INSTITUTO DOCTUM DE EDUCAÇÃO E TECNOLOGIA

FACULDADES INTEGRADAS DE CARATINGA

CIÊNCIA DA COMPUTAÇÃO

NECESSIDADE DOS TESTES DE UNIDADE NO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS COM *ZEND FRAMEWORK*

BEATRIZ GOMES MEDINA

CARATINGA
2015

Beatriz Gomes Medina

NECESSIDADE DOS TESTES DE UNIDADE NO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS COM *ZEND FRAMEWORK*

Monografia apresentada à banca examinadora da Faculdade de Ciência da Computação das Faculdades Integradas de Caratinga, como requisito parcial para obtenção do título de bacharel em Ciência da Computação, sob orientação da professora Msc. Fabrícia Pires Souza Tiola.

Caratinga
2015

FACULDADES INTEGRADAS DE CARATINGA

TRABALHO DE CONCLUSÃO DE CURSO

TERMO DE APROVAÇÃO

TÍTULO DO TRABALHO

NECESSIDADE DOS TESTES DE UNIDADE NO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS COM ZEND FRAMEWORK

por

Beatriz Gomes Medina

Este Trabalho de Conclusão de Curso foi apresentado perante a Banca de Avaliação composta pelos professores Fabrícia Pires Souza Tiola, Bruno Vieira Becattini e Maicon Vinícius Ribeiro, às 19 horas do dia 09 de dezembro de 2015 como requisito parcial para a obtenção do título de bacharel. Após a avaliação de cada professor e discussão, a Banca Avaliadora considerou o trabalho aprovado, com a qualificação: EXCELENTE.

Trabalho indicado para publicação: (X)SIM ( )NÃO

Caratinga, 09 de dezembro de 2015

Professor Orientador e Presidente da Banca

Bruno Vieira Becattini

Professor Avaliador 1

Maicon Ribeiro

Professor Avaliador 2

Beatriz Gomes Medina.

Aluno(a)

Coordenador (a) do Curso

AGRADECIMENTOS

Primeiramente agradeço a Deus pela vida, pela saúde, e por ter me dado forças e iluminado meu caminho para que pudesse concluir mais uma etapa da minha vida.

Aos meus pais que me proporcionaram condições de estudar e sempre me apoiaram e incentivaram a prosseguir nesta jornada. A toda a minha família que foi um alicerce seguro para os momentos difíceis.

Aos meus amigos por compreenderem nos momentos que não pude estar presente. Aos meus colegas que tive o prazer de conhecer e compartilhar grandes momentos de alegria e aprendizado. Em especial aos que se tornaram verdadeiros amigos nessa caminhada.

Agradeço também a todos os professores, pelos valorosos ensinamentos nesses anos, em especial a minha orientadora Fabrícia Pires por ter me guiado para que esse trabalho se realizasse, me apoiando durante toda a caminhada até este momento. Muito obrigada!

“Porque, como imaginou no seu coração, assim é ele”.

Provérbios 23:7

RESUMO

Um dos fatores que interferem na pouca propagação dos testes de software é que essa atividade não é uma tarefa simples, ela exige um bom planejamento para ser bem sucedida, e muitos desenvolvedores pensam ser uma perca de tempo e uma queda na produtividade por causa do tempo gasto para escrever os testes de unidade automatizados, além disso a falta de conhecimento sobre o processo de testes faz com que muitos erros não sejam encontrados, sucedendo em softwares sem qualidade e clientes insatisfeitos.

No entanto, garantir o desenvolvimento de software seguro e de qualidade é fator indispensável para a atual sociedade, onde os sistemas estão se tornando uma obrigatoriedade do mercado. E, com a busca pela satisfação dos clientes, e a complexidade dos sistemas, vem a necessidade de investir em meios profissionais para reduzir erros, falhas ou defeitos. Visualizando este cenário, este trabalho teve por objetivo analisar a adequação dos testes de unidade durante o processo de desenvolvimento de software com *Zend Framework* visando a correção de falhas e a qualidade das aplicações.

O questionário aplicado foi desenvolvido tendo como base a pesquisa bibliográfica realizada sobre as principais técnicas, tipos e fases dos testes de software, além dos fundamentos sobre o *Zend Framework*, o PHPUnit e o TDD (Desenvolvimento orientado a testes).

Foram coletadas respostas de 85 profissionais da área de Tecnologia da Informação, e os resultados demonstraram que os profissionais têm ciência da importância de se realizar um processo de testes, apesar de grande parte ainda não utilizar metodologias e técnicas apropriadas para a realização efetiva dos testes e consequente correção e redução do número de falhas no processo de desenvolvimento de softwares.

Palavras-chave: Testes de software, teste de unidade, TDD, PHPUnit, *Zend Framework*.

LISTA DE SIGLAS

MVC – *Model-View-Controller* (Modelo-Visão-Controlador)

PHP – *Hypertext Preprocessor* (Pré-processador de hipertexto)

TDD – *Test Driven Developement* (Desenvolvimento Orientado a Testes)

TI – Tecnologia da Informação

ZF – *Zend Framework*

LISTA DE ILUSTRAÇÕES

FIGURA 1 – Desenvolvimento orientado a testes (TDD) (SOMMERVILLE, 2011) ...27
FIGURA 2 – O padrão MVC (SOMMERVILLE, 2011) ...30

GRÁFICO 1 - A necessidade de utilização dos Testes de Software visando a qualidade e sucesso das aplicações ...38
GRÁFICO 2 - Tempo de experiência sobre Processo de Testes ...39
GRÁFICO 3 - Tempo que o Processo de Testes foi implantado no projeto ...40
GRÁFICO 4 - Como o Processo de Testes utilizado é avaliado ...40
GRÁFICO 5 - Testes de software que são realizados no projeto ...41
GRÁFICO 6 - Responsáveis por realizar os testes de software ...42
GRÁFICO 7 - Início da execução dos testes ...43
GRÁFICO 8 - Maior incidência dos erros ...44
GRÁFICO 9 - Erros mais comuns encontrados nos sistemas ...44
GRÁFICO 10 - Quantidade de erros encontrados após a entrega do produto ao cliente, depois da implantação dos testes de software ...45
GRÁFICO 11 - Os custos após a implantação dos testes ...46
GRÁFICO 12 - Como é desenvolver sistemas com o Zend Framework utilizando o PHPUnit e o TDD ...47
GRÁFICO 13 – Vantagens dos testes automatizados em relação aos testes feitos de forma manual ...48
GRÁFICO 14 – Processo de Testes baseado no Desenvolvimento Orientado a Testes (TDD) ...49
GRÁFICO 15 – Aderência do TDD no projeto por parte dos colaboradores ...50
GRÁFICO 16 – Nível da capacidade de progresso e correção de falhas que os Testes de unidade agregam ao processo de desenvolvimento ...51
GRÁFICO 17 – Nível de dificuldade em utilizar o PHPUnit ...52
GRÁFICO 18 – Nível de eficácia que o PHPUnit apresenta no projeto ...53
GRÁFICO 19 - Como o Zend Framework é avaliado ...54
GRÁFICO 20 - Nível de eficiência que o Zend Framework apresenta no projeto ...54

