	89
Gráfico 14: Fatores considerados no âmbito das atividades de testes combinados vs. localização dos membros.....	89
Gráfico 15: Tempo ideal para codificar e testar um software .....	90
Gráfico 16: Tempo ideal de iteração vs. função .....	92
Gráfico 17: Porcentagem de times que conseguem testar o software durante toda a iteração	92
Gráfico 18: Níveis de testes utilizados em ambientes de desenvolvimento ágil .....	94
Gráfico 19: Envolvimento dos <i>stakeholders</i> no teste de software.....	95
Gráfico 20: Capacidade envolver os <i>stakeholders</i> vs. porte da empresa.....	96
Gráfico 21: Práticas que mais corroboram para se ter um software testado em ágil.....	97
Gráfico 22: Agrupamento de práticas que mais corroboram para o software testado.....	99
Gráfico 23: Práticas que mais corroboram para o software testado em porcentagem vs.	

função .....	100
Gráfico 24: Abordagem utilizada para testes manuais .....	102
Gráfico 25: Abordagem manual vs. função .....	104
Gráfico 26: Camadas utilizadas na automação de testes .....	106
Gráfico 27: Níveis do software automatizado vs. função .....	108
Gráfico 28: Verificação do software quanto aos requisitos não funcionais .....	110
Gráfico 29: Execução de testes não funcionais vs. porte da empresa.....	110
Gráfico 30: Utilização de ferramentas na execução de testes não funcionais .....	111
Gráfico 31: Utilização de ferramentas para requisitos não funcionais vs. porte da empresa. .....	112
Gráfico 32: Abordagens utilizadas no desenvolvimento ágil.....	113
Gráfico 33: Possuem ambiente de teste gerenciável .....	115
Gráfico 34: Métricas de testes utilizadas por equipes de desenvolvimento ágil .....	116
Gráfico 35: Métricas utilizadas em teste de software em ambientes ágeis. Contagem por agrupamento. ....	118
Gráfico 36: Formação das equipes ágeis .....	119
Gráfico 37: Papel do testador em equipes de desenvolvimento ágil .....	120

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>18</b>
1.2 Questão de pesquisa e objetivo.....	20
1.3 Organização .....	20
<b>2 PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE .....</b>	<b>22</b>
2.1 Métodos de desenvolvimento Ágeis.....	24
2.2 O teste de software no contexto dos métodos de desenvolvimento ágeis .....	25
<b>3 TESTE DE SOFTWARE .....</b>	<b>28</b>
3.1 Estratégias de testes .....	29
3.2 Níveis de testes .....	29
3.3 Tipos e abordagens para teste de software .....	31
3.4 Testes manuais.....	32
3.4.1 Casos de teste – design predefinido .....	32
3.4.2 Testes exploratórios.....	33
3.5 Testes automatizados.....	34
3.6 Equipe de teste.....	36
3.7 Testes de software no contexto ágil.....	37
3.7.1 Estratégias de teste em métodos ágeis .....	41
3.7.2 Níveis de testes em métodos ágeis .....	46
3.7.3 Tipos de testes em métodos ágeis .....	48
3.7.4 Testes manuais em métodos ágeis .....	50
3.7.5 Testes automatizados em métodos ágeis .....	52
3.7.6 O papel do testador dentro da equipe ágil.....	53
3.7.7 Indicadores de testes de software.....	55
3.7.8 Relatos de experiências em testes ágeis .....	57
<b>4 PESQUISA .....</b>	<b>60</b>
4.1 Levantamento de evidências do processo de teste de software em métodos ágeis .....	62
4.1.1 Levantamento baseado na experiência do desenvolvimento ágil.....	62
4.1.2 Visão geral da implantação do método ágil.....	63



4.1.3 Implementação das atividades de teste .....	65
4.2 Aplicação do formulário e validação .....	68
4.2.1 Abordagem no questionário sobre a caracterização dos respondentes .....	68
4.2.2 Abordagem no questionário sobre as atividades de teste .....	69
4.3 Estudo Piloto .....	72
4.3.1 Abordagem no estudo piloto sobre a caracterização dos respondentes .....	73
4.3.2 Abordagem no estudo piloto sobre as atividades de teste.....	74
<b>5 RESULTADOS .....</b>	<b>77</b>
5.1 Caracterização dos respondentes .....	77
5.2 Apresentação e análise dos resultados.....	85
5.2.1 Fatores considerados no âmbito de testes no planejamento de uma iteração .....	85
5.2.2 Tempo para codificar e testar o software em uma iteração.....	90
5.2.3 Execução dos testes na iteração.....	92
5.2.4 Níveis de teste.....	93
5.2.5 Envolvimento dos Stakeholders .....	94
5.2.6 Práticas que corroboram para o software testado.....	96
5.2.7 Testes manuais.....	102
5.2.8 Testes automatizados.....	106
5.2.9 Testes não funcionais .....	109
5.2.10 Abordagens de desenvolvimento ágil .....	112
5.2.11 Gerenciamento de testes em ambientes ágeis.....	114
5.2.12 Formação de equipe ágeis.....	118
<b>6 CONCLUSÃO.....</b>	<b>122</b>
<b>REFERÊNCIAS .....</b>	<b>125</b>
<b>APÊNDICE A – PRIMEIRO QUESTIONÁRIO .....</b>	<b>131</b>
<b>APÊNDICE B – SEGUNDO QUESTIONÁRIO .....</b>	<b>138</b>

## 1 INTRODUÇÃO

A atual demanda do mercado requer novas e diferentes soluções em software em um pequeno espaço de tempo. Para as organizações que desenvolvem software, entregá-lo de forma rápida, confiável, com qualidade e de acordo com as exigências dos clientes é uma realidade difícil. Neste contexto, observa-se a utilização de métodos ágeis de desenvolvimento de software, que tem por objetivo simplificar os processos, reduzindo a complexidade de planejamento, focando nas necessidades do cliente e moldando equipes colaborativas e participativas (HANSMANN; STOBER, 2010).

Muitos fatores podem ser citados para apontar as razões pelas quais os métodos ágeis são adotados. De acordo com pesquisa da VersionOne (2013), os motivos mais expressivos são a maior rapidez no tempo de resposta ao mercado, o gerenciamento de prioridade de mudanças e o melhor alinhamento entre TI e o negócio.

Vários estudos relacionam a adoção de métodos ágeis com melhoria da qualidade de software. Uma pesquisa realizada por Ambler (2008) destaca que a grande maioria dos respondentes da pesquisa indicou que a utilização de métodos ágeis produziu software com qualidade superior em comparação com o software produzido com métodos tradicionais. A pesquisa da versionOne (2013) também indica essa tendência, mostrando que 82% dos respondentes tiveram melhorias na qualidade de software ao adotar métodos ágeis.

Observa-se que times que implementam métodos ágeis com sucesso conseguem produzir software que atendam às reais necessidades dos clientes de maneira mais rápida, com custo reduzido e com melhor qualidade. No entanto, a implementação de métodos ágeis requer bastante esforço, tanto da equipe de desenvolvimento, quanto do resto da organização para que este seja implementado com sucesso e atinja os resultados esperados no desenvolvimento de software (COHN, 2013).

Na busca por implementar métodos ágeis de forma a atingir os resultados esperados, identificam-se vários desafios em sua utilização relacionado a testes de software, que são a forma mais importante de se assegurar ou controlar a qualidade do software (HERZLICH, 2007).

Observa-se que os processos no desenvolvimento ágil consideram pequenas iterações com entregas ao final de cada iteração (BECK, *et al.*, 2001) e, por causa da natureza do ciclo

de vida deste desenvolvimento, os testes devem ser executados de forma contínua e integrada desde o início do projeto, no qual o teste de software é parte integrante do processo de desenvolvimento (CRISPIN; GREGORY, 2009). Para integrar o teste de software ao desenvolvimento de software, muitas práticas podem ser utilizadas, tais como o TDD (*Test Driven Development*) proposta por BECK (2002).

Apesar das diversas práticas utilizadas nos métodos ágeis, distribuir as atividades de teste dentro de uma iteração a fim de garantir que estas atividades sejam executadas de forma contínua requer bastante esforço. Além disso, deixar os testes para o final do ciclo leva ao risco de apenas segmentar o desenvolvimento em pequenos ciclos mantendo em cada um deles a mesma sequência de etapas de desenvolvimento do modelo tradicional (COHN, 2013). Para que isso não aconteça, é necessário adaptar as técnicas e práticas de teste ao contexto ágil de forma que ambas possam estar de acordo com os princípios ágeis que valorizam mais os indivíduos e as iterações do que os processos e as ferramentas, mais o software funcional do que a documentação abrangente, mais a colaboração do cliente do que a negociação de contrato, e mais as respostas às mudanças do que a orientação a um planejamento (BECK, 2001).

É importante observar ainda que várias dificuldades são encontradas na adoção do teste de software em ambientes ágeis, tais como as apresentadas por Ambler (2012):

- 50% dos respondentes têm dificuldade em fazer com que todos os testes sejam feitos durante a iteração em curso.
- 37% não conseguem adotar abordagens ágeis, tais como o TDD.
- 33% têm dificuldade para a validação de requisitos não funcionais.
- 33% enfrentam problemas para ter os *stakeholders* ou clientes envolvidos com teste.
- 33% apresentam dificuldade em ter o teste feito pelo próprio desenvolvedor.
- 21% não conseguem ter teste de interface de usuário.
- 16% acham difícil aprender a testar durante todo o ciclo de desenvolvimento ágil.
- 13% têm dificuldade em adotar novas ferramentas de testes ágeis.
- 12% destacam que é difícil migrar os testes existentes e profissionais de qualidade para ambientes ágeis.

- 8% têm dificuldade para utilizar ferramentas de teste existentes para apoiar o desenvolvimento ágil.
- 8% acreditam ser difícil a permanência de um ambiente regulatório complacente.

Verifica-se que a adesão aos métodos ágeis tem sido crescente e que seu objetivo é melhorar a qualidade do software, mas a execução do teste de software, que é uma atividade importante para a qualidade do produto, apresenta vários desafios na adoção de métodos ágeis. Neste contexto, observa-se a necessidade de entender como as atividades de testes são implementadas dentro das empresas de desenvolvimento de software, além de avaliar se estas empresas estão de acordo com os princípios ágeis e são coerentes com atividades de teste necessárias para garantir a qualidade do software.

## **1.2 Questão de pesquisa e objetivo**

Este trabalho busca responder a seguinte questão: Quais as atividades de testes têm sido executadas pelos diversos tipos de empresas, times e pessoas em equipes que usam métodos ágeis para desenvolvimento de software?

Neste sentido, o objetivo do presente trabalho é verificar como atividades de teste têm sido implementadas em equipes de desenvolvimento de software que utilizam métodos ágeis de desenvolvimento. Essa verificação é feita a partir do levantamento e da análise das principais atividades de testes de software e da atuação das equipes de desenvolvimento em empresas de diversos portes, destacando tendências e práticas a partir de pesquisa realizada com desenvolvedores.

## **1.3 Organização**

O primeiro capítulo fez a contextualização do tema e dos principais objetivos, assim como a apresentação da organização deste trabalho.

O segundo capítulo faz uma breve descrição de métodos de desenvolvimento, observando a data de sua criação com foco nos métodos ágeis e no processo de teste dentro de cada método.

O terceiro capítulo destaca os principais fatores acerca do teste de software tradicional, assim como do teste de software em ambientes ágeis.

O quarto capítulo apresenta como a pesquisa foi realizada, destacando a experiência do pesquisador em testes ágeis, a pesquisa-piloto e a pesquisa por meio de questionário.

O quinto capítulo apresenta os resultados da pesquisa, assim como caracteriza os respondentes, apresenta graficamente os resultados e faz uma discussão sobre os valores alcançados.

O último e sexto capítulo apresenta a conclusão deste trabalho. Nele são apresentadas as contribuições alcançadas depois da aplicação do questionário e também propostas de trabalhos de futuros frutos da pesquisa.

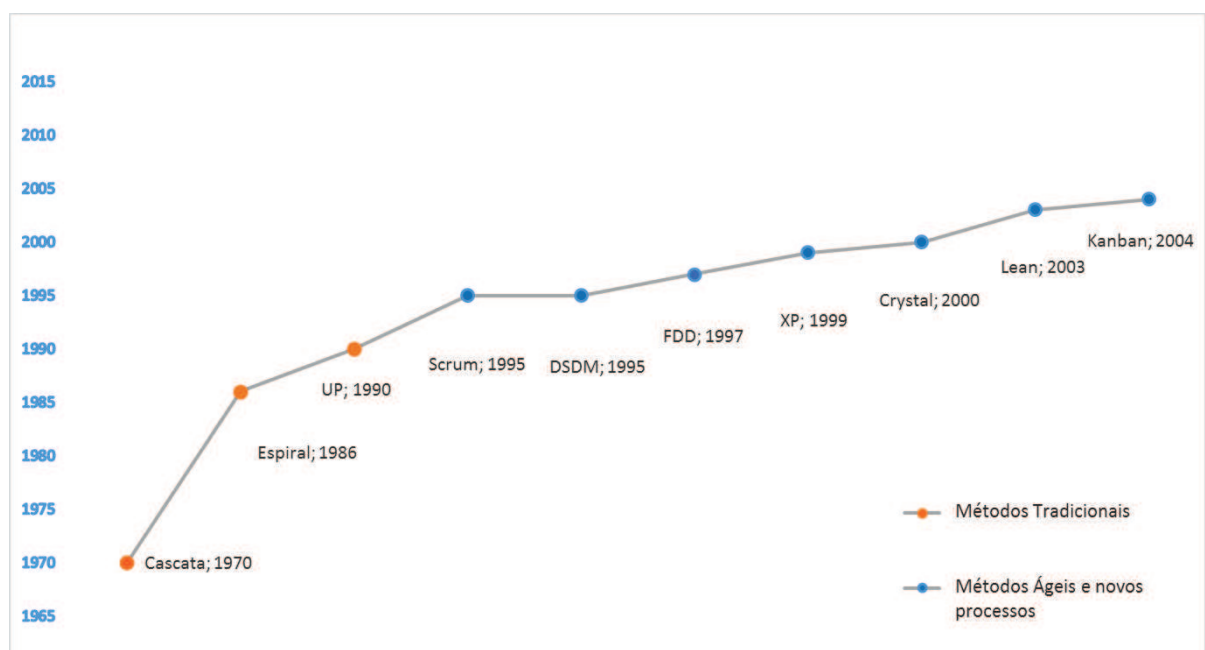
## 2 PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

A engenharia de software descreve um método como sendo uma abordagem estruturada com o objetivo de auxiliar na produção de software. Diferentes métodos possuem abordagens distintas que fazem com que cada um seja mais adequado para uma determinada área (SOMMERVILLE, 2007).

Os métodos tradicionais de desenvolvimento são preditivos e se baseiam na definição de requisitos no início do projeto, o que torna difícil o gerenciamento de mudanças de forma rápida. Estes métodos dependem de documentação bem definidas, se concentram em funções e enfatizam o planejamento ao longo do processo de construção do software. Já os métodos ágeis são baseados em planejamento adaptativo, desenvolvimento iterativo e evolutivo, flexíveis quanto a mudanças e com promoção da comunicação (KUMAR; BHATIA, 2014).

Observa-se a seguir a classificação dos métodos em tradicionais ou ágeis, métodos estes que são descritos de acordo com a data aproximada de publicação, conforme coletados nos trabalhos de Bassi (2008), Anderson (2010) e Pressman (2011). A Figura 1 mostra os métodos de desenvolvimento de software e suas respectivas datas de criação.

Figura 1: Métodos de desenvolvimento de software



Os métodos de desenvolvimento ágeis são baseados em pequenas iterações e com entregas ao final de cada iteração, seguindo os princípios ideológicos do manifesto ágil que foi criado para definir princípios comuns dos métodos ágeis que já existiam até então (BECK, *et al.*, 2001):

- Indivíduos e interações são mais importantes que processos e ferramentas.
- Software funcionando é mais importante do que documentação completa e detalhada.
- Colaboração com o cliente é mais importante do que negociação de contratos.
- Adaptação a mudanças é mais importante do que seguir o plano inicial.

Rico, Sayani e Sone (2009) fazem uma reflexão sobre estes valores. Para os autores, quando os métodos ágeis dão mais valor aos indivíduos, eles conferem maior poder aos desenvolvedores para formar a equipe e o gerenciamento próprio, requisitando, desta forma, bastante habilidade de comunicação e técnicas eficazes. Verifica-se, também, a importância do software funcionando, em que são priorizadas as tarefas de desenvolvimento, tendo em vista que o produto que será entregue é o software, e não a documentação. Espera-se que o cliente tenha mais interação com o desenvolvimento do produto, pois, desta forma, é possível atender às suas necessidades de maneira mais assertiva. As respostas às mudanças fazem com seja possível captar as necessidades dos clientes, desenvolver o software, mostrar o resultado e repetir o processo até que o cliente esteja satisfeito, respondendo, assim, às mudanças necessárias.

Dos diversos métodos e *frameworks* ágeis existentes na atualidade, os mais reconhecidos são *Scrum*, *Extreme Programming*, *Feature Driven Development*, *Crystal* e *Dynamic Systems Development*. Todos eles apresentam processos efetivos para o desenvolvimento ágil e, apesar de suas abordagens diferentes, seguem os princípios preconizados pelo manifesto ágil (RICO; SAYANI; SONE, 2009). Destaca-se, também, a importância do princípio *Lean* e do *Kanban* por serem úteis em diversas etapas de uma empresa (BORIA *et al.*, 2013). Uma breve descrição destes métodos, assim como a contextualização do teste de software nestes métodos serão descritos a seguir.

## 2.1 Métodos de desenvolvimento Ágeis

São diversos os métodos ágeis utilizados na atualidade. Os métodos mais conhecidos e descritos na literatura são o Scrum, o Dynamic Systems Development, o Feature Driven Development, o Extreme Programming, o Crystal, o Lean e o Kanban. O Scrum, que é a abordagem mais utilizada (VERSIONONE, 2013), é um framework no qual são empregados diversos processos e técnicas a fim de gerenciar o desenvolvimento de produtos complexos. O framework não define qual o processo e a técnica a serem utilizados para a elaboração do projeto (SCHWABER; SUTHERLAND, 2013). Ele é utilizado para orientar atividades de desenvolvimento dentro de um processo que incorpora as atividades de especificação de requisitos, análise, projeto, evolução e entrega do software (PRESSMAN, 2011).

O Dynamic Systems Development (DSDM) é baseado em oito princípios: foco no cliente, entrega no tempo certo, colaboração, não-comprometimento da qualidade, construção de forma incremental a partir de bases sólidas, desenvolvimento iterativo, comunicação clara e contínua, e demonstração de controle (DAVIS, 2013). O método é composto de um ciclo de vida que define três ciclos: iteração de modelos funcionais, iteração de projeto e desenvolvimento e implementação, que são precedidos por duas atividades adicionais: estudo da viabilidade e estudo do negócio (PRESSMAN, 2011).

No Feature Driven Development (FDD) a funcionalidade é o objeto valorizado e, portanto, essa deve ser entregue em duas semanas ou menos. O método preconiza a colaboração entre pessoas e equipes, a decomposição do projeto em funcionalidades para melhoria da gerência de problemas, a complexidade dos projetos e a comunicação verbal (PRESSMAN, 2011). Define-se um modelo de domínio com os requisitos iniciais dos usuários que provê uma lista de funcionalidades, nas quais as mais importantes são priorizadas. Durante a fase de desenvolvimento, estas funcionalidades são convertidas em programas e integradas ao sistema (UNHELKAR, 2013).

O método Extreme Programming (XP) é um método focado em técnicas de programação, comunicação clara e trabalho em equipe. O método está baseado em valores, princípios e práticas. Os valores são os motivadores das ações e as práticas são as evidências destes valores. Para que exista uma ligação entre valores e práticas, devem existir princípios que são como guias de domínio entre práticas e valores (BECK, 2004). Observa-se que as práticas do método XP devem ser aplicadas no contexto de quatro atividades: planejamento, projeto, codificação e testes (PRESSMAN, 2011).



O Crystal é uma família de métodos ágeis que priorizam a adaptabilidade, em que cada método é mais apropriado para um tipo de projeto. Para se ter este ajuste, os métodos possuem elementos essenciais e comuns a todos, mas com papéis, padrões de processos, produto de trabalho e práticas únicas para cada método (PRESSMAN, 2011). Os métodos do Crystal seguem sete princípios: entregas frequentes, feedback contínuo, comunicação constante, segurança, foco, acesso aos usuários, automação de testes e testes de integração (UNHELKAR, 2013).

O Lean é baseado em sete princípios que podem ser aplicados ao desenvolvimento de software, conforme descrito: eliminar o desperdício, desenvolver com qualidade, criar conhecimento, adiar comprometimentos, entregar rápido, respeitar as pessoas, otimizar o todo (AMBLER; LINES, 2012).

O Kanban significa literalmente “registro ou placa visível” e foi introduzido junto com o Sistema Toyota de Produção. O Kanban é baseado em cinco propriedades básicas: visualização do fluxo de trabalho, trabalho em progresso limitado, fluxo medido e gerenciado, políticas de processos explícitas e uso de modelos para reconhecer oportunidades de melhoria (ANDERSON, 2010). Para implementar o Kanban no desenvolvimento de software, deve-se mapear o fluxo de trabalho existente, definindo um ponto de início e um de fim e, então, pontos de interface entre o começo e o fim do controle (ANDERSON, 2010).

A partir da descrição feita nessa seção que contextualizou os principais métodos utilizados na atualidade é possível descrever como o teste de software é executado em cada método descrito anteriormente. Essa descrição será feita na próxima seção.

## **2.2 O teste de software no contexto dos métodos de desenvolvimento ágeis**

No contexto de teste de software o framework Scrum não define técnica para elaboração do projeto de teste. A equipe se auto-organiza para alcançar o resultado esperado. Todos os membros da equipe são responsáveis pela entrega do software com qualidade em todas as iterações (SCHWABER; SUTHERLAND, 2013).

No DSDM o teste de software não é definido como uma fase do processo de desenvolvimento, pois este acontece em todo o processo: tanto na iteração do modelo funcional, quanto na iteração de projeto e desenvolvimento. O teste não irá necessariamente satisfazer todas as necessidades identificadas, mas irá cobrir todas as exigências que foram acordadas para o incremento atual. Um núcleo de requisitos (subconjunto mínimo utilizável) deve ser testado

com as outras partes de acordo com o tempo disponível (DSDM, 2008).

Observa-se que no FDD os testes são realizados após o desenvolvimento das funcionalidades. Estas devem ser integradas ao software já existente e então testadas para que uma nova iteração ocorra (UNHELKAR, 2013).

No XP o teste faz parte do processo. O desenvolvedor primeiramente desenvolve o teste de unidade e posteriormente desenvolve o código que deverá ser construído para passar no teste. Destaca-se que a programação em dupla é bastante difundida no XP, em que dois programadores trabalham juntos no processo de desenvolvimento e, desta forma, podem resolver problemas em tempo real. Os testes de unidade devem ser desenvolvidos no início da codificação, o que auxilia a implantação de testes de regressão. Posteriormente, são executados os testes de aceitação, também conhecidos como testes de clientes, nos quais identificam-se as características e funcionalidades do software (PRESSMAN, 2011).

O teste de software no Crystal varia de acordo com o método da família. Observa-se que o Crystal possui várias formas de verificação, e que devem ser feitos controles para que o versionamento, os testes automatizados e as integrações de sistema sejam suportados (UNHELKAR, 2013).

No Lean se estabelece um conjunto de testes que faça com que o objetivo do negócio esteja claro para a implementação. O objetivo do teste não deve ser apenas de encontrar defeitos, mas, também, criar um ambiente que ajude a evitá-los (POPPENDIECK; POPPENDIECK, 2007).

O Kanban não define um processo de desenvolvimento, a empresa deve se adaptar de acordo com os seus processos e, desta forma, distribuir as atividades de testes dentro do painel de Kanban (ANDERSON, 2010). Ao definir o fluxo, reforçam-se as práticas de integração contínua e a gestão da qualidade. Destaca-se que o Kanban permite à equipe mais visibilidade, foco e rigor na prática de gestão de teste e filas de defeito (VALLET, 2014).

Observa-se que apesar de seguir os mesmos valores, cada método ágil adota uma abordagem de teste. Os testes estão alinhados com os valores ágeis e a forma como os métodos ágeis abordam o desenvolvimento de software.

Apesar das diversas opções existentes em métodos ágeis, implementá-los pode ser difícil para algumas organizações. Muitos líderes não percebem a dificuldade que o time tem em se adaptar para um modelo ágil, no qual os programadores devem mudar a abordagem técnica e social e os gerentes devem mudar a forma de interagir com os times. Ao se implementar os métodos ágeis, os líderes buscam entregar valor de forma rápida, com

qualidade, fazendo o necessário, engajando e inspirando as equipes. Para se alcançar estes resultados as equipes, no entanto, devem agir de forma ágil, se adaptando, explorando, facilitando e buscando soluções que possam integrar o pensamento de todos no time (HIGHSMITH, 2013).

Observa-se que, no contexto de teste de software, muitas atividades de testes deixam de ser executadas ou são adiadas pela dificuldade de se adaptar à transformação ágil. Em geral, problemas são encontrados na automação de testes, em que a equipe perde muito tempo depurando e mantendo testes automatizados e no gerenciamento de filas de defeitos, que muitas vezes não conseguem ser consertados de forma rápida, além de sofrerem impactos com a inexistência de uma política para prevenção de defeitos (CRISPIN; GREGORY, 2015).

É neste contexto que as atividades de teste serão abordadas no próximo capítulo e na pesquisa realizada, no qual as atividades buscam se adaptar aos modelos ágeis, mas no qual o time nem sempre consegue agir forma de ágil.

### 3 TESTE DE SOFTWARE

O desenvolvimento de software está sujeito a diversos tipos de problemas que acabam resultando na produção de um produto diferente do que se espera. A grande maioria destes problemas são causados por erro humano e, para que estes erros não perdurem, existe uma série de atividades de Validação, Verificação e Teste, ou VVT, que têm por objetivo garantir que tanto o modo de desenvolvimento, quanto o produto em si estejam em conformidade com o que foi especificado (DELAMARO, MALDONADO e JINO, 2007).

Segundo Sommerville (2007), verificação é o processo de se avaliar se o software está de acordo com os requisitos, ao passo que validação é a avaliação do sistema que visa assegurar que ele atenda às necessidades dos clientes. A atividade de teste é a principal técnica de verificação e validação de software (SOMMERVILLE, 2007). O teste de software é o processo de execução de um programa com a finalidade de se encontrar erros, no qual se aumenta a confiabilidade do software à medida que os erros são identificados e corrigidos (MYERS *et al.*, 2011).

De acordo com Myers *et al.* (2011), o teste de software é guiado por princípios considerados vitais para o processo de teste, sendo os princípios listados a seguir considerados os mais importantes:

- Teste é o processo de execução de um programa com o intuito de encontrar erros.
- O teste é mais bem-sucedido quando não é executado por aqueles que desenvolvem o programa, pois estes possuem uma visão diferente de quem está desenvolvendo.
- Um caso de teste bem-sucedido é aquele que detecta um erro desconhecido.
- Teste bem-sucedido inclui definição da entrada e saída esperadas, assim como as não esperadas. Deve-se verificar dados de entrada e saída válidos, assim como os inválidos, cobrindo o sistema amplamente dessa forma.
- Teste bem-sucedido inclui estudar cuidadosamente os resultados dos testes, pois um resultado plausível pode ser interpretado de forma equivocada.

Muitos fatores implicam na execução de testes de software. As seções seguintes irão descrever os principais elementos que compõem o processo de teste de software.

### 3.1 Estratégias de testes

A estratégia de testes é uma descrição dos níveis de testes a serem executados e quais testes serão executados dentro destes níveis. A escolha da estratégia de teste é um fator importante para o sucesso dos esforços em teste, assim como para a precisão dos planos de testes (GRAHAM, 2008).

Pressman (2011) destaca que a estratégia de teste fornece um roteiro com os passos a serem executados como parte do teste. Ele observa que existem várias estratégias de testes na literatura, mas que todas elas oferecem um modelo para o teste com algumas características em comum, tais como a execução de revisões técnicas eficazes para que muitos erros sejam eliminados no começo do teste, o início pelo nível de componente para depois progredir para a integração do sistema como um todo, a aplicação de diferentes técnicas de teste para diferentes abordagens de desenvolvimento, a execução de teste pelo desenvolvedor e pelo testador.

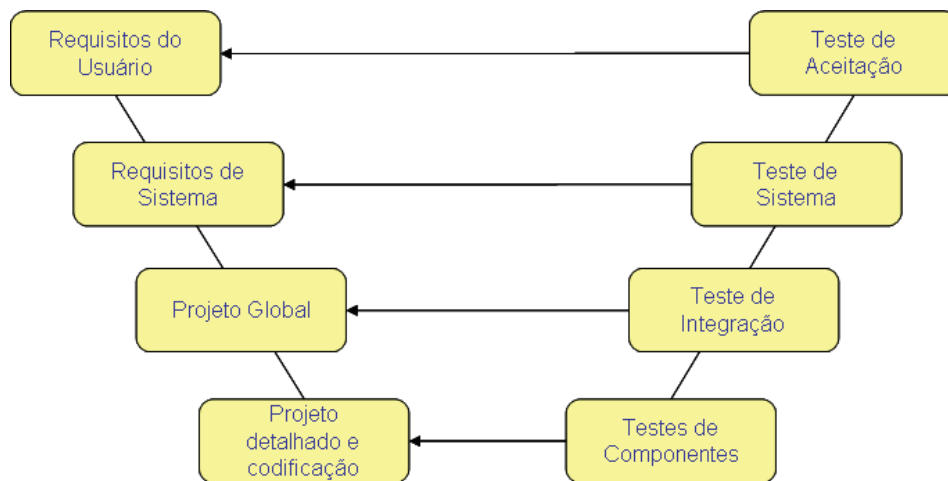
Kasurinen (2010) generaliza várias proposições encontradas na literatura e determina que uma estratégia de teste deve ter, pelo menos, os seguintes aspectos: ferramentas de teste (incluindo regressão e automação), métodos de testes, equipe de teste (incluindo responsabilidades e níveis de independência na organização), *design* de casos de testes e métodos de seleção e critérios de qualidade (tanto de entrada, quanto de saída).

Em estratégias de testes tradicionais, é comum se utilizar as fases de testes de acordo com as fases de desenvolvimento. Estas etapas de testes serão explicadas na próxima seção.

### 3.2 Níveis de testes

De acordo com Ammann e Offutt (2008), os testes podem ser derivados de requisitos, especificações, artefatos do projeto, ou códigos-fonte. Cada atividade de desenvolvimento é acompanhada por um nível diferente de teste. A fim de mostrar esta relação do desenvolvimento com o teste, apresenta-se o modelo em “V”, conforme a Figura 2, que geralmente possui quatro níveis: a fase de codificação com testes de componentes, o projeto global com os testes de integração, os requisitos de sistemas com os testes de sistemas, e a fase de definição de requisitos pelo usuário com os testes de aceitação.

Figura 2: Modelo em “V”



Fonte: Adaptado de ISTQB (2011)

No nível de teste de componente, também conhecido como teste de unidade, o objetivo principal é garantir que cada unidade individual do software esteja funcionando de acordo com sua especificação. O foco está em testar módulos separadamente, tais como programas, objetos e classes. Os testes são muitas vezes desenvolvidos e executados de forma isolada do resto do sistema e, geralmente, envolvem o programador que escreveu o código (ISTQB, 2011). Estes testes são úteis, pois podem ser criados de maneira rápida, executados mais cedo e com mais frequência. Esses testes são mais fáceis de serem depurados quando ocorre falha no teste (KOROŠEC; PFARRHOFER, 2015).

O teste de integração verifica a interface entre os componentes e iterações com diferentes partes de um sistema. Pode haver mais de um nível de teste de integração, tais como a integração de componente, que testa as iterações entre os componentes de software e é feito após testes de componentes, e os testes de integração do sistema, que testam as iterações entre sistemas diferentes, ou entre hardware e software (ISTQB, 2011).

No nível de teste de sistema, o foco está no comportamento do sistema como um todo. Os casos de teste são derivados de especificações de requisitos, processos de negócios, casos de uso, histórias de usuários ou outras descrições de texto de alto nível ou modelos de comportamento do sistema (ISTQB, 2011).

Os testes de aceitação servem para criar confiança no sistema. Eles não têm por objetivo encontrar defeitos. Casos de teste de aceitação são baseados nos requisitos dos clientes e devem demonstrar que o sistema ou componente está pronto para ser entregue aos *stakeholders* (ISTQB, 2011).

O teste de regressão não é um nível de teste, mas um reteste do software quando este foi modificado para uma nova versão, em que as funcionalidades do software são mantidas, mas com algumas modificações que podem introduzir defeitos. O teste de regressão pode ocorrer em qualquer nível do teste de software (BURNSTEIN, 2003).

Dentro dos níveis de teste, diversos tipos de testes podem ser utilizados para garantir que o software testado esteja de acordo com o especificado. Estes tipos de testes serão cobertos na próxima seção.

### **3.3 Tipos e abordagens para teste de software**

Observa-se que existem diferentes tipos de testes, tais como testes de segurança, teste de carga, entre outros. Os tipos de teste estão focados em um objetivo específico e podem ser executados dentro de qualquer nível (GRAHAM, 2008). Estes testes podem ser classificados em quatro categorias: testes funcionais, testes não funcionais, testes estruturais e testes relacionados à mudança (ISTQB, 2011).

Os testes funcionais verificam a capacidade de um software realizar as funcionalidades requisitadas pelos clientes. Nesta abordagem, considera-se a funcionalidade externa do software (ISTQB, 2011). Este tipo de teste também é conhecido como teste de caixa preta, pois, para a projeção e a execução do teste, são fornecidos dados de entrada e são avaliadas as saídas para verificar a conformidade do software (FABBRI *et al.*, 2007).

Já os testes não funcionais verificam características, tais como carga e portabilidade. Características que não estão diretamente relacionadas às funcionalidades do software. O objetivo principal deste tipo de teste é verificar como o software realiza determinada função (GRAHAM, 2008).

O objetivo do teste estrutural é verificar a estrutura do sistema ou componente. Este tipo de teste também é conhecido como caixa branca, no qual se utiliza da estrutura conhecida do software para a projeção e a execução dos testes (GRAHAM, 2008).

Os testes relacionados a mudanças têm por objetivo verificar se as alterações feitas no software, geralmente advindas do conserto de um defeito, não causaram outros problemas no software. Para verificar se o defeito foi consertado, retesta-se o software e o teste de regressão

é executado a fim de verificar se outras partes foram quebradas devido à mudança feita (GRAHAM, 2008).

Existem várias formas de se verificar se um software está funcionando de acordo com o esperado. As próximas seções irão mostrar alguns conceitos na definição de testes manuais e automatizados.

### 3.4 Testes manuais

Os testes manuais são processos de execução do software feitos pelo testador sem o apoio do software que automatiza este processo com o objetivo de encontrar defeitos. Em geral, o testador executa a aplicação como o usuário final para garantir o funcionamento correto do software (KUMAR, 2012).

Observa-se que encontrar defeitos relevantes de forma eficiente é um desafio na engenharia de software, e que grande parte dos defeitos funcionais são encontrados por testes manuais. A fim de encontrar estes defeitos, os testadores aplicam numerosas técnicas e estratégias durante a execução dos testes, não confiando apenas na documentação fornecida pelos casos de testes; durante a execução de testes exploratórios, também se aplicam as técnicas descritas na definição de casos de testes (ITKONEN; MÄNTYLÄ; LASSENIUS, 2009).

A seguir, serão apresentados os principais conceitos sobre as duas principais abordagens de testes manuais: definição de casos com *design* predefinido e testes exploratórios.

#### 3.4.1 Casos de teste – *design* predefinido

A fim de otimizar o teste de software e criar testes que possam verificar o software de maneira efetiva, vários métodos podem ser aplicados em uma estratégia de *design* predefinido. Dentro das estratégias de caixa preta, em que não se tem acesso ao código durante o teste, destaca-se a equivalência de partição, análises de valores limites, gráfico de causa e efeito e erros por adivinhação. Nas estratégias de caixa branca com acesso ao código, destaca-se cobertura de declarações, cobertura de decisão, cobertura de condição, cobertura de condição/decisão e cobertura de múltiplas decisões (MYERS *et al.*, 2011). Estes métodos para



criar casos de testes mostram a preocupação que se tem no planejamento dos testes. Uma vez que não se consegue testar tudo, as técnicas ajudam a criar testes que possam ser efetivos.