SUMÁRIO

INTRODUÇÃO 11
1 REFERENCIAL TEÓRICO 13
1.1 TESTES DE SOFTWARE 13
1.1.1 Técnicas de testes 15
1.1.1.1 Teste funcional (caixa preta) 16
1.1.1.2 Teste estrutural (caixa branca) 17
1.1.2 Fases de teste 17
1.1.2.1 Teste de Unidade 18
1.1.2.2 Teste de Integração 18
1.1.2.3 Teste de Validação 19
1.1.2.4 Teste de Sistema 20
1.1.2.5 Teste de Regressão 20
1.1.3 Tipos de teste 21
1.1.3.1 Teste de Carga (Estresse) 21
1.1.3.2 Teste de Configuração (Disponibilização) 22
1.1.3.3 Teste de Desempenho (Performance) 22
1.1.3.4 Teste de Instalação 22
1.1.3.5 Teste de Recuperação 23
1.1.3.6 Teste de Segurança 23
1.1.3.7 Teste de Usabilidade 24
1.1.3.8 Teste de Volume 24
1.1.3.9 Teste Paralelo 25
1.1.4 Benefícios do teste de unidade 25
1.2 DESENVOLVIMENTO ORIENTADO A TESTES (TDD) 26

1.3 PHPUNIT....28
1.4 ZEND FRAMEWORK....29
2 METODOLOGIA....32
2.1 PÚBLICO ALVO DO QUESTIONÁRIO....32
2.2 ELABORAÇÃO DO QUESTIONÁRIO....32
2.3 O QUESTIONÁRIO....33
2.4 COLETA DE DADOS....34
2.5 TRATAMENTO DE DADOS....34
3 RESULTADOS....36
3.1 ANÁLISE DAS RESPOSTAS COLETADAS....36
3.1.1 Perfil do Respondente....36
3.1.2 Processo de Testes....37
3.1.3 Processo de Testes Automatizados....47
3.1.4 Desenvolvimento Orientado a Testes....49
3.1.5 Testes de Unidade....50
3.1.6 Framework PHPUnit....51
3.1.7 Zend Framework....53
3.2 DISCUSSÃO DOS RESULTADOS....55
CONCLUSÃO....57
TRABALHOS FUTUROS....58
REFERÊNCIAS....59
APÊNDICE 1 - QUESTIONÁRIO....61

INTRODUÇÃO

Os softwares são uma realidade para a atual sociedade. Desde aplicações para celular até o controle de uma nave espacial, os sistemas estão se tornando uma obrigatoriedade do mercado. Garantir o desenvolvimento de software seguro e de qualidade é fator indispensável para a reputação das empresas de desenvolvimento que buscam a satisfação dos clientes. Com a complexidade dos sistemas vem a necessidade de investir em meios profissionais para reduzir erros, falhas ou defeitos.

Neste contexto entra a necessidade dos testes de software, que é um processo de execução que explora as funcionalidades do software durante o desenvolvimento com o objetivo de encontrar defeitos e vulnerabilidades, possibilitando que as correções sejam realizadas antes da entrega final ao cliente. O teste de unidade é uma das etapas do processo de testes, que tem o objetivo de avaliar componentes individuais de uma aplicação.

A proposta deste trabalho é descobrir se o desenvolvimento das aplicações com o apoio dos testes de unidade é uma boa solução para correção de falhas não identificadas no processo de desenvolvimento de softwares. O principal problema dessa pesquisa é entender o porquê de muitos profissionais ligados ao desenvolvimento de sistemas não utilizarem um processo de testes ou não atribuírem a devida importância à essa prática.

Pensando nisso, o objetivo desse trabalho é apresentar o ponto de vista dos profissionais de TI, em relação ao processo de testes de software, por meio de uma pesquisa científica do tipo qualitativa, que busca responder às hipóteses de que o uso dos testes de unidade no desenvolvimento de sistemas aumenta a capacidade de progresso e correção de falhas não identificadas no processo, e a utilização do *framework* de testes PHPUnit com o apoio da técnica de desenvolvimento orientado a testes é uma boa opção para desenvolvimento de sistemas com o *Zend Framework*, contribuindo assim para o desenvolvimento de softwares de alta qualidade.

O questionário utilizado na pesquisa foi dividido em sete grupos: perfil do respondente; processo de testes; processo de testes automatizados; desenvolvimento orientado a testes; testes de unidade; framework PHPUnit; e Zend Framework. A pesquisa contou com respostas de 85 profissionais da área de TI, que de maneira

geral avaliaram positivamente o uso dos testes de unidade no desenvolvimento das aplicações.

Este trabalho está estruturado em três capítulos principais. O capítulo 1 descreve o referencial teórico deste trabalho, o qual apresenta as principais técnicas, tipos, fases e benefícios dos testes de software. Apresenta a ferramenta de desenvolvimento *Zend Framework*, o *framework* de testes PHPUnit e a metodologia de desenvolvimento orientado a testes (TDD). O capítulo 2 apresenta a metodologia utilizada para alcançar os resultados. E o capítulo 3 descreve os resultados obtidos com a pesquisa realizada com os profissionais de Tecnologia da Informação. Por fim, são apresentadas as conclusões e sugestões para trabalhos futuros.

1 REFERENCIAL TEÓRICO

O avanço da Tecnologia da Informação vem sendo acompanhado pelo aumento da complexidade no processo de desenvolvimento, e junto vem uma crescente exigência por qualidade, eficácia, eficiência e redução de custos, devido à concorrência acirrada e às reclamações dos clientes.

Nesse cenário caracterizado por mudanças constantes, heterogeneidade dos ambientes e pouca tolerância a falhas, os testes de software têm se tornado importantes para a garantia da qualidade dos sistemas. No entanto, a atividade de teste não é tarefa simples, é preciso um bom planejamento para que o processo de testes seja bem-sucedido, e para isso faz se necessário o conhecimento sobre testes de software.

Nesta seção será descrito o referencial teórico deste trabalho, o qual aborda as principais técnicas, tipos, fases e benefícios dos testes de software, além de apresentar a ferramenta de desenvolvimento *Zend Framework*, o *framework* de testes PHPUnit e a metodologia de desenvolvimento orientado a testes (TDD). Permitindo assim uma melhor compreensão dos assuntos que são tratados no capítulo de metodologia.

1.1 TESTES DE SOFTWARE

O constante avanço do desenvolvimento de software, num cenário mundial cada vez mais dependente da tecnologia, exige que sejam utilizados processos de garantia da qualidade para que a demanda seja atendida satisfatoriamente.

Os softwares são uma realidade na atual sociedade, eles são encontrados nos comércios, na cultura e até nas mais simples atividades cotidianas. Segundo Pressman (2011), o software “é um produto e, ao mesmo tempo, o veículo para distribuir um produto”, ele é tanto um transformador de informações como uma base para o controle do computador.

> O software distribui o produto mais importante da nossa era – a informação. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis num determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informações (Internet) e os meios para obter informações sob todas as suas formas (PRESSMAN, 2011, P. 31).

Para garantir a qualidade dos softwares os desenvolvedores utilizam uma estrutura chamada de engenharia de software, que envolve um processo, um conjunto de práticas e várias ferramentas, que capacita o desenvolvimento de sistemas com alta qualidade. Os processos de engenharia definem metodologias que possibilitam o desenvolvimento dentro do prazo e com qualidade garantida, as práticas ou métodos fornecem informações técnicas que incluem tarefas de comunicação, análise de requisitos, modelagem, construção, testes e suporte, as ferramentas fornecem suporte automatizado para o processo e para as práticas da engenharia de software (PRESSMAN, 2011).

Apesar da preocupação com a qualidade dos sistemas ter crescido à medida que o software se torna mais importante na vida cotidiana, muitas empresas “não percebem que implantar um processo de garantia da qualidade de software não é uma opção a ser estudada, mas parte de uma estratégia de sobrevivência em um mercado cada vez mais exigente e competitivo” (BARTIÉ, 2002).

> Em essência, todos querem construir sistemas de alta qualidade, mas o tempo e o esforço necessários para produzir um software “perfeito” simplesmente não existem em um mundo orientado ao mercado. A pergunta passa a ser: devemos construir software “bom o suficiente”? (PRESSMAN, 2011, P. 371).

Produzir software de alta qualidade abrange o investimento em qualidade durante toda a fase de desenvolvimento, pois qualquer decisão tomada durante o processo pode comprometer a qualidade final. Para Bartié (2002), “Qualidade de Software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos”. É nesse processo de qualidade que surge o conceito de testes de software que é uma fase muito importante, onde são descobertos os erros, inconsistências e falhas no software que serão corrigidos, evitando grandes prejuízos tanto para os desenvolvedores quanto para os usuários.

De forma geral, teste de software é entendido como o processo de demonstrar que não existem defeitos, que algo funciona corretamente ou ainda que determinadas coisas fazem o que deveriam fazer, todavia, o real objetivo dos testes é provar que algo não funciona, o que exige um esforço muito maior do que provar que algo funciona (BARTIÉ, 2002).

Os erros podem ser introduzidos no sistema durante todo o processo de desenvolvimento, desde os requisitos até o final do desenvolvimento, e quanto mais tarde os erros são descobertos, maiores são os custos de correção. Por isso, a execução dos testes deve começar desde o planejamento e especificação dos requisitos até a fase de manutenção, assim os testes passam a atuar não somente na detecção de erros, mas também na prevenção de problemas, na melhoria dos processos e no aumento da produtividade, diminuindo o prazo de entrega e aumentando a satisfação dos clientes.

1.1.1 Técnicas de testes

Um software pode ser avaliado a partir de duas abordagens diferentes e complementares que determinam como os testes serão conduzidos para ampliar a detecção de erros. O teste funcional e o teste estrutural são as duas estratégias de validação que buscam “identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo” (BARTIÉ, 2002).

O teste funcional é conhecido por caixa preta, pois usa uma visão externa garantindo que os requisitos do sistema são atendidos pelo software. O teste estrutural ou caixa branca, usa uma visão interna identificando defeitos na estrutura interna do software (PRESSMAN, 2011).

Os testes de caixa preta são voltados para as especificações do programa e utilizados quando o grau de complexidade é normal, e os testes de caixa branca examinam o código-fonte do programa testando seus principais elementos e são utilizados quando o grau de complexidade é alto (LOPES, 2013).

Uma técnica não elimina a outra, ambas precisam ser usadas. “Essas estratégias são complementares e não exclusivas, o que significa que teremos um

produto de maior qualidade se ambos os processos foram aplicados nas etapas de validação de software” (BARTIÉ, 2002).

1.1.1.1 Teste funcional (caixa preta)

O teste caixa preta se concentra nos requisitos funcionais do software. Segundo Pressman (2011), é também chamado de teste comportamental e seus critérios “permitem derivar séries de condições de entrada que utilizarão completamente todos os requisitos funcionais para um programa”.

A grande vantagem dessa estratégia é que não requer conhecimento de conceitos complexos, por isso, é considerada mais simples de se implantar do que os testes caixa branca. As organizações realizam testes funcionais “em forma de testes manuais executados por profissionais ou mesmo usuários do sistema”, o desafio é começar a exigir um planejamento mais apurado e substituir o trabalhoso processo manual por um processo automatizado e confiável (BARTIÉ, 2002).

De acordo com Pressman (2011), o teste caixa preta encontra erros em funções incorretas ou ausentes, erros de interface, erros nas estruturas de dados ou no acesso a banco de dados externos, erros de desempenho, e erros de iniciação e término. Além disso o foco é no domínio das informações, no conhecimento dos requisitos, características e comportamentos esperados, o software é avaliado pelos resultados da aplicação, por isso os testes funcionais são executados nos estágios finais do processo de testes.