A especificação de um caso de teste deve conter detalhes de cada caso de teste, sendo esta especificação composta de: uma identificação única, identificação de itens e funcionalidades a serem testados, especificação de dados de entrada de cada caso de teste, especificação de dados de saídas de cada caso de teste, ambiente requerido para o teste, especificação de procedimentos ou configurações específicas para o caso de teste e, quando necessário, a lista de dependência de outros casos de testes. Um caso de teste deve ter, também, a especificação dos procedimentos de teste com os passos de testes que determinam se um teste passou ou falhou (COPELAND, 2004).

#### 3.4.2 Testes exploratórios

No teste exploratório, o testador controla o *design* de casos de testes no mesmo momento em que estes são criados. Observa-se que a informação que o testador adquire ao executar um conjunto de testes o guia para definir e executar os próximos testes (COPELAND, 2004).

Os testes exploratórios são importantes, pois permitem ao testador utilizar as habilidades de ouvir, ler, pensar e reportar rigorosamente e efetivamente sem ter que utilizar instruções de um pré-roteiro. Nele, o testador controla ativamente o *design* dos testes enquanto estes são executados e utiliza estas novas informações para projetar novos e melhores testes. Muitos autores destacam que o testes exploratórios podem ser qualquer teste executado a fim de encontrar erros. Existem, no entanto, níveis de controle para execução destes testes que, nestes casos, resultam em uma série de notas que podem atualizar o material de teste, assim como a massa de dados para teste (BACH, 2003).

Na execução dos testes exploratórios gerenciados, o testador interage com o produto a fim de cumprir uma missão de testes e resultados em relatórios. A missão é cumprida através de um ciclo contínuo para alinhar com a própria missão, concebendo perguntas sobre o produto que, se respondidas, permitem concluir a missão, que é projetar testes para aquelas questões; caso contrário, ajustam-se os testes e continua-se explorando o software até que se encontre resultados. A fim de gerenciar os testes exploratórios, muitas equipes utilizam guias de cobertura para organizar o esforço de teste (BACH, 2003).

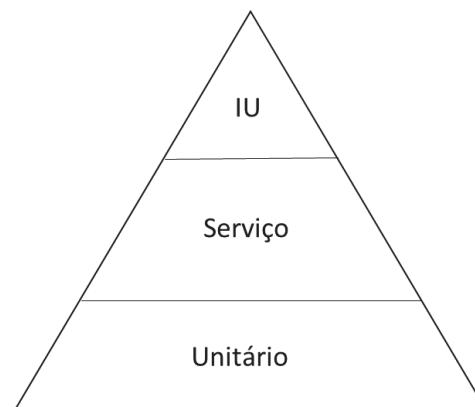
Com a utilização de testes exploratórios, é possível priorizar quais testes podem ser executados de acordo com o tempo disponível. No teste exploratório, é possível utilizar o seguinte processo: criação de um mapa mental das funcionalidades do sistema, desenho de um ou mais testes, execução de testes e observação de resultados, avaliação de resultados e repetição do processo até que se aprove ou não a funcionalidade testada. Para se ter uma missão clara do teste exploratório, pode-se definir o que testar, o que documentar, qual tática utilizar, que tipos de defeitos procurar e quais os riscos envolvidos (COPELAND, 2004).

### 3.5 Testes automatizados

Testes automatizados são programas ou *scripts* que executam as funcionalidades do sistema e, desta forma, verificam se o software está funcionando de acordo com o esperado. Destaca-se que a grande vantagem de utilizar automação é a facilidade para executar os testes repetidos (BERNARDO; KON, 2008).

Cohn (2013) destaca que mesmo antes da ascendência dos métodos ágeis, já se dava importância à automação de testes, mas por ser considerada uma atividade cara e que levava tempo para a conclusão, não era realizada constantemente pelas equipes. Um dos motivos para este problema é que muitas vezes se aplicava uma estratégia de automação no nível errado. Uma estratégia de automação efetiva é realizada em níveis de unidade, de serviços e de IU (Interface de Usuário), conforme descrito na pirâmide da Figura 3:

Figura 3: Pirâmide do teste de automação



Fonte: Adaptado de Cohn (2013) – *Succeeding with agile*

A base da pirâmide é composta pelos testes de unidade, que devem ser o alicerce para uma boa estratégia de teste de automação. Estes testes representam a parte mais larga da pirâmide. Os testes de unidade permitem maior facilidade aos desenvolvedores para encontrar a causa raiz dos defeitos. Por serem escritos com a mesma linguagem do sistema, os programadores ficam mais confortáveis para escrevê-los (COHN, 2013).

Os testes de serviço, que ficam no meio da pirâmide, servem para verificar as respostas da aplicação em relação a diferentes entradas no sistema. Portanto, ao invés de se aplicar vários testes em nível de interface, estes testes são aplicados em nível de serviço. Observa-se que, desta forma, é possível testar várias combinações de entradas e saídas de resultados (COHN, 2013).

No último nível da pirâmide ficam os testes de IU, que validam a interface com o usuário. A cobertura destes testes é menor, pois são mais frágeis, isto é, qualquer alteração do sistema pode quebrar a automação. São mais caros de escrever, pois um teste de interface de usuário que permanece útil por várias vezes requer mais trabalho em sua elaboração do que um teste criado apenas através de captura de *playback*, além de levarem mais tempo para serem executados. Experiências mostram que, muitas vezes, nem mesmo um conjunto de servidores permitem executar todos os testes em uma noite (COHN, 2013).

O custo da automação está diretamente ligado ao número de vezes que um teste é executado. Observa-se que a automação será mais vantajosa quando o custo de automação, que é composto pelo custo de ferramentas mais o custo para criação do *script* mais o custo para manutenção do *script*, for menor que o custo de se executar o teste manualmente várias vezes. Isto é, quanto maior a quantidade de vezes que um teste é executado em um projeto, mais vantajoso se torna automatizar este teste (SCHWABER; GILPIN, 2005). Alsmadi (2012) destaca que o custo de automação pode ser maior em um primeiro momento, mas, uma vez implantado, o custo tende a cair com o tempo, tornando-se mais baixo que a execução manual.

Já Ramler e Wolfmaier (2006) destacam vários fatores que devem ser analisados quando se pretende automatizar os testes, tais como: a natureza de cada teste, pois estes são processos diferentes com dinâmicas diferentes e, conseqüentemente, trazem resultados diferentes; a importância dos testes manuais e automatizados na probabilidade de se encontrar defeitos e os impactos destes; o contexto em que estes testes são executados, levando-se em consideração os recursos disponíveis e o orçamento destinado para o teste, assim como custos adicionais que podem ocorrer durante o projeto, como mudanças de funcionalidades que afetam diretamente os testes automatizados fazendo com que muitas vezes estes tenham que ser abandonados, além

do aumento do custo por falsos positivos na execução e custos de ferramentas e treinamento para a automação.

Uma vez que se tem definido como os testes podem ser executados, torna-se importante definir como as pessoas executam estes testes. As equipes de testes muitas vezes são partes integrantes do time de desenvolvimento, outras vezes são independentes. A seguir serão apresentadas as principais abordagens sobre a formação de uma equipe com o foco em teste de software.

### **3.6 Equipe de teste**

Quanto à equipe de desenvolvimento, o *International Software Testing Qualifications Board* (ISTQB, 2011) defende que, apesar dos desenvolvedores serem capazes de testar seus próprios códigos, ter a separação de responsabilidade traz vantagens, pois o testador, por não estar envolvido com o desenvolvimento, consegue obter uma visão diferente do software e, assim, encontrar diferentes defeitos, além de ser um profissional treinado para executar testes.

Myers *et al.* (2011) destacam, ainda, a possibilidade de organizações independentes para o teste de software. Para os autores, a organização que desenvolve o software tem dificuldade em testar objetivamente o mesmo programa, enquanto as organizações independentes não são influenciadas pelas mesmas pressões de gestão que controlam as organizações de teste, trazendo, desta forma, mais eficiência ao teste de software.

Observa-se que, muitas vezes, as técnicas e os tipos de testes utilizados nos métodos tradicionais são também utilizados nos métodos ágeis (VEENENDAAL, 2010). No entanto, no desenvolvimento ágil, deve-se distribuir as atividades de testes de forma contínua e, em geral, os especialistas de teste são parte integrante do time de desenvolvimento (CRISPIN; GREGORY, 2009).

A seguir, serão apresentados alguns estudos sobre o teste de software em ágil, como as técnicas de teste podem ser empregadas neste contexto, e como a equipe de desenvolvimento trabalha para garantir a qualidade do software.

### 3.7 Testes de software no contexto ágil

A estratégia de teste de software no contexto ágil é diferente dos métodos tradicionais. Isto se dá por causa da natureza do ciclo de vida de um projeto ágil, no qual os testes contínuos e integrados devem ser realizados desde o início do projeto. Destaca-se que o teste em ágil não é uma fase do desenvolvimento, mas parte integrante deste processo (CRISPIN; GREGORY, 2009). Cohn (2013) observa que distribuir as atividades de teste de software dentro de uma iteração não é tarefa fácil e que não se deve deixar para fazer o teste apenas no final da iteração, pois se cria pequenos modelos em cascata que não se adequam aos princípios ágeis.

O teste ágil é uma forma colaborativa de teste, no qual todos estão envolvidos nos processos de definição, implementação e execução do plano de testes. Os clientes devem colaborar com os testes de aceitação, e os desenvolvedores, com os testes automatizados. Observa-se que o teste no contexto ágil requer colaboração e comunicação de todos os envolvidos no processo de desenvolvimento (MYERS *et al.*, 2011).

Muitas técnicas são utilizadas para se alcançar qualidade nos métodos ágeis. Khalane e Tanner (2013) destacam, no entanto, que no Scrum, que é a abordagem mais utilizada na atualidade, não existe um guia de estratégias de qualidade de software. Os autores observam que esta falta causa dificuldades para o time de desenvolvimento em atingir os atributos de qualidade.

Observa-se que, apesar das diferenças que existem no teste de software entre os métodos tradicionais e ágeis, os tipos de testes e as técnicas utilizadas são as mesmas para ambos (VEENENDAAL, 2010). Analisando o contexto do desenvolvimento ágil, Itkonen, Rautiainen e Lassenius (2005) discorrem sobre a dificuldade de se adaptar os princípios ágeis ao teste de software, assim como os princípios de teste nos métodos ágeis.

Entregar software de valor ao cliente cedo e continuamente resulta em um desafio para o teste quando as equipes não estão alinhadas de forma efetiva aos valores ágeis. Observa-se que o ciclo com liberação rápida coloca prazos fixados sobre as atividades de teste e não permite que o período de testes seja estendido se forem encontrados mais defeitos do que a quantidade estimada (ITKONEN, RAUTIAINEN, LASSENIUS, 2005).

As mudanças de requisitos também representam um desafio, pois as técnicas tradicionais de teste são baseadas em especificações que são completadas em certa fase do desenvolvimento e que servem de base para as outras atividades de teste. Neste contexto,

destaca-se também a comunicação, que em ágil deve ser face a face, ou seja, as pessoas da área de negócios e os desenvolvedores devem trabalhar juntos nos projetos; além disso, os detalhes sobre os resultados esperados são de conhecimento dos desenvolvedores e analistas de negócio, mas nem sempre são fáceis de serem absorvidos pela equipe de teste (ITKONEN, RAUTIAINEN, LASSENIUS, 2005).

Outra premissa que se observa é de que o software funcionando é a principal métrica de progresso. Isto significa que o teste não pode ser deixado como uma etapa final de uma iteração, uma vez que deve fornecer informações sobre a qualidade alcançada o quanto antes a fim de permitir a avaliação do código produzido e assegurar que ele realmente funciona como esperado (ITKONEN, RAUTIAINEN, LASSENIUS, 2005).

Finalizando, verifica-se a simplicidade como um princípio desafiador para a equipe de garantia de qualidade, pois este princípio faz com que seja difícil manter as práticas de qualidade tradicionais ao processo de desenvolvimento, visto que as atividades podem ser percebidas como desnecessárias e improdutivas e que não adicionam valor em termos de código e características (ITKONEN, RAUTIAINEN, LASSENIUS, 2005). Estes desafios podem ser vistos de forma resumida na Tabela 1:

Tabela 1: Desafios do teste de software em métodos ágeis

Princípio ágil	Desafio
Frequente entrega de software com valor	- Tempo curto para o teste em cada ciclo - Teste não pode exceder o prazo final
Respostas às mudanças mesmo que em etapas avançadas do desenvolvimento	- Teste não pode ser baseado em uma especificação completa
Baseada na comunicação face a face	- Desenvolvedores e pessoas da área de negócio devem estar envolvidas nos testes
Software funcionando é a principal métrica de progresso	- Informações de qualidade são requisitadas cedo e frequentemente durante o desenvolvimento
Simplicidade é essencial	- Algumas práticas de testes podem ser eliminadas na busca pela simplicidade

Adaptado de Itkonen, Rautiainen e Lassenius (2005): *Toward an understanding of quality assurance in agile software development*

Conforme a Tabela 1, verifica-se que entregar um software com valor pode ser um

desafio para o teste em ágil, pois o tempo de uma iteração pode não ser suficiente para executar todos os testes com a finalidade de se entregar o software funcionando, tendo em vista que nem sempre é possível exceder o prazo final da iteração para que se possa executar todos os testes de maneira tradicional.

Observa-se que lidar com mudanças no contexto de teste software também traz alguns desafios no planejamento dos testes, pois não é possível se basear em uma especificação completa. A comunicação também pode ser um desafio no teste de software, pois há dificuldades em se engajar as pessoas da área de negócio e os desenvolvedores nas atividades de testes.

Gerar métricas de qualidade é outro desafio a ser destacado, pois as informações de qualidade são requisitadas cedo e de forma frequente durante o desenvolvimento. A principal métrica é ter o software funcionando, e nem sempre é possível trazer valores que contribuam para a melhoria dos testes dessa forma, pois os testes são executados frequentemente. Verifica-se também que algumas equipes podem eliminar determinadas práticas de testes na busca pela simplicidade por não estarem adaptadas aos valores dos métodos ágeis.

Observando os desafios descritos, verifica-se que o teste de software deve sofrer modificações tanto no processo, quanto no comportamento daqueles que os executam para que esteja alinhado aos princípios ágeis, e para que as equipes consigam superar os desafios de adaptação aos princípios ágeis.

Da perspectiva do teste de software, Itkonen, Rautiainen e Lassenius (2005) também fazem uma reflexão. Eles destacam que, quando o desenvolvedor testa o próprio código, ele tem dificuldade em encontrar problemas naquilo que ele mesmo desenvolveu. Em algumas metodologias, desenvolvedores assumem o papel de testador, mas, ainda assim, não chegam a ser totalmente independentes. Destaca-se que as atividades de testes requerem habilidades específicas que nem sempre os desenvolvedores e clientes estão aptos a fazer.

Os autores Itkonen, Rautiainen e Lassenius (2005) destacam, também, a especificação dos resultados de teste, em que a dependência de testes automatizados nem sempre consegue detectar com eficiência o resultado correto esperado pelo software. Observam, também, que os testadores trabalham sob uma ação destrutiva, isto é, eles tendem a testar o software com o objetivo de quebrar e encontrar defeitos, já os desenvolvedores trabalham com uma filosofia construtiva que visa criar um produto. A análise é finalizada com o princípio de avaliação da qualidade alcançada, que só pode ser medida através de acompanhamento de conformidades com a qualidade. A Tabela 2 mostra estes desafios de maneira resumida sob a visão dos principais princípios de teste de software.

Tabela 2: Princípios de teste vs. práticas ágeis

Princípio do teste	Práticas contraditórias nos métodos ágeis
Independência do teste	<ul style="list-style-type: none"> <li>- Desenvolvedores escrevem o teste para o próprio código</li> <li>- Definição do papel do testador que é rotacionado entre a equipe de desenvolvimento</li> </ul>
Testar requer habilidades específicas	<ul style="list-style-type: none"> <li>- Desenvolvedores fazem do teste parte do desenvolvimento de código</li> <li>- Clientes têm um importante papel para a qualidade do software que requer bastante colaboração e responsabilidade.</li> </ul>
Problema do oráculo	<ul style="list-style-type: none"> <li>- Baseado em testes automatizados para revelar defeitos</li> </ul>
Atitude destrutiva	<ul style="list-style-type: none"> <li>- Desenvolvedores são voltados para atividades construtivas, já os testadores, para atividades destrutivas, com o intuito de encontrar defeitos</li> </ul>
Avaliação da qualidade	<ul style="list-style-type: none"> <li>- A confiança na qualidade vem através da conformidade com um conjunto de boas práticas.</li> </ul>

Adaptado de Itkonen, Rautiainen e Lassenius (2005): *Toward an understanding of quality assurance in agile software development*

Na Tabela 2, observa-se que diversas práticas de métodos ágeis podem ser contraditórias aos princípios de testes. Verifica-se que é importante ter a independência do teste, mas, em muitas implantações de métodos ágeis, o próprio desenvolvedor testa o próprio código. Além disso, em algumas abordagens o teste é rotacionado entre membros da mesma equipe. Porém, mesmo neste caso o testador não é totalmente independente, o que pode impactar nos resultados do teste.

Verifica-se também que testar requer habilidades específicas e nem sempre os desenvolvedores e clientes estão preparados para esta atividade. Considera-se ainda que nas práticas ágeis de teste de software, nem sempre é possível definir os resultados esperados de forma consistente e, assim, deixar descobrir defeitos. Observa-se que as atividades requerem um olhar destrutivo para o software com o intuito de se encontrar defeitos, e que a equipe que está focada em atividades construtivas nem sempre consegue averiguar o software de maneira crítica. Além do mais, destaca-se que muitas vezes a avaliação da qualidade do software pode estar comprometida, já que este software vem da conformidade com um conjunto de boas práticas em muitos métodos ágeis, e não da utilização de indicadores de teste de software.

É importante ressaltar que as atividades de teste fazem parte do processo de



desenvolvimento e o software é considerado concluído quando essas atividades são realizadas. Observa-se, por exemplo, que o conceito de “pronto” no Scrum pode variar entre várias equipes. No entanto, os membros devem ter um entendimento comum do que significa para o trabalho “estar concluído”, a fim de que se possa garantir a transparência. Além disso, cada incremento é aditivo para todos os incrementos anteriores, tendo que ser completamente testado para garantir que todos os incrementos trabalhem juntos. Em equipes mais maduras, espera-se que a definição de pronto irá expandir para incluir critérios mais rigorosos para uma maior qualidade (SCHWABER; SUTHERLAND, 2013).

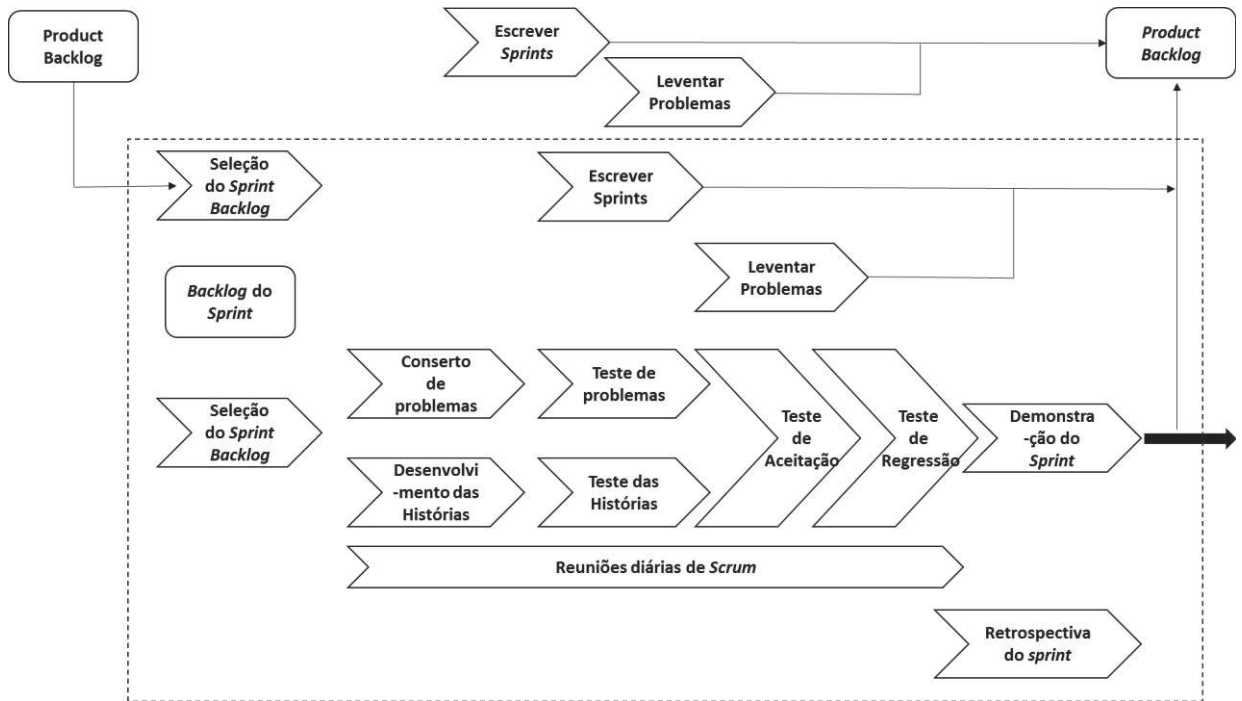
Embora ainda existam desafios para que algumas equipes executem o teste de software no contexto ágil seguindo os seus princípios de maneira efetiva, várias técnicas podem ser aplicadas por estas equipes para que elas consigam executar esses testes com sucesso e atingir o conceito de “pronto” para o incremento ou produto. Essas técnicas serão discutidas nas próximas seções.

### 3.7.1 Estratégias de teste em métodos ágeis

De acordo com a norma ISO/IEC/IEEE 29119-1, uma estratégia de teste ágil deve refletir o modelo de desenvolvimento, utilizando-se do mesmo vocabulário e incluindo os mesmos conceitos, tais como: *backlogs*, *sprints*, scrum diário, além de considerar o perfil de risco para projetos e produtos cobertos pelo teste.

No scrum, por exemplo, o gerenciamento de teste deve ser integrado ao gerenciamento do *product backlog*, aos *sprints* individuais e às reuniões diárias. O time deve planejar atividades de teste e atribuir responsabilidades aos membros. Os métodos ágeis seguem uma estratégia durante todo o ciclo de desenvolvimento de produto, e o teste pode ser executado como demonstra a Figura 4, que representa o desenvolvimento em um *sprint*.

Figura 4: Exemplo do ciclo de vida de um *sprint* em ágil



Fonte: Adaptado da norma ISO/IEC/IEEE 29119-1

Na Figura 4, verifica-se que todas as atividades de desenvolvimento são acompanhadas por atividades de testes, e que estas são complementadas pelos testes de aceitação e regressão para que, assim, o time esteja preparado para uma demonstração do que foi desenvolvido. As reuniões diárias acompanham o processo e o time de teste deve estar integrado nestas reuniões. Ao final do *sprint*, deve-se fazer uma retrospectiva de todo o processo de desenvolvimento para que se alcance melhorias no processo.

Crispin e Gregory (2009) sugerem uma proposta de estratégia para métodos ágeis como demonstra o Quadro 1, na qual a equipe de teste deve sempre participar de todo o processo.

Quadro 1: Estratégia de teste em métodos ágeis

Início do projeto	Obter compreensão do projeto
Planejamento	Participação no dimensionamento das histórias Criação de plano de testes
Iterações	Participação no planejamento dos <i>sprints</i> , estimando tarefas Criação e execução de testes das histórias Teste em pares com outros testadores e desenvolvedores Validação do negócio Automação de novos testes funcionais Execução de testes de regressão automatizados Execução de testes de carga Demonstração para os <i>stakeholders</i>
Fim do jogo	Simulação de testes para a gestão de lançamento Execução de teste de fumaça Execução de teste de carga (se necessário) Execução de testes de aceitação Participação do lançamento
Lançamento para produção	Participação no lançamento para produção Participação nas retrospectivas

Fonte: Adaptado de Crispin e Gregory (2009) – *Agile Testing*

De acordo com a estratégia proposta por Crispin e Gregory (2009), o testador deve participar durante todo o processo de desenvolvimento dentro de uma iteração. Inicialmente, o testador deve obter uma compreensão do projeto para que possa posteriormente ajudar no planejamento e, desta forma, mostrar à equipe quais são os pontos de atenção no dimensionamento das histórias quanto aos fatores que incluem o teste de software. Observa-se, também, que o testador participa ativamente durante as iterações, participando desde o planejamento até a demonstração para os *stakeholders*. Durante o processo, o time verifica e valida o software e também automatiza testes que auxiliam na integração contínua do software. Ao final do processo, o testador também participa de testes que auxiliam no lançamento do produto e nas retrospectivas com *feedback*.

Agarwal *et al.* (2014) destacam ainda que, em se tratando de desenvolvimento ágil, o envolvimento daqueles que irão participar do teste de software deve começar no início do projeto para que estes tenham conhecimento de todos os fatores que compõem o desenvolvimento, desde a arquitetura até o custo do projeto, pois todos estes fatores implicam na qualidade de software. Os autores (Agarwal *et al.*; 2014) observam também que o teste é um componente-chave no desenvolvimento ágil, em que se fazem necessários testes efetivos para se entregar valor ao cliente em intervalos frequentes. Para isso, todos os membros do time devem estar envolvidos com o processo de teste, que deve contar com a contribuição de um *expert* em teste de software. Destacam ainda que, dentro do time, deve-se encorajar as pessoas a desenvolver múltiplos papéis para tornar o processo de desenvolvimento mais efetivo. Além disso, deve-se fazer com que o processo de desenvolvimento e teste movam em paralelo, garantindo que os defeitos sejam consertados nos estágios iniciais de desenvolvimento e, dessa forma, maximizando os recursos.

Entre as várias técnicas e atividades existentes para o desenvolvimento e o teste em ambientes ágeis, pode-se destacar o *Test Driven Development* (TDD), o *Acceptance Test Driven Development* (ATDD) e o *Behavior Driven Development* (BDD), que são importantes para a adequação do teste aos princípios ágeis (ISTQB, 2014).

O TDD proposto por Beck (2002) defende que não se deve escrever o código novo antes que se tenha um teste automatizando falhando e que toda a duplicação tenha sido eliminada. De acordo com a ISO/IEC/IEEE 29119-1, TDD é uma prática na qual testes codificados são escritos antes da codificação do software. Os testes são baseados nas histórias de usuário e geralmente são desenvolvidos pelos desenvolvedores e testadores juntos. Eles normalmente são implementados por ferramentas de testes que automatizam o código e, como resultado do TDD, é gerada a codificação do programa. Delamaro *et al.* (2009), após um estudo sistemático sobre teste de software na metodologia ágil, verificaram que a estratégia mais utilizada em ambientes ágeis é o TDD para testes de unidade. Agarwal e Deep (2014), em um estudo empírico, apontam que a utilização do TDD traz melhor produtividade aos desenvolvedores, reduz a injeção de defeitos no software, aumenta a cobertura de código por teste, e ajuda no processo de *design*.

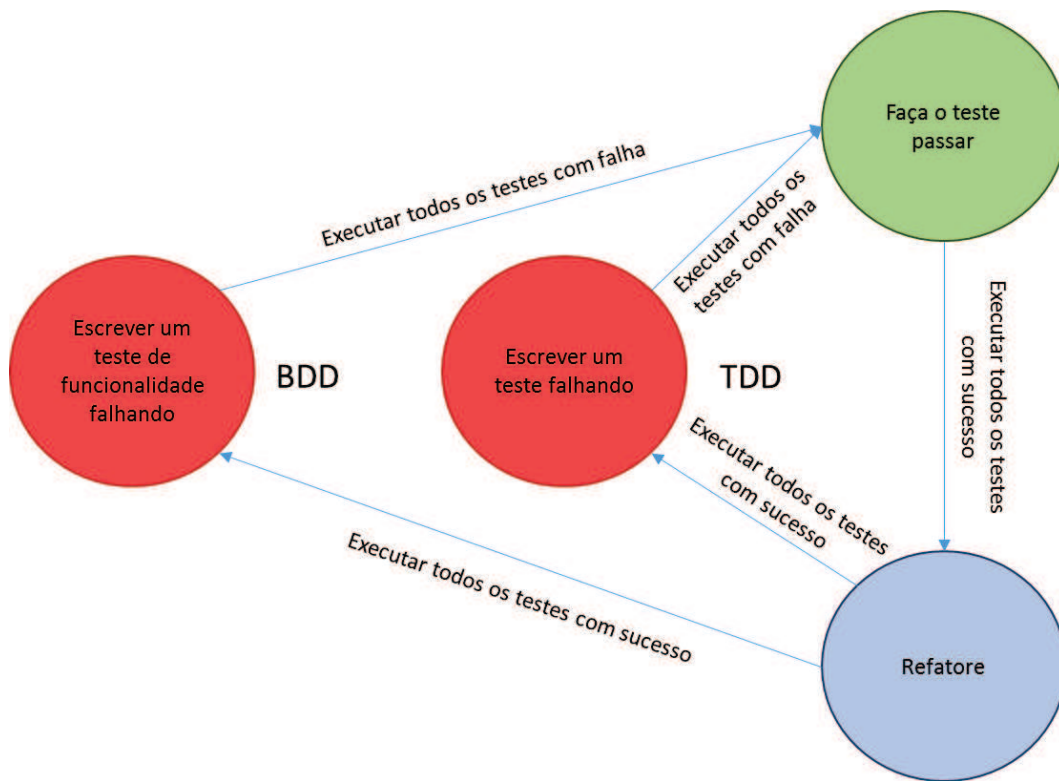
A abordagem do *Acceptance Test Driven Development* (ATDD) define o critério de aceitação e os testes durante a criação das histórias dos usuários. Com esta abordagem, é possível que os *stakeholders* entendam o comportamento de cada componente e como os desenvolvedores e testadores devem trabalhar para criar estes componentes (ISTQB, 2014). A utilização do ATDD faz com que todo o time colabore na identificação dos requisitos do

software, assim como garante melhor entendimento destes antes que o desenvolvimento comece. Antes da implementação de cada recurso, os membros da equipe colaboram para criar exemplos concretos do recurso que será desenvolvido e, em seguida, estes exemplos são traduzidos para testes de aceitação automatizados. Estes exemplos e testes se tornam importantes para a definição de “pronto”, ou seja, quando este recurso pode ser utilizado pelo usuário final. (GARTNER, 2013). Hoffmann *et al.* (2014) verificaram que, com a aplicação do ATDD para o desenvolvimento de um sistema acadêmico de tempo real, eles conseguiram não apenas facilidade na especificação dos requisitos, mas também a consciência do time a respeito da importância dos testes como mecanismo auxiliar na garantia de qualidade. Observa-se que o ATDD pode ser uma abordagem interessante para o teste de software em ambientes ágeis, garantido o entendimento dos requisitos por todos os interessados, assim como a execução de testes dentro do processo de desenvolvimento.

O desenvolvimento dirigido por comportamento (BDD - *Behavior Driven Development*) permite que os desenvolvedores estejam focados na criação de um código de testes baseado no comportamento do software. Uma vez que estes são criados sob a perspectiva do comportamento, fica mais fácil para os membros do time e os *stakeholders* compreenderem o software (ISTQB, 2014). Chelimsky *et al.* (2010) recomendam que se utilize o mesmo ciclo do TDD, mas que se utilize uma linguagem ubíqua entre o cliente e o time de desenvolvimento. O BDD captura o comportamento do software antes do desenvolvimento e segue o mesmo ciclo do TDD, no qual o teste escrito em BDD inicialmente falha e passa assim que a implementação preenche o comportamento esperado. Verifica-se na Figura 5 este paralelismo entre TDD e BDD.

North (2012) observa que BDD pode ser o mesmo que TDD quando se tem apenas programadores no time, mas quando se tem uma audiência mais ampla, tais como testadores, gerentes de projetos, entre outros envolvidos no projeto, o BDD se torna a comunicação entre todas essas partes interessadas para criar uma visão única e coerente do projeto no qual todos possam entender o objetivo do teste. Desta forma, cria-se uma documentação que sofre modificações juntamente com o desenvolvimento do software.

Figura 5: Ciclo de TDD e BDD



Fonte: Adaptado de Rahman e Goa (2015)

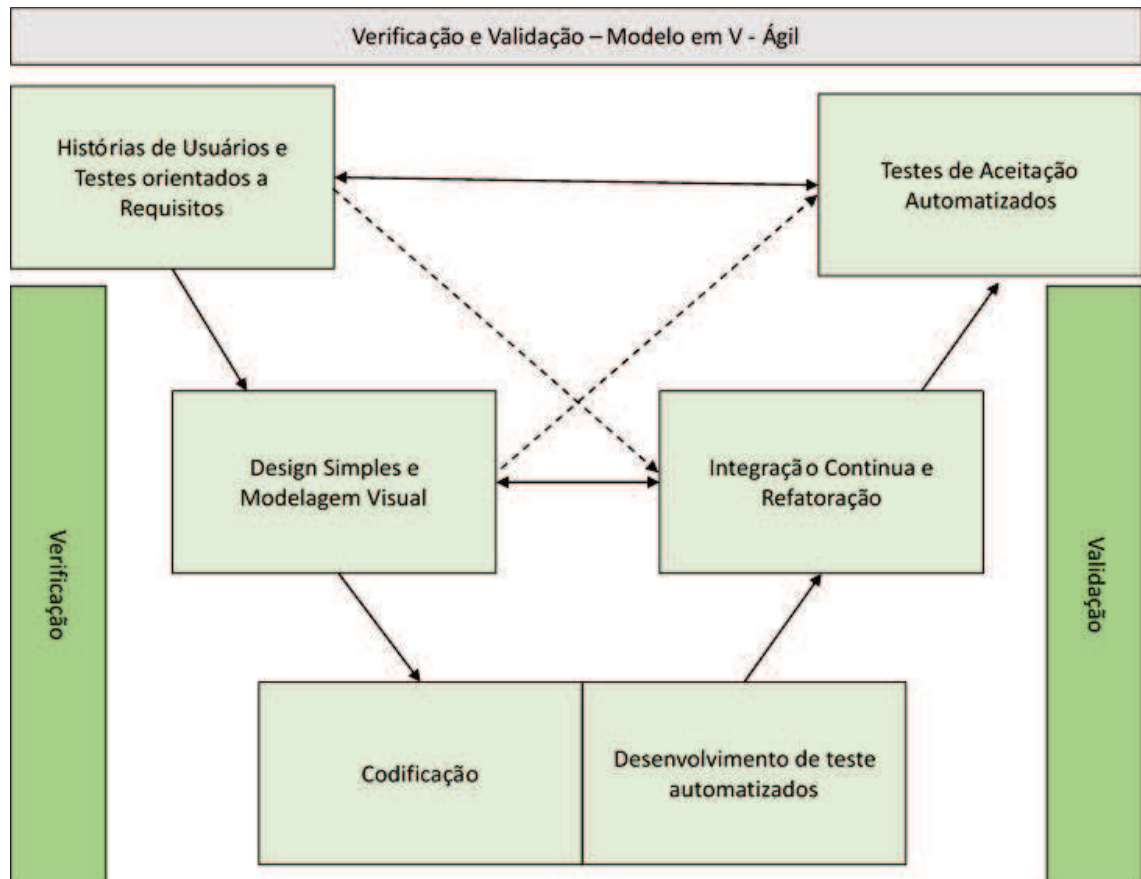
Uma vez que a equipe define uma estratégia de teste em ágil, é importante como estes serão executados dentro de uma iteração, visto que apesar de ter abordagens de testes para métodos ágeis, muitas vezes se utiliza as mesmas técnicas e práticas que o teste de software tradicional. As seções seguintes irão mostrar como o teste ágil lida com estes conceitos.

### 3.7.2 Níveis de testes em métodos ágeis

Os níveis de teste são atividades de teste logicamente relacionadas entre si, muitas vezes pela maturidade ou pela integridade do item de teste. Em modelos de ciclo de vida sequenciais, os níveis de teste são muitas vezes definidos de tal forma que os critérios de saída de um nível fazem parte dos critérios de entrada para o próximo nível. Em alguns modelos iterativos, esta regra não se aplica e os níveis de teste se sobrepõem (ISTQB, 2014).

Expedith (2012) apresenta uma abordagem para a utilização dos níveis de testes, fazendo uma adaptação do modelo em “V” e contemplando alguns níveis de testes mais utilizados em ágil.

Figura 6: Modelo em “V” para métodos ágeis



Fonte: Adaptado de Expedith (2012) – *Agile Testing: Key Points for Unlearning*

Observa-se na Figura 6 que a validação e a verificação do software são feitas por meio de práticas ágeis, tais como a utilização de testes de aceitação para validar as histórias de usuários, as práticas de integração contínua para validar o desenvolvimento iterativo e o desenvolvimento de testes automatizados para a validação do código.