Os critérios aplicados com essa técnica são o particionamento de equivalência, análise de valor limite e grafo de causa e efeito.

1.1.1.2 Teste estrutural (caixa branca)

O teste caixa branca é baseado na arquitetura interna do software. É também chamado de teste da caixa de vidro e “usa a estrutura de controle descrita como parte do projeto no nível de componentes para derivar casos de teste” (PRESSMAN, 2011).

De acordo com Pressman (2011), o teste caixa branca possibilita a criação de casos de teste que garantam que todos os caminhos independentes foram testados pelo menos uma vez, testam todas as decisões lógicas nos estados verdadeiro e falso, executam todos os ciclos em seus limites e dentro de suas fronteiras operacionais, e testam estruturas de dados internas para assegurar a validade.

Com essa técnica são aplicados os critérios baseados em fluxo de controle, baseados em fluxo de dados e baseados na complexidade.

Essa estratégia requer conhecimento da tecnologia usada no software e da arquitetura interna, bem como o acesso a fontes e estruturas dos bancos de dados, por isso, apesar da alta eficiência na detecção de erros, o teste caixa branca é considerado difícil de se implementar. Pode ser realizado pelos próprios desenvolvedores, porém gerará resultados mais rápidos se for realizado por profissionais de testes (BARTIÉ, 2002).

1.1.2 Fases de teste

Para aumentar o nível de confiabilidade do software testado, a atividade de teste é dividida em várias fases, principalmente quando o tamanho e a complexidade do projeto são grandes, assim é possível focar em diferentes tipos de erros e aspectos diferentes do software testado.

As fases de teste de software são: teste de unidade, teste de integração, teste de validação, teste de sistema e teste de regressão. As descrições de cada fase serão apresentadas a seguir.

1.1.2.1 Teste de Unidade

Muitas são as maneiras e técnicas para se testar um software, que são aplicadas dependendo de cada projeto, de acordo com Blanco (2012), “o teste de unidade é o primeiro a ser realizado e analisa cada unidade do sistema (uma unidade é a menor parte do sistema que possa ser testada) separadamente na busca de erros”.

> O teste de unidade focaliza o esforço de verificação na menor unidade de projeto do software – o componente ou módulo de software. Usando como guia a descrição de projeto no nível de componente, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo. A complexidade relativa dos testes e os erros que revelam são limitados pelo escopo restrito estabelecido para o teste de unidade. Esse teste enfoca a lógica interna de processamento e as estruturas de dados dos limites de um componente. Esse tipo de teste pode ser conduzido em paralelo para diversos componentes (PRESSMAN, 2011, P.407).

Para Bartié (2002), “a validação de uma unidade de software somente será completa se aplicarmos testes em sua estrutura (caixa branca) e testes que validem seus requisitos (caixa preta)”.

Pressman (2011) lembra que “O teste de unidade normalmente é considerado um auxiliar para a etapa de codificação. O projeto dos testes de unidade pode ocorrer antes de começar a codificação ou depois que o código-fonte tiver sido gerado”.

É recomendado que os testes de unidade sejam automatizados, assim é possível executar um conjunto de testes em poucos segundos, e a cada vez que uma alteração é feita no programa (SOMMERVILLE, 2011).

1.1.2.2 Teste de Integração

O teste de integração é uma técnica para construir o software, ao mesmo tempo que tenta descobrir erros associados com as interfaces, a partir de testes. “O objetivo é construir uma estrutura de programa determinada pelo projeto a partir de componentes testados em unidade” (PRESMANN, 2011).

> Os testes de integração podem ser entendidos como o segundo estágio do processo de validação de componentes de software. Esses têm por objetivo validar a compatibilidade entre componentes da mesma arquitetura tecnológica. A compatibilidade de um componente é quebrada sempre que existe uma alteração ou exclusão de uma rotina ou propriedade pública de um componente. Quando isso ocorre, todos os outros componentes que empregam essas rotinas ou propriedades estão automaticamente incompatíveis com o componente modificado, gerando erros durante a execução do software. São exatamente esses erros que devem ser identificados nesse estágio dos testes de software (BARTIÉ, 2002, P. 148).

Os testes de integração são uma continuidade dos testes de unidade, são importantes, pois, os testes de unidade não garantem que os componentes individuais irão funcionar corretamente quando forem colocados todos juntos. Segundo Pressman (2011), os testes de integração verificam se dados não foram perdidos através de uma interface, se um componente não teve um efeito inesperado sobre outro, se subfunções produzem a função principal desejada, entre outros.

A melhor forma de aplicar os testes de integração é construindo e testando o software em pequenos fragmentos, dessa forma é mais fácil corrigir e isolar os erros.

1.1.2.3 Teste de Validação

Quando os componentes individuais já foram testados e o teste de integração termina, o software está montado e os testes de validação começam. O teste de validação é a garantia final de que o software satisfaz todos os requisitos informativos, funcionais, comportamentais e de desempenho (PRESSMAN, 2011).

> [...] a validação tem sucesso quando o software funciona de uma maneira que pode ser razoavelmente esperada pelo cliente. Nesse ponto, um desenvolvedor de software veterano pode protestar: “Quem ou o quê é o árbitro para decidir o que são expectativas razoáveis?”. Se foi desenvolvida uma Especificação de Requisitos de Software, ela descreve todos os atributos do software visíveis ao usuário e contém uma seção denominada Critério de Validação que forma a base para uma abordagem de teste de validação (PRESSMAN, 2011, P. 417).

Durante a validação uma série de testes demonstra conformidade com os requisitos, após cada caso de teste ter sido executado, temos duas condições, ou a característica está de acordo com os requisitos, e é aceita, ou é descoberto um desvio

dos requisitos e é criada uma lista de deficiências. Segundo Bartié (2002), se essa fase apresentar muitas falhas, é sinal de que as fases de teste anteriores não foram implementadas corretamente.

1.1.2.4 Teste de Sistema

Uma vez que o software é validado, o teste de sistema verifica se todos os elementos do sistema, como hardware, pessoas e base de dados, se combinam corretamente.