Veenendaal (2010) observa que muitos autores estão focados apenas nos testes de unidade e de aceitação em ágil, mas destaca que os testes de integração e de sistema são de extrema importância para a detecção de defeitos. Ele observa que, em geral, os testes de unidade descobrem apenas cerca de 30% a 40% dos defeitos, enquanto os testes de aceitação são utilizados como orientação para a validação do cliente.

Uma vez que se define como o teste de software lida com as fases de testes, torna-se importante verificar como os tipos de testes são executados dentro das iterações e a separação lógica quanto ao objetivo de teste.

### *3.7.3 Tipos de testes em métodos ágeis*

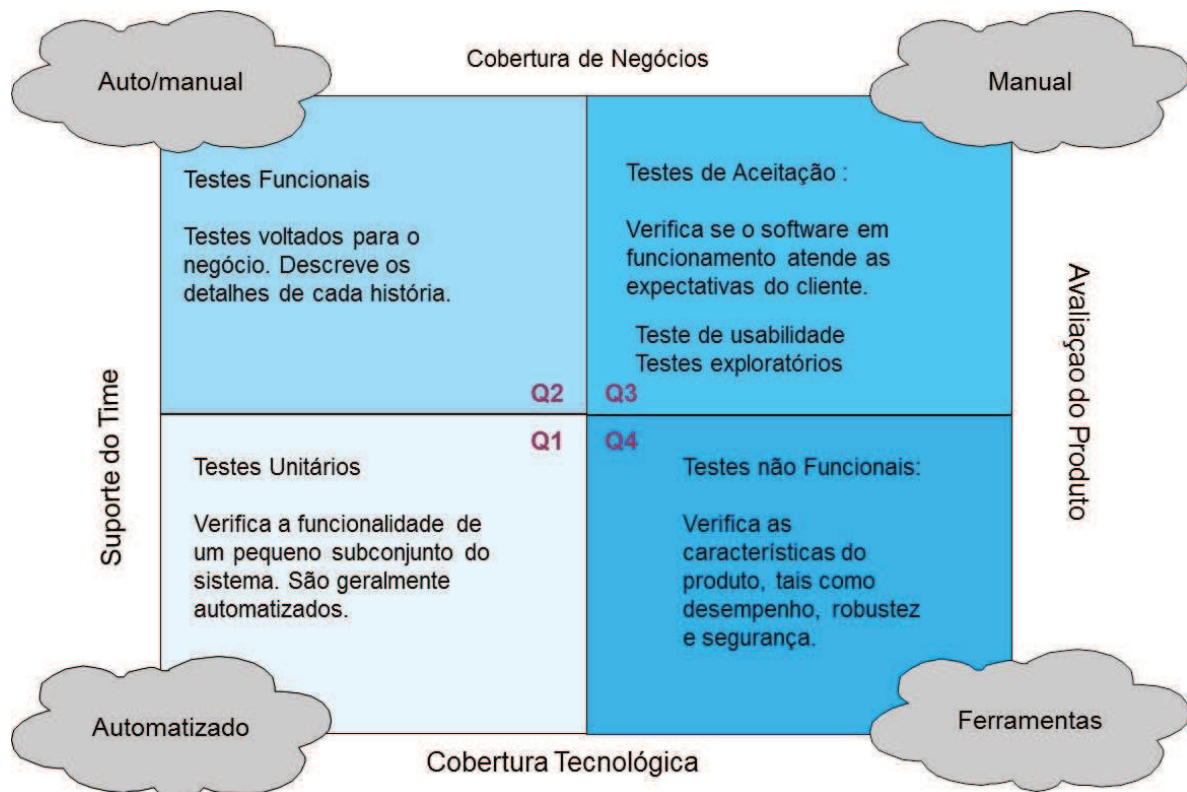
Veenendaal (2010) destaca que nem sempre é possível executar todos os tipos de testes dentro de uma iteração. Por exemplo, os testes de performance levam em geral mais de quatro semanas para serem executados. Observando que estes tipos de teste não podem ser executados dentro de uma iteração, o que torna difícil alinhar as práticas de testes dentro e fora das iterações.

Já Expedith (2012) observa que, apesar de muitas vezes não conseguir fazer testes de aspectos não funcionais dentro de uma iteração, é possível criar uma iteração separada para estes tipos de testes e também realizar um ciclo de testes de aceitação para garantir que o sistema continue funcionando depois do conserto de possíveis defeitos.

Crispin e Gregory (2009) fazem uma adaptação atualizada da proposta da divisão dos tipos de testes em quadrantes feita por Brian Marick, como uma forma de distribuir atividades de teste analisando a cobertura tecnológica e de negócios, o suporte do time e a avaliação do produto, demonstrando onde a automação é mais e menos utilizada, como pode-se observar na Figura 7.



Figura 7: Quadrante de teste ágil



Fonte: Adaptado de Crispin e Gregory – *Agile Testing*

No quadrante 1 (Q1) aparecem os testes de unidade que são, geralmente, automatizados. Estes testes ajudam o programador a entender o que o código precisa realmente fazer, sendo, portanto, um guia de projeto.

No quadrante 2 (Q2) aparecem os testes funcionais. Estes testes devem ser escritos para cada história antes de se começar a codificação, pois eles ajudam o time a entender o código que deve ser escrito. Seu principal objetivo é verificar se o software atingiu as expectativas de negócio, sendo as histórias utilizadas para desenvolvimento. Parte destes casos de testes deve ser automatizada e utilizada para a verificação contínua do software.

No quadrante 3 (Q3) aparece a verificação quanto ao fato de o software ter ou não atendido às necessidades dos clientes ou de mercado. Costuma-se fazer uma demonstração no final da iteração para que os clientes analisem o que está sendo entregue. Testes exploratórios e de usabilidade também são executados neste quadrante.

No quadrante 4 (Q4) aparecem as ferramentas para verificar requisitos não funcionais, tais como performance, segurança e robustez. Quando se está planejando as atividades, não se deve pensar apenas nas necessidades de negócio, mas também nos riscos que se pode ter em termos não funcionais, e incluir estes testes no planejamento.

Crispin e Gregory (2009) destacam, ainda, que quando os testes não funcionais não conseguem ser realizados dentro de uma iteração, conforme o quadrante citado, podem ser criadas histórias para estes testes e, assim, tais histórias podem ser priorizadas no *backlog* de acordo com a necessidade. As autoras citam a seguinte história como exemplo: “Eu, como usuária, preciso recuperar um relatório *X* em menos de 20 segundos e, assim, posso tomar uma decisão rapidamente”.

A utilização de ferramentas e diferentes abordagens para a execução desses tipos de testes se faz importante para cumprir os objetivos do desenvolvimento ágil. As próximas seções irão detalhar como estes testes podem ser executados, analisando o contexto de teste manuais e automatizados.

#### 3.7.4 Testes manuais em métodos ágeis

Os testes exploratórios são uma técnica bastante utilizada nos métodos ágeis e são importantes devido à limitação de tempo para a atividade de análise de teste e limitação de detalhe nas histórias (ISTQB, 2014). Watkins (2009) destaca que os testes exploratórios ajudam a reduzir a complexidade de planejamento e são bastante úteis em histórias de complexidade baixa.

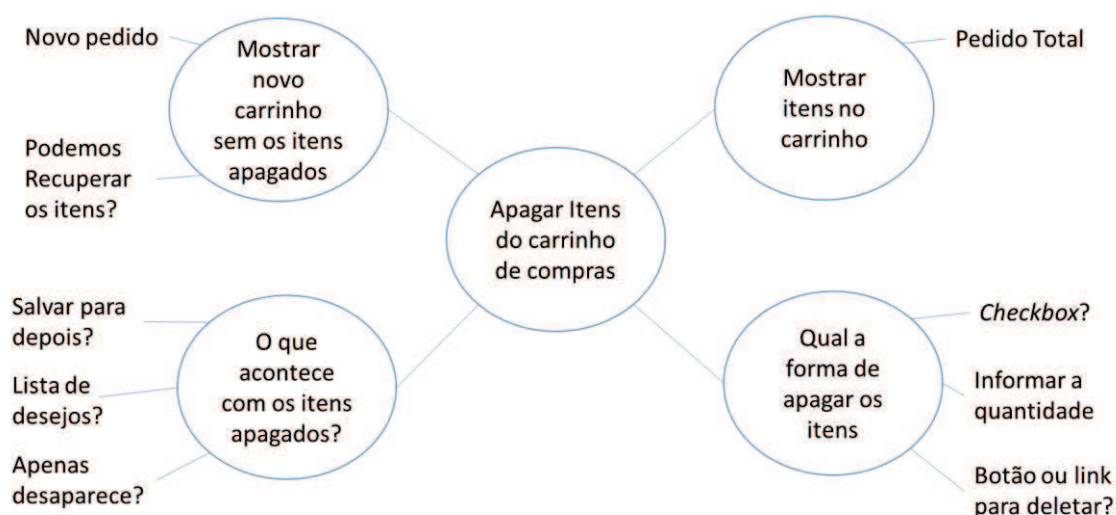
Segundo Veenendaal (2010), no contexto ágil, em geral os testes manuais são vistos, primariamente, como testes exploratórios. Neste sentido, são feitos testes através de um ciclo rápido de passos para o planejamento de testes, *design* e execução. Além de encontrar defeitos, estes testes podem identificar casos de testes que podem ser adicionados aos níveis de testes automatizados e descobrir ideias que poderiam estar faltando nas histórias de usuários.

Observa-se que os testes manuais são importantes principalmente em níveis mais elevados, tais como os testes de aceitação. Nos times ágeis, observa-se que as práticas de testes devem seguir os princípios ágeis e, para isso, priorizar os *backlogs* de testes. É importante que os testadores tenham conhecimento e domínio do sistema a ser testado e se adaptem ao princípio

da simplicidade, utilizando planilhas de trabalho e lista de objetivos de teste para priorizar os esforços de testes, deixando públicos todos os *backlogs* de teste para que exista transparência com o time (GERAS, 2010).

A utilização de mapas mentais, embora não seja uma técnica presente apenas em testes ágeis, é abordada como uma forma bastante útil nestes métodos para se descobrir ideias de testes. As autoras Crispin e Gregory (2009) mostram um exemplo de como utiliza-los para teste como um meio eficiente para a criação de testes no desenvolvimento ágil. Elas citam a seguinte história como exemplo: “Como um comprador em nosso site, eu quero deletar itens do meu carrinho de compras e, dessa forma, eu não preciso comprar itens extras que eu decidi não levar”. A partir desta proposta, as autoras mostram o exemplo de mapa mental representado na Figura 8.

Figura 8: Exemplo de mapa mental em teste de software



O exemplo dado na Figura 8 mostra as várias possibilidades que podem ser abordadas durante o teste de software. Esta abordagem contempla de forma rápida e abrangente vários aspectos do sistema que podem ser abordados durante o teste e, dessa forma, caracteriza-se como uma abordagem de teste útil em ágil.

Discute-se também a falta de documentação na execução de testes no ambiente ágil. Essa, no entanto, pode ser suprida pela própria dinâmica do desenvolvimento ágil que busca entregar valor frequentemente em um ritmo sustentável. O próprio teste automatizado pode ser um exemplo de documentação que é sempre atualizado para estar compatível com as mudanças

feitas no decorrer do processo de desenvolvimento. Em ambientes que requerem auditoria, técnicas de guardar evidências dos testes, tais como fotos das telas, podem ser utilizadas. Utilizar da organização do método ágil para que os *stakeholders* participem da solução e ajam como auditores é uma alternativa para a falta de documentação e eficiência da auditoria (CRISPIN e GREGORY, 2015).

### 3.7.5 Testes automatizados em métodos ágeis

A utilização de algumas ferramentas se faz necessária dentro do contexto ágil. Crispin e Gregory (2009) destacam que é muito difícil entregar valor em tempo hábil sem a utilização de ferramentas.

Pham e Pham (2011) abordam a importância de testes automatizados e testes de integração contínua, pois estes testes sustentam o objetivo de métodos ágeis, como o scrum, que é entregar um produto ao final de cada *sprint*. Com testes automatizados, é possível fazer testes de regressão mais facilmente quando novas funcionalidades são implementadas e os testes de integração permitem verificar que a nova funcionalidade está funcionando juntamente com o todo. Delamaro *et al.* (2009) analisaram que a automação de testes é uma atividade recorrente e bastante utilizada no desenvolvimento ágil.

Destaca-se ainda a relevância da automação nos resultados obtidos por Shaye (2008) que, em uma implantação ágil, utilizou a automação dos testes para validação de *builds* automáticos e diminuiu em 67% o custo do teste manual. Observou-se que os recursos humanos teriam que fazer testes manuais caso não houvesse a automação de testes, porém, neste caso eles puderam se focar em técnicas mais avançadas de testes.

Como visto anteriormente, várias práticas são utilizadas para garantir que o teste de software esteja adequado aos princípios ágeis. Nas equipes ágeis, em geral, se tem pessoas multidisciplinares, que procuram testar o software durante toda a iteração. Na próxima seção será descrito como os testadores estão integrados dentro destas equipes.

### 3.7.6 O papel do testador dentro da equipe ágil

Muitas abordagens são utilizadas para definir a constituição de uma equipe de desenvolvimento ágil. Alguns autores destacam a importância do trabalho em equipe e ainda o papel do testador dentro deste contexto.

Cockburn e Highsmith (2001) observam que as equipes ágeis focam nas competências individuais como fator crítico de sucesso. Destacam ainda a importância do trabalho em equipe, no qual as pessoas que trabalham em conjunto com boa comunicação e interação podem operar em níveis mais elevados do que quando eles usam seus talentos individuais. Observa-se também que a participação dos *stakeholders* é importante neste processo. Estes devem estar ativamente envolvidos com testes de aceitação durante todo o ciclo de desenvolvimento (AMBLER, 2014).

Crispin e Gregory (2009) verificam que a comunicação é fator crítico para o sucesso da implementação ágil. Por isso, é importante a integração de testadores e desenvolvedores em um mesmo time, pois isso faz com que todos os membros da equipe estejam preocupados e engajados com a qualidade do software. Observa-se que esta integração é de fundamental importância, mas que existem razões para se manter o time de teste separado da equipe, tais como a independência para verificar e auditar o software, o fornecimento de uma visão imparcial e externa, a manutenção do ponto de vista do cliente sem intervenção do ponto de vista do desenvolvedor e o fornecimento de reporte para pessoas diferentes, o que ajuda na entrega de um código testado de fato.

Carter (2010) destaca o papel do testador. Para o autor, este deve ser incorporado ao time de desenvolvimento ágil. Ele destaca que o engenheiro de qualidade deve desenvolver habilidades em testes exploratórios, automação de testes, além de se adaptar à dinâmica dos times ágeis e aos métodos de comunicação. Segundo Carter (2010), apesar da visão de que qualquer pessoa pode testar e que um recurso dedicado ao teste pode não ser uma estratégia valiosa, a presença do testador nas equipes ágeis é bastante importante, pois este preserva os benefícios trazidos pelos times de garantia de qualidade em times de desenvolvimento iterativo.

O testador nas equipes de desenvolvimento ágil ajuda provendo *feedback* durante todo o processo de desenvolvimento, ajuda ou tem conhecimento de nível de código do teste a ser realizado, assume a liderança nos testes de aceitação, de regressão, planos de testes e revela cenários de teste adicionais através de testes exploratórios. Além disso, o testador ágil garante que a cobertura de teste esteja adequada, lidera os esforços de automação, integração, testes de nível de sistema, mantém ambientes de teste e dados disponíveis, além de identificar problemas

de regressão e partes técnicas de teste. Os testadores também identificam a necessidade de testes adicionais que estão fora do escopo do planejamento da história, tais como performance, que muitas vezes são executados em histórias orientadas para o teste (CARTER, 2010).

Veenendaal (2010) também observa a importância das habilidades do testador. Para o autor, o testador em uma estratégia de desenvolvimento ágil precisa ter muitas habilidades que são compatíveis com as habilidades de um testador sênior com boa capacidade de comunicação. Os testadores devem estar aptos a fazer testes técnicos, testes sobre o negócio e gerenciamento de problema. O autor destaca que o testador com menos experiência pode não estar apto para trabalhar em equipes ágeis.

Myers *et al.* (2011) destacam que os testadores em métodos ágeis devem ser colaborativos para que todos estejam envolvidos no processo, o que exige esforços de comunicação e colaboração. Observa que os clientes devem ajudar nos testes de aceitação e os desenvolvedores devem colaborar na construção de códigos que podem ser facilmente automatizados. Isto é, todos na equipe devem estar envolvidos no processo de teste.

Utilizando-se da prerrogativa de que todos no time de desenvolvimento ágil devem trabalhar juntos em prol de um objetivo comum, o ISTQB (2014) ressalta que o testador deverá trabalhar com os desenvolvedores e representantes do negócio para atingir o nível de qualidade desejada. Isso inclui trabalhar com os desenvolvedores e os clientes para criar testes de aceitação adequados, trabalhar com os desenvolvedores para criar uma estratégia de teste e abordagens para a automação, além de transferir o conhecimento sobre teste e influenciar os membros da equipe com este conhecimento.

Broek *et al.* (2014), em um estudo de caso sobre a integração do teste dentro do processo de desenvolvimento ágil, verificaram que, em *sprints* nos quais não havia um profissional dedicado ao teste, foram encontrados menos defeitos do que em *sprints* com este profissional. O profissional de teste foi incorporado já no andamento do projeto e este mostrou, através da descoberta de defeitos, que a qualidade do software não foi adequadamente aferida antes de sua participação, o que fez com a equipe tivesse que utilizar alguns *sprints* para a correção de defeitos que já poderiam ter sido resolvidos anteriormente. Observando este contexto, os autores destacam a importância do testador na equipe durante todo o projeto e manutenção de uma equipe constante. Uma equipe constante não enfrenta problemas relacionados com a adaptação das pessoas na equipe e no projeto e é, portanto, mais capaz de aprender como uma equipe. Além disso, com uma equipe multifuncional que inclui um testador, a continuidade da

qualidade do produto é melhor assegurada porque os esforços de teste podem ser uniformemente distribuídos em todas os *sprints*.

No contexto ágil, observa-se que o fator humano é bastante requerido para o sucesso da implantação de métodos ágeis, destacando-se que ter pessoas especializadas em teste pode trazer grande melhoria de qualidade. Observa-se, também, que o trabalho em equipe e a comunicação são fatores que possibilitam às pessoas desenvolver softwares de qualidade.

Com o estudo das equipes de desenvolvimento, encerra-se a análise dos fatores que podem impactar no teste de software ágil. A seguir, será apresentado como as equipes podem medir e gerenciar as atividades de testes dentro do contexto ágil.

### 3.7.7 Indicadores de testes de software

Observa-se que as atividades de testes são importantes para a aferição da qualidade de software. Por isso, torna-se bastante relevante a utilização de indicadores. Para criar indicadores, deve-se criar medidas que indiquem extensão, quantidade, capacidade ou tamanho de algum atributo de um processo ou produto e, assim, desenvolver métricas, isto é, relacionar as medidas individuais (PRESSMAN, 2011).

Hartmann e Dymond (2006) destacam que métricas inadequadas podem desperdiçar recursos e distorcer a cultura inerente aos métodos ágeis e, por isso, apontam que as métricas em desenvolvimento ágil devem afirmar e reforçar os princípios de *lean* e ágil, isto é, elas devem estar focadas nos clientes, sem gerar desperdício. Para os autores, deve-se medir o resultado e não apenas as saídas, isto é, os resultados dos valores entregues ao cliente. Deve-se, também, seguir tendências e não apenas números. As métricas precisam responder a uma questão em particular para uma determinada pessoa, isto é, ela precisa servir a um propósito real. É importante ressaltar que estas métricas devem pertencer a um conjunto pequeno de métricas e diagnósticos, pois o excesso de importação pode esconder tendências importantes. Estas métricas devem ser coletadas de maneira fácil e revelar seu contexto e variáveis para que estas sejam significativas ao time de desenvolvimento, promovendo *feedback* e possibilitando o aprendizado. Os autores observam também que a qualidade deve ser boa suficiente para o cliente ou para o negócio.

Crispin e Gregory (2009) destacam que em ágil é importante que o time saiba qual o propósito das métricas a serem coletadas. Independente do fator que o time decidir medir, deve



fazê-lo de maneira simples. As autoras observam também que o time não precisa continuar coletando e usando métricas quando estas não são mais úteis no processo.

Baseado nos princípios apresentados anteriormente e nas práticas mais utilizadas em métodos ágeis, Vicente (2010) seleciona uma lista de métricas de teste de software em métodos ágeis, dentre os quais se destaca:

- A cobertura de código a fim de atingir a cobertura de código desejada é verificada.
- O fator de teste que serve para indicar a relação entre o tamanho do código de testes e o tamanho do código em produção com o objetivo de verificar a evolução do código de teste e o esforço para criação de testes.
- A quantidade de casos de teste e assertivas para mensurar a produção.
- A porcentagem de assertivas de teste de unidade passando e falhando.
- A quantidade de testes de aceitação por funcionalidade.
- A porcentagem de assertivas de teste de aceitação passando e falhando.
- As funcionalidades testadas e entregues com o objetivo de medir a quantidade de valor de negócio entregue em cada funcionalidade do sistema, sob o ponto de vista do cliente.
- O tempo de execução.
- A quantidade de defeitos encontrados.

Crispin e Gregory (2009) também fazem um levantamento de métricas úteis com base em experiência. A partir deste levantamento, as autoras destacam os seguintes indicadores:

- A execução de testes por história.
- O status de automação.
- O número de testes passando e falhando.
- Os defeitos.

Com o estudo dos indicadores de teste, verificou-se como criar métricas para teste de software ágil e, assim, avaliar o processo de desenvolvimento de software. A seguir, serão



apresentados alguns relatos de experiência encontrados na literatura para que se possa dimensionar os problemas e benefícios que as equipes têm ao adotar métodos ágeis sob a perspectiva do teste de software.

### *3.7.8 Relatos de experiências em testes ágeis*

Cavalcante *et al.* (2011) relatam uma experiência da implementação de testes em ambientes ágeis de desenvolvimento utilizando Scrum. Entre as lições aprendidas nesta implementação, destacam-se que, em equipes de desenvolvimento ágil, é de extrema importância que se encare a equipe toda como um único time, na qual todos são responsáveis pelo software, e que ter o time alocado no mesmo ambiente ajuda na comunicação, na cooperação para execução de tarefas e mantém todos informados sobre o contexto de cada *sprint*. Os autores observam também a importância de se ter uma estratégia de teste para que todos entendam claramente todas as atividades de testes, facilitando, assim, a colaboração de membros da equipe.

Cavalcante *et al.* (2011) destacam ainda a adoção de integração contínua, que forçou a implementação de testes de unidade para todos os *builds*, e a automação de testes funcionais, que reduziu o tempo de execução de testes e possibilitou que as equipes ficassem focadas na descoberta e correção de defeitos. A cooperação de testadores e desenvolvedores na criação de testes automatizados fez com que se pudesse acomodar as mudanças no software com mais confiança, além de incentivar os testadores a contribuir com uma visão diferente sobre o planejamento e o projeto.

Um dos problemas enfrentados foi a maneira de se lidar com os defeitos encontrados. Muitas vezes, no começo da implementação ágil, não se encontrava tempo para consertar e testar todos os defeitos dentro de uma iteração. Com o amadurecimento do processo, foi possível fazer com que todos os defeitos encontrados no desenvolvimento de uma história pudessem ser resolvidos, e aqueles de menor impacto pudessem ser adiados para iterações seguintes.

Observou-se também que foi necessário dar ao time a oportunidade de cometer erros. Uma vez que a equipe percebe que algo está errado no processo, a equipe deve ter a oportunidade de discutir os problemas e possíveis soluções. Deve-se ter cuidado especial com

novos times para que se aloque tempo necessário para o teste, pois uma história só é considerada pronta depois de testada.

Collins e Lucena (2012) trazem uma contribuição no que se refere a práticas de automação em ambientes ágeis. Através de uma pesquisa de estudo de caso com diversos projetos e práticas de automação, chegou-se às conclusões relatadas a seguir:

- A colaboração é um fator essencial para o sucesso da automação de testes e projetos ágeis: o envolvimento dos testadores nas revisões de *backlog*, projetos de interface e configuração de ambientes foram importantes para evitar a separação de testadores e desenvolvedores, mantendo a unidade da cooperação no projeto. Quando possível, a equipe de teste pode contar com a ajuda dos desenvolvedores para resolver problemas de automação, e a equipe de desenvolvimento pode contar com a ajuda dos testadores para a revisão de testes de unidade a fim de se atingir a cobertura ideal.
- Encontrar a ferramenta que se adapte às estratégias de testes e ao método ágil foi um fator que impactou na implantação ágil. Verificou-se que o projeto que escolheu as ferramentas de testes mais adequadas teve mais sucesso, enquanto o outro projeto teve que trocar de ferramenta ao final do *sprint* a fim de encontrar uma ferramenta que se adequasse às necessidades do projeto, tendo maiores problemas para finalizar a iteração com o software testado.
- A estratégia de automação foi analisada como fator de impacto, no qual verificou-se que o time que tentou automatizar tudo não conseguiu testar outras funcionalidades do sistema que teriam agregado mais valor ao projeto.
- A automação de testes deve ser simples e as atividades de automação devem estar acessíveis para todos os membros da equipe. Por isso, não se deve concentrar em ferramentas complexas, em que o conhecimento pertence apenas a alguns membros da equipe. Observou-se que o time que implantou processos mais simples pode contar com a contribuição tanto de testadores quanto de desenvolvedores para criar e executar testes de integração.
- Deve priorizar a execução de testes de regressão. É importante executar primeiro os testes que têm mais probabilidade de falhar. Na experiência, verificou-se que depois de várias vezes executando os mesmos testes de regressão, não se notou o reaparecimento de defeitos.

- Recomenda-se executar testes não funcionais no começo. Nesta experiência, analisou-se como as equipes executaram testes não funcionais, no qual verificou-se que a equipe que executou os testes de segurança e estresse no final do projeto enfrentou vários problemas que acarretaram no atraso do projeto.
- Utilização de testes automatizados para documentação e informação de *feedback*. Verificou-se que as ferramentas de testes serviram para controlar algumas atividades de testes, especificações, ambientes de testes e rastreamento de defeitos.

As experiências citadas mostram vários fatores que podem influenciar na implementação de testes ágeis. Nota-se que garantir um software testado requer bastante empenho das equipes na adequação de processos para a execução do teste de software, tanto manual quanto automaticamente.

Com a apresentação destes relatos de experiência finaliza-se a fundamentação teórica. O próximo capítulo aborda os aspectos relacionados a pesquisa realizada neste trabalho.

## 4 PESQUISA

A pesquisa desenvolvida tem abordagem quantitativa. Para alcançar o objetivo geral do trabalho, foi realizada uma pesquisa descritiva por meio de levantamento bibliográfico inicial. Em caráter exploratório, foi feito um levantamento pelo pesquisador de evidências sobre o processo de teste de software por sua experiência em um laboratório de desenvolvimento de software de uma grande empresa multinacional da área de tecnologia da informação, que desenvolve produtos de gestão de acordo com as necessidades do mercado. A partir dos elementos encontrados nas pesquisas descritas anteriormente, foi construído um questionário, cuja aplicação busca identificar como atividades de teste têm sido implementadas em equipes de desenvolvimento, que utilizam métodos ágeis para a produção de software, assim como as principais estratégias utilizadas nas atividades de testes de software e como atuam as equipes de desenvolvimento nos diversos portes de empresas sob o ponto de vista do desenvolvedor de sistemas.

Para definição da amostra, utilizaram-se os seguintes grupos de interesse em desenvolvimento ágil e teste de software para divulgação da pesquisa:

- Agile Floripa (<https://www.facebook.com/AgileFloripa/?fref=ts>),
- Agile Momentun (<https://www.facebook.com/agilemomentum/?fref=ts>),
- Agile Testers (<http://agiletesters.com.br/>),
- Agile Trend (<https://www.facebook.com/AgileTrends/?fref=ts>),
- Agilidade Recife (<https://www.facebook.com/AgilidadeRecife/?fref=ts>),
- Caipira Ágil (<https://www.facebook.com/Caipira-%C3%81gil-355971547755488/?fref=ts>),
- Fábrica de Testes (<https://www.linkedin.com/groups/3733631>),
- Qualidade de Software (<https://www.linkedin.com/groups/3125862>),
- Quality Applied (<https://www.facebook.com/groups/336893809728742/?ref=browser>),
- Testadores (<https://www.linkedin.com/groups/2897468>),
- Testes e Qualidade de Software (<https://www.facebook.com/groups/testesqa/?ref=browser>).

A pesquisa foi compartilhada na rede profissional do LinkedIn do pesquisador. Ela foi

enviada para 40 pessoas da rede do pesquisador, juntamente com um pedido para compartilhamento da pesquisa. O pesquisador também utilizou o LinkedIn para enviar a pesquisa para pessoas que não participavam diretamente de sua rede. Para isso, utilizou-se como filtro “pessoas com habilidade em desenvolvimento ágil” na localidade “Brasil”. Um convite para fazer parte da rede do pesquisador foi enviado para 300 pessoas, das quais 72 aceitaram o convite. A pesquisa ficou disponível de 26/11/2015 a 08/01/2016, e 167 pessoas responderam a pesquisa, sendo que 149 pessoas possuíam experiência em times de desenvolvimento ágil e essas respostas foram consideradas válidas.

A pesquisa foi de caráter transversal, pois não foram avaliadas tendências ou mudanças temporais. Foram coletados dados de empresas que utilizam métodos ágeis. O tratamento de dados se deu por meio de técnicas estatísticas descritivas, em sua maioria, nas quais a análise e interpretação dos dados levou em consideração o ambiente em que são coletados os dados, bem como as formas de controle das variáveis envolvidas utilizando a caracterização dos respondentes e forma como eles utilizam o teste de software.

Inicialmente foi feita uma análise estatística descritiva, no qual buscou verificar em porcentagem o perfil dos participantes da amostra e quais as práticas utilizadas por eles no teste de software em métodos ágeis. Posteriormente, buscou-se correlacionar os dados para verificar se diferentes características dos respondentes impactavam na forma de selecionar práticas. Valores numéricos foram atribuídos às variáveis do questionário de forma que pudesse verificar a execução ou não de uma prática de acordo com a característica analisada.

Utilizou-se o conceito de correlação de postos de Spearman para calcular o coeficiente de correlação entre as variáveis mensuradas, que é uma medida de correlação não-paramétrica que avalia a relação entre duas variáveis, sem fazer suposições sobre a distribuição de frequências destas variáveis.

O coeficiente de correlação por postos de Spearman pode variar de  $-1,00$  a  $+1,00$ . O valor do coeficiente calculado explica o grau de relacionamento entre as variáveis e a existência de consistência nesse relacionamento. Um coeficiente próximo a  $1,00$  ou  $-1,00$  indica forte grau de relacionamento entre os dois conjuntos de dados. Um coeficiente próximo de zero indica não haver grau de correlação significativa (BRYMAN; BELL, 2003).

Para as análises onde foram avaliadas duas variáveis qualitativas (categóricas ou ordinais), e era necessário verificar se havia ou não alguma associação entre elas, foi usado o coeficiente de Yule, voltado para variáveis binárias, qualitativas com apenas duas categorias mutuamente exclusivas.

Observa-se que as correlações de dados não trouxeram valores com alta significância e

por isso utilizou-se para exploração dos dados a ferramenta de análise de dados da IBM, *Watson Analytics*, a ferramenta é baseada em um sistema de computação cognitiva que permite a utilização de linguagem natural para a consulta de dados (IBM, 2014).

#### **4.1 Levantamento de evidências do processo de teste de software em métodos ágeis**

O levantamento baseado na experiência do pesquisador na implementação de métodos ágeis é relatado em dois aspectos. Primeiramente, analisa-se como o processo foi organizado para se adequar aos princípios ágeis, baseando-se no *framework* scrum para a organização e enfatizando a visão do teste de software neste processo geral. Logo após, são analisadas as práticas utilizadas no teste de software neste contexto, observando-se quais as vantagens e desvantagens de utilizá-las em um ambiente ágil.

A fim de verificar externamente o processo de teste de software e a visão de outros profissionais, um questionário foi elaborado a partir da pesquisa bibliográfica e da experiência do pesquisador relatada neste trabalho.

##### *4.1.1 Levantamento baseado na experiência do desenvolvimento ágil*

Runeson e Höst (2009) observam que o paradigma de pesquisas analíticas nem sempre é suficiente para investigar problemas complexos da vida real, os quais envolvem os seres humanos e suas interações com a tecnologia. Por isso, a utilização de relatos em engenharia de software se torna importante. É nesse sentido que se busca captar as práticas utilizadas, assim como os desafios e vantagens dos métodos ágeis para uma equipe de testes de software de forma a poder realizar pesquisas em outras experiências na implementação de métodos ágeis com equipes de testes de software.

Este relato é baseado na experiência de implantação de métodos ágeis em uma grande empresa de desenvolvimento de software. A empresa já vem utilizando métodos ágeis para desenvolvimento por cerca de cinco anos e muito pode ser observado sobre as práticas de testes utilizadas neste ambiente.

A equipe de desenvolvimento é composta em média por três desenvolvedores, dois testadores e um redator técnico. Existe, também, uma equipe dedicada apenas à execução e ao suporte à automação de testes. Todos os membros da equipe são qualificados em engenharia de software e muitos dos membros possuem certificações, tanto na área de desenvolvimento, quanto na área de teste de software. A grande maioria do time está na empresa desde a implantação dos métodos ágeis e são engenheiros seniores. A equipe também conta com participantes juniores que têm se adaptado aos métodos ágeis e aos processos da empresa.

O software desenvolvido pela empresa é um produto de mercado para gestão de ativos de TI. Novas versões do produto e pacotes com correções do sistema são lançados durante o decorrer do ano, de acordo com as necessidades de mercado. Existe um *feedback* do software ao final de todos os *sprints*, mas o software é entregue de uma só vez com todas as funcionalidades novas selecionadas para aquela versão ou pacote com correções.

#### 4.1.2 Visão geral da implantação do método ágil

O processo de implantação de métodos ágeis foi inicializado com *sprints* de quatro semanas, em que se desenvolvia testes para cada funcionalidade selecionada para entrega ao final do *sprint*. Durante as fases iniciais do *sprint*, os testadores trabalhavam na construção de ambientes para teste e desenvolviam casos de testes. Após a entrega do software pelos desenvolvedores, todos os casos de testes eram executados e buscava-se automatizar estes testes. Ao final de todo *sprint*, uma reunião era feita para apresentar o software desenvolvido e obter *feedback*. A demonstração ajudou na melhoria da qualidade do software. Estas reuniões trouxeram ao time novas formas de avaliar o software com a ajuda de especialistas do mercado, e que permitiram ao time compreender melhor o software e as necessidades dos clientes.

Na primeira fase, transição, foram dedicados, ao final de todos os *sprints*, alguns meses para a fase de teste de sistema e de integração, na qual o sistema era testado com a utilização de casos de testes mais complexos e todas as funcionalidades do sistema eram testadas de maneira integrada. Os meses dedicados apenas aos testes foram importantes para a transição. Neste momento, o time ainda não tinha maturidade para automatizar todos os testes e estava se adaptando ao processo. A execução serviu, inclusive, para verificar se os testes dentro dos *sprints* estavam sendo suficientes para garantir uma cobertura de testes adequada ao sistema.

Como forma de aprimorar os testes de cobertura, o time foi solicitado a fazer esforços quanto à automação de casos de testes.

Para dar suporte às equipes de testadores que automatizavam as funcionalidades que estavam sendo desenvolvidas dentro de um *sprint*, uma equipe de automação de testes foi criada para desenvolver e dar suporte a um *framework* de automação. A criação desta equipe trouxe muitos benefícios ao time de teste, pois este pode contar com ajuda de especialistas em automação durante todo o processo de desenvolvimento, além de ter uma equipe para analisar constantemente os resultados de testes antes do *build* ser testado pela equipe, mantendo todos informados sobre os resultados sem desperdício quanto à análise de resultados semelhantes por outros membros.

Umas das dificuldades encontradas durante a finalização dos *sprints* era depurar todos os defeitos descobertos. Por isso, com o tempo, tornou-se prática priorizar os defeitos que impactavam diretamente o funcionamento do sistema e postergar os menos importantes para o começo dos *sprints* posteriores. A triagem de defeitos ajudou para diminuir a sobrecarga de trabalho ao término dos *sprints*.

Na segunda fase de transição, houve a redução dos *sprints* para duas semanas. Os testes de sistemas ao final de todos os *sprints* foram suprimidos, uma vez que já existia uma grande parte de testes cobertos pelos testes automatizados. Os casos de testes também foram simplificados a fim de otimizar o tempo para teste e o controle, que passou a ser feito juntamente com as histórias dos *sprints* e a execução de testes automatizados. Em alguns casos, profissionais com habilidades de codificar casos de testes passaram a criar seus artefatos através de código, utilizando ferramentas e *frameworks* específicos de acordo com o escopo do software.

Houve a tentativa de utilizar *sprints* de apenas uma semana. No entanto, este tempo foi muito pequeno para o time que tinha que se dedicar ao planejamento e ao *feedback*, sobrando pouco tempo para o desenvolvimento e, praticamente, impossibilitando a execução de testes mais complexos dentro dos *sprints*. Por este motivo, optou-se por *sprints* de duas semanas.



#### 4.1.3 Implementação das atividades de teste

No ambiente analisado, foram utilizados os seguintes níveis de testes de forma constante: de unidade, componentes e testes de sistemas. A empresa utiliza-se de terminologia diferente da utilizada na literatura. Os testes de unidade eram criados pela equipe de desenvolvimento com auxílio, muitas vezes, da equipe de automação. Os testes de componentes eram executados para verificar cada funcionalidade criada durante o *sprint* e os testes de sistemas, para verificar a funcionalidade juntamente com outros componentes do software. Não foram executados testes de aceitação de usuários, visto que o software é desenvolvido para o mercado. No entanto, as reuniões de revisão foram bastante úteis para validação do software.

Uma equipe de teste de integração dedicou-se para verificar a integração do sistema desenvolvido com outros softwares da mesma linha de produtos. O foco da equipe foi de analisar a integração do software com outros produtos, e não com os componentes do mesmo produto. Essa equipe não trabalhava de forma conjunta com o time de desenvolvimento e possuía um planejamento próprio para o teste de integração. A separação da equipe para testes de integração se fez necessária, pois a integração de vários outros produtos era testada continuamente pela mesma equipe e o time de desenvolvimento, que estava focado em novas funcionalidades, não tinha capacidade para execução de todos os testes necessários.

Observa-se que a utilização de testes dos níveis de testes da forma como foi organizada, fazia com que os testes fossem executados ao final dos *sprints* e, por isso, nem sempre se conseguia terminar um *sprint* com todos os defeitos testados. No entanto, a priorização de defeitos era feita para postergar para próximos *sprints* os defeitos de menor impacto para a aplicação. Devido à complexidade do software quanto a integração com outros produtos, também não foi possível manter todos os testes sendo executados pelo mesmo time de desenvolvimento.

A análise das práticas de teste utilizadas é feita a partir das evidências das práticas de uso dos testes automatizados e manuais. No ambiente analisado, a implantação de testes automatizados se deu em dois níveis: nos testes de unidade e nos de UI (interface com usuário). Diferentemente do que é encontrado na literatura (COHN, 2013), optou-se por não se ter o nível intermediário, em que os testes funcionais na camada de API (interface de programação de aplicação) são usados para conseguir executar testes de verificação sem os gastos para a criação e manutenção quando se utiliza a camada de UI.

Os testes de unidade foram criados pelos desenvolvedores do software. Estes testes foram criados na mesma linguagem de desenvolvimento do software. Manter os desenvolvedores encarregados destes testes se tornou bastante eficaz, já que eles conheciam o código e podiam fazer esta automação de maneira mais rápida. Não houve a participação de engenheiros de teste na criação, especificação ou planejamento destes testes.

A automação de testes foi realizada por todos os membros durante o desenvolvimento de novas versões. Isto é, após a criação de novas funcionalidades, novos testes automatizados eram criados no nível de UI. Um *framework* foi criado para que todos seguissem o mesmo padrão e desenvolvessem testes utilizando técnicas de programação que ajudaram na reutilização de código e no melhor aproveitamento de estruturas de classes, tais como definidas na programação orientada a objeto.

Nos pacotes com correções do sistema, não houve a automação das alterações, pois as exceções corrigidas não trariam vantagens ao time na execução de testes de regressão em relação à cobertura de novas funcionalidades. Nestes pacotes, priorizou-se a execução dos testes automatizados de regressão para aferir que nenhuma correção pudesse quebrar o software que já estava em funcionamento.

A fim de validar o software a cada novo *build* (versão compilada deste software), foram criados testes de BVT (*Build Verification Test*). Estes testes executavam a instalação de um *build* automaticamente e também um conjunto de casos de testes automatizados que asseguravam que as funcionalidades básicas estavam funcionando corretamente. Os testes de BVT foram escalados para serem executados ao final do lançamento de cada novo *build*, e estes eram de responsabilidade do time de automação de testes. Com esta prática, o time sempre garantiu que as funcionalidades básicas para execução do sistema não estivessem comprometidas.

Todos os testadores deveriam criar testes automatizados das funcionalidades por eles testadas em um *sprint*, e a equipe de automação deveria dar apenas o suporte necessário para esta criação. Em alguns momentos, no entanto, por falta de tempo ou conhecimento técnico dos testadores, alguns testes foram automatizados pela própria equipe de automação.

A execução de testes de unidade e de BVT foi importante para garantir a confiabilidade do software durante os *sprints*. Já os testes de regressão automatizados, por utilizarem a camada de UI, levavam mais tempo para execução e análise de logs. O exercício dos testes automatizados de forma integral, mesmo utilizando-se várias máquinas virtuais e com execução

de vários testes simultaneamente, não era viável dentro do *sprint*, pois a análise dos logs se dava de maneira manual e não havia tempo para interpretação de todos os testes de regressão dentro do tempo destinado para um *sprint*. Por isso, optou-se pela execução em paralelo aos *sprints* com resultados ao final de um conjunto de *sprints*. Esta prática não comprometeu a qualidade final do produto, pois os outros testes, tais como o de BVT, puderam garantir que o sistema estivesse disponível para testes dentro dos *sprints*, e trouxe um conjunto de testes automatizados que puderam verificar o software de forma ampla durante a regressão, validando o sistema em três bancos de dados de forma contínua.

Os testes manuais foram utilizados para verificar as novas funcionalidades desenvolvidas durante o *sprint* e, posteriormente, realizar sua automatização. No início da implantação do desenvolvimento ágil, a documentação e a utilização de ferramentas de controle de execução de casos de testes foram mantidas, o que ajudou muito na mudança de paradigma. Posteriormente, a documentação dos testes passou a ser nas histórias, assim como o controle das atividades. Neste contexto, preocupou-se primeiramente com a definição de cenários de testes. As práticas de documentação e o controle nas histórias foram utilizadas tanto nas novas versões do produto quanto nos pacotes de atualizações.

Foram utilizadas ferramentas de apoio à gestão para a administração dos testes manuais. Com elas, foi possível acompanhar a evolução das tarefas de cada história, assim como as atividades de testes que estavam representadas no *sprint backlog*, que podia ser acessado por todos da equipe. Grande parte dos novos defeitos foi encontrada na execução de testes manuais, destacando, assim, a sua importância neste contexto, tanto para validação quanto para verificação do software. A execução manual dos testes permitiu que os testadores obtivessem melhor entendimento do software e pudessem criar testes automatizados a partir delas.

No ambiente analisado, havia a separação de papéis entre os desenvolvedores e os testadores. Neste ambiente, a validação e a verificação eram de responsabilidade dos testadores apenas. As reuniões diárias ajudavam o time a manter o status do projeto atualizado, e todos possuíam conhecimento do que estava sendo desenvolvido. A comunicação entre a equipe para entendimento das funcionalidades se dava por meio de troca de mensagens, e-mails e conferências, de acordo com a necessidade do testador para entendimento do software.

Para elaborar a estratégia de teste, o testador quem liderava as iniciativas de revisões de escopo e elaboração de testes que pudessem cobrir as funcionalidades desenvolvidas. A responsabilidade pela validação e verificação do software estava concentrada na equipe de testes. Assim como a definição de ambientes de testes, que permitiam utilizar diferentes

sistemas operacionais, banco de dados e servidores de aplicativos e desta forma cobrir diversos cenários e encontrar defeitos específicos para esses cenários. O time de desenvolvimento concentrava-se nas práticas de programação e elaboração de testes de unidade.

Para coletar métricas, a equipe de testes cadastrou os defeitos na ferramenta de controle e os classificou de acordo com a aplicação com defeito, a severidade, o impacto, a fase encontrada e outras informações que podem ser facilmente resgatadas para a elaboração de relatórios que possam mostrar várias informações estratégicas ao time.

## **4.2 Aplicação do formulário e validação**

Com base na revisão bibliográfica do referencial e o relato de experiência apresentado, foi construído um questionário. O objetivo deste questionário é verificar quais técnicas e estratégias de testes têm sido utilizadas no processo de teste de software quando utilizados métodos ágeis de desenvolvimento de software. O questionário inicia-se com a caracterização dos respondentes, empresas e equipes. Posteriormente, identifica-se através do questionário quais atividades são utilizadas por equipes ágeis e como estas equipes as implementam.

### *4.2.1 Abordagem no questionário sobre a caracterização dos respondentes*

Para caracterização dos respondentes, primeiramente, verificou-se o porte da empresa, utilizando a classificação de acordo com o SEBRAE (2013): microempresa (até 9 funcionários), pequena empresa (de 10 a 49 funcionários), média empresa (de 50 a 99 funcionários), grande empresa (acima de 99 funcionários).

Para saber qual a maturidade dos respondentes com método ágeis, verificou-se o tempo de experiência em desenvolvimento ágil utilizando a escala likert com cinco níveis: 0 a 1 ano, 1 a 3 anos, 3 a 5 anos, 5 a 10 anos, acima de 10 anos.

Com o intuito de verificar qual método ou abordagem de desenvolvimento os participantes utilizam, criou-se uma pergunta com a opção de selecionar os métodos abordados

na fundamentação teórica: Scrum, Dynamic Systems Development (DSDM), Feature Driven Development (FDD), Extreme Programming (XP), Crystal, Lean, Kanban e Outros.

#### 4.2.2 Abordagem no questionário sobre as atividades de teste

Observou-se na fundamentação teórica que existem várias estratégias de testes na literatura (PRESSMAN, 2011), e que a escolha da estratégia de teste é um fator importante para o sucesso dos esforços em teste (GRAHAM, 2008), justificando, assim, a preocupação dos times para a elaboração de uma estratégia de teste.

Verificou-se que, no contexto ágil, a estratégia de teste ágil deve refletir o modelo desenvolvimento (ISO/IEC/IEEE 29119-1). Segundo Crispin e Gregory (2009), os testadores devem participar no planejamento dos *sprints*. Os autores apontam os seguintes elementos de teste para o planejamento do *sprint*: criação e execução de testes das histórias, teste em pares com outros testadores e desenvolvedores, validação do negócio, automação de novos testes funcionais, execução de testes de regressão automatizados, execução de testes de carga, demonstração para os *stakeholders*.

Na experiência do pesquisador, destacou-se a importância de se testar os elementos da história, criar testes automatizados e ter um conjunto de testes automatizados que podem ser executados dentro de um *sprint* a fim de dar confiança ao time sobre as condições do software testado. Sendo, assim, verificou-se se os seguintes fatores são utilizados no âmbito de planejamento das atividades de teste: criação e execução de testes de histórias, testes em pares com outros testadores e desenvolvedores, validação do negócio, automação de novos testes funcionais, execução de testes de regressão automatizados, execução e automação de testes não funcionais, demonstração para os *stakeholders* e outros.

Verificou-se que, no contexto ágil, a estratégia de teste ágil deve refletir o modelo de desenvolvimento (ISO/IEC/IEEE 29119-1). Na experiência do pesquisador, foram feitas algumas experiências em relação ao tempo das iterações. Observou-se que uma semana não foi suficiente para codificar e testar todas as funcionalidades. A pergunta foi proposta para verificar se o tempo utilizado para codificação e teste interfere no processo de teste e também na cobertura de teste. Para isso, utilizou as seguintes opções de medição de tempo de uma iteração: 1 semana, 2 semanas, 3 semanas, 4 semanas, depende da complexidade das histórias. A fim de

validar o questionário, foi incluído um item para que os respondentes pudessem colocar seus comentários.

Observou-se na literatura que cada atividade de desenvolvimento é acompanhada por um nível diferente de teste. Estas são, geralmente, apresentadas no modelo em “V” (AMMANN; OFFUTT, 2008). Verifica-se que em alguns modelos iterativos, esta regra não se aplica, os níveis de teste se sobrepõem (ISTQB, 2014). Observa-se que várias abordagens podem ser utilizadas para aplicar este conceito, como o quadrante proposto por Crispin e Gregory (2009) ou a adaptação feita por Expedith (2012). Esta pergunta se faz necessária para investigar como as equipes verificam e validam cada atividade de desenvolvimento e se estas utilizam conceitos tradicionais de teste de software no contexto ágil.

Na experiência do pesquisador, verificou-se a utilização de testes de unidade, testes de componentes e de sistemas. A equipe também contou com uma equipe de teste de integração que tinha por objetivo verificar a integração do sistema desenvolvido com outros softwares da mesma linha de produtos. Desta forma, verificou-se se os respondentes utilizavam este conceito com as seguintes alternativas: não se aplica este conceito, executa-se os testes de unidade, executa-se os testes de integração, executa-se os testes de sistema, executa-se os testes de aceitação de usuário. No momento de validação, também foi incluído a opção para comentário para identificar a opinião dos respondentes sobre o assunto.

Observa-se que existe diversos tipos de testes, sendo que cada um deles está focado em um objetivo específico e pode ser executado dentro de qualquer nível (GRAHAM, 2008). Veenendaal (2010) destaca que nem sempre é possível executar todos os tipos de testes dentro de uma iteração. Na experiência do pesquisador, os testes funcionais se deram dentro do *sprint*, enquanto outra equipe foi responsável por testes não funcionais. A pergunta relativa a este assunto foi feita de forma descritiva no estudo piloto para que se pudesse obter os cenários utilizados pelos respondentes de forma mais detalhada. Posteriormente, ela foi dividida em duas questões uma para verificar em que momento a verificação não funcional é feita e outra para verificar a utilização de ferramentas nesse processo.

Muitas práticas podem ser utilizadas para a validação e verificação do software; destaca-se a importância dos seguintes fatores na literatura dentro deste contexto: Automação de testes [Shaye (2008); Crispin e Gregory (2009); Delamaro *et al.* (2009); Pham e Pham (2011); Collins e Lucena (2012); Cohn (2013)], Testes exploratórios [Veenendaal (2010); ISTQB, 2014], Testes de unidade [Crispin e Gregory (2009)], Testes de aceitação com os *stakeholders* [Crispin e Gregory (2009)], Boas práticas de desenvolvimento [Delamaro *et al.* (2009)]. Neste contexto,

buscou-se identificar quais fatores mais corroboram para a eficiência do teste de software. Na experiência do pesquisador, destacou-se a importância dos testes automatizados e de aceitação com os *stakeholders*. Sendo assim, perguntou-se aos respondentes quais práticas mais corroboravam para se ter um teste executado em ágil, utilizando as seguintes opções: automação de testes, testes exploratórios, testes de unidade, testes de aceitação com os *stakeholders* e boas práticas de desenvolvimento, também se deu a opção ao respondente inserir outras práticas.

Visto que a automação de testes é bastante recorrente em ambientes ágeis, buscou verificar qual a estratégia de automação utilizada pelos times de desenvolvimento ágil. Cohn (2013) apresenta uma estratégia de teste na qual se automatiza de forma mais ampla a camada de código, depois, a camada de serviço e, por último, a camada de interface de usuário. Destaca-se, também, a vantagem de se utilizar os testes automatizados para testes que são executados várias vezes Schwaber e Gilpin (2005), Alsmadi (2012) e Ramler e Wolfmaier (2006) destacam vários itens que devem ser analisados para a implantação de uma estratégia de automação eficiente. Na experiência do pesquisador, levou-se em conta, para automação de testes, os testes utilizados para regressão no nível de Interface de usuário.

Com objetivo de verificar a estratégia de automação, perguntou-se quais são os níveis da aplicação selecionados para automação: unitário (Automação de testes de unidade - código), serviço (Automação na camada de API) e UI (Automação na camada de UI). Para a validação do formulário, também, foi deixado em aberto comentários para que os respondentes pudessem dar *feedback* sobre a técnica utilizada para estratégia de automação.

Observa-se que muitas abordagens podem ser utilizadas para criação e execução de testes manuais, e que estas podem impactar quanto à adequação dos princípios ágeis. Observa-se que vários métodos podem ser aplicados em uma estratégia de *design* predefinido, e isso ajuda na otimização dos testes. Já os testes exploratórios são importantes devido à limitação de tempo e de detalhe nas histórias (ISTQB, 2014). Na experiência do pesquisador, inicialmente começou com a execução de testes com *design* predefinido e, posteriormente, passou para o exploratório. Observando o contexto de teste manuais verificou-se quais abordagens utilizadas em ágil para este teste, dando como a opção aos respondentes os seguintes itens: casos de testes com *design* predefinido, testes exploratórios sem planejamento (*freestyle*), testes exploratórios gerenciados - baseado em sessões ou outra abordagem.

Verifica-se que, em equipes ágeis, as competências individuais são fatores críticos para o sucesso do desenvolvimento do software (COCKBURN; HIGHSMITH, 2001). Por isso, estudar o fator humano neste contexto se torna importante. Destacando-se que o testador deve



estar incorporado ao time (CARTER, 2010). Na experiência do pesquisador, destaca-se a importância da competência do testador para se adaptar aos princípios ágeis e criar formas de criar testes que estejam alinhados com estes princípios. Para verificar o papel do testador dentro das equipes ágeis, verificou-se se existe uma separação de papéis entre os desenvolvedores e os testadores, se existem recursos dedicados apenas ao teste de software, se existe colaboração entre todos os membros do time para se alcançar a qualidade de um software, se os desenvolvedores e os testadores trabalham alinhados no mesmo objetivo e a comunicação é efetiva entre eles.

Uma vez que podem existir pessoas apenas dedicadas ao teste de software dentro de uma equipe, verificou-se quais são as contribuições deste ao time. Destacando os seguintes itens: prover *feedback* durante todo o processo de desenvolvimento, ajudar ou ter conhecimento de nível de código do teste a ser realizado, assumir a liderança nos testes de aceitação, assumir a liderança nos testes de regressão, desenvolver os planos de testes, revelar cenários de teste adicionais através de testes exploratórios, garantir que a cobertura de teste é adequada, liderar os esforços de automação, liderar os esforços de testes de integração, executar testes de nível de sistema, manter ambientes de teste e os dados disponíveis, identificar problemas de regressão e partes técnicas de teste e outros

Para a validação do questionário, também foi incluída uma pergunta para que as pessoas pudessem dar sugestão de questões ou ideias do que poderia ser incluído no questionário, assim como prover *feedback* sobre as questões respondidas.

### **4.3 Estudo Piloto**

Neste estudo piloto, o questionário foi aplicado a 14 profissionais de desenvolvimento de software que utilizam métodos ágeis. A primeira parte do questionário buscou caracterizar os respondentes. A seção seguinte buscou identificar como teste de software é utilizado pelas equipes ágeis de desenvolvimento de software.

O estudo piloto serviu para melhorar e modificar o questionário para a pesquisa efetiva. Nas próximas seções, serão descritas as mudanças feitas nas perguntas do questionário. Por aplicar este questionário a um número pequeno de respondentes, os resultados obtidos não serão mostrados, pois não são relevantes no resultado final.



Perguntas referentes a qualidade de software e que não estavam diretamente relacionadas ao processo de teste e método ágil foram excluídas para que se pudesse estabelecer prioridades e focar no objetivo do trabalho para as atividades de teste em desenvolvimento de software em métodos ágeis. Observa-se que a discussão e aprimoramento das questões foram realizadas apenas para as questões que foram priorizadas neste contexto.

#### *4.3.1 Abordagem no estudo piloto sobre a caracterização dos respondentes*

A seguir serão apresentadas as questões relacionadas à caracterização dos respondentes e como estas perguntas foram aprimoradas para que se pudesse obter o perfil das empresas, equipes e indivíduos participantes da pesquisa.

Na primeira questão, foi verificado o porte da empresa que tinha sido construído para seguir a classificação do SEBRAE (2013). Após a aplicação do primeiro questionário, verificou-se a necessidade de incluir mais uma classificação, pois não se conseguiu caracterizar empresas que eram muito grandes, tais como as multinacionais ou de grande relevância no mercado brasileiro.

Nesta validação, também se optou por verificar qual a região de atuação da empresa e, assim, analisar se este fator interfere em como as empresas aplicam os processos de testes. Para caracterizar as empresas, foi adicionada uma pergunta para verificar se as empresas que têm como atividade principal o desenvolvimento de software, possuem processos diferentes daquelas que têm como atividades principais outros produtos, mas desenvolvem software para suprir suas necessidades. Uma pergunta sobre a natureza do software quanto ao risco também foi adicionada, pois um software de alto risco requer processos de testes mais rígidos, e isso pode impactar nas respostas dos entrevistados.

Além de caracterizar a empresa, notou-se que a formação das equipes dentro das empresas poderia interferir em como os processos são executados e, por isso, duas perguntas foram acrescentadas: uma para verificar o tamanho das equipes, e outra para saber se as equipes são distribuídas ou trabalham no mesmo local.

Em relação ao tempo de experiência em desenvolvimento ágil dos respondentes da pesquisa, não foram feitas modificações, mas, além da pergunta sobre o tempo de experiência,

foi adicionada uma questão para saber se o respondente possui alguma certificação em métodos ágeis, verificando a preparação formal do respondente. Além do mais, foi adicionada uma pergunta sobre a função dos respondentes, para analisar a percepção dos indivíduos em relação aos testes de software de acordo com a atividade que este trabalha.

A pergunta sobre a indicação dos métodos ágeis utilizados pelos respondentes não foi modificada, pois entender como estes métodos funcionam pode ajudar a trazer informações relevantes para contexto da pesquisa.

#### *4.3.2 Abordagem no estudo piloto sobre as atividades de teste*

Nesta seção serão discutidas as perguntas sobre atividades de testes. Novas alternativas e perguntas foram incluídas depois da aplicação do questionário no estudo piloto que permitiram melhor aproximação das práticas e técnicas executadas pelos respondentes do estudo piloto.

Na pergunta sobre a elaboração de uma estratégia de teste em ambientes ágeis, além dos itens selecionados para o planejamento de uma iteração, verificou-se a preocupação dos respondentes em fazer os testes durante o desenvolvimento, ressaltando que não se deve ter uma etapa para o teste de software. Uma questão foi adicionada para cobrir essa preocupação. Além disso, foram apontados dois itens que devem ser considerados durante o planejamento: a criação dos ambientes de testes e a revisão dos cenários de teste pelo time.

Verificou-se também que vários fatores podem influenciar no período selecionado, tais como a maturidade do time (times com menos experiência podem precisar de mais tempo) e a natureza do software (alguns tipos de software requerem mais tempo de desenvolvimento e teste). Ressalta-se, ainda, os processos utilizados pela empresa, em que processos mais eficientes podem resultar em períodos menores. Por último, observou-se a tendência de se utilizar entrega contínua, no qual deve estar sempre pronto para entregar o software.

Em relação à pergunta sobre níveis de testes, enquanto alguns respondentes observam que todos os níveis de testes devem ser executados dentro de uma iteração, outros não aplicam este conceito. Alguns respondentes tiveram dificuldade em entender a pergunta, verificando, assim, a necessidade de readaptar a pergunta para que o respondente não tenha dúvidas do que

está sendo perguntado ou, até mesmo, levando-se em consideração a falta de conhecimento de definições utilizadas comumente na área de teste.

Na experiência do pesquisador, observou-se que, apesar de a equipe de teste possuir várias atividades de testes durante a iteração, a grande parte dos testes ficava para o final da iteração. Por isso, inclui-se uma pergunta para verificar se as equipes conseguem manter um ambiente em que o teste é feito continuamente. Muitos respondentes do questionário piloto mostraram preocupação quanto à continuidade do teste durante as iterações, observando que muitas vezes modelos em cascata, em menor escala, eram criados dentro das iterações

Para a validação de requisitos funcionais e não funcionais, observou-se que, para requisitos funcionais, a validação é feita com testes exploratórios guiados por mapas mentais originados dos casos de uso dos requisitos; em muitos casos, a validação é feita com a utilização de testes manuais e automatizados de forma mesclada. Já para a validação de requisitos não funcionais, são utilizados testes que têm geralmente como base ferramentas que geram relatórios a serem analisados pelos desenvolvedores. Alguns times apenas executam testes não funcionais quando se tem tempo, outros não utilizam nenhum tipo de validação para requisitos não funcionais. Observou-se a incidência alta do uso de ferramentas para a validação de requisitos não funcionais. Após a validação do questionário, acrescentou-se uma pergunta de múltipla escolha que cobrisse quando os testes não funcionais são executados e se estes são feitos a partir do uso de ferramentas.

Para que os times possam fazer investimentos que otimizem os resultados do teste de software, verificou-se com os respondentes quais práticas mais corroboram para entrega de um software testado ao final de uma iteração. Após a aplicação do questionário, verificou-se a necessidade de adicionar opção sobre a revisão ou criação dos cenários de teste junto ao desenvolvedor, para que ele possa prevenir a identificação de defeitos durante o desenvolvimento. A revisão foi citada por mais de um respondente. Observou-se também a necessidade de adicionar a utilização de testes manuais neste contexto.

Uma vez que foi analisado o quanto é importante a automação para o teste de software em ágil, verificou-se a necessidade de investigar como as equipes tem utilizado a automação de testes nestes ambientes. Não houve modificações nesta pergunta após o teste piloto.

Visto que para o teste dentro das histórias utilizam-se técnicas de testes manuais com bastante frequência, tornou-se necessário investigar como se dá esta validação, após a validação do questionário adicionou a opção: “testes exploratórios guiados por mapas mentais”.

Durante a validação do questionário, verificou-se que algumas estratégias utilizadas por times ágeis poderiam influenciar em como os testes são executados e por isso acrescentou-se uma pergunta sobre as abordagens: Acceptance Test Driven Development (ATDD), Behavior Driven Development (BDD) e Test Driven Development (TDD). Observou-se, também, a importância da participação dos *stakeholders* para validação do software, acrescentando-se ao questionário uma pergunta sobre o tema.

Com a pesquisa, verificou-se, também, que as equipes possuíam um ambiente de teste gerenciável, com estratégia, monitoramento e controle. Para complementar a pergunta sobre o gerenciamento do ambiente de teste utilizado, adicionou-se uma pergunta sobre a utilização de métricas neste contexto, verificando assim como se dá o gerenciamento de testes.

As perguntas sobre a existência de pessoas dedicadas ao teste de software foram reformuladas para conseguir verificar como o teste de software é executado pelas equipes e assim analisar se existe separação de papéis, equipes independentes ou multidisciplinares. Não houve modificações nas perguntas sobre o papel do testado nas equipes ágeis.

Durante a validação do formulário, vários respondentes apontaram a possibilidade de ser ter níveis intermediários em perguntas que aceitavam apenas “sim” e “não”; por isso, estas perguntas foram alteradas para que se pudesse utilizar escalas.

## **5 RESULTADOS**

A pesquisa feita a partir de uma análise bibliográfica e a experiência do pesquisador resultou em um questionário piloto que foi validado e alterado para a versão final conforme apêndice 2. Este questionário foi enviado de forma eletrônica de acordo com a amostra definida na pesquisa. A pesquisa contou com 167 respondentes, sendo que 149 possuíam experiência em times de desenvolvimento de software ágeis. A análise de dados foi feita para os respondentes com experiência em ágil, excluindo aqueles que não tinham experiência prática no desenvolvimento de software ágil.

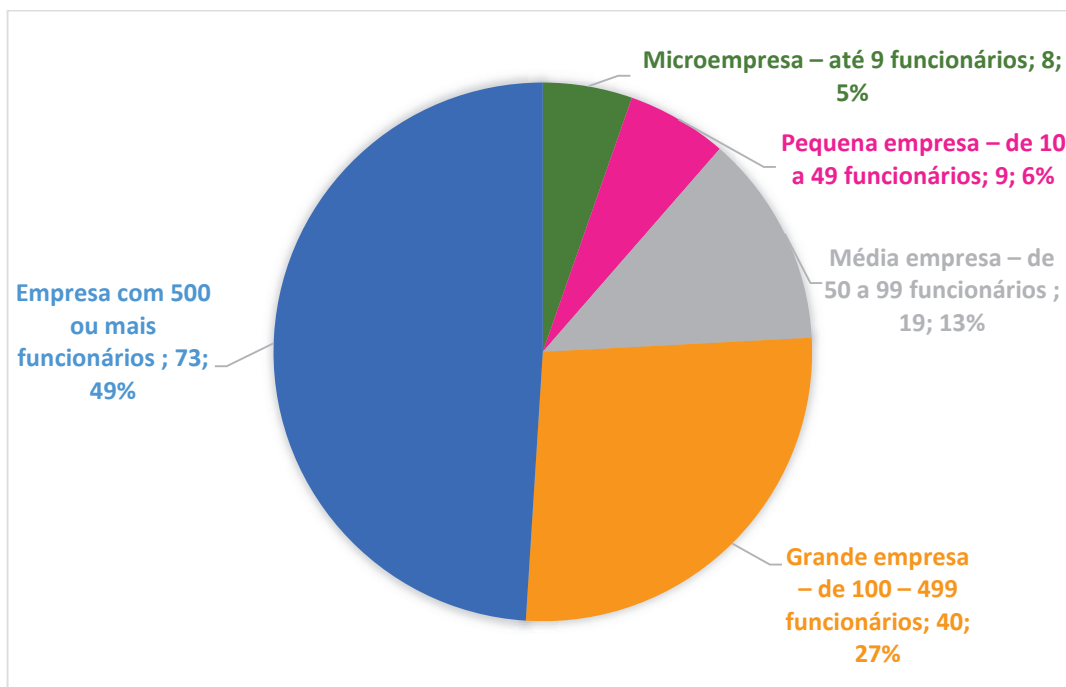
A pesquisa ficou disponível publicamente para respostas entre dia 26 de novembro de 2015 até o dia 08 de janeiro de 2016.

### **5.1 Caracterização dos respondentes**

Inicialmente foi feito um levantamento de dados dos respondentes. Este levantamento tem por objetivo verificar as características da empresa, do software, da equipe, do indivíduo e do método de desenvolvimento.

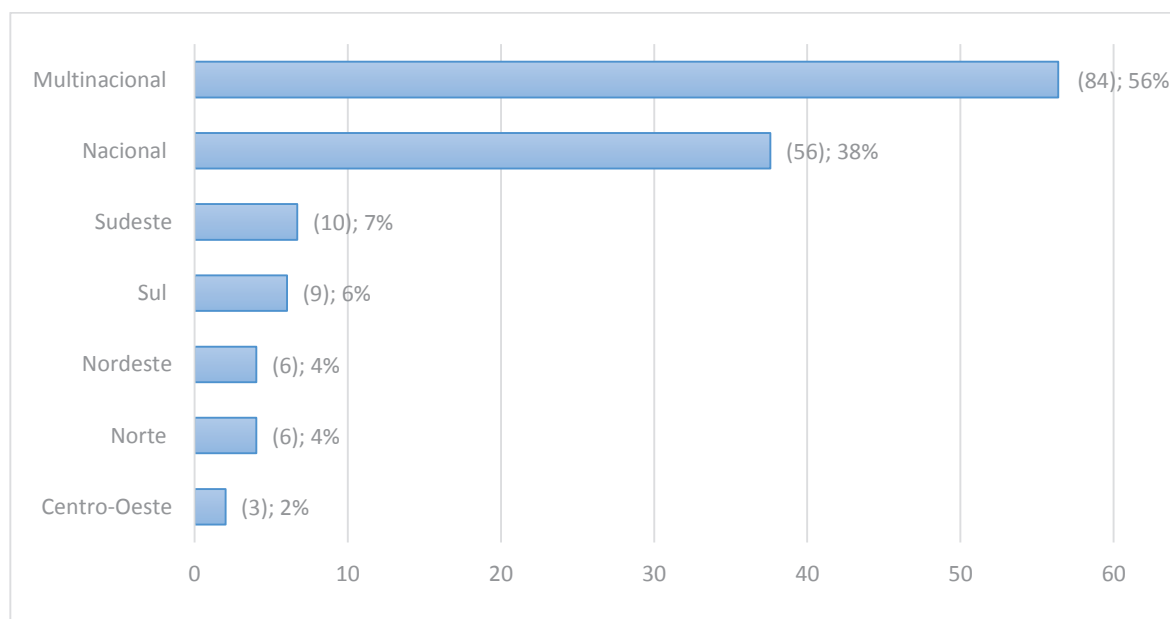
A primeira questão da pesquisa tem por objetivo verificar qual o porte de empresa onde atuam os respondentes. Observa-se que 49% dos participantes trabalham em empresas com mais ou 500 funcionários, 27% trabalham em grande empresa, 13% em média empresa, 6% em pequena empresa e 5% em microempresa conforme mostrado no Gráfico 1.

Gráfico 1: Porte da empresa onde atuam os respondentes da pesquisa



A segunda pergunta teve por objetivo verificar qual a extensão de atuação da empresa, isso é em qual localidade essas empresas atuam (Gráfico 2).

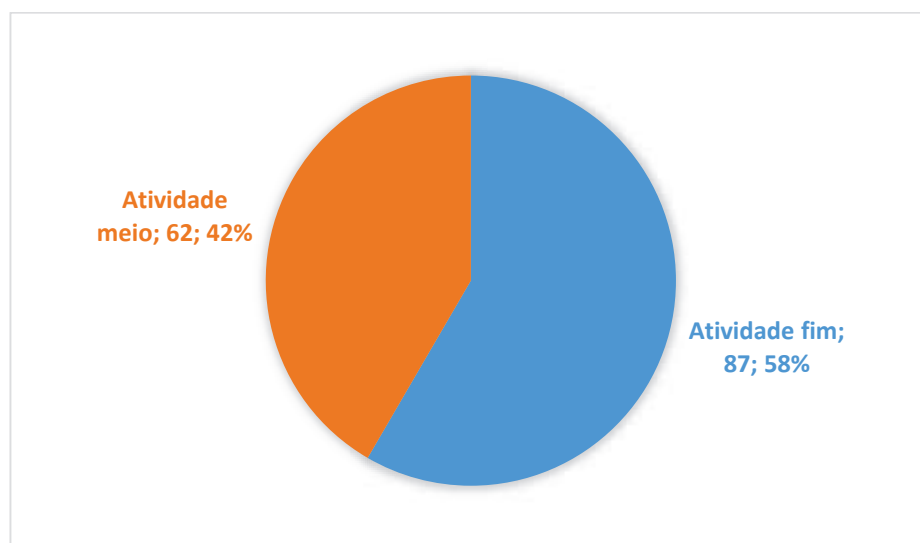
Gráfico 2: Extensão de atuação da empresa



Observa-se no Gráfico 2 que 56% dos participantes trabalham em empresas com atuação multinacional, 38% trabalham em empresas nacionais. Regionalmente, a pesquisa conseguiu atingir 7% dos respondentes em empresas com atuação no Sudeste, 6% de empresas no Sul, 4% no Nordeste, 4% norte e 2% centro-oeste. Como as empresas podem atuar em múltiplas regiões foi permitido que se seleccionasse mais de um local, a porcentagem é calculada pelo número total de participantes da pesquisa (149).

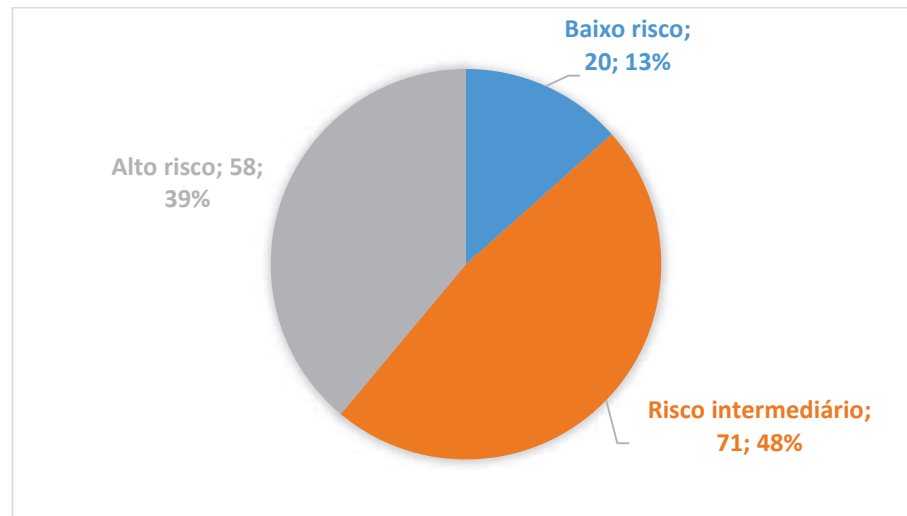
Verificou-se também se o desenvolvimento de software é atividade meio ou fim da empresa. Foi observado que 58% dos respondentes trabalhavam em empresas onde o software é atividade fim e 42% é atividade meio, de acordo com o Gráfico 3. Essa verificação é importante, pois pode haver diferença nas práticas de teste, quando o software é a atividade fim e quando este é apenas parte do processo de produção da empresa.