> Teste de sistema é na realidade uma série de diferentes testes cuja finalidade primária é exercitar totalmente o sistema. Embora cada um dos testes tenha uma finalidade diferente, todos funcionam no sentido de verificar se os elementos do sistema foram integrados adequadamente e executam as funções a eles alocadas (PRESSMAN, 2011, P. 419).

Alguns dos tipos de testes de sistema são: teste de recuperação, teste de segurança, teste de desempenho e teste de disponibilização, estes são elucidados na seção Tipos de teste.

1.1.2.5 Teste de Regressão

Segundo Bartié (2002), teste de regressão trata-se de reexecutar um subconjunto total ou parcial de testes previamente executados. Com o objetivo de garantir que as alterações ou inserções de determinados segmentos do produto, durante a manutenção ou melhorias, não causaram efeitos colaterais indesejados.

> Sempre que o software é corrigido, algum aspecto da configuração do software (o programa, sua documentação, ou os dados que o suportam) é alterado. O teste de regressão ajuda a garantir que as alterações (devido ao teste ou por outras razões) não introduzam comportamento indesejado ou erros adicionais (PRESSMAN, 2011, P. 412).

Realizar o teste de regressão é quase que obrigatório, já que todo sistema, pelo menos uma vez, terá que passar por alguma alteração. É muito importante garantir que não surgirão erros ao modificar alguma funcionalidade, e o teste de regressão verifica principalmente isso.

1.1.3 Tipos de teste

Existem vários tipos de teste de software, desde testes ligados aos requisitos de negócios, relacionados ao desempenho do software, relativos à compatibilidade com outros ambientes de execução, até testes de segurança do sistema. Para não perder tempo nem recursos tentando encontrar todo e qualquer tipo de defeito, é bom planejar uma estratégia e estabelecer quais tipos de erros devem ter prioridade nos testes. As descrições de alguns tipos de testes de software serão apresentadas a seguir.

1.1.3.1 Teste de Carga (Estresse)

Tem o objetivo de simular condições atípicas de utilização do software, provocando aumentos e reduções sucessivas de transações para testar se o sistema funciona corretamente em diversas condições de carga de trabalho. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Elevando e reduzindo sucessivamente o número de transações simultâneas;
- Aumentando e reduzindo o tráfego de rede;
- Aumentando o número de usuários simultâneos;
- E, combinando todos esses elementos.

1.1.3.2 Teste de Configuração (Disponibilização)

Tem o objetivo de testar se o software funciona corretamente sobre as diversas configurações de hardware e software exigidas, para garantir que o programa rode adequadamente nos variados ambientes de produção. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Variando os sistemas operacionais e versões;
- Variando os *browsers*;
- Variando os hardwares que irão interagir com a solução;
- E, combinando todos esses elementos.

1.1.3.3 Teste de Desempenho (Performance)

Tem o objetivo de testar se o desempenho está de acordo com os requisitos do sistema, o volume de transações e tempo de resposta obtidos nos testes são comparados com valores limite especificados, assim é estabelecido o critério de sucesso. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Validar todos os requisitos de desempenho identificados;
- Simular *n* usuários acessando a mesma informação, ao mesmo tempo;
- Simular *n* usuários processando ao mesmo tempo a mesma transação;
- Simular *n*% de tráfego de rede;
- E, combinar todos esses elementos.

1.1.3.4 Teste de Instalação

Tem o objetivo de validar se o software pode ser instalado corretamente sob diferentes alternativas de instalação. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Realizar a primeira instalação do software;
- Realizar a instalação de um software já instalado;
- Realizar a instalação de atualização de um software;
- Realizar a instalação de software em vários ambientes distintos;
- Realizar todas as alternativas de instalação;
- Validar pré-requisitos de instalação de software.

1.1.3.5 Teste de Recuperação

Tem o objetivo de avaliar o comportamento do software após determinadas condições anormais, verificando se o sistema consegue se recuperar adequadamente depois da ocorrência de um erro ou falha. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Interromper o acesso à rede, por alguns instantes e por um longo período;
- Interromper o processamento, desligar o micro e desligar o servidor;
- Gerar arquivos, cancelar o processamento e avaliar se existem arquivos gerados.

1.1.3.6 Teste de Segurança

Tem o objetivo de avaliar a segurança que a infraestrutura oferece, detectando falhas que podem comprometer o sigilo e fidelidade das informações, provocar perdas de dados ou interrupções de processamento, testando se o sistema realmente protege de acessos não autorizados. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Validar todos os requisitos de segurança identificados;
- Tentar acessar funcionalidades e informações que requerem perfil avançado;
- Tentar invadir/derrubar o servidor de dados/internet;
- Tentar extrair backups de informações sigilosas;

- Tentar descobrir senhas e quebrar protocolos de segurança;
- Tentar processar transações geradas de fontes inexistentes;
- Tentar simular comportamento/infecção por vírus.

1.1.3.7 Teste de Usabilidade

Tem o objetivo de testar se o software é de fácil utilização pelo usuário final, de forma a deixar o sistema mais simples e intuitivo, verificando padronização e clareza de textos e mensagens, entre outros aspectos. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Entrar em cada tela e avaliar a facilidade de navegação entre elas;
- Realizar *n* operações e depois desfazê-las;
- Realizar procedimentos críticos e avaliar mensagem de alerta;
- Avaliar número de passos para realizar as principais operações;
- Avaliar a existência de ajuda em todas as telas;
- Realizar *n* buscas no manual de ajuda e validar os procedimentos sugeridos.

1.1.3.8 Teste de Volume

Tem o objetivo de testar os limites de processamento e carga do software e de toda infraestrutura, focalizando o aumento continuo dos parâmetros de execução para verificar se o sistema consegue lidar com grandes volumes de dados. Esses testes podem ser executados da seguinte forma (BARTIÉ, 2002):

- Aumentando sucessivamente o volume de transações;
- Aumentando sucessivamente o volume de consultas;
- Aumentando sucessivamente o tamanho de arquivos a serem processados.

1.1.3.9 Teste Paralelo

Tem o objetivo de testar se os resultados do sistema atual são iguais aos resultados do antigo sistema, os mesmos dados são executados em duas versões do sistema e os dados de saída são comparados para verificar a consistência do software.