Gráfico 3: Classificação do software como atividade meio ou fim da empresa



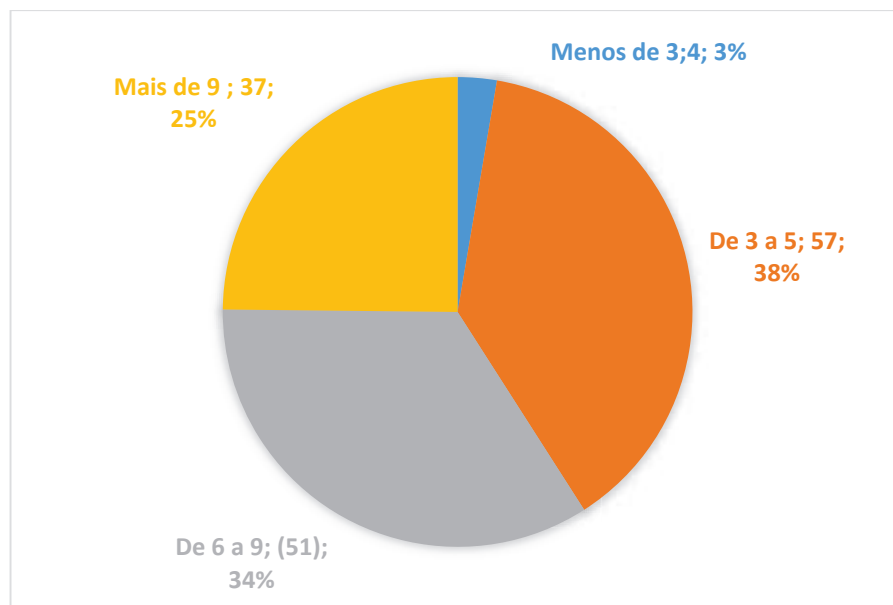
Em relação ao software desenvolvido, verificou-se a classificação quanto ao risco, no qual 48% é de risco intermediário, 39% de alto risco e 13% de baixo risco, conforme mostrado no Gráfico 4. Essa verificação é importante, pois espera-se que software de alto risco tenham mais rigor no processo de teste do que softwares com risco menor.

Gráfico 4: Classificação do software em relação ao risco



A pesquisa classificou também em relação ao tamanho da equipe de trabalho. Conforme mostrado no Gráfico 5, onde 3% participavam de equipes com menos de 3 membros, 38% com equipes de 3 a 5 membros, 34% em equipes de 6 a 9 membros e 25% em equipes de mais de 9 membros.

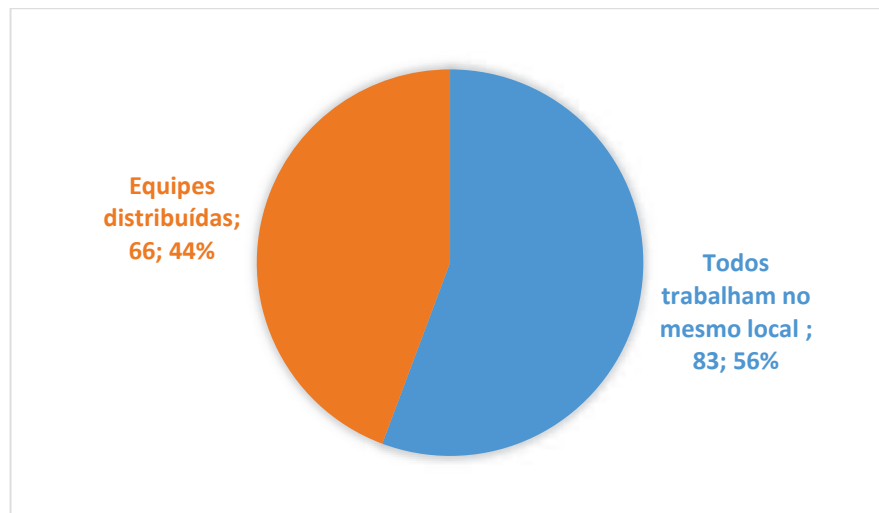
Gráfico 5: Tamanho das equipes: número de membros





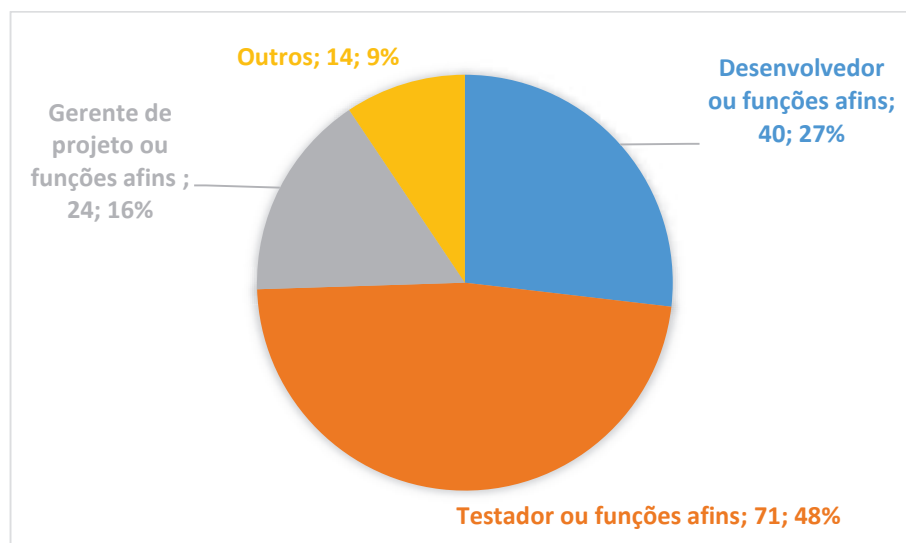
Para caracterização das equipes dos respondentes verificou-se, também, se estes trabalhavam em equipes distribuídas ou locais, dos quais 56% trabalham com as equipes no mesmo local e 44% trabalham em equipes distribuídas, conforme mostrado no Gráfico 6.

Gráfico 6: Distribuição das equipes



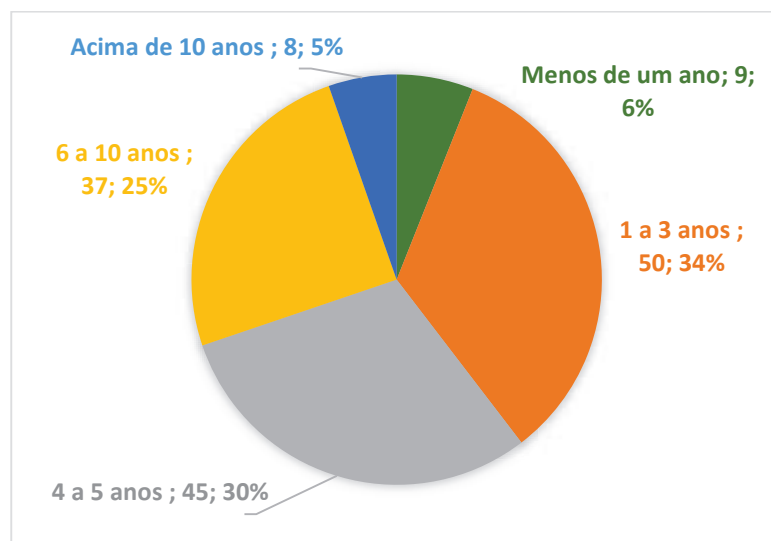
Também se caracterizou os respondentes em relação à função e verificou que 48% dos respondentes são testadores ou desempenham funções afins, 27% são desenvolvedores, 16% gerentes de projeto e 9% executam outras atividades. Conforme observado no Gráfico 7.

Gráfico 7: Função dos respondentes da pesquisa



Verificou-se, também, o tempo de experiência dos participantes no desenvolvimento de software utilizando métodos ágeis. De acordo com Gráfico 8, observou-se que 6% dos respondentes tem menos de um ano de experiência, 34% tem de 1 a 3 anos de experiência, 30% tem de 4 a 5 anos de experiência, 25% tem de 6 a 10 anos de experiência e 5% tem mais de 10 anos de experiência.

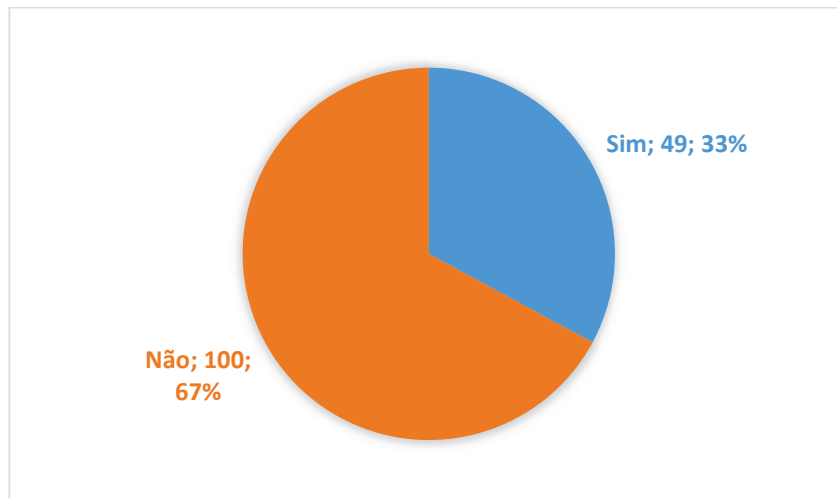
Gráfico 8: Tempo de experiência em desenvolvimento de software com métodos ágeis



Observa-se que apenas 6% tem pouca experiência (menos de um ano) em desenvolvimento ágil. Esse fator é importante para garantir que os processos são executados por aqueles que já possuem tempo de experiência em processos de desenvolvimento ágil na prática.

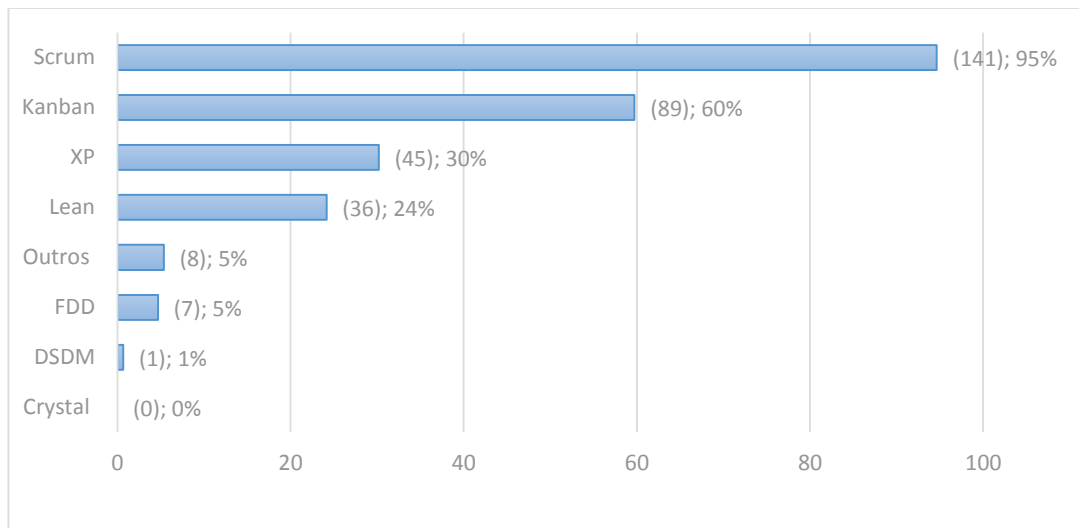
A pesquisa verificou, também, se os participantes tinham alguma certificação em métodos ágeis com intuito de verificar a educação formal sobre o assunto. Observou-se que 67% não possuem certificação e 33% possuem algum tipo de certificação em ágil como mostrado na Gráfico 9.

Gráfico 9: Certificação em métodos ágeis



Por fim, verificou-se quais os métodos de desenvolvimento utilizados pelos participantes. Os participantes podiam escolher mais de um método. Verificou-se que 95% utilizam Scrum para desenvolvimento de software, 60% utilizam Kanban, 30% utilizavam XP, 24% Lean, 5% outros métodos, 5% FDD, 1% DSDM e 0% Crystal conforme o Gráfico 10.

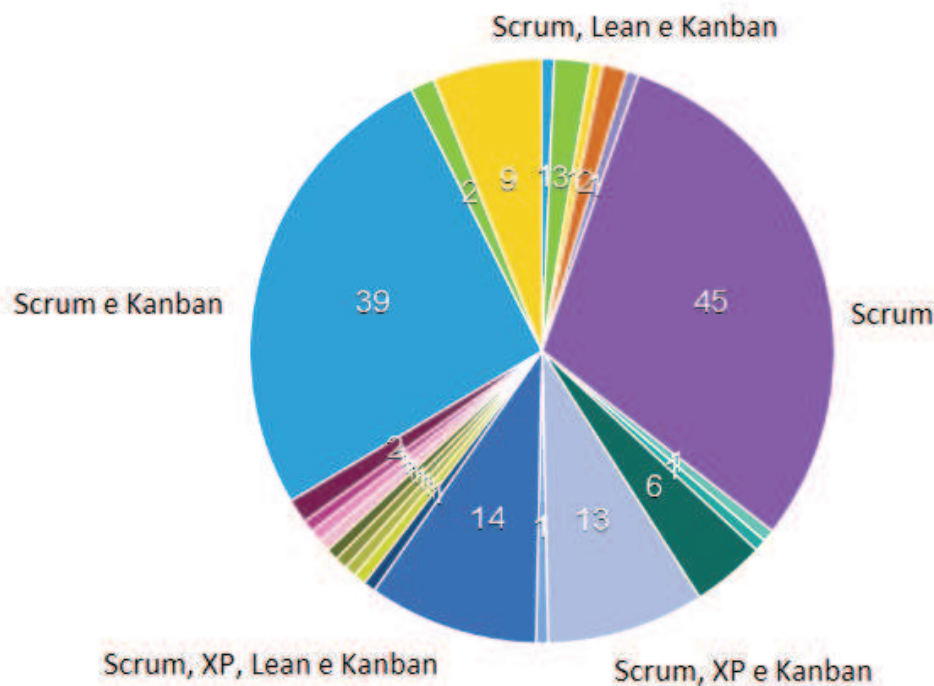
Gráfico 10: Métodos ágeis utilizados pelos participantes da pesquisa



Observa-se, ainda, no Gráfico 11, quais as combinações mais utilizadas pelos respondentes. Verifica-se que 45 respondentes utilizam apenas Scrum, 39 utilizam Scrum e Kanban, 14 utilizam Scrum, XP, Lean e Kanban, 13 utilizam Scrum, XP e Kanban e 9 utilizam

Scrum, Lean e Kanban. As outras combinações por serem pouco utilizadas não foram mostradas.

Gráfico 11: Métodos ágeis utilizados de forma conjunta pelos participantes da pesquisa



Com os dados do Gráfico 11, verificou-se que 30,20% dos participantes da pesquisa utilizam apenas o Scrum como método de desenvolvimento, 26,17% utilizam Scrum e Kanban, 9,40% utilizam Scrum, XP, Lean e Kanban, 8,72% utilizam Scrum, XP e Kanban e 6,04% utilizam Scrum, Lean e Kanban.

Por meio das perguntas sobre as características verificou-se o perfil dos respondentes. Destacando-se que 49% trabalham em empresas com mais de 500 funcionários, 56% em empresas multinacionais, 58% tem o software como a atividade fim da empresa, 48% desenvolvem software de risco intermediário, 38% das equipes são compostas por 3 a 5 membros, 56% trabalham com times locais, 34% utilizam métodos ágeis por 1 a 3 anos, 67% não possuem nenhum tipo de certificação ágil e o método de desenvolvimento mais utilizado pelos respondentes é o Scrum com 95% de respostas.

## 5.2 Apresentação e análise dos resultados

Em síntese, os dados da pesquisa indicam as práticas e abordagens mais utilizadas no teste de software em equipes de desenvolvimento ágil. Destaca-se, ainda, que algumas práticas sugeridas na literatura não são utilizadas na prática de forma ampla por parte das equipes dos participantes da pesquisa. Os resultados apontam também que existe pouca diferença na forma de como os testes são executados por diferentes características de empresas, equipes e pessoas.

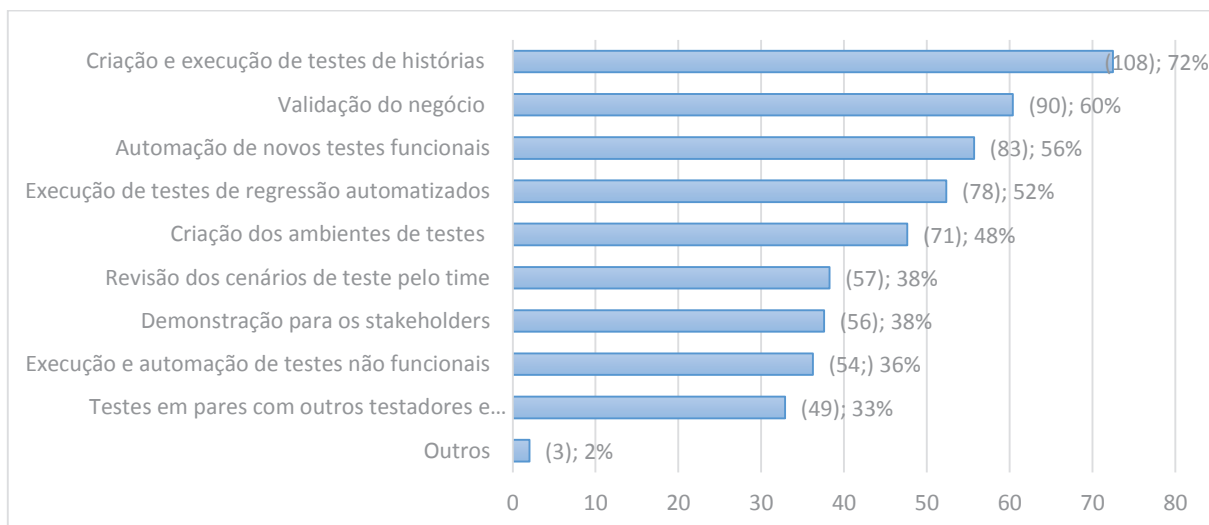
Considerando a baixa correlação encontrada em grande parte das variáveis, a ferramenta de análise de dados Watson *Analytics* foi utilizada na busca de evidências que corroborassem a baixa correlação. A análise com a ferramenta considerou todas as perguntas que envolviam o processo de desenvolvimento, bem como as características das empresas e das equipes. Para as perguntas que envolviam a opinião sobre melhores práticas ou tempo, foram utilizadas as características das pessoas para a análise. Com isso foi possível ilustrar, por meio de gráficos, a baixa correlação identificada pela análise estatística convencional.

As seções seguintes irão mostrar e discutir os resultados obtidos para cada questão abordada sobre as práticas de testes. Os resultados não considerados significativos não serão mostrados. Apenas resultados com tendência discreta de algumas análises são apresentadas.

### 5.2.1 Fatores considerados no âmbito de testes no planejamento de uma iteração

Os fatores mais considerados no planejamento de uma iteração podem ser observados no Gráfico 12. A pergunta foi de múltipla escolha, com 10 opções e em média foram selecionadas 4,36 opções por respondente.

Gráfico 12: Fatores considerados no âmbito de testes no planejamento de uma iteração



Destaca-se que o fator mais considerado é a criação e execução de testes de histórias com 72%, posteriormente a validação do negócio com 60%, a automação de novos testes funcionais com 56%, execução de testes de regressão automatizados com 52%, criação dos ambientes de testes com 48%, revisão dos cenários de teste pelo time com 38%, demonstração para os *stakeholders* com 38%, execução e automação de testes não funcionais com 36%, testes em pares com outros testadores e desenvolvedores com 33% e outros fatores com 2%.

Observa-se que a principal preocupação é a criação e execução de testes de histórias e posteriormente a validação do negócio. De forma geral os respondentes garantem que as novas funcionalidades estão sendo verificadas e validadas na iteração.

Observou-se que as atividades de automação e execução de testes de regressão são consideradas por cerca de 50% das equipes (automação de novos testes funcionais - 56%, execução de testes de regressão automatizados - 52%). Fator que mostra que apesar das equipes se preocuparem com a execução de testes das novas funcionalidades desenvolvidas, quase metade das equipes não se preocupam em verificar de forma automatizada se o software, integrado à nova funcionalidade, não foi danificado. Observa-se que estes times não planejam a execução de novos testes para as próximas integrações e a execução de regressão de forma automatizada.

Verifica-se a preocupação da criação dos ambientes de testes por 48% dos respondentes. Na experiência do pesquisador observa-se que diferentes defeitos podem ser encontrados em diferentes configurações de ambientes. Nessa experiência foi de grande valia a criação de

diferentes ambientes utilizando a combinação de diversos bancos de dados, servidores de aplicação e sistemas operacionais (página 68).

A revisão dos cenários de teste pelo time foi indicada por 38% dos respondentes. Esse fator foi indicado durante o teste do questionário piloto. No qual a discussão de cenários de teste com toda a equipe, segundo indicações dos respondentes do questionário piloto, trouxe melhorias qualitativas ao software e manteve o time alinhado sobre a cobertura de teste.

A demonstração para os *stakeholders* foi indicada por 38%. Apesar de um número baixo de indicações, na experiência do pesquisador a demonstração foi útil para validar o software e alinhar toda a equipe sobre a necessidade do cliente.

Outros fatores como a execução e automação de testes não funcionais com 36% , testes em pares com outros testadores e desenvolvedores com 33% também foram indicados.

Buscou-se correlacionar os fatores de teste com características das empresas, equipes e respondentes. Valores numéricos foram atribuídos para os fatores e soma destes valores foram correlacionados com as características. A análise, também, foi feita com a correlação de cada fator analisado individualmente.

Buscou-se verificar o total de fatores selecionados para uma iteração e sua correlação com características das empresas e software: porte da empresa e a natureza do software desenvolvido por ela. Para o cálculo foi utilizado o coeficiente de postos de Spearman. Os valores de coeficiente correlação podem ser observados na Tabela 3.

Tabela 3: Correlação de fatores planejados em uma iteração vs. características das empresas e software

	<i>Porte da empresa</i>	<i>Natureza do software</i>
Quantidade de Fatores	0,18	0,20

Observa-se que os valores de correlação tiveram baixa significância, desta forma verifica-se que quando cresce o porte da empresa, não é possível dizer que mais fatores são indicados, pois o valor de significância é de apenas 0,18. Verifica-se também que houve baixa significância (0,20) para o risco do software com a quantidade de fatores indicados.

Com a análise de correlação não é possível concluir que as características das empresas analisadas apresentam diferentes comportamento na seleção de fatores de teste para o planejamento.

A correlação também foi analisada observando cada fator separadamente e suas correlações com as características: atividade da empresa e localização dos membros. Foi utilizado para o cálculo o coeficiente de Yule, conforme observado na Tabela 4.

Tabela 4: Correlação de fatores planejados em uma iteração vs. características das empresas

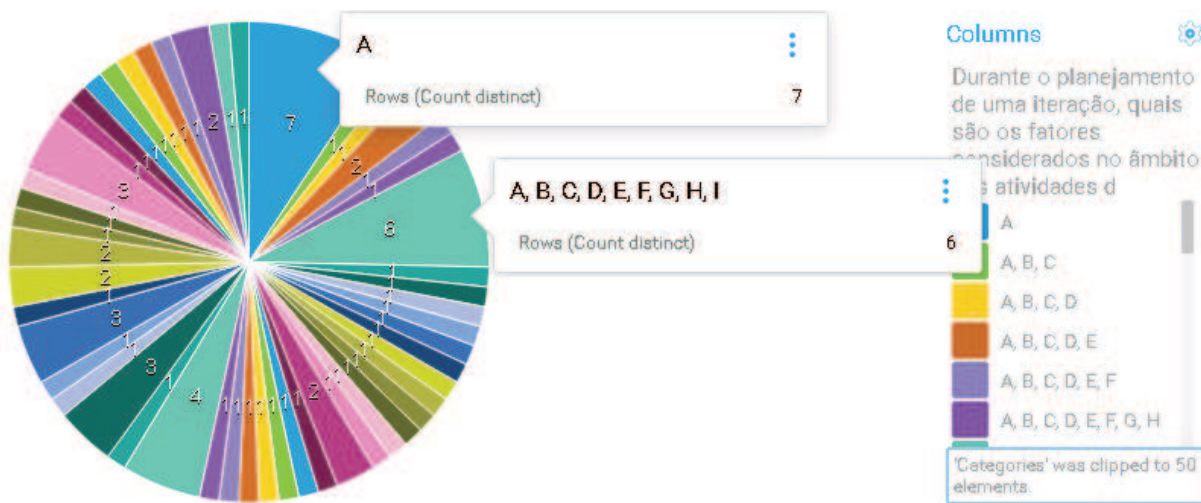
	<i>Atividade da empresa</i>	<i>Localização dos membros</i>
Criação e execução de testes de histórias	-0,20	<b>0,35</b>
Testes em pares com outros testadores e desenvolvedores	0,16	0,14
Validação do negócio	0,03	-0,05
Automação de novos testes funcionais	-0,09	0,18
Execução de testes de regressão automatizados	-0,19	<b>0,39</b>
Execução e automação de testes não funcionais	-0,15	0,18
Demonstração para os stakeholders	<b>0,32</b>	0,01
Criação dos ambientes de testes	-0,03	0,30
Revisão dos cenários de teste pelo time	0,02	-0,01

Verifica-se na Tabela 4 que mesmo analisando a indicação por fator, a correlação foi muito discreta. Apesar disso, é possível observar que para empresas com atividade meio existe maior preocupação com a demonstração para os *stakeholders*. Observa-se, também, maior proporção na preocupação de equipes distribuídas para a criação e execução de testes de histórias, execução de testes de regressão automatizados e na criação de ambientes de testes.

Uma vez que a análise de dados não permitiu identificar correlação significativa a ferramenta *Watson Analytics*, da IBM, foi utilizada para corroborar essa falta de correlação significativa. Ao se analisar de forma isolada a combinação de fatores selecionados no planejamento de uma iteração, verifica-se que não existe fatores combinados que são selecionados de forma comum. Explicando-se, assim, o motivo pelo qual a análise por diferentes características não mostra tendências ou diferenças em correlações, pois não foram encontrados padrões de respostas como ilustra o Gráfico 13.



Gráfico 13: Fatores considerados no âmbito das atividades de testes combinados



Verificou-se maior incidência para “A - Criação e execução de testes de histórias” e para a seleção de todos os fatores: “A - Criação e execução de testes de histórias”, “B - Testes em pares com outros testadores e desenvolvedores”, “C - Validação do negócio”, “D - Automação de novos testes funcionais”, “E - Execução de testes de regressão automatizados”, “F - Execução e automação de testes não funcionais”, “G - Demonstração para os stakeholders”, “H - Criação dos ambientes de testes”, “Revisão dos cenários de teste pelo time”. Embora destaca-se nos gráficos, os valores (6 e 7) são considerados baixos para análise de dados.

A mesma tendência é observada quando se verifica os fatores selecionados e características dos respondentes. O Gráfico 14 exemplifica a distribuição de como os fatores são selecionados de acordo com a localização dos membros.

Gráfico 14: Fatores considerados no âmbito das atividades de testes combinados vs. localização dos membros

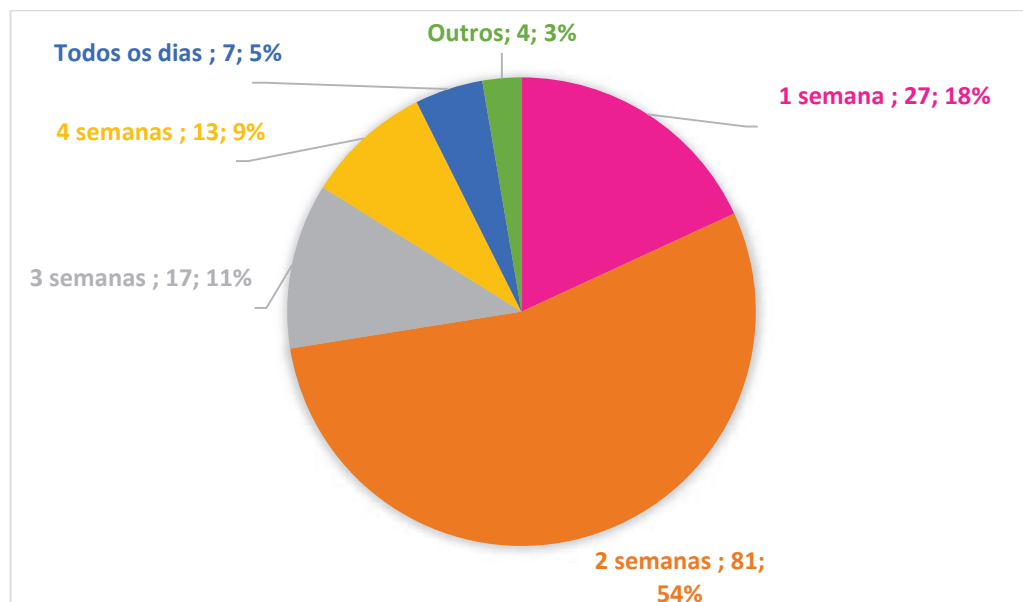


O Gráfico 14 obtido pela ferramenta *Watson Analytics* ilustra a falta de tendência ou diferenças significativas quanto as características dos respondentes em relação a localização dos membros quando se seleciona fatores de teste no planejamento de uma iteração. A mesma análise foi feita para as outras características citadas nesta pesquisa e não foram encontradas tendências sobre o conjunto de fatores que são selecionados pelos respondentes.

### 5.2.2 Tempo para codificar e testar o software em uma iteração

Verificou-se também o tempo de iteração que as pessoas julgam ideal para codificar e testar as funcionalidades desenvolvidas e observou que 54% acreditam que duas semanas é o tempo ideal para codificar e testar um software, 18% acreditam que uma semana é o tempo ideal, 11% acreditam que três semanas é o tempo ideal, 9% acreditam que quatro semanas é o tempo ideal, 5% acreditam que se deve ter o software pronto todos os dias e 3% apontaram outras alternativas. Esses dados podem ser observados no Gráfico 15.

15: Tempo ideal para codificar e testar um software



Na experiência do pesquisador, verificou-se que o tempo dedicado para uma iteração impactou nas atividades de teste. O tempo que a equipe achou ideal para codificar e testar o

software foi de duas semanas. Esse tempo também se refletiu na pesquisa, no qual 54% apontaram o período como o ideal de uma iteração. O tempo de iteração de uma semana foi o segundo período selecionado pelos respondentes, com 18%. Destacando-se, assim que as iterações pequenas de 1 ou 2 semanas são consideradas boas pelos times para que se consiga ter o software pronto, codificado e testado, ao final de uma iteração.

Ekas e Will (2014) destacam, também, na experiência deles como consultores de implantação de métodos ágeis, que muitas equipes são restritivas na adoção de iterações curtas. As principais razões destacadas pelas equipes são o tempo insuficiente para teste e também por manter a equipe sempre esforçada ao máximo para conseguir entregar o software funcionando. Os autores, Ekas e Will (2014), recomendam, no entanto, que os times utilizem 1 ou 2 semanas no máximo. Eles apontam como fatores para isso a melhor capacidade que o time tem para estimar em períodos menores de tempo e a divisão do trabalho em pequenas histórias que permite melhor conexão entre desenvolvimento e teste.

Constata-se, desta forma, que iterações pequenas, de uma e duas semanas, são períodos ideais para os participantes da pesquisa para que seja possível codificar e testar o software. Observando, desta forma, tendência em utilizar períodos pequenos (uma ou duas semanas) do que períodos grandes (três ou quatro semanas).

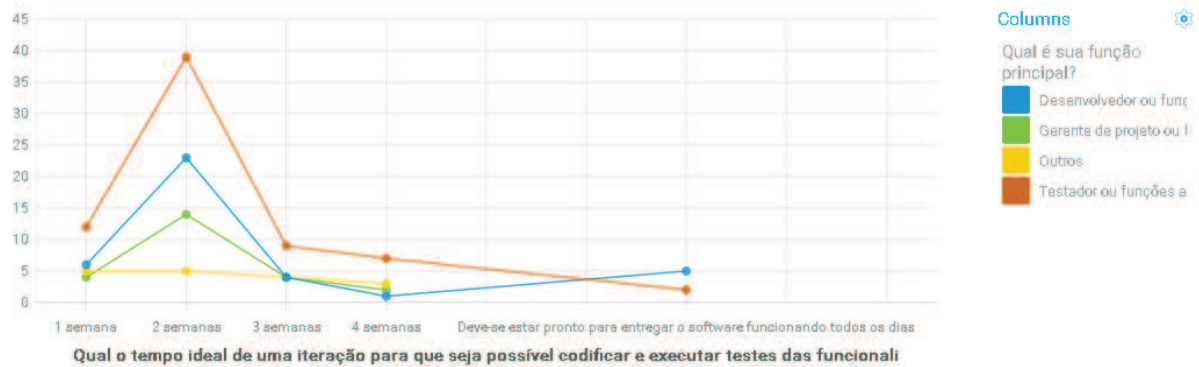
Verificou-se a correlação sobre a opinião de tempo de uma iteração para características de tempo de experiência. Os valores desta correlação não foram significantes, conforme observado na Tabela 5. Observa-se assim a tendência de utilizar duas semanas como tempo ideal para se codificar e testar um software pelos participantes da pesquisa sem correlação com o tempo de experiência.

Tabela 5: Correlação de tempo de iteração vs. tempo de experiência

<i>Tempo de Experiência</i>	
Tempo de iteração	0,07

A análise feita pela ferramenta *Watson Analytics* verificou também esta tendência utilizando os valores absolutos (não porcentagem) para função, conforme o Gráfico 16.

Gráfico 16: Tempo ideal de iteração vs. função



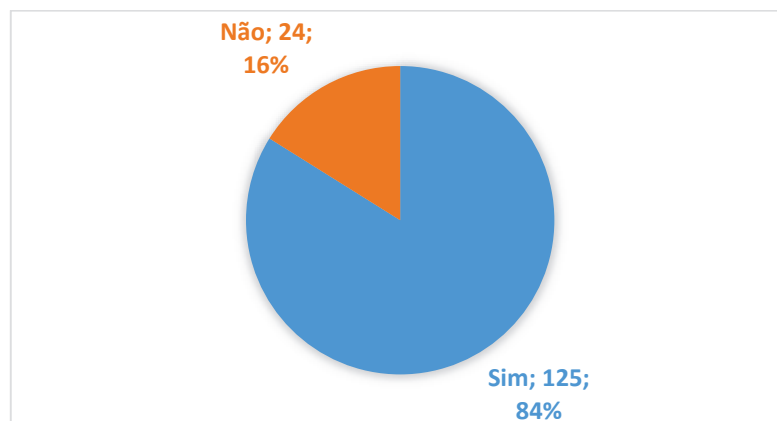
Observa-se, no Gráfico 16, que o número de respondente por função é diferente, no entanto verifica-se maior incidência na indicação de tempo de duas semanas.

Outras análises foram feitas para verificar se as características dos respondentes impactavam no tempo definido, mas não foram encontradas tendências. Com estudo pode se observar que os respondentes desta pesquisa, independentemente de suas características, acreditam que o tempo ideal para se codificar e testar um software é de duas semanas.

### 5.2.3 Execução dos testes na iteração

Uma vez que os testes devem ser contínuos em métodos ágeis, verificou-se se os times conseguiam testar o software durante toda a iteração. Observou-se que 84% dos respondentes conseguem testar o software continuamente, enquanto 16% não conseguem (Gráfico 17).

Gráfico 17: Porcentagem de times que conseguem testar o software durante toda a iteração



Observa-se que os times pesquisados conseguem, na sua maior parte, executar continuamente o teste durante a iteração e não apenas o executá-lo durante algumas fases do desenvolvimento, caracterizando-os alinhados com os métodos de desenvolvimento ágeis.

Na análise de correlação, utilizando o coeficiente de Yule, verificou se o tempo de iteração (0 <= 2 semanas e 1 >= 3 semanas), a utilização de testes automatizados e o tamanho da equipe iteração (0 = equipes pequenas e 1 = equipes grandes) influenciam na capacidade de testar durante toda a iteração. Os valores de correlação, no entanto, se mostraram com baixa significância, conforme mostrado na Tabela 6.

Tabela 6: Correlação de realização de testes durante toda a iteração vs. tempo de iteração, utilização de testes automatizados e tamanho da equipe

	<i>Tempo da iteração</i>	<i>Tamanho equipe</i>	<i>Utiliza automação</i>
Os testes são realizados durante toda iteração	<b>0,41</b>	-0,12	<b>0,46</b>

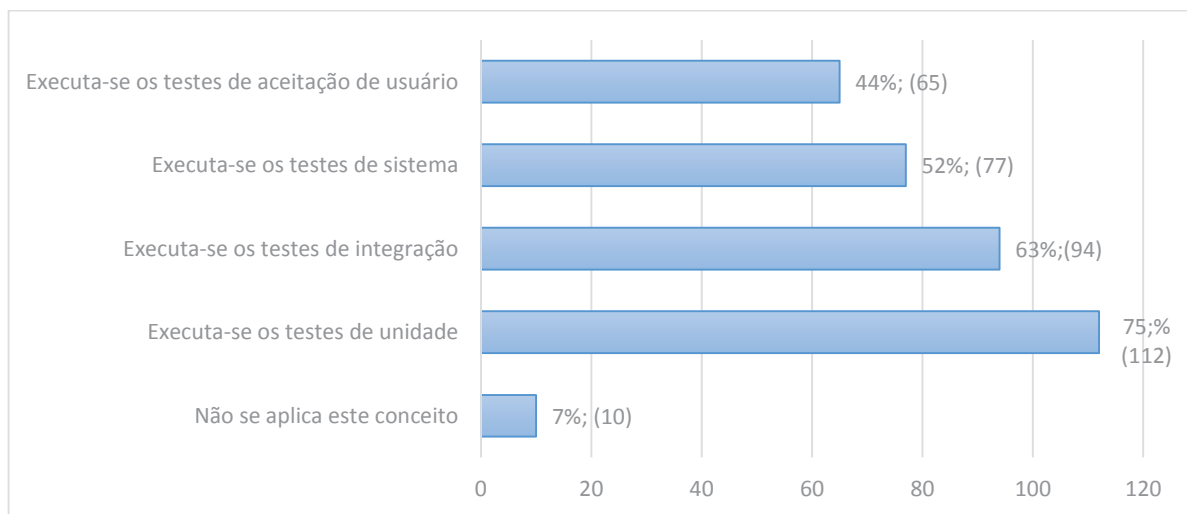
Observa-se que os valores de correlação indicam baixa associação entre as variáveis, por isso não é possível, nessa amostra, concluir que o tempo de iteração, tamanho da equipe e a utilização de testes automatizados impactam na capacidade de se testar durante toda a iteração.

Apesar do nível de correlação ser baixo, é possível verificar que em iterações pequenas os testes são proporcionalmente mais realizados durante toda iteração do que em iterações grandes. Observa-se, também, que a utilização de automação faz com que estes também sejam mais executados durante toda a iteração.

#### 5.2.4 Níveis de teste

Para verificar como os testes são executados pelas equipes de desenvolvimento ágil, verificou se o conceito de níveis de teste, que é utilizado em métodos tradicionais, também é utilizado em métodos ágeis. Observou-se que apenas 7% dos respondentes não aplicam este conceito. Verificou-se também, nessa pergunta, quais os níveis de testes utilizados e observou-se que 75% executam os testes de unidade, 63% executam os testes de integração, 52% executam os testes de sistema e 44% executam os testes de aceitação. Conforme mostrado no Gráfico 18.

Gráfico 18: Níveis de testes utilizados em ambientes de desenvolvimento ágil



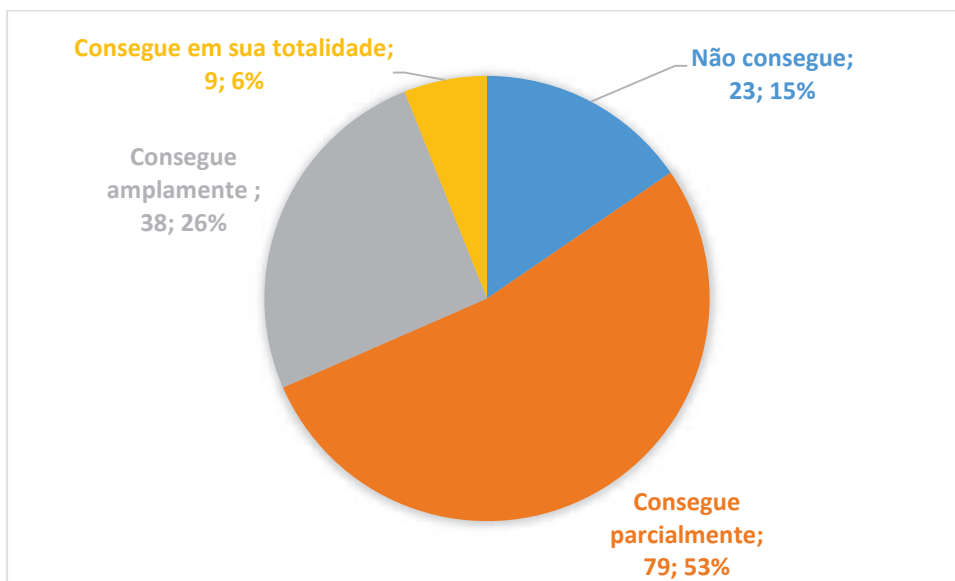
Observa-se que em modelos iterativos muitas vezes os níveis de testes se sobrepõem e não se aplica este conceito (ISTQB, 2014 – página 45). Alguns autores sugerem, no entanto, a adaptação para o modelo ágil usando os níveis: unidade, integração e aceitação (EXPEDIT, 2012 - página 45).

A análise dos dados permite verificar que para os respondentes da pesquisa, os níveis de testes são claros e as equipes conseguem executá-los. Observa-se, no entanto, ao analisar os níveis de testes executados que as atividades de testes não estão alinhadas com o valor ágil de colaboração com o cliente, pois este nível de teste, que permitiria melhor integração com o cliente é executado por apenas 44% dos respondentes, sendo esse o menos executado.

#### 5.2.5 Envolvimento dos Stakeholders

Visto que a integração com os *stakeholders* é o um desafio para o teste de software em times que implementam métodos ágeis, verificou-se que 15% não conseguem envolver os *stakeholders* no teste de software, 53% conseguem parcialmente, 26% conseguem amplamente e apenas 6% em sua totalidade, conforme mostrado no Gráfico 19

Gráfico 19: Envolvimento dos *stakeholders* no teste de software



Observa-se que apesar da literatura mostrar a importância da integração com os *stakeholders* [ (RICO; SAYANI; SONE, 2009 - página 22), (MYERS et al., 2011 - página 35), (CRISPIN E GREGORY (2015) – página 50)], poucos times (32%) conseguem envolvê-los com os testes de forma ampla ou integral.

Buscou-se por correlação de dados, utilizando o coeficiente de postos de Spearman, verificar se existe entre as características das empresas dos participantes diferentes comportamentos quanto ao envolvimento do *stakeholder*. Os valores de correlação, no entanto, foram considerados sem significância, como mostrado na Tabela 7, e não foram encontradas evidências que pudessem mostrar quando há mais envolvimento.

Tabela 7: Correlação do envolvimento com os *Stakeholders* vs. porte da empresa e natureza do software

	Porte da empresa	Natureza do Software
Envolvimento do Stakeholders	0,06	-0,05

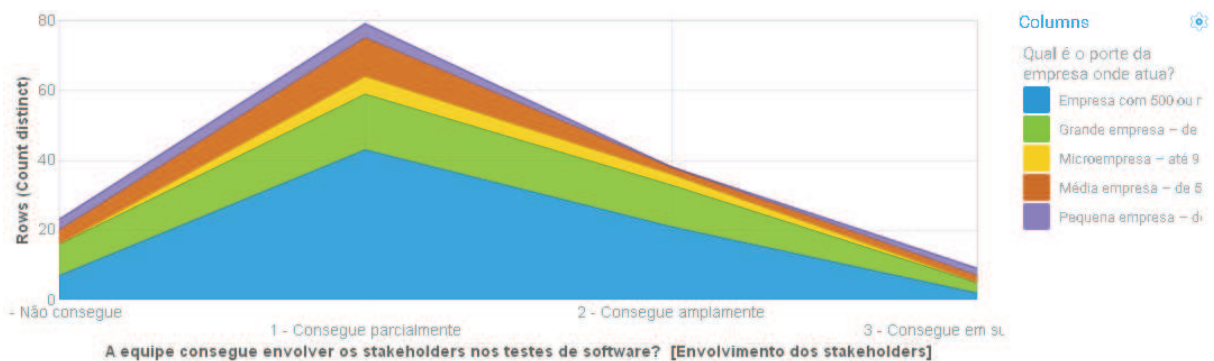
Utilizou-se, ainda, a análise utilizando o coeficiente de Yule (utilizando 0 para “não consegue” e “consegue parcialmente” e 1 para “consegue amplamente” e “consegue em sua totalidade”). Verificou-se, no entanto, independência das variáveis, conforme resultados da Tabela 8.

Tabela 8: Correlação do envolvimento com os *Stakeholders* vs. atividade da empresa e localização dos membros

	Atividade da empresa	Localização
Envolvimento do Stakeholder	0,09	0,07

A pesquisa buscou verificar ainda utilizando a ferramenta *Watson Analytics* para encontrar evidências quanto as características que pudessem diferenciar este envolvimento, verificou-se, entretanto, que estas características não tiveram impacto. O Gráfico 20 ilustra a falta de correlação entre o porte da empresa e o envolvimento com os *stakeholders*. Mostrando que a tendência de falta do envolvimento do *stakeholder* existe independente do porte da empresa.

Gráfico 20: Capacidade envolver os *stakeholders* vs. porte da empresa



Observa-se no Gráfico 20 que apesar de diferentes quantidades de respondentes por porte de empresa, existe a mesma tendência de conseguir envolver parcialmente o *stakeholder*.

#### 5.2.6 Práticas que corroboram para o software testado

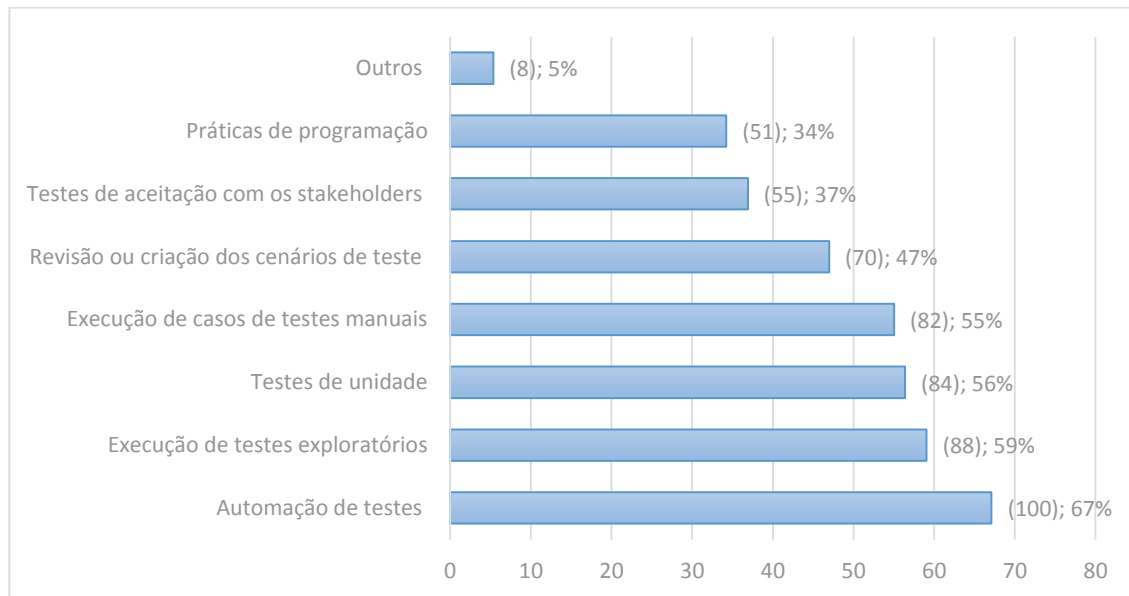
Verificou-se também quais as práticas os participantes acreditam ser mais importantes para se ter o software testado ao final de uma iteração e observou-se que 67% acreditam que a automação de testes é atividade que mais corrobora, 59% apontaram a execução de testes exploratórios, 56% selecionaram os testes de unidade, 55% a execução de casos testes manuais, 47% a criação e revisão de cenários de testes, 37% os testes de aceitação com os *stakeholders*,



34% as práticas de programação e 5% apontaram outras atividades não listadas, tais como integração contínua e demonstração para os *stakeholders*.

A pergunta foi de múltipla escolha e em média foram selecionadas 3,56 respostas, de um total de 8, por respondente. Os dados são mostrados no Gráfico 21.

Gráfico 21: Práticas que mais corroboram para se ter um software testado em ágil



Verificou-se quais as práticas os participantes acreditam ser mais importantes para se ter o software testado ao final de uma iteração e observou-se que 67% acreditam que a automação de testes é atividade que mais corrobora para ter um software codificado e testado. A automação de testes é destacada por vários autores como forma de entregar valor em tempo hábil em todas as fases de desenvolvimento [ (CRISPIN; GREGORY, 2009 – página 50), (PHAM; PHAM, 2011 – página 50), (DELAMARO *et al.*; 2009 – página 50), (SHAYE, 2008) – página 50]. Confirmando, assim, que as equipes encontram valor na utilização da automação, assim como na teoria.

Destaca-se, também, a execução de testes exploratórios com 59% indicações. Na literatura os testes exploratórios são apontados como técnica bastante utilizada em ambientes ágeis. Estes ajudam a reduzir a complexidade de planejamento de testes e a gerenciar o detalhamento das histórias [ (ISTQB, 2014 – página 48), (WATKINS, 2009 – página 48) (VEENENDAAL, 2010 – página 48) ]. Observa-se, assim, que tanto na literatura quanto na

prática, os testes exploratórios são destacados e considerados úteis para se alcançar os objetivos em equipes ágeis.

Os testes de unidade foram selecionados por 56% dos respondentes como atividade que corrobora para ter um software testado ao final de uma iteração. Estes testes são considerados importantes nos métodos ágeis, pois podem ser criados de maneira rápida, executados com antecedência e com bastante frequência (KOROŠEC; PFARRHOFER, 2015 – página 28).

A execução de testes manuais com *design* predefinido foi indicada por 55% dos respondentes. Verificando-se, assim, que apesar dos recursos disponíveis para automação e técnicas manuais, como os testes exploratórios, muitas equipes optam pela execução de casos de testes com *design* predefinido.

A criação e revisão de cenários de testes é indicada por 47% dos respondentes. Essa prática foi indicada por participantes do projeto piloto, mostrando que atividades indicadas pelos participantes do projeto piloto também são úteis para os respondentes de forma mais ampla.

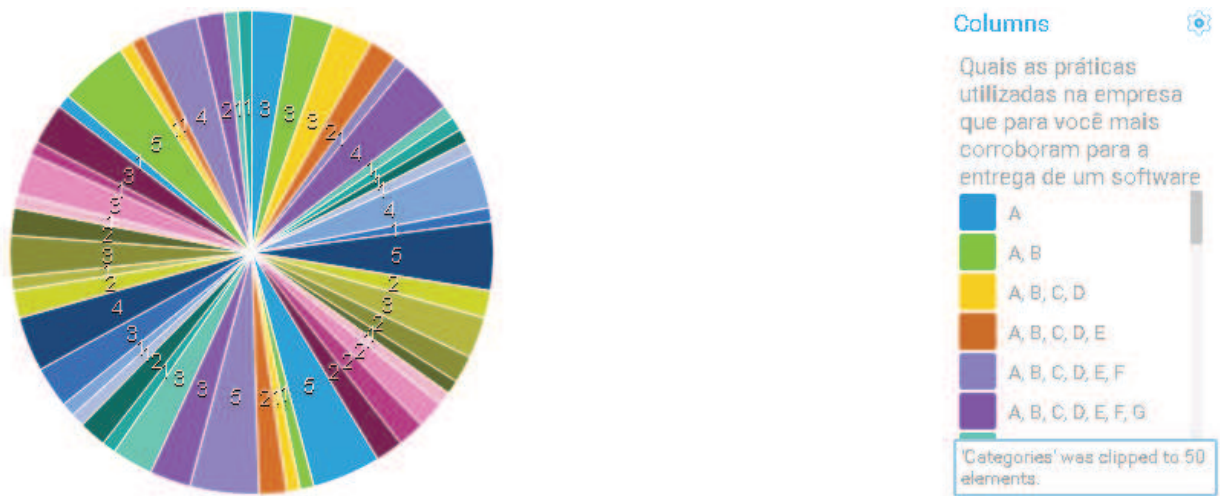
Os testes de aceitação com os *stakeholders* foram indicados por apenas 37% dos respondentes. Mostrando desta forma que as equipes dos respondentes dão pouca importância à integração dos *stakeholders* ao processo de teste de software.

As práticas de programação foram apontadas por 34% dos respondentes. Indicando que a maior parte do time se preocupa mais com a validação e verificação do produto final do que do processo como um todo.

Outras atividades não listadas, tais como integração contínua e demonstração para os *stakeholders* foram indicadas na opção aberta para comentários. Essas práticas também são importantes no processo de teste. A integração contínua para validar o desenvolvimento iterativo (EXPEDITH, 2012 – página 46) e a demonstração aos *stakeholders* que ajuda na validação do software, como citada na experiência do pesquisador (página 61).

O Gráfico 22, obtido pela ferramenta Watson *Analytics*, ilustra a falta de um conjunto de práticas específico que são selecionadas de forma contínua pelos respondentes da pesquisa, não indicando assim quais seriam as práticas que juntas poderiam corroborar mais para o software testado ao final de uma iteração.

Gráfico 22: Agrupamento de práticas que mais corroboram para o software testado.



A correlação de dados utilizou o coeficiente Spearman para correlacionar o total de fatores indicados e o tempo de experiência dos respondentes. Verificou-se baixa significância, conforme a Tabela 9.

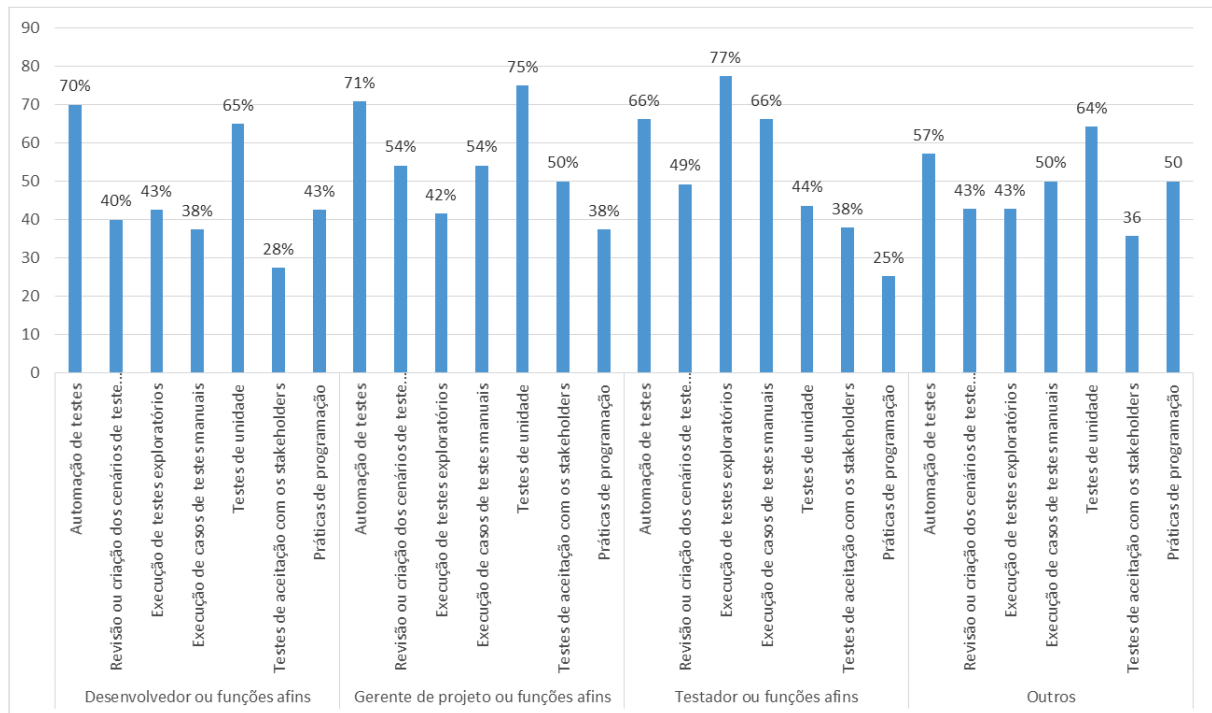
Tabela 9: Correlação de práticas de teste vs. tempo de experiência dos respondentes

	<i>Tempo de Experiência</i>
Quantidade de práticas que corroboram para o software testado	0,29

Observa-se que por análise de correlação, o impacto do tempo de experiência dos respondentes na quantidade de práticas selecionadas que mais corroboram para o software testado durante a iteração se apresenta de forma muito discreta.

Buscou-se na análise, ainda, fazer a contagem de opções por cada tipo de função, conforme o Gráfico 23.

Gráfico 23: Práticas que mais corroboram para o software testado em porcentagem vs. função



Observa-se no Gráfico 23 que algumas práticas são mais apontadas por determinadas funções: práticas de programação para os desenvolvedores, testes de aceitação com os *stakeholders* para os gerentes de projeto e testes exploratórios para os testadores. Observa-se ainda que o testador indica menos os testes de unidade. Desta forma a correlação de dados foi feita para se analisar se existe significância quanto as práticas destacadas em relação a função.

Observa-se que 43% dos desenvolvedores destacam as práticas de programação como atividade importante no processo, enquanto os gerentes as apontam em 38% e os testadores em 25%. Quando se aplica a correlação de dados usando o coeficiente de Yule com os dados da Tabela 10, em que se analisa apenas o desenvolvedor e as outras funções para a prática de programação, observa-se, no entanto que o valor de significância é de 0,24, isto é, baixa significância.

Tabela 10: Práticas de programação vs. desenvolvedor e outras funções

Prática de programação		
	Desenvolvedor	Outros
Corrobora	17	34
Não corrobora	23	75

Para os testes de aceitação verificou, em porcentagem, mais importância para os gerentes de projeto com 50%, enquanto para os testadores foi de 38% e para os programadores de 28%. Verificou-se, no entanto, que o valor do coeficiente de correlação de Yule apresentou baixa significância (-0,18), quando se analisa apenas o gerente de projetos e as outras funções, conforme dados da Tabela 11.

Tabela 11: Testes de aceitação com os stakeholders vs. gerente de projeto e outras funções

Testes de aceitação com os stakeholders		
	Gerente de projeto	Outros
Corrobora	12	74
Não corrobora	12	51

Nos testes exploratórios verificou-se maior importância para os testadores com 77% de indicações, enquanto os desenvolvedores por 43% e os gerentes de projeto por 42%. O valor de significância de 0,65 de correlação foi calculado conforme dados da Tabela 12. Este valor, embora, não seja de alta significância, mostra tendência em o testador acreditar mais no uso desta prática.

Tabela 12: Execução de testes exploratórios vs. testador e outras funções

Execução de testes exploratórios		
	Testador	Outros
Corrobora	55	33
Não corrobora	16	45

Já para os testes de unidade verificou-se menor importância para os testadores com 44% de indicações, enquanto os desenvolvedores por 65% e os gerentes de projeto por 75%. O valor de correlação de -0,46 de correlação, calculado conforme dados da Tabela 13, embora aponte que os testadores dão menos importância aos testes de unidade, não apresenta valor alto de associação.

Tabela 13: Execução de testes de unidade vs. testador e outras funções

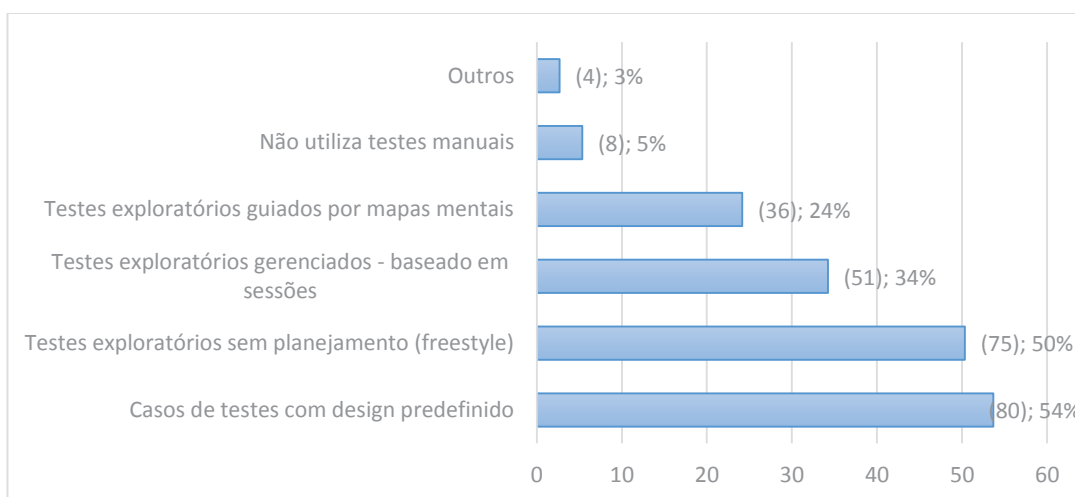
Testes de unidade		
	Testador	Outros
Corroborar	31	53
Não corroborar	40	25

Observa-se desta forma que apesar de existir diversificação de respostas, na amostra, para determinadas funções, não é possível concluir que as algumas práticas estão mais associadas ou não a uma determinada função.

### 5.2.7 Testes manuais

Em termos de abordagem manual, verificou-se que apenas 5% não utilizam testes manuais, 24% executam testes exploratórios guiados por mapas mentais, 34% executam testes exploratórios gerenciados - baseado em sessões, 50% executam testes exploratórios sem planeamento (*freestyle*) e 54% executam casos de testes com *design* predefinido. Outras formas de execução foram apontadas por 3% dos respondentes. O Gráfico 24 mostra estes valores.

Gráfico 24: Abordagem utilizada para testes manuais



Na figura 9 é mostrada a intersecção da utilização das diferentes abordagens manuais indicadas pelos respondentes da pesquisa.

Figura 9: Abordagem manual utilizada para teste – número de indicações



Nesta intersecção verifica-se em destaque a utilização de abordagens com definição de casos de testes com *design* predefinido e testes exploratórios sem planeamento (*freestyle*) por 17 respondentes. Estas duas abordagens, também, se destacam quando analisadas de forma isolada. Na qual 27 participantes utilizam de forma exclusiva os casos de testes com *design* predefinido e 26 participantes utilizam unicamente os testes exploratórios sem planeamento (*freestyle*).

Verificou-se que apesar das várias alternativas que se tem para executar o teste de forma automatizada, apenas 5% não utilizam testes manuais. Verificou-se que, ainda que exista outras formas de testes manuais mais ágeis e que requerem menos documentação, tais como os testes exploratórios e definição de cenários em alto nível (citado por vários respondentes como abordagem para execução manual), muitas equipes utilizam a execução casos de testes com *design* predefinido. Sendo essa a abordagem com a maior indicação dos respondentes (54%).

A correlação de dados foi feita, utilizando o coeficiente Yule, para analisar cada tipo de teste manual executado por diferentes características, conforme Tabela 14.

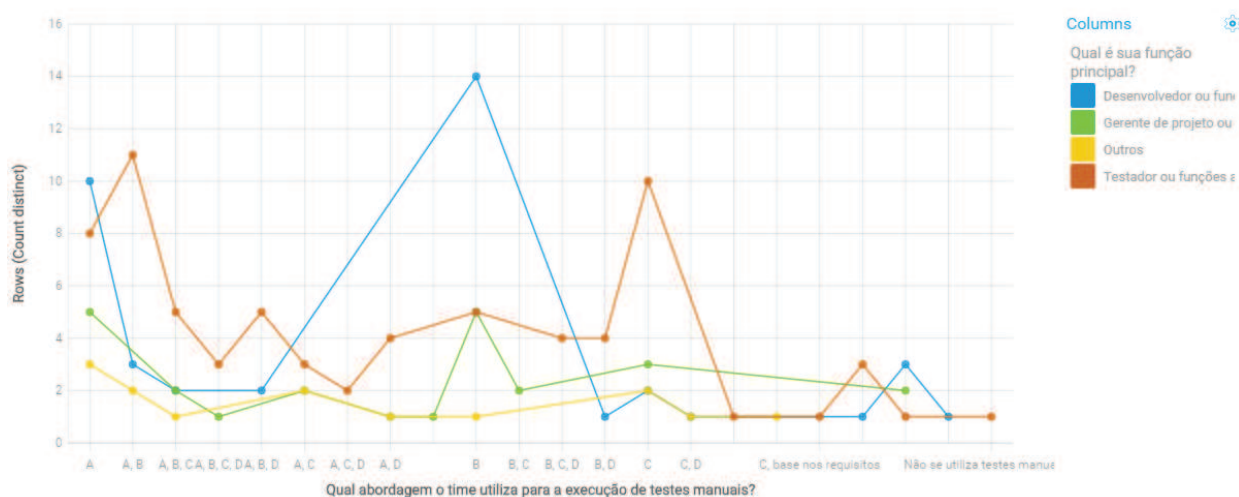
Tabela 14: Correlação da utilização de testes manuais vs. características da empresa

	Atividade da empresa	Localização dos membros
Não se utiliza testes manuais	<b>0,42</b>	-0,15
Casos de testes com design predefinido	-0,07	<b>0,30</b>
Testes exploratórios sem planejamento (freestyle)	-0,18	-0,01
Testes exploratórios gerenciados - baseado em sessões	-0,17	0,08
Testes exploratórios guiados por mapas mentais	0,08	0,08

Verifica-se, nesta amostra, baixa correlação das diferentes características dos respondentes e a forma com que se executa os testes manuais. Observa-se uma discreta tendência em empresas com atividade fim utilizar menos os testes manuais e equipes distribuídas usar mais casos de testes predefinidos.

Buscou-se fazer também uma investigação sobre a execução de testes manuais com a ferramenta *Watson Analytics* e verificou-se diferenças na perspectiva do desenvolvedor e do testador para utilização de testes exploratórios sem planejamento, não combinados com outros testes. Enquanto para os desenvolvedores esta prática é utilizada por 14 respondentes, para os testadores é utilizada por 5. Já os testes exploratórios baseados em sessões são utilizados mais pelos testadores, enquanto 10 pessoas selecionaram este tipo de teste, para os desenvolvedores apenas 2 selecionaram teste tipo de teste. Estes valores podem ser vistos no Gráfico 25.

Gráfico 25: Abordagem manual vs. função



- A - Casos de testes com design predefinido
- B - Testes exploratórios sem planejamento (freestyle)
- C - Testes exploratórios gerenciados - baseado em sessões
- D - Testes exploratórios guiados por mapas mentais



Em relação as funções e o teste manual, verifica-se diferença na tendência para os desenvolvedores, com a utilização de testes exploratórios sem planejamento e para os testadores a tendência em utilizar testes exploratórios gerenciados.

Uma vez que foi analisado esta tendência, buscou verificar se existe correlação de dados na execução de testes exploratórios sem planejamento para os desenvolvedores e testes exploratórios gerenciados para os testadores. Os dados para o cálculo do valor podem ser observados nas Tabelas 15 e 16. A total da quantidade da utilização destes testes foram somados por função, sem levar em conta o agrupamento da utilização destas práticas com outras.

Tabela 15: Testes exploratórios sem planejamento vs. desenvolvedor

Testes exploratórios sem planejamento		
	Desenvolvedor	Outros
Utiliza	23	52
Não Utiliza	17	57

Tabela 16: Testes exploratórios gerenciados vs. testador

Testes exploratórios gerenciados		
	Testador	Outros
Utiliza	29	42
Não Utiliza	22	56

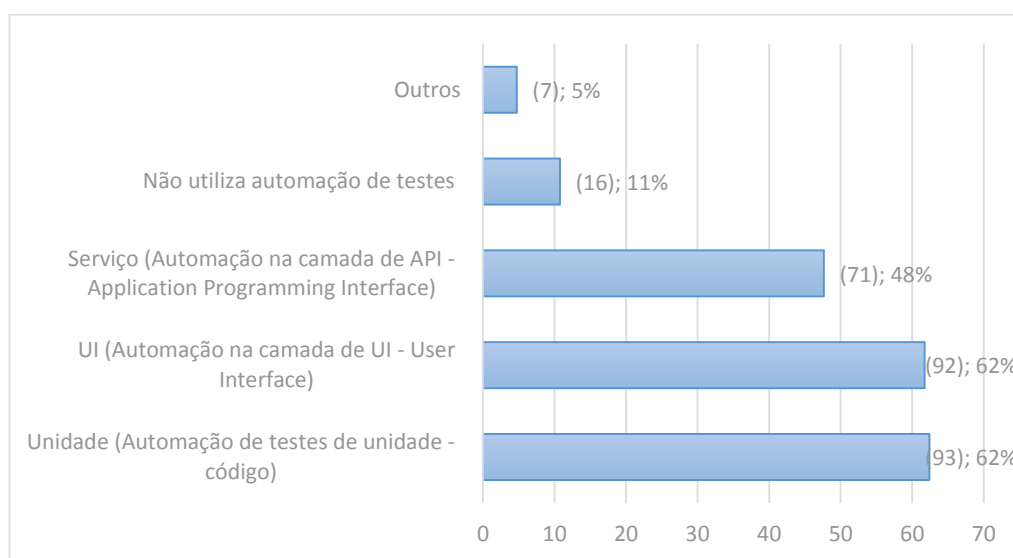
O coeficiente de Yule foi de 0,19 para os testes exploratórios sem planejamento e de 0,27 para os testes exploratórios gerenciados. Indicando, assim, baixa significância e não comprovando que estes podem ser executados diferentemente por função.

Observa-se, assim, que apesar de esperar das diferentes formas para executar os testes manuais de acordo com a função do respondente, não se observou valores de significativos que comprovem que diferentes funções executem os testes manuais de maneira diferente.

### 5.2.8 Testes automatizados

Para os testes automatizados verificou-se em quais camadas os testes eram automatizados, conforme mostrado no Gráfico 26.

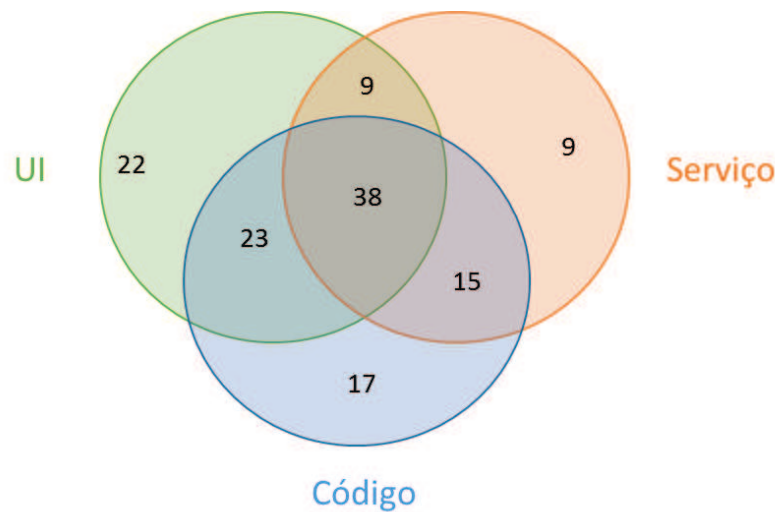
Gráfico 26: Camadas utilizadas na automação de testes



Observou-se que 11% dos respondentes não utilizam automação de testes, 48% automatizam na camada de serviço, 62% automatizam na camada de interface com o usuário e 62% na camada de código, 5% identificaram outras camadas. Observa-se que apesar da automação ser considerada a atividade que mais corrobora para um software testado, 11% dos times não utilizam automação de testes. Verifica-se, também, que apesar da literatura indicar que se deve automatizar menos na camada de interface com o usuário (COHN, 2013 – página 33), ela é utilizada por 62% dos respondentes, enquanto a camada de serviço é por 48%.

Na Figura 10 é mostrada a intersecção da utilização de camadas pelos respondentes da pesquisa no diagrama de Veen. Desta forma, verifica-se quais camadas são utilizadas em conjunto.

Figura 10: Utilização das camadas do software na automação de testes – representação por número de indicações



Observa-se que 38 respondentes utilizam as três camadas de forma conjunta na automação. A camada de UI com a camada de código é utilizada por 23 respondentes. Já a camada de API com código é utilizada por 15 respondentes e a camada de API com a de UI por 9 respondentes. De forma isolada a camada mais utilizada é a camada de UI com 22 indicações, posteriormente a camada de código com 17 indicações. Por último, a camada de API com 9 indicações.