1.1.4 Benefícios do teste de unidade

Um dos benefícios do teste de unidade é que ele pode ser automatizado. Nas empresas de pequeno e médio porte os testes são realizados manualmente, o que torna o processo demorado e sujeito a falhas humanas. Seria ideal que os processos fossem automatizados, onde ferramentas avaliam se o software funciona como esperado, os requisitos do sistema são validados e um número maior de testes são executados em menos tempo (LUFT, 2012). Segundo Ferreira (2011), os testes de unidade “devem ser automáticos e executados em intervalos regulares de validação e verificação sobre o correto funcionamento das unidades testadas”.

> O teste de uma unidade é o tipo mais importante de teste para a grande maioria das situações, já que é ele que deve testar se um algoritmo faz o que deveria ser feito e garantir que o código encapsulado por uma unidade deve produzir o efeito colateral esperado. Outra vantagem importante é que o teste de unidade é focalizado em um trecho específico do código, desta forma os erros encontrados são facilmente localizados, diminuindo o tempo gasto com depuração (BERNARDO E KON, 2008).

De acordo com o parecer de Justin Searls, relatado na obra “The Grumpy Programmer’s PHPUnit Cookbook”, os testes de unidade proporcionam um rápido *feedback*, o que ajuda o desenvolvedor a controlar o progresso do software. Searls (2015), expressa:

> A magia dos testes de unidade, em particular no contexto do desenvolvimento orientado a testes, é que ele dá ao desenvolvedor a capacidade de controlar o seu próprio senso de progresso. O tradicional *feedback* exige a nossa aplicação totalmente integrada e os nossos olhos para afirmar se estamos no caminho certo ou no caminho errado. O teste de unidade permite-nos estabelecer um ciclo de *feedback* em qualquer nível de integração que queremos (por exemplo, talvez um monte de objetos em coordenação, talvez uma função em isolamento puro), desde que podemos imaginar e implementar uma maneira de afirmar o comportamento de trabalho que não requer inspeção manual dos nossos olhos (SEARLS, 2015).

Os testes de unidade ajudam o desenvolvedor a pensar sobre os requisitos da aplicação e sobre a rota de implementação que ele deve escolher, dando uma visão imediata sobre o bom funcionamento do software e facilitando a localização dos defeitos. Searls (2015), complementa:

> Desta forma, mesmo que se depare com a implementação de uma solução de complexidade assustadora, testes de unidade geralmente pode nos ajudar a dividir o problema de tal maneira que nós podemos fazer (e mais importante, notar!) o progresso incremental em nosso caminho para a total conclusão. Ao dominar as ferramentas e o ofício de testes unitários, um *feedback* rápido é atingível, independentemente da idade ou da complexidade do projeto (SEARLS, 2015).

Quando um erro é introduzido no código, os testes de unidade conseguem localizá-lo rapidamente, facilitando a refatoração que é importante para garantir a longevidade do sistema. Além disso, quando os testes são executados corretamente, refletem uma confiança no processo de desenvolvimento, que vai desde os desenvolvedores até os clientes.

1.2 DESENVOLVIMENTO ORIENTADO A TESTES (TDD)

O desenvolvimento orientado a testes (TDD) é uma abordagem para o desenvolvimento de programas onde são intercalados testes e desenvolvimento de código.

Segundo Cardoso e Aniche, esse ciclo de desenvolvimento possui uma ideia bem simples, "o desenvolvedor deve começar a implementação pelo teste e deve, o tempo todo, fazer de tudo para que seu código fique simples e com qualidade".

O processo fundamental do TDD é mostrado na FIG 1.

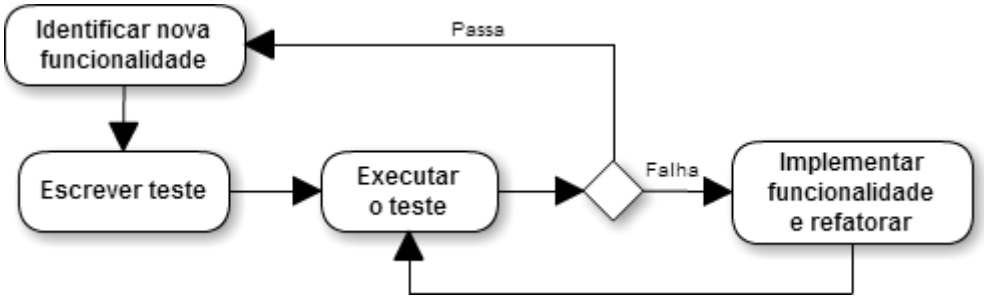

FIGURA 1 – Desenvolvimento orientado a testes (TDD) (SOMMERVILLE, 2011)

Inicialmente deve ser identificada a nova funcionalidade, o incremento deve ser pequeno, o segundo passo é escrever o teste automatizado e depois executar todos os testes, que deveram falhar já que a funcionalidade ainda não foi implementada, o próximo passo é implementar a funcionalidade e refatorar, para depois executar os testes novamente e repetir o processo até que os testes passem, quando todos os testes passarem é hora de implementar mais uma parte da funcionalidade (SOMMERVILLE, 2011).

Cobertura de código, teste de regressão, depuração simplificada e documentação de sistema são alguns dos benefícios do TDD. Segundo Sommerville (2011), um outro forte argumento a favor do desenvolvimento orientado a testes é que ele ajuda a clarear as ideias sobre o que o código supostamente deve fazer.

> Para escrever um teste, você precisa entender a que ele se destina [...]. Se você não sabe o suficiente para escrever os testes, não vai desenvolver o código necessário. Por exemplo, se seu cálculo envolve divisão, você deve verificar se não está dividindo o número por zero. Se você se esquecer de escrever um teste para isso, então o código para essa verificação nunca será incluído no programa (SOMMERVILLE, 2011, P. 156).

Para facilitar o desenvolvimento de software surgiram os *frameworks* que fazem com que os sistemas sejam criados de acordo com o padrão do mercado, assim quando um novo integrante ingressa no projeto, por exemplo, se ele conhece o *framework* rapidamente será capaz de se inteirar sobre o sistema, pois o entendimento fica mais fácil. Além disso, de acordo com Minetto (2007), “como todos os desenvolvedores que usam determinado *framework* programam usando as mesmas convenções, classes e bibliotecas, a manutenção de um programa é muito mais fácil”.

> *Framework* é um “arcabouço de software”. [...] Em outras palavras, um *framework* de desenvolvimento é uma “base” de onde se pode desenvolver algo maior ou mais específico. É uma coleção de códigos-fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de novos softwares (MINETTO, 2007, P.17).

A utilização de *frameworks*, segundo Bonoto (2012), “facilita a organização de uma aplicação, realiza injeção de dependência se necessário, valida campos com lógica de negócio e ajuda a tornar a estrutura do projeto organizada na maioria das vezes dentro de um modelo MVC” nas aplicações Web. Além disso, os *frameworks* fornecem uma arquitetura de esqueleto para a aplicação, dando suporte ao reúso de projeto e de classes específicas de sistema (PRESSMAN, 2011).

Com todos esses benefícios, fica claro a utilidade que as ferramentas de testes automatizados têm para o processo de desenvolvimento orientado a testes, que apesar de exigir um tempo maior de planejamento, possibilita que os testes de cada unidade sejam executados quantas vezes forem necessárias e com um custo muito baixo se comparado com testes manuais.

Como o TDD faz com que o código seja desenvolvido em incrementos muito pequenos, e os testes devem ser executados toda vez que o programa é refatorado, é essencial usar um ambiente de testes automatizados, como o PHPUnit, assim é possível executar centenas de testes separados em poucos segundos.

1.3 PHPUNIT

O PHPUnit é um framework para automatizar testes de unidade na linguagem de programação PHP (*Hypertext Preprocessor*). Baseado no XUnit, o PHPUnit foi criado por Sebastian Bergmann, e é mantido pelo mesmo com a ajuda de contribuidores, atualmente está na versão 5.0 e é gratuito, com código aberto (BERGMANN, 2015).

Segundo Hartjes (2015), PHPUnit pode ser intimidante, mesmo para obter somente o esqueleto de um teste, devido a sua imensa documentação e grande número de opções de configuração, mas, é possível começar com apenas alguns princípios básicos antes de ir para configurações mais complicadas, como ignorar certos tipos de testes ou alterar as configurações padrão.

As duas regras fundamentais do framework de acordo com Bergmann (2015), são que as classes devem terminar com "Test" e os métodos de teste devem começar com "test".

A melhor maneira de se beneficiar com o uso do PHPUnit é começar projetando a classe, então é necessário criar o conjunto de testes, implementar a classe e executar os testes verificando as falhas e erros para, então, executar novamente os testes. Pode parecer que esse processo exige muito tempo, mas o PHPUnit gasta apenas alguns segundos para executar o conjunto de testes.

Os testes para uma classe vão dentro de uma *ClasseTest* que herda na maioria das vezes de *PHPUnit_Framework_TestCase*. Os testes são métodos públicos, dentro deles são usados métodos de confirmação como *assertEquals()*, para confirmar que um valor real equivale a um valor esperado (BERGMANN, 2015).

> Uma das grandes vantagens do PHPUnit é que os erros são isolados a fim de permitir uma execução rápida dos testes. Assim é possível um *feedback* de testes com maior qualidade, pois um erro encontrado no início da execução não interrompe a verificação. O processo continua mantendo isolados os códigos onde erros foram encontrados, resultando uma melhor performance (TONIAZZO, 2007).

A complexidade pode ser uma desvantagem do PHPUnit, porém, o uso da metodologia adequada e de estratégias claras para a escrita dos testes apresentam resultados que compensam a utilização do *framework*, que se apresenta eficiente no processo de encontrar falhas nos softwares.

1.4 ZEND FRAMEWORK

A fim de diminuir a complexidade no desenvolvimento de sistemas são realizados esforços no desenvolvimento de novos *frameworks*, como o *Zend Framework* (ZF) que foi criado com a proposta de ser flexível e facilitar o uso da linguagem PHP, já que os requisitos mais complexos para o desenvolvimento de sistemas tornam inviável a utilização da linguagem simples. O ZF permite uma fácil personalização por ser uma biblioteca leve e fracamente acoplada, "usa também uma base de códigos exaustivamente testada e extensível, uma arquitetura flexível, além

da integração com diversas ferramentas modernas aumentando a acessibilidade e a usabilidade do sistema" (FERREIRA, 2011).

> O *Zend Framework* faz parte do projeto PHP *Collaboration*, uma iniciativa da empresa *Zend Technologies*. Ele é um *framework* escrito em PHP5 que utiliza todos os recursos de orientação a objetos fornecidos por esta versão da linguagem, além de prezar a utilização de padrões de projetos, visando à construção de componentes altamente reutilizáveis (BONOTO, 2012, P. 3).

Com uma coleção de componentes que podem ser estendidos, o *Zend Framework* pode ser usado com outras aplicações e *frameworks*, possibilitando aos desenvolvedores adaptarem a estrutura conforme as necessidades do sistema, além de estar em conformidade com as convenções que são consideradas as melhores práticas de programação.

O ZF segue o padrão *Model-View-Controller* (MVC), dando suporte à apresentação de dados de maneiras diferentes, e permitindo a interação com cada uma dessas apresentações.

A FIG. 2 apresenta o padrão MVC com as três camadas lógicas nas quais a aplicação é dividida, cada camada possui responsabilidades distintas:

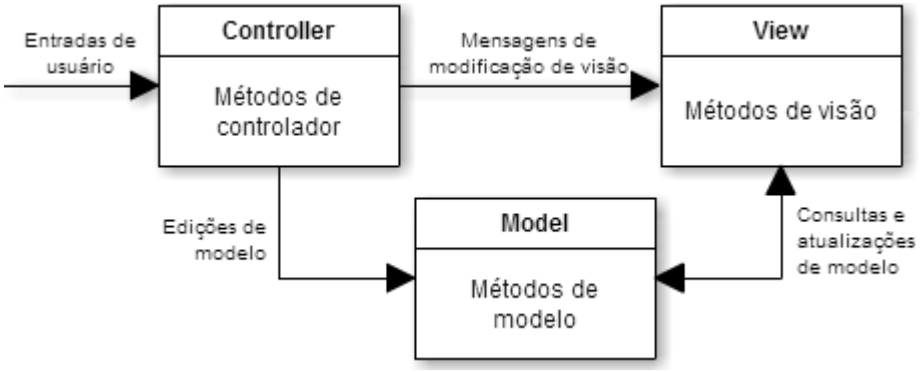

FIGURA 2 – O padrão MVC (SOMMERVILLE, 2011)

Enquanto a *view* define a apresentação e coleta dados do usuário, o *model* define a funcionalidade básica detrás das abstrações e o *controller* vincula todo o padrão do conjunto, manipulando o *model* e definindo qual apresentação a *view* vai exibir.