Verifica-se na literatura que para o sucesso da automação de testes é preciso ter uma estratégia bem definida que otimize os recursos: automatizando em sua totalidade ou em grande maioria os testes de unidades, depois os testes na camada de serviço e por último na camada de interface de usuário (COHN, 2013 – página 33). A pesquisa identificou que os respondentes utilizam em sua maioria a automação de testes de unidade e de interface com o usuário. Ambos com 62% de indicações. Já o teste na camada de serviço é utilizado por 48% dos respondentes.

Quando se observa a conjunção da utilização de camadas, verifica-se que a maior parte dos respondentes utilizam as três camadas discutidas anteriormente, fator que indica que estes participantes podem ter estratégias de automação conforme o discutido na literatura. Por outro lado, quando se verifica de forma isolada, observa-se que a maior parte dos respondentes utilizam apenas a automação na camada de interface com o usuário, que são considerados por muitos especialistas como os testes mais difíceis de serem executados e automatizados.

Verificou-se, ainda, que 11% dos respondentes não utilizam nenhuma abordagem para automação. Visto que a automação impacta na entrega de valor em ágil (CRISPIN; GREGORY,

2009 - página 50), este indicativo mostra que ainda existem equipes que não conseguem se adaptar totalmente aos processos de desenvolvimento de software em métodos ágeis.

A correlação de dados, utilizando o coeficiente de Yule, foi feita para analisar se a forma de automação é impactada por diferentes características: atividade da empresa e localização, conforme Tabela 17.

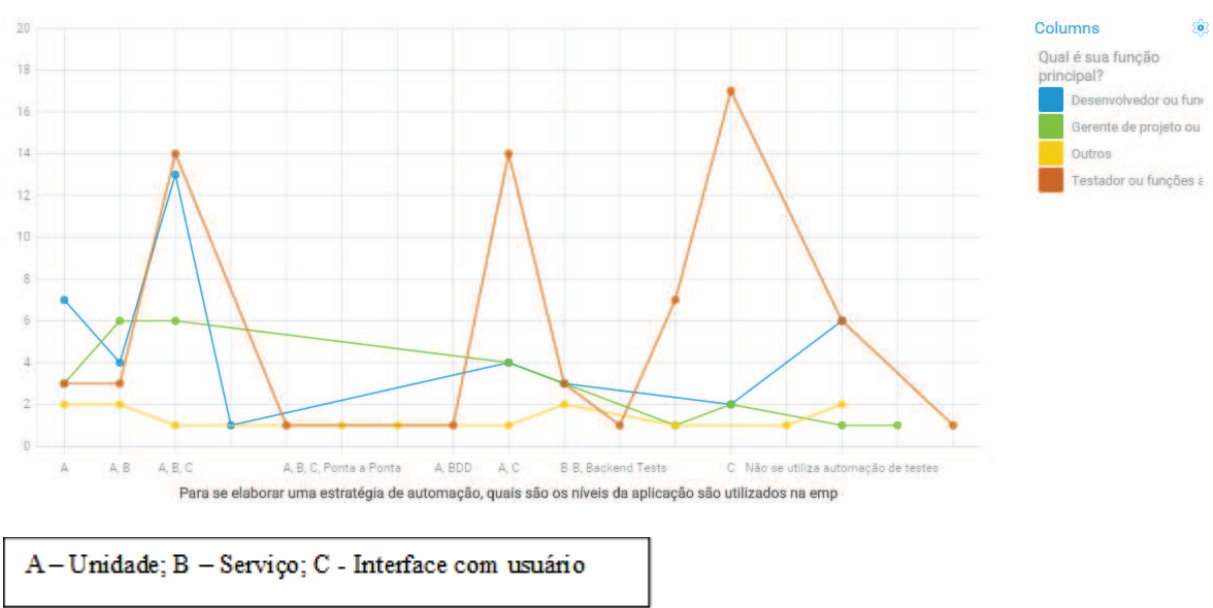
Tabela 17: Correlação de níveis do software automatizados vs. testador

	Atividade da empresa	Localização dos membros
Não se utiliza testes automatizados	<b>-0,32</b>	<b>0,30</b>
Unidade	-0,10	0,24
Serviço	-0,03	0,08
UI (User Interface)	-0,02	-0,13

Observa-se que, por análise de correlação, é muito discreta a correlação da não utilização de testes automatizados quanto a atividade da empresa, que apresenta a tendência de sua não utilização em empresas onde o desenvolvimento não é a atividade fim. A mesma discreta tendência é observada quanto a localização dos membros, na qual a não utilização de testes automatizados é menor em equipes locais do que distribuídas.

Buscou-se, também, com análise feita com a ferramenta *Watson Analytics*, verificar a execução de testes automatizados em relação à função destacada para os testadores mais automação para testes de interface com o usuário; já para os desenvolvedores verifica-se aumento na curva para a não utilização de automação de testes, conforme Gráfico 27.

Gráfico 27: Níveis do software automatizado vs. função



Analisou-se, portanto, se existe correlação significativa entre automação de testes nos níveis de interface para o testador e a não automação de testes para o desenvolvedor, conforme os dados da Tabela 18 e 19.

Tabela 18: Automação de nível de Interface vs. testador

Nível de Interface		
	Testador	Outros
Utiliza	53	39
Não Utiliza	18	39

Tabela 19: Não automação de testes vs. desenvolvedor

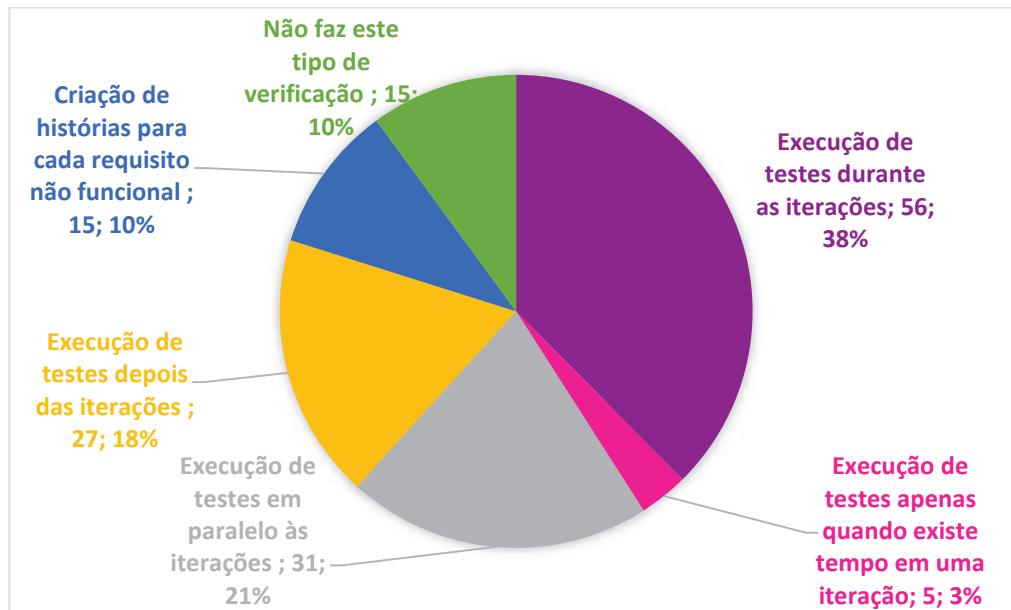
Não automatiza os testes		
	Desenvolvedor	Outros
Utiliza	6	34
Não Utiliza	10	99

Observa-se que os valores de significância, também, são baixos quando se analisa a função e possíveis tendências na automação de testes: 0,49 para o testador e os testes de interface (discreta tendência) com o usuário e 0,27 para o desenvolvedor e a não utilização de automação de testes verificando assim, que nesta amostra não se pode concluir que diferentes funções impactam na forma com que se automatiza os testes de software.

#### 5.2.9 Testes não funcionais

Na pesquisa verificou-se, também, como os testes não funcionais são executados. Observou-se que 38% dos times executam os testes não funcionais durante as iterações, 21% executam em paralelo, 18% executam os testes depois das iterações, 10% criam histórias para cada requisito não funcional, 10% não fazem este tipo de verificação e 3% executam os testes não funcionais quando se tem tempo, conforme mostrado no Gráfico 28.

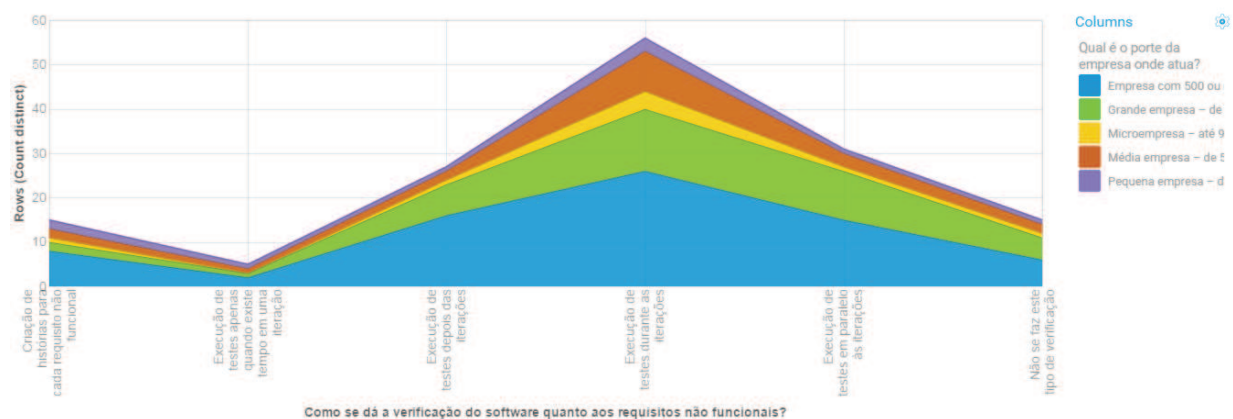
Gráfico 28: Verificação do software quanto aos requisitos não funcionais



Alguns autores [(VEENENDAAL,2010 – página 46); (EXPEDITH, 2012 – página 46)], indicam que nem sempre é possível executar os testes não funcionais dentro de uma iteração, propondo formas de execução para estes testes. Na pesquisa verificou-se que a maioria dos respondentes (38%) conseguem executar os testes não funcionais durante as iterações, 21% executam em paralelo, 18% executam os testes depois das iterações, 10% criam histórias para cada requisito não funcional, 10% não fazem este tipo de verificação e 3% executam os testes não funcionais quando se tem tempo.

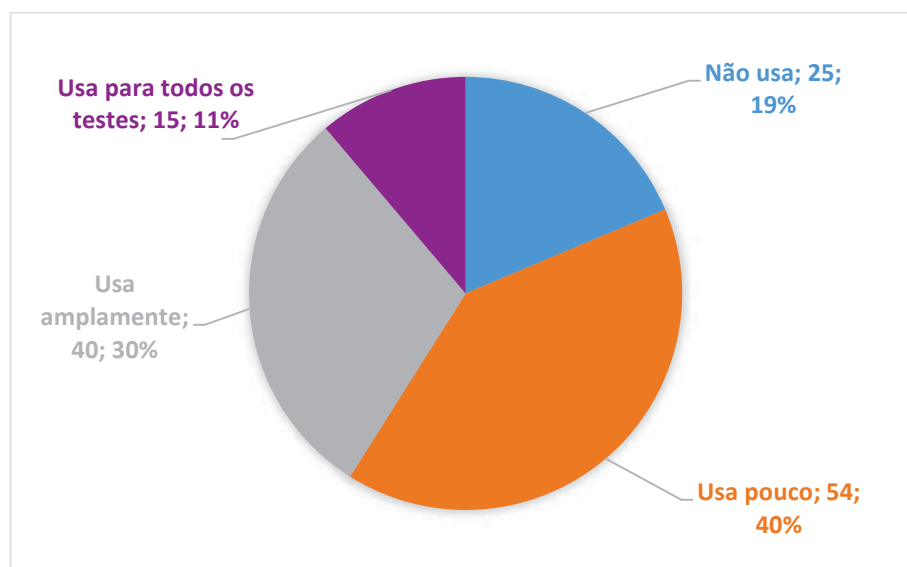
A ferramenta *Watson Analytics* foi utilizada para análise e verificou tendência na forma como os testes são executados pelas empresas, conforme pode ser visto no Gráfico 29 que mostra a distribuição na forma que os testes são executados por porte de empresa, no qual destaca-se a execução durante a iteração.

Gráfico 29: Execução de testes não funcionais vs. porte da empresa



No contexto de execução de testes não funcionais também foi analisado como as ferramentas são utilizadas. Para aqueles que fazem os testes não funcionais foi perguntado sobre a utilização destas e os resultados podem ser observados no Gráfico 30.

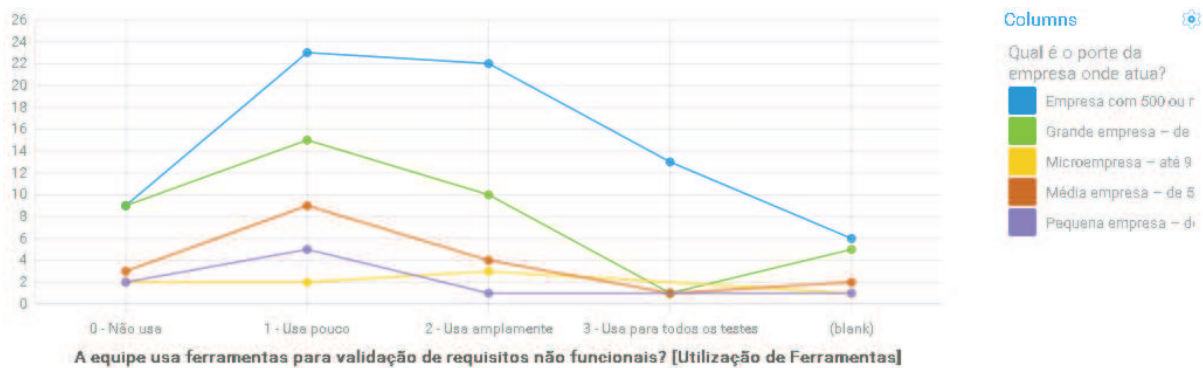
Gráfico 30: Utilização de ferramentas na execução de testes não funcionais



Crispin e Gregory (2009) destacam (página 46) que a utilização de ferramentas é essencial para o sucesso da execução de testes não funcionais. Na pesquisa observou-se que 19% não utilizam ferramentas, 40% usam pouco ferramentas, 30% usam amplamente e 11% utilizam para todos os testes. Observa-se que apesar das indicações para o uso de ferramentas neste processo, mais da metade (59%) ainda utilizam pouco ou não utilizam ferramentas para o processo de teste não funcional.

Utilizando a ferramenta de análise *Watson Analytics* verificou-se que empresas com 500 ou mais funcionários utilizam mais as ferramentas do que os outros portes, observa-se que estas empresas, no entanto, tiveram mais participação na pesquisa. O Gráfico 31 mostra o número de respondentes de acordo com o porte da empresa e a utilização de ferramentas. Os valores mostrados como *blank* fazem parte daqueles que não responderam à pergunta por não fazer testes não funcionais. O Gráfico 31 mostra, ainda, maior diferença de proporção quando se utiliza ferramentas para todos os testes.

Gráfico 31: Utilização de ferramentas para requisitos não funcionais vs. porte da empresa.



Uma vez que a ferramenta de análise do Watson *Analytics* apontou indícios de diferenças para a utilização de ferramentas para empresas com mais de 500 funcionários, verificou em termos de correlação destas variáveis (utilização para todos os testes e empresa com mais de 500 funcionários) e verificou o valor de coeficiente de correlação de Yule com significância de 0,78. Indicando que, nessa amostra, as grandes empresas, com mais de 500 funcionários tem mais capacidade de utilizar ferramentas para todos os testes não funcionais.

Outras análises utilizando as diversas características das empresas e dos respondentes foram verificados, mas não foram encontrados evidências de que estas poderiam ser diferentes do padrão de respostas.

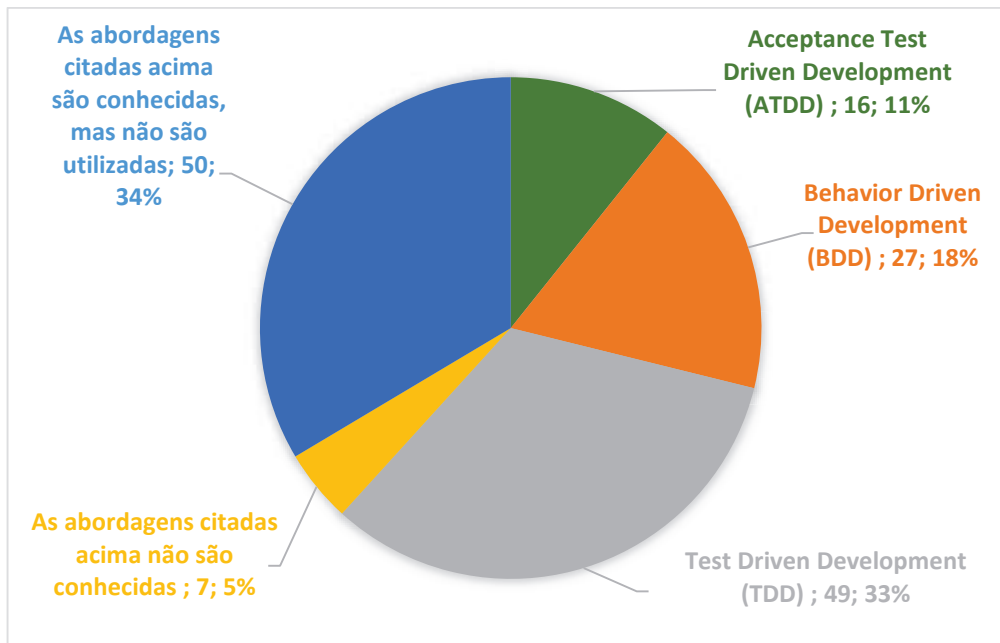
#### 5.2.10 Abordagens de desenvolvimento ágil

Várias abordagens podem ser utilizadas com intuito de otimizar o processo de desenvolvimento. Essas abordagens são abordadas de maneira frequente na literatura [(BECK, 2002 – página 42), (DELAMARO et al., 2009 – página 43), (CHELIMSKY et al., 2010 – página 43) (NORTH, 2012 - página 44) (GARTNER, 2013 – página 43), (AGARWAL; DEEP, 2014 – página 42), (ISTQB, 2014 – página 42)] e por isso a pesquisa buscou verificar se as equipes as utilizam durante o processo desenvolvimento. Observou-se, no Gráfico 33, que 11% utilizam o Acceptance Test Driven Development (ATDD), 18% utilizam o Behavior Driven Development (BDD), 33% Test Driven Development (TDD), 34% conhecem as abordagens



acima citadas, mas não utilizam para desenvolvimento e 5% não conhecem as abordagens acima citadas.

Gráfico 32: Abordagens utilizadas no desenvolvimento ágil



A pesquisa identificou, também, quais abordagens utilizadas no processo de desenvolvimento: ATDD, BDD e TDD. Essas abordagens se mostram importantes para conseguir se adequar aos princípios ágeis (ISTQB, 2014). Observou-se que 11% utilizam o Acceptance Test Driven Development (ATDD), 18% utilizam o Behavior Driven Development (BDD), 33% Test Driven Development (TDD). Verificou-se, desta forma, que a maior parte dos respondentes (62%) utilizam algum tipo abordagem para o desenvolvimento de software.

Na análise de dados verificou-se se a utilização de abordagens de desenvolvimento impactava na capacidade de testar o software durante toda a iteração. Os valores absolutos daqueles que conseguem testar durante as iterações são mostrados na Tabela 20 e posteriormente a porcentagem é calculada para utilização ou não de abordagens para o desenvolvimento de software.

Tabela 20: Estratégias de teste vs. capacidade de testar durante toda a iteração

	Estratégias ágeis		Não consegue testar durante toda iteração	Consegue testar durante toda iteração
	Não consegue testar durante toda iteração	Consegue testar durante toda iteração		
Acceptance Test Driven Development (ATDD)	3	13	12	80
Behavior Driven Development (BDD)	3	24		
Test Driven Development (TDD)	6	43		
As abordagens citadas acima não são conhecidas	2	5	12	45
As abordagens citadas acima são conhecidas, mas não são utilizadas	10	40		

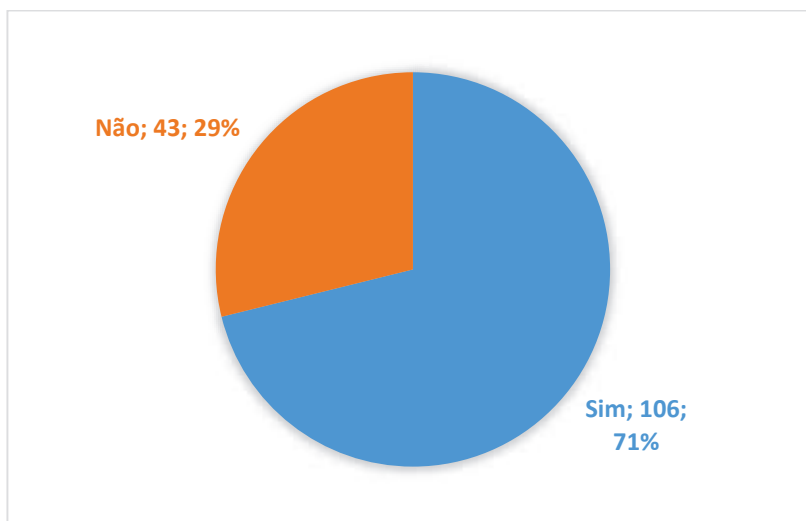
Com os valores apresentados na Tabela 20 verificou-se que 86,97% das equipes que utilizam alguma abordagem ágil conseguem executar o teste durante toda a iteração, enquanto 78,94% que não utilizam a abordagem conseguem testar durante toda a iteração. Verifica-se uma porcentagem maior de times que executam o teste continuamente quando se utiliza alguma abordagem ágil para o desenvolvimento.

Para verificar a correlação destes valores foi utilizado o cálculo de coeficiente de Yule e chegou-se a um valor de  $Y = -0,28$ . Verifica-se que o valor de correlação indica independência entre as variáveis, não sendo possível afirmar, nessa amostra, que a utilização de abordagens esteja relacionada com a capacidade de se testar durante toda a iteração.

### 5.2.11 Gerenciamento de testes em ambientes ágeis

A pesquisa investiga, ainda, como é o ambiente de teste das equipes de desenvolvimento ágil e verifica que 71% possuem um ambiente de teste gerenciável, com estratégia de teste, monitoramento e controle de teste. Já 29% não possuem um ambiente gerenciável conforme o Gráfico 33.

Gráfico 33: Possuem ambiente de teste gerenciável



Para a análise dos dados foi feita a correlação usando o coeficiente de Yule para verificar se existe impacto no gerenciamento de testes quanto ao envolvimento do *stakeholder* (1 para quando se consegue amplamente e totalmente e 0 para quando não se consegue ou consegue parcialmente), a presença do testador na equipe, a capacidade de se testar o software durante toda a iteração e a execução ou não de testes automatizados.

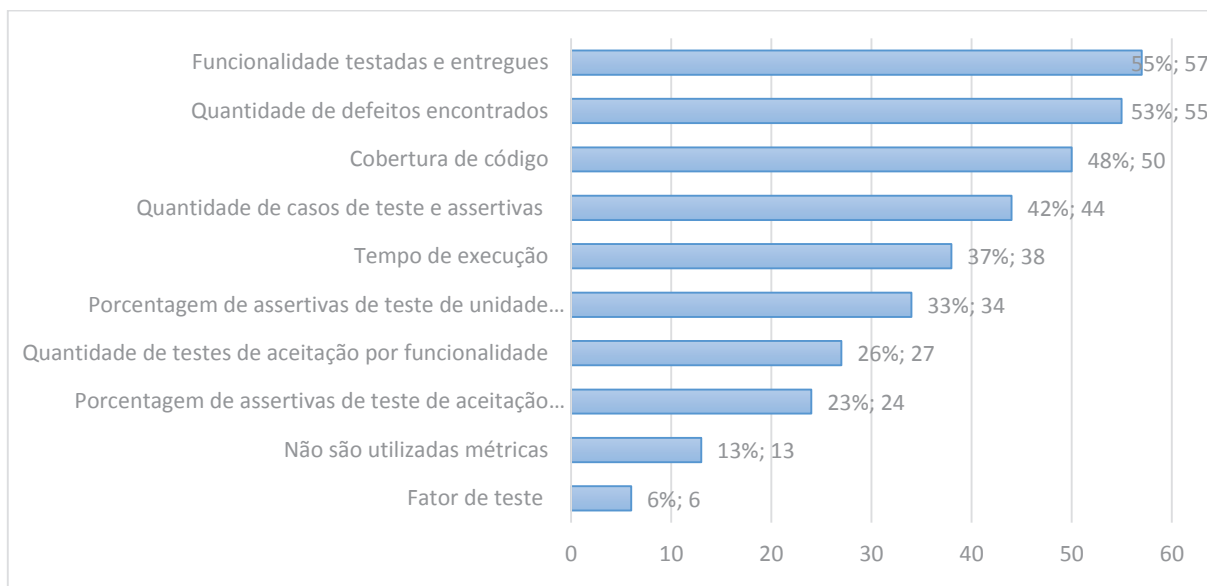
Tabela 21: Correlação de ambiente de teste vs. características do ambiente

	<i>Envolvimento do Stakeholder</i>	<i>Presença do testador</i>	<i>Realiza teste durante toda a iteração</i>	<i>Não automatiza testes</i>
Ambiente de Teste Gerenciável	0,04	0,29	0,23	-0,58

Observou-se na Tabela 21 que as características analisadas, nesta amostra, não mostraram coeficiente de correlação significativo na gestão de ambientes de teste. Observa-se, apenas, que apesar de correlação não forte para automação de testes, verifica-se que a utilização deste pode impactar no ambiente de teste

Para todos aqueles que possuem um ambiente gerenciável, verificou-se que quais métricas são utilizadas para a gestão conforme o Gráfico 34.

Gráfico 34: Métricas de testes utilizadas por equipes de desenvolvimento ágil



Observou-se, no Gráfico 34, que o indicador mais utilizado é a funcionalidade testada e entregue com 55%, posteriormente os indicadores: quantidade de defeitos encontrados com 53%, cobertura de código com 48%, quantidade de casos de teste e assertivas com 42%, tempo de execução com 37%, porcentagem de assertivas de teste de unidade passando e falhando com 33%, quantidade de testes de aceitação por funcionalidade com 26%, porcentagem de assertivas de teste de aceitação passando e falhando com 23%, fator de teste com 6%.

Observou-se que o indicador mais utilizado é a funcionalidade testada e entregue com 55%. Desta forma, é possível verificar o valor de negócio entregue em cada funcionalidade do sistema com objetivo de verificar se as necessidades dos clientes foram alcançadas.

A quantidade de defeitos encontrados foi selecionada por 53% dos respondentes. Ao se analisar um defeito é possível verificar várias informações, tais como o nível do defeito, a origem, a severidade, facilidade de remoção, complexidade, confiabilidade, manutenção, cronograma, orçamento, portabilidade e conformidade com os requisitos. A satisfação do usuário em relação ao software pode ser medida de várias formas, em relação aos defeitos encontrados em um software, é possível dizer que quanto menos defeitos encontrados, mais satisfeitos os usuários estarem (JONES, 2008). Além de considerar a satisfação do cliente e também o risco de software falhar em operações que requerem que o software funcione com a precisão, deve-se analisar o custo de um defeito. Verifica-se que quanto antes um defeito é descoberto, mais barato ele é para ser consertado (GRAHAM, 2008). Considerando o risco de

um defeito, custo e satisfação do cliente quando não encontra defeitos este é um indicador que traz bastante valor às equipes de desenvolvimento ágil.

Observa-se, também, que uma das formas de verificar a amplitude do que foi testado é analisar qual a cobertura de código-fonte utilizando-se critérios estruturais e assim atingir a cobertura necessária para a validação do software. Este indicador foi selecionado por 48% dos respondentes e se mostra bastante útil no desenvolvimento de software ágil.

A quantidade de casos de teste e assertivas foi indicada por 42% dos respondentes. Essa é uma forma de verificar quanto trabalho tem sido feito para se ter um software testado ao final de uma iteração.

Os métodos ágeis devem ter apenas as disciplinas necessárias para se alcançar as necessidades dos clientes. Fazer um plano de projeto e desenvolver um software com alta qualidade para que se atenda às necessidades de mercado rapidamente com soluções inovadoras (RICO, *et al.*, 2009) uma vez que os métodos ágeis devem ser eficientes, o tempo de teste deve ser verificado a cada iteração para que se possa verificar quanto se tem gasto com esta atividade e assim refletir sobre os processos e melhorias que possam ser feitas no projeto. O indicador de tempo de execução foi selecionado por 37% dos respondentes.

Já a porcentagem de assertivas de teste de unidade passando e falhando foi selecionada por 33% dos respondentes. Como já observado anteriormente os testes de unidade são mais fáceis de serem criados e mais rápidos de serem executados. Os indicadores trazidos por eles podem ajudar as equipes sobre o processo de teste de software.

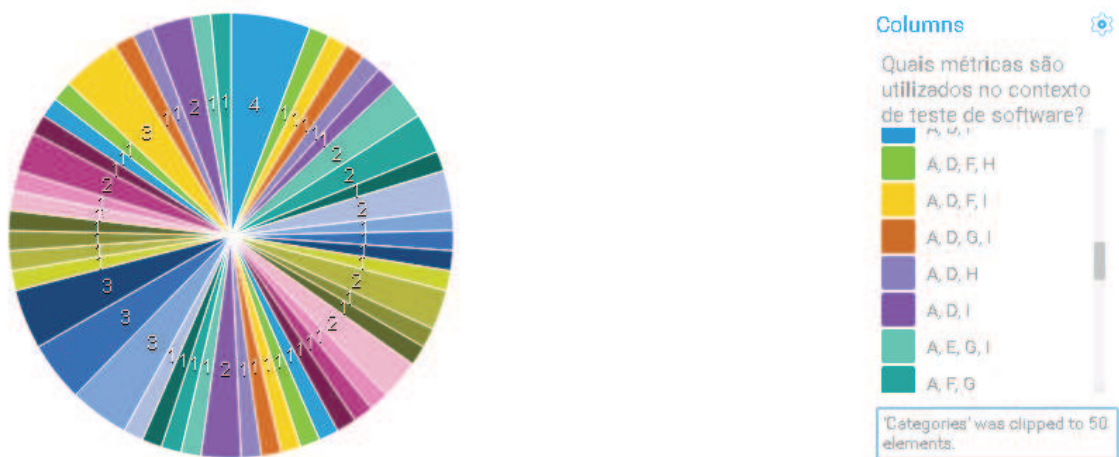
A quantidade de testes de aceitação por funcionalidade é importante para verificar a quantidade de teste gerada por funcionalidade para demonstração para os clientes. Este indicador foi selecionado por apenas 26%. Neste mesmo contexto verifica-se a porcentagem de assertivas de teste de aceitação passando e falhando com 23% de indicação. Estes testes são importantes para alinhar se as necessidades dos clientes estão sendo alcançadas.

O fator de teste foi indicado por 6%. Essa métrica indica a relação entre o tamanho do código de testes e tamanho do código em produção. Métrica que pode ser utilizada quando se cria testes automatizados e assim verificar o quanto de esforço está sendo feito para criação de teste e de código.

Apesar desta pergunta ser respondida por pessoas que possuíam um ambiente de teste gerenciável, 13% não utilizam nenhum tipo de métrica para teste.

O Gráfico 35 obtido pela ferramenta *Watson Analytics* ilustra a falta de um conjunto de métricas que se destacam na seleção feita pelos respondentes, mostrando assim que, na amostra, não existe um padrão de métricas utilizadas de forma conjunta.

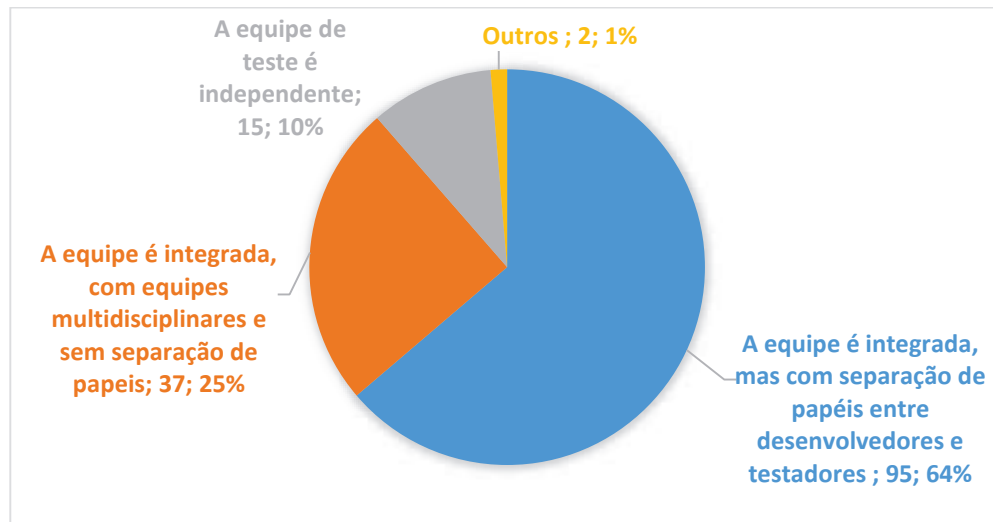
Gráfico 35: Métricas utilizadas em teste de software em ambientes ágeis. Contagem por agrupamento.



#### 5.2.12 Formação de equipe ágeis

A pesquisa investigou ainda como são formadas as equipes de desenvolvimento ágil. Desta forma, pode-se verificar a organização dos times que compõem a amostra. Observou-se, conforme o Gráfico 36, que a equipe ágil dos respondentes é composta em 64% dos casos por equipes integradas, mas com separação de papéis entre desenvolvedores e testadores. Em 25% ela é integrada, com equipes multidisciplinares e sem separação de papéis. Em 10% dos casos, apesar de se definir como equipe de desenvolvimento ágil, são compostas por equipes de independentes. Outros formatos foram apontados por apenas 1% dos respondentes.

Gráfico 36: Formação das equipes ágeis



Observa-se que 64% dos casos as equipes são integradas, mas com separação de papéis entre desenvolvedores e testadores. Destaca-se que em métodos ágeis as equipes devem ser colaborativas (MYERS *et al.*; 2011 – página 53) e que ter a diferença de papéis pode trazer benefícios ao processo de desenvolvimento (BROEK *et al.*; 2014 – página 53).

Em 25% ela é integrada, com equipes multidisciplinares e sem separação de papéis. Destaca-se que as competências individuais são fator crítico para o sucesso no desenvolvimento ágil (COCKBURN; HIGHSMITH, 2001 – página 51).

Em 10% dos casos as equipes de testes independentes foram selecionadas. Apesar da integração com a equipe estar comprometida devido a independência da equipe de teste, observa-se que estas organizações trazem alguns benefícios, quanto a gestão e eficiência ao processo de teste de software (MYERS *et al.*; 2011- página 52).

A correlação de dados foi utilizada para analisar se equipes de teste independente possuíam ambiente de teste mais gerenciável que as outras equipes, esta correlação, teve significância baixa de 0,07, mostrando assim que para os participantes da pesquisa a equipe de teste independente e a gestão do ambiente de teste não estão correlacionadas.

A análise de correlação também foi realizada para analisar se a presença de pessoas dedicadas ao teste de software, isso é em equipes integradas, mas com separação de papéis e equipes de testes independentes impactavam na gestão de ambiente de teste e constatou-se, também, baixa significância de 0,15.

Para se analisar possíveis diferenças no processo de teste de acordo com a formação da equipe, verificou-se, também, a capacidade de testar o software durante toda a iteração para cada tipo de formação de equipe conforme a Tabela 22.

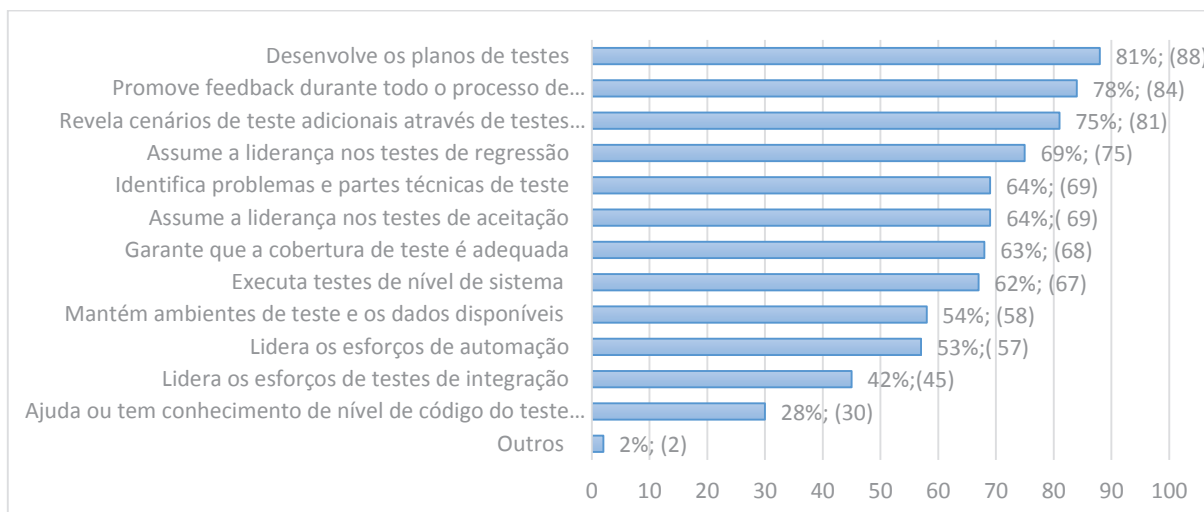
Tabela 22: Organização das equipes vs. capacidade de testar durante toda a iteração

Como se organiza as equipes de desenvolvimento em relação ao teste de software?		
	Não consegue testar durante toda iteração	Consegue testar durante toda iteração
A equipe é integrada, mas com separação de papéis	15	80
A equipe é integrada, com equipes multidisciplinares e sem separação de papeis	7	30
A equipe de teste é independente	2	13
Outros	0	2

Com os valores da Tabela 24 verificou-se que para as equipes integradas com separação de papéis 84,21% conseguem testar durante todo o período, em equipes integradas, mas com separação de papéis 81,08% conseguem testar durante todo o período e equipes independentes 86,67% conseguem testar durante toda a iteração. Observa-se que diferença de proporções nestas equipes são baixas e por isso não é possível dizer que a formação da equipe impacta diretamente em como os testes são executados.

Para as equipes que tenham pessoas dedicadas ao teste de software, a pesquisa investigou qual a contribuição deste no desenvolvimento ágil e verificou as principais colaborações conforme o Gráfico 37.

Gráfico 37: Papel do testador em equipes de desenvolvimento ágil





Em relação ao papel do testador nas equipes identificou várias atividades selecionadas no processo, em média 7,34, atividades foram selecionadas pelos respondentes conforme destacado a seguir:

- Desenvolver planos de testes (81%),
- Promover feedback durante todo o processo de desenvolvimento (78%),
- Revelar cenários de teste adicionais através de testes exploratórios (75%),
- Liderar os testes de regressão (69%),
- Identificar problemas e partes técnicas de teste (64%),
- Liderar os testes de aceitação (64%),
- Garantir que a cobertura de teste é adequada (63%),
- Executar testes de nível de sistema (62%),
- Manter ambientes de teste e os dados disponíveis (54%),
- Liderar os esforços de automação (53%),
- Liderar os esforços de testes de integração (42%),
- Ajudar ou ter conhecimento de nível de código do teste a ser realizado (28%)

Não foram correlacionados dados nas atividades do testador nas equipes, o objetivo foi apenas para verificar quais são as principais contribuições deste na equipe de desenvolvimento ágil.

Com os resultados apresentados na pesquisa, verificou-se, na amostra, como as atividades de testes são executadas pelos participantes do processo produtivo. A pesquisa abordou vários fatores que acercam as equipes ágeis e desta forma pode atingir o objetivo da pesquisa que foi verificar como atividades de teste têm sido implementadas em equipes de desenvolvimento de software que utilizam métodos ágeis de desenvolvimento.

## 6 CONCLUSÃO

O presente trabalho possibilitou verificar como os participantes do processo produtivo executam o teste de software em ambientes de desenvolvimento ágil. A partir da pesquisa bibliográfica que permitiu identificar as práticas, técnicas e abordagens indicadas na execução de teste ágil e da experiência na implantação de métodos ágeis sob a perspectiva do teste de software do pesquisador, foi possível criar um questionário para a pesquisa de identificação.

O relato mostra que, apesar de utilizar vários conceitos do desenvolvimento ágil, as equipes sofrem um processo de adaptação na sua implantação que nem sempre permite o alinhamento total no teste de software.

Na pesquisa realizada por meio de questionário, foi possível verificar as práticas e abordagens mais utilizadas no teste de software em equipes de desenvolvimento ágil. No entanto, percebe-se pela pesquisa que algumas práticas sugeridas na literatura não são utilizadas de forma ampla pelas equipes de desenvolvimento ágil selecionadas nesta amostra.

Verificou-se, também, que muitas das práticas utilizadas não estão aderentes aos métodos ágeis. Observa-se equipes que não utilizam testes automatizados, equipes utilizando estratégias de teste manuais com *design* predefinido, sendo este o mais utilizado, além de baixa utilização de ferramentas para testes não funcionais. Práticas que não corroboram para a implantação eficiente de métodos ágeis e, por isso, muitas equipes não conseguem adaptar o teste de software de forma eficaz em ambientes de desenvolvimento ágil.

Ao verificar as diversas práticas que podem ser utilizadas no processo de testes de software em ambientes ágeis, observou-se que no planejamento das iterações se prioriza a criação e a execução de testes de histórias e a validação do negócio. Em relação ao tempo, verificou-se que a maior parte dos respondentes acredita que o tempo ideal para se codificar e testar um software é de duas semanas. Verifica-se ainda que os testes são realizados durante toda a iteração por grande parte das equipes.

Em relação aos conceitos de níveis de testes, verificou-se, na amostra, que estes também são largamente utilizados em ambientes ágeis e que o nível menos executado é o de teste de aceitação. Observando a integração das atividades de teste com o envolvimento dos *stakeholders*, verificou-se que a maior parte dos respondentes não consegue integrá-los ao processo de teste de software de forma ampla ou total.

Já na identificação de práticas que corroboram para o software testado ao final de uma iteração, verificou-se que automação de testes é a atividade mais indicada pelos participantes. Em relação à utilização prática dos testes, observou-se que, para práticas manuais, muitas equipes ainda utilizam a execução de casos de testes com *design* predefinido. Já na estratégia de automação, verificou-se que as camadas do software com mais automação de testes são as camadas de código e de interface do usuário.

A pesquisa não se limitou em verificar apenas como os testes funcionais eram executados, verificou-se, também, o âmbito não funcional. Observou-se que os times conseguem fazer essa verificação não funcional durante as iterações, depois das iterações ou em iterações paralelas. Verificou-se, no entanto, que a utilização de ferramentas, conforme recomendado na literatura (CRISPIN e GREGORY, 2009) para estes testes, não é utilizada largamente pelas equipes.

Analizou-se inclusive a utilização de abordagens de desenvolvimento de software e verificou-se que o TDD é a abordagem mais utilizada (33% dos respondentes) e que, ainda, apesar destas práticas serem conhecidas, 34% não as utilizam.

Verificou-se que 71% dos participantes possuem um ambiente de teste gerenciável, no qual a funcionalidade entregue e testada é a métrica mais utilizada pelas equipes. As equipes em geral são integradas, mas com separação de papéis. O testador nessas equipes tem como principal função prover *feedback* durante todo o processo de desenvolvimento.

A pesquisa buscou identificar, ainda, como o processo de teste era executado por diversas equipes ágeis, verificando, desta forma, o impacto das funções e as diferentes formações de equipes, assim como a diversificação no planejamento de testes, a função do tempo neste processo e a formação do indivíduo, o conhecimento de abordagens ágeis e sua execução. Com a análise feita não foi possível concluir, na amostra, que diferentes características estudadas impactam na forma em que o teste é executado. Observa-se a pluralidade de padrões de respostas ao questionário que mostra a diversidade na aplicação no desenvolvimento usando métodos ágeis.

Além das contribuições já citadas, a pesquisa ainda alerta e informa as empresas sobre práticas e estratégias que podem ser modificadas de forma a se tornarem mais alinhadas aos valores ágeis e obterem eficiência no processo de teste de software. Tais práticas e estratégias são levantadas no referencial teórico deste trabalho.

Observa-se que, de forma geral, as equipes participantes desta pesquisa utilizam parcialmente práticas que estejam alinhadas aos valores ágeis. Ao utilizar apenas alguns conceitos dos métodos ágeis e manter a cultura dos métodos tradicionais ou se adaptar parcialmente ao novo paradigma as equipes podem não obter as vantagens, também descritas neste trabalho, na utilização dos métodos ágeis.

A alta dependência dos testes manuais, estratégias de automação não muito eficientes e pouca utilização de ferramentas na execução de testes não funcionais identificados neste trabalho dão margem a pesquisas futuras para verificar quais as dificuldades das equipes em aderir a estes processos.

A dificuldade em integrar os *stakeholders* no processo de teste pode ser investigado em trabalhos futuros e, assim, verificar quais as dificuldades nessa integração e os impactos desta falta de envolvimento na qualidade final do software. Além disso, pode-se propor formas de melhorar a integração dos *stakeholders* e as pessoas que executam o teste de software no processo ágil.

A quantidade de dados levantada e as contribuições dessa pesquisa dão margem a uma série de outras possíveis pesquisas, tanto quantitativas quanto qualitativas, que venham a analisar aspectos específicos levantados aqui.

## REFERÊNCIAS

- AGARWAL, N; DEEP, P. Obtaining Better Software Product by Using Test First Programming Technique. **IEEE**, 2014
- AGARWAL, A; GARG, N,K; JAIN, A. Quality Assurance for Product Development using Agile. **IEEE**, 2014
- ALSMADI, I. **Advanced Automated Software Testing: Frameworks for Refined Practice**. 1. ed. Hershey: IGI Global, 2012.
- AMBLER, S. **Agile Adoption Rate Survey Results: February 2008**. Disponível em: <<http://www.ambysoft.com/surveys/agileFebruary2008.html>>. Acesso em: 04/06/2015., 2008.
- \_\_\_\_\_. **Agile Testing and Quality Strategies: Discipline Over Rhetoric**. Disponível em: <<http://www.ambysoft.com/essays/agileTesting.html>>. Acesso em: 10/09/2015., 2014.
- \_\_\_\_\_. S. **Results from the November 2012 Agile Testing Survey**. Disponível em: <<http://www.ambysoft.com/surveys/agileTesting201211.html>>. Acesso em: 08/07/2015., 2012.
- \_\_\_\_\_.; LINES, M. **Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise**. IBM Press, 2012.
- AMMANN, P; OFFUTT, J. **Introduction to Software Testing**, 1. ed. New York: Cambridge University Press, 2008.
- ANDERSON, DJ. **Kanban: Successful Evolutionary Change for Your Technology Business**. Washington: Blue Hole Press, 2010.
- BASSI, DL. **Experiências com desenvolvimento ágil**. Dissertação de Mestrado apresentada ao Instituto de Matemática e Estatística da Universidade de São Paulo. São Paulo, 2008.
- BACH, J. **Exploratory testing explained**, 2003. Disponível em <<https://www.satisfice.com/articles/et-article.pdf>> Acesso em: 01/05/2015.
- BECK, *et al.*. **Manifesto for Agile Software Development**, 2001. Disponível em: <<http://Agilemanifesto.org>>. Acesso em: 06/06/2014.
- \_\_\_\_\_. **Extreme Programming Explained: Embrace Change**. Boston: Addison Wesley, 2004.
- \_\_\_\_\_. **Test Driven Development: By Example**. Boston: Addison Wesley, 2002.
- BERNARDO, PC; KON, F. A Importância dos Testes Automatizados. **Engenharia de Software Magazine**, 2008.
- BORIA, *et al.* **Melhoria de Processos de Software com Métodos Ágeis e Modelo MPS**.

PBQP Software, 2013.

BROEK, RVD. *et al.*. Integrating Testing into Agile Software Development Processes. **IEEE: Model-Driven Engineering and Software Development (MODELSWARD)**, 2014

BRYMAN, A; EMMA, B. **Business Research Methods**. Oxford University Press, 2003.

BURNSTEIN, I. **Practical Software Testing: A Process-Oriented Approach**. New York: Springer, 2003.

CARTER, J. The agile tester. **VersionOne**, 2010.

CAVALCANTE, AM; CORREIA, IB; KARLSSON, BF; SANTOS, AM; SILVA E. Testing in an Agile Product Development Environment: An Industry Experience Report. **IEEE**, 2011.

CHELIMSKY, *et al.* **The RSpec Book Behaviour-Driven Development with RSpec, Cucumber, and Friends** Pragmatic Bookshelf, 2009.

CRISPIN, L; GREGORY J. **Agile Testing: A Practical Guide for Testers and Agile Teams**. 1. ed. Boston: Addison Wesley, 2009.

\_\_\_\_\_. **More Agile Testing: Learning Journeys for the whole team**. 1. ed. Boston: Addison Wesley, 2015.

COCKBURN, A; HIGHSMITH, J. Agile Software Development: The People Factor. **Software Management**, 2001.

COHN, M. **Succeeding with agile. Software development using scrum**. Boston: Addison Wesley, 2013

COLLINS, EF; LUCENA, VFJ. Software Test Automation Practices in Agile Development Environment: An Industry Experience Report. **IEEE**, 2012.

COPELAND, L. **A Practitioner's Guide to Software Test Design**. Boston: Artech House, 2004.

DSDM Public Version 4.2 Manual. (n.d.). DSDM Consortium - Enabling Business Agility. Disponível em: <<http://www.dsdm.org/version4/2/public/default.asp>> Acesso em: 01/04/2015

DAVIS, B. **Agile Practices for Waterfall Projects: Shifting Processes for Competitive Advantage**. J. Ross Publishing: 2013.

DELAMARO, M, E; MALDONADO, J, C; JINO, M. **Introdução ao Teste de Software**. Rio de Janeiro: Elsevier, 2007.

DELAMARO, M, E; *et al.* Uma Revisão Sistemática sobre a atividade de Teste de Software no contexto de Métodos Ágeis. **XXXV Conferencia Latino Americana de Informática**. Pelotas: CLEI, 2009.

EKAS, L; WILL, S. Being Agile: Eleven Breakthrough Techniques to Keep You From

Waterfalling Backwards. IBM Press, 2014.

EXPEDITH, MVL. Agile Testing: Key Points for Unlearning. **Scrum Alliance**. 2012. Acesso em 05/04/2014. Disponível em: <https://www.scrumalliance.org/community/articles/2012/january/agile-testing-key-points-for-unlearning>

FABBRI, SCPF; VINCENZI, AMR; MALDONADO, JC. **Introdução ao teste de software**, cap. 2 (Teste Funcional) Rio de Janeiro: Elsevier, 2007.

GARTNER, M. **ATDD by example**. Westford, Massachusetts: Pearson Education, 2013.

GRAHAM, D; *et al.* **Foundations of Software Testing: ISTQB Certification**. 2. ed. London. Cengage Learning. 2008.

GERAS, Adam. Leading Manual Test Efforts with Agile Methods. *Agile 2008 Conference*, 2008.

HANSMANN, U; STOBER, T. **Agile Software Development: Best Practices for Large Software Development Projects**. 1. ed. New York: Springer, 2010.

HARTMANN, D; DYMOND, R. Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. Proceedings of Agile 2006 Conference. Minnesota, USA, Washington, DC, USA: **IEEE**, 2006.

HERZLICH, P. **The Need For Software Testing**. Ovum Summit. 2007.

HIGHSMITH, J. **Adaptive Leadership: Accelerating Enterprise Agility**. Addison-Wesley Professional, 2013.

HOFFMANN, *et al.* Applying acceptance test driven development to a problem based learning academic real-time system. 11th International Conference on Information Technology: New Generations. **IEEE**, 2014.

IBM, **Analytics for all: the time has come**, 2014. Disponível em: <http://public.dhe.ibm.com/common/ssi/ecm/yt/en/ytw03395usen/YTW03395USEN.PDF?>> Acesso em: 11/03/2015.

ITKONEN, J; RAUTIAINEN, K; LASSENIUS, C. Toward an understanding of quality assurance in agile software development. **International Journal of Agile Manufacturing**, volume 8, number 2, pages 39-49. 2005.

ITKONEN, J; MÄNTYLÄ, M, V; LASSENIUS, C. How Do Testers Do It? An Exploratory Study on Manual Testing Practices. Third International Symposium on Empirical Software Engineering and Measurement, **IEEE**, 2009.

ISO/IEC/IEEE 29119-1. **Concepts and definitions**. Disponível em: <[http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=45142](http://www.iso.org/iso/catalogue_detail.htm?csnumber=45142)> Acesso em: 14/11/2014.

ISTQB. International Software Testing Qualifications Board. **Agile Tester Extension Syllabus**. 2014. Disponível em: < <http://www.istqb.org/downloads/syllabi/agile-tester-extension-syllabus.html>> Acesso em: 14/11/2014.

ISTQB. International Software Testing Qualifications Board. **Foundation Level Syllabus**. 2011. Disponível em: <<http://www.istqb.org/downloads/viewcategory/16.html>> Acesso em: 15/11/2013.

JONES, C. **Applied Software Measurement: Global Analysis of Productivity and Quality**. McGraw-Hill/Osborne, 2008

KASURINEN, J. *Elaborating Software Test Processes and Strategies*, **IEEE**, 2010.

KHALANE, T; TANNER, M. **Software Quality Assurance in Scrum**: The need for concrete guidance on SQA strategies in meeting user expectations. **IEEE**. 2013.

KOROŠEC, D, I, R; PFARRHOFER, D, R. Supporting the Transition to an Agile Test Matrix. **IEEE**, 2015

KUMAR, G; BHATIA, P, K. Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies. **IEEE**, 2014

KUMAR, V. Comparison of Manual and Automation Testing. **International Journal of Research in Science and Technology**, 2012

NORTH, D. **BDD is like TDD if**. 2012. Disponível em: < <http://dannorth.net/2012/05/31/bdd-is-like-tdd-if/>> Acesso em: 30/08/2015.

MYERS, *et al.*. **The Art of Software Testing**, 3. ed. New Jersey: Wiley Publishing, 2011.

PRESSMAN, RS. **Engenharia de Software: Uma abordagem Profissional**. 7. ed. Porto Alegre: AMGH, 2011.

PHAM, A; PHAM, PV. **Scrum in Action: Agile Software Project Management and Development**. Boston: Cengage Learning. 2011.

POPPENDIECK, M; POPPENDIECK, T. **Implementing Lean Software Development: From concept to cash**. Boston: Addison Wesley, 2007.



RAHMAN, M; GAO, J. A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. IEE Symposium on Service-Oriented System Engineering. **IEEE**, 2015

RAMLER, R; WOLFMAIER, K. "Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost". **ACM**, 2006.

RICO, DF; SAYANI, HH; SONE, S. **The Business Value of Agile Software Methods: Maximizing ROI with Just-in-Time Processes and Documentation**. 1. ed. Fort Lauderdale J. Ross Publishing. 2009.

RUNESON, P; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. **Empirical Software Engineering**. Volume 14, Issue 2, pp 131-164. 2009.

SCHWABER, C; Gilpin, M. Evaluating Automated Functional Testing Tools, **Forrester Research**. 2005.

SCHWABER, K; SUTHERLAND, J. **Scrum Guide**. 2013. Disponível em <<https://www.scrum.org/Scrum-Guide>> Acesso em: 05/06/2014.

SEBRAE. **Anuário do Trabalho na Micro e Pequena Empresa**. Disponível em: <[http://www.sebrae.com.br/Sebrae/Portal%20Sebrae/Anexos/Anuario%20do%20Trabalho%20Na%20Micro%20e%20Pequena%20Empresa\\_2013.pdf](http://www.sebrae.com.br/Sebrae/Portal%20Sebrae/Anexos/Anuario%20do%20Trabalho%20Na%20Micro%20e%20Pequena%20Empresa_2013.pdf)>. Acesso em: 21/07/2015.

SHAYE, SD. Transitioning a Team to Agile Test Methods. Agile 2008 Conference. **IEEE** 2008.

SOMMERVILLE, I. **Engenharia de Software**. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

UNHELKAR, Bhuvan. **The Art of Agile Practice: A Composite Approach for Projects and Organizations**. Auerbach Publications: 2013.

VALLET, B. Kanban at Scale – A Siemens Success Story. 2014. InfoQ. Disponível em <<http://www.infoq.com/articles/kanban-siemens-health-services>> Acesso em: 03/02/2016.

VEENENDAAL, EV. SCRUM & Testing: Assessing the risks. **Agile Record: The magazine for Agile Developers and Agile Testers**. 2010.

\_\_\_\_\_. Test Process Improvement and Agile: Friends or Foes? **Testing experience: the magazine for professional testers**. 2014.

VERSION ONE. **Survey: The state of agile survey**. 2013. Disponível em: <<http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf>>. Acesso em: 05/06/2014.

VICENTE, A. A. **Definição e gerenciamento de teste no contexto de métodos ágeis**. Dissertação de Mestrado. Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, São Paulo, SP. Orientador: Márcio Eduardo Delamaro. 2010.

WATKINS, J. **Agile Testing: How to Succeed in an Extreme Testing Environment.**  
Cambridge University Press, 2009.

## APÊNDICE A – PRIMEIRO QUESTIONÁRIO

<http://goo.gl/forms/semsIsPP6M>

O objetivo deste questionário é verificar quais as técnicas e estratégias de testes tem sido utilizadas no processo de teste de software quando utilizados métodos ágeis de desenvolvimento de software.

### **Informações importantes:**

1. As respostas são de caráter sigiloso e confidencial.
2. Os dados da pesquisa serão utilizados exclusivamente para fins acadêmicos.
3. Apenas responda o questionário se tiver experiência em desenvolvimento ágil.

1) Qual é o porte da empresa onde atua?

- a. Microempresa – até 9 funcionários
- b. Pequena empresa – de 10 a 49 funcionários
- c. Média empresa – de 50 a 99 funcionários
- d. Grande empresa – acima de 99 funcionários

2) Quanto tempo de experiência você possui em desenvolvimento de software utilizando métodos ágeis?

- a. 0 a 1 ano
- b. 1 a 3 anos
- c. 3 a 5 anos
- d. 5 a 10 anos
- e. Acima de 10 anos

3) Qual(is) método(s) ou abordagem(s) de desenvolvimento de software são utilizados?

- a. Scrum
- b. Dynamic Systems Development (DSDM)
- c. Feature Driven Development (FDD)
- d. Extreme Programming (XP)
- e. Crystal
- f. Lean
- g. Kanban
- h. Outros: \_\_\_\_\_

4) Durante o planejamento de uma iteração, quais são os fatores considerados no âmbito das atividades de testes?

- a. Criação e execução de testes de histórias
- b. Testes em pares com outros testadores e desenvolvedores
- c. Validação do negócio
- d. Automação de novos testes funcionais
- e. Execução de testes de regressão automatizados
- f. Execução e automação de testes não funcionais
- g. Demonstração para os *stakeholders*
- h. Outros: \_\_\_\_\_

5) Qual o tempo ideal de uma iteração para que seja possível codificar e executar testes das funcionalidades selecionadas para esta iteração?

- a. 1 semana
- b. 2 semanas
- c. 3 semanas
- d. 4 semanas

- e. Depende da complexidade das histórias.
- f. Comentário: \_\_\_\_\_

6) Observa-se que os níveis de teste estão diretamente relacionados com os artefatos gerados no processo de desenvolvimento de software. Em modelos iterativos, no entanto, estes níveis de teste, muitas vezes, se sobrepõem. Como o time de desenvolvimento trabalha com este conceito a fim de validar e verificar cada etapa do desenvolvimento de software?

- a. Não se aplica este conceito
- b. Executa-se os testes de unidade
- c. Executa-se os testes de integração
- d. Executa-se os testes de sistema
- e. Executa-se os testes de aceitação de usuário
- f. Comentário: \_\_\_\_\_

7) Como se dá a validação do software quanto aos requisitos funcionais e não funcionais?

\_\_\_\_\_

8) Quais as práticas que mais corroboram para a entrega de um software testado no final de uma iteração?

- a. Automação de testes
- b. Testes exploratórios
- c. Testes de unidade
- d. Testes de aceitação com os *stakeholders*
- e. Boas práticas de desenvolvimento, tais como: TDD, programação em pares e fatoração.
- f. Outras: \_\_\_\_\_

9) Para se elaborar uma estratégia de automação, quais são os níveis da aplicação selecionados para automação?

- a. Unitário (Automação de testes de unidade - código)
- b. Serviço (Automação na camada de API - *Application programming interface*)
- c. GUI (Automação na camada de GUI - *Graphical user interface*)
- d. Comentário: \_\_\_\_\_

10) Quais fatores são utilizados para se considerar a automação de um teste?

- a. Utilização para teste de regressão
- b. Facilidade de automação
- c. Custo para automatizar
- d. Outro: \_\_\_\_\_

11) Qual abordagem o time utiliza para a execução de testes manuais?

- a. Casos de testes com *design* predefinido
- b. Testes exploratórios sem planejamento (*freestyle*)
- c. Testes exploratórios gerenciados - baseado em sessões
- d. Outra abordagem, comentário: \_\_\_\_\_

12) Existe uma separação de papéis entre os desenvolvedores e os testadores?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

13) Existem recursos dedicados apenas ao teste de software?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

14) Existe colaboração entre todos os membros do time para se alcançar a qualidade de um software?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

15) Os desenvolvedores e os testadores trabalham alinhados no mesmo objetivo e a comunicação é efetiva entre eles?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

16) Se existirem pessoas no time apenas dedicadas ao teste de software, qual é a contribuição destes no processo de desenvolvimento?

- a. Promove *feedback* durante todo o processo de desenvolvimento
- b. Ajuda ou tem conhecimento de nível de código do teste a ser realizado
- c. Assume a liderança nos testes de aceitação
- d. Assume a liderança nos testes de regressão
- e. Desenvolve os planos de testes
- f. Revela cenários de teste adicionais através de testes exploratórios
- g. Garante que a cobertura de teste é adequada

- h. Lidera os esforços de automação
- i. Liderança os esforços de testes de integração
- j. Executa testes de nível de sistema
- k. Mantém ambientes de teste e os dados disponíveis
- l. Identifica problemas de regressão e partes técnicas de teste
- m. Outra: \_\_\_\_\_

17) Durante uma iteração quais conjuntos de atributos de qualidade são verificados:

- a. Confiabilidade: atributos que incidem com a capacidade do software em manter o desempenho do software a partir de condições estabelecidas.
- b. Usabilidade: atributos que se relacionam quanto à utilização do produto.
- c. Eficiência: atributos que incidem sobre a relação entre o desempenho do software e a quantidade de recursos.
- d. Manutenibilidade: atributos que evidenciam o esforço necessário para fazer modificações específicas.
- e. Portabilidade: atributos que se relacionam com a capacidade do software de ser transferido de um ambiente para outro.
- f. Nenhum
- g. Outro

18) Existe algum processo de desenvolvimento, no qual nem todos os interessados da equipe tem conhecimento sobre este processo?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_



19) Os processos são adaptativos?

- a. Sim
- b. Não
- c. Se sim, como se dá esta adaptação: \_\_\_\_\_

20) É possível avaliar a maturidade dos processos de desenvolvimento ágil do qual faz parte?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

21) Existem um ambiente de teste gerenciável, com estratégia de teste, monitoramento e controle de teste?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

22) Existe um ambiente otimizado no qual são aplicados conceitos de melhoria contínua, tais como prevenção de defeitos, otimização de processos de testes e controle de qualidade durante o decorrer das iterações?

- a. Sim
- b. Não
- c. Comente: \_\_\_\_\_

23) Gostaria de acrescentar alguma sugestão para a pesquisa?

## APÊNDICE B – SEGUNDO QUESTIONÁRIO

<http://goo.gl/forms/Zp6150avyf>

### O uso de testes de software com métodos ágeis

Prezado(a)

Esta pesquisa faz parte de um trabalho de mestrado de Raquel Bortoluci sob a orientação do Prof. Dr. Marcelo Duduchi no curso de Gestão e Tecnologia em Sistemas Produtivos do Centro Paula Souza.

O principal objetivo do trabalho é verificar como as atividades de teste têm sido implementadas em equipes de desenvolvimento, que utilizam métodos ágeis de desenvolvimento para a produção de software.

Esse questionário deverá ser respondido em aproximadamente 15 minutos.

Contatos:

Aluna/autora: Raquel Bortoluci – [raquelborto@gmail.com](mailto:raquelborto@gmail.com)

Orientador: Prof. Dr. Marcelo Duduchi – [mduduchi@gmail.com](mailto:mduduchi@gmail.com)

Informações importantes:

1. As respostas são de caráter sigiloso e confidencial.
2. Os dados da pesquisa serão utilizados exclusivamente para fins acadêmicos.
3. Responda o questionário se tiver experiência em desenvolvimento ágil

**Você tem experiência em desenvolvimento de software utilizando métodos ágeis?**

- ☐ Sim
- ☐ Não

Continuar »

# O uso de testes de software com métodos ágeis

**\*Obrigatório**

**Qual é o porte da empresa onde atua? \***

- ☐ Microempresa – até 9 funcionários
- ☐ Pequena empresa – de 10 a 49 funcionários
- ☐ Média empresa – de 50 a 99 funcionários
- ☐ Grande empresa – de 100 – 499 funcionários
- ☐ Empresa com 500 ou mais funcionários

**Qual a extensão de atuação da empresa? \***

- ☐ Multinacional
- ☐ Nacional
- ☐ Norte
- ☐ Nordeste
- ☐ Centro-Oeste
- ☐ Sudeste
- ☐ Sul

**O desenvolvimento de software é qual atividade da empresa \***

Considerar a principal atividade da empresa

- ☐ Atividade fim
- ☐ Atividade meio

**Qual a natureza do software desenvolvido pela empresa? \***

Considerar os principais softwares desenvolvidos

- ☐ Baixo risco
- ☐ Risco intermediário
- ☐ Alto risco

Em média, por quantos membros são compostos o time de desenvolvimento de um determinado projeto ágil na empresa que você trabalha? \*

- ☐ Menos de 3
- ☐ De 3 a 5
- ☐ De 6 a 9
- ☐ Mais de 9

Como se dá a formação do time quanto a localização dos membros? \*

- ☐ Todos trabalham no mesmo local
- ☐ Equipes distribuídas

Qual é sua função principal? \*

- ☐ Desenvolvedor ou funções afins
- ☐ Testador ou funções afins
- ☐ Gerente de projeto ou funções afins
- ☐ Outro:

Quanto tempo de experiência você possui em desenvolvimento de software utilizando métodos ágeis? \*

- ☐ Menos de um ano
- ☐ 1 a 3 anos
- ☐ 4 a 5 anos
- ☐ 6 a 10 anos
- ☐ Acima de 10 anos

Você possui alguma certificação relacionada à métodos ágeis? \*

- ☐ Sim
- ☐ Não

Qual(is) método(s) ou abordagem(s) de desenvolvimento de software são utilizados? \*

- ☐ Scrum
- ☐ Dynamic Systems Development (DSDM)
- ☐ Feature Driven Development (FDD)
- ☐ Extreme Programming (XP)
- ☐ Crystal
- ☐ Lean
- ☐ Kanban
- ☐ Outro:

« Voltar

Continuar »

**Durante o planejamento de uma iteração, quais são os fatores considerados no âmbito das atividades de testes?**

- ☐ Criação e execução de testes de histórias
- ☐ Testes em pares com outros testadores e desenvolvedores
- ☐ Validação do negócio
- ☐ Automação de novos testes funcionais
- ☐ Execução de testes de regressão automatizados
- ☐ Execução e automação de testes não funcionais
- ☐ Demonstração para os stakeholders
- ☐ Criação dos ambientes de testes
- ☐ Revisão dos cenários de teste pelo time
- ☐ Outro:

**Qual o tempo ideal de uma iteração para que seja possível codificar e executar testes das funcionalidades selecionadas para esta iteração?**

- ☐ 1 semana
- ☐ 2 semanas
- ☐ 3 semanas
- ☐ 4 semanas
- ☐ Deve-se estar pronto para entregar o software funcionando todos os dias
- ☐ Outro:

**Os testes são realizados durante toda a iteração?**

- ☐ Sim
- ☐ Não

Em que níveis o time de desenvolvimento valida e verifica o software? \*

- ☐ Não se aplica este conceito
- ☐ Executa-se os testes de unidade
- ☐ Executa-se os testes de integração
- ☐ Executa-se os testes de sistema
- ☐ Executa-se os testes de aceitação de usuário

Como se dá a verificação do software quanto aos requisitos não funcionais? \*

- ☐ Execução de testes durante as iterações
- ☐ Execução de testes apenas quando existe tempo em uma iteração
- ☐ Execução de testes em paralelo às iterações
- ☐ Execução de testes depois das iterações
- ☐ Criação de histórias para cada requisito não funcional
- ☐ Não se faz este tipo de verificação

« Voltar

Continuar »

## Ferramentas

A equipe usa ferramentas para validação de requisitos não funcionais? \*

0 - Não usa

1 - Usa pouco

2 - Usa  
amplamente

3 - Usa para todos  
os testes

Utilização de  
Ferramentas



« Voltar

Continuar »



Quais as práticas utilizadas na empresa que para você mais corroboram para a entrega de um software testado no final de uma iteração? \*

- ☐ Automação de testes
- ☐ Revisão ou criação dos cenários de teste junto ao desenvolvedor
- ☐ Execução de testes exploratórios
- ☐ Execução de casos de testes manuais
- ☐ Testes de unidade
- ☐ Testes de aceitação com os stakeholders
- ☐ Práticas de programação, tais como fatoraçoão e programação em pares.
- ☐ Outro:

Qual abordagem o time utiliza para a execução de testes manuais? \*

- ☐ Não se utiliza testes manuais
- ☐ Casos de testes com design predefinido
- ☐ Testes exploratórios sem planejamento (freestyle)
- ☐ Testes exploratórios gerenciados - baseado em sessões
- ☐ Testes exploratórios guiados por mapas mentais
- ☐ Outro:

Para se elaborar uma estratégia de automação, quais são os níveis da aplicação são utilizados na empresas? \*

- ☐ Não se utiliza automação de testes
- ☐ Unitário (Automação de testes de unidade - código)
- ☐ Serviço (Automação na camada de API - Application programming interface)
- ☐ GUI (Automação na camada de GUI - Graphical user interface)
- ☐ Outro:



**Em termos de estratégias ágeis, qual a principal abordagem utilizada?**

- ☐ Acceptance Test Driven Development (ATDD)
- ☐ Behavior Driven Development (BDD)
- ☐ Test Driven Development (TDD)
- ☐ As abordagens citadas acima não são conhecidas
- ☐ As abordagens citadas acima são conhecidas, mas não são utilizadas

**A equipe consegue envolver os stakeholders nos testes de software?**

	<input type="radio"/> - Não consegue	1 - Consegue parcialmente	2 - Consegue amplamente	3 - Consegue em sua totalidade
Envolvimento dos stakeholders	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Como se organiza as equipes de desenvolvimento em relação ao teste de software?**

- ☐ A equipe é integrada, mas com separação de papéis entre desenvolvedores e testadores
- ☐ A equipe é integrada, com equipes multidisciplinares e sem separação de papéis
- ☐ A equipe de teste é independente
- ☐ Outro:

« Voltar

Continuar »

## Papel do testador

Qual é a contribuição da equipe de teste no processo de desenvolvimento?

- ☐ Promove feedback durante todo o processo de desenvolvimento
- ☐ Ajuda ou tem conhecimento de nível de código do teste a ser realizado
- ☐ Assume a liderança nos testes de aceitação
- ☐ Assume a liderança nos testes de regressão
- ☐ Desenvolve os planos de testes
- ☐ Revela cenários de teste adicionais através de testes exploratórios
- ☐ Garante que a cobertura de teste é adequada
- ☐ Lidera os esforços de automação
- ☐ Lidera os esforços de testes de integração
- ☐ Executa testes de nível de sistema
- ☐ Mantém ambientes de teste e os dados disponíveis
- ☐ Identifica problemas e partes técnicas de teste
- ☐ Outro:

« Voltar

Continuar »

Existe um ambiente de teste gerenciável, com estratégia de teste, monitoramento e controle de teste?

- ☐ Sim
- ☐ Não

« Voltar

Continuar »

## Métricas

Quais métricas são utilizados no contexto de teste de software?

- ☐ Não são utilizadas métricas
- ☐ Cobertura de código
- ☐ Fator de teste (indica a relação entre o tamanho do código de testes e tamanho do código em produção)
- ☐ Quantidade de casos de teste e assertivas
- ☐ Porcentagem de assertivas de teste de unidade passando e falhando
- ☐ Quantidade de testes de aceitação por funcionalidade
- ☐ Porcentagem de assertivas de teste de aceitação passando e falhando
- ☐ Funcionalidade testadas e entregues
- ☐ Tempo de execução
- ☐ Quantidade de defeitos encontrados

« Voltar

Enviar