> O *controller* será responsável por conter toda a lógica de domínio de negócio da aplicação, envolvendo desde um cálculo complexo, até a lógica de acesso ao banco de dados. A *view* fica responsável em exibir para o usuário dados do *model*, ou interfaces gráficas para entrada de dados. O *controller* é o responsável por intermediar requisições da *view* e delegar lógica do *model*, assim como notificar a *view* sobre possíveis mudanças nos dados do *model* (BONOTO, 2012, P.4).

Segundo Sommerville (2011), “quando os dados são modificados por meio de uma das apresentações, o modelo de sistema é alterado e os controladores associados a cada visão atualizam sua apresentação”.

Uma base de códigos exaustivamente testada e extensível é essencial para o desenvolvimento de grandes projetos, em especial aqueles que tem muitas pessoas envolvidas. O PHPUnit é o *framework* que compõe a suíte de testes sólida do ZF, isso porque os testes de unidade ajudam a aliviar a carga, visto que testar manualmente cada componente individual após cada mudança é impraticável, e os testes automatizados conseguem mostrar quando algo para de funcionar depois que as mudanças são feitas, quando os testes são usados corretamente (ZEND FRAMEWORK, 2015).

Um cenário muito comum no contexto de testes de software é que a preparação e execução dos testes são feitas de maneira muito superficial. Geralmente os testes são deixados para última etapa do processo de desenvolvimento, e por problemas como falta de tempo para planejamento, os testes ficam incompletos e são tratados como causadores do aumento dos custos e prazos dos projetos. O grande problema é que as empresas não possuem um processo de teste estruturado com documentos e regras bem definidas, e os testes são executados pelos próprios desenvolvedores, que não são profissionais qualificados para essa atividade, por isso os sistemas são falhos e tem custos elevados.

2 METODOLOGIA

O presente trabalho teve por objetivo analisar a adequação dos testes de unidade durante o processo de desenvolvimento de software com *Zend Framework* visando a correção de falhas e a qualidade das aplicações.

Após a realização do estudo através da pesquisa bibliográfica, foi realizada uma pesquisa com profissionais relacionados à produção de software, a fim de conhecer suas experiências sobre o assunto, através de um questionário que se encontra no APÊNDICE A e que será detalhado nas subseções seguintes.

2.1 PÚBLICO ALVO DO QUESTIONÁRIO

Como público respondente do questionário, foram escolhidos profissionais relacionados à produção de softwares e estudantes da área de Tecnologia da Informação (TI), com o intuito de identificar o atual perfil destes e entender o ponto de vista dos mesmos em relação ao tema em questão, testes de software.

Os entrevistados foram sensibilizados a participar do questionário que foi divulgado por *e-mails* enviados a profissionais de TI, alunos e ex-alunos do curso de Ciência da Computação das Faculdades Integradas de Caratinga, contendo uma mensagem explicando o objetivo da pesquisa e um *link* que possibilita o acesso ao questionário. A pesquisa também foi divulgada em grupos de discussão e nas redes sociais *Facebook* e *LinkedIn* a profissionais relacionados à programação, qualidade e testes de software e desenvolvimento com *Zend Framework*.

2.2 ELABORAÇÃO DO QUESTIONÁRIO

As questões que compõem o questionário de teste de software foram elaboradas com base no conhecimento obtido através das leituras realizadas.

Algumas questões foram baseadas no trabalho de Luft (2012), cujo objetivo era verificar os métodos de testes existentes nas empresas de software na área de TI.

Foram feitas na pesquisa quarenta e sete perguntas que foram organizadas em sete grupos. Para proporcionar um fluxo de leitura agradável e lógico e para tentar tornar o processo de coleta de dados menos cansativo, o formulário de pesquisa foi dividido em quinze páginas, de forma que as páginas estavam condicionadas às respostas das questões das páginas anteriores, ou seja, os entrevistados não responderam todas as quarenta e sete questões. Além disso, a primeira página contém uma mensagem explicando o objetivo da pesquisa, informando que os dados são confidenciais e agradecendo pela colaboração. As questões não foram numeradas e a maioria delas são obrigatórias, assim o entrevistado só conseguia enviar o formulário após responder todas as perguntas.

2.3 O QUESTIONÁRIO

As quarenta e sete questões do questionário foram organizadas em sete grupos, cada um deles buscando coletar dados sobre um tipo de informação.

O primeiro grupo engloba o perfil do respondente, com questões como *e-mail*, cidade, estado, idade e gênero, para identificar o profissional e, formação acadêmica, cargo que ocupa profissionalmente e clientela, para saber o meio em que ele trabalha.

O segundo grupo é sobre o processo de testes, são vinte e uma questões onde os entrevistados são indagados sobre a necessidade de utilização dos testes de software, o tempo de experiência, como os testes são realizados, quais testes e critérios são utilizados, e qual a contribuição que os testes realizados apresentam no projeto.

O terceiro grupo é sobre o processo de testes automatizados, são três questões direcionadas aos entrevistados que realizam testes automáticos, para saber como esses testes são avaliados, quais as principais vantagens e quais ferramentas são utilizadas.

O quarto grupo sobre desenvolvimento orientado a testes (TDD), contém uma pergunta para saber se o entrevistado conhece a metodologia e mais três para saber

como a metodologia é avaliada, qual a aderência por parte dos colaboradores e se o desenvolvimento orientado a testes é utilizado pelos profissionais no processo de desenvolvimento dos softwares.

O quinto grupo é composto por três questões sobre testes de unidade, para saber o quanto agregam ao projeto, quais ferramentas são utilizadas e se o entrevistado conhece o PHPUnit para ir às questões do sexto grupo, sobre o *framework* PHPUnit, que contém quatro questões para saber como o *framework* é avaliado, qual a dificuldade e a eficácia e se é utilizado.

O sétimo grupo sobre *Zend Framework* é composto por mais quatro questões, para saber se é conhecido, como o *framework* é avaliado, qual a eficiência que apresenta no projeto e se é utilizado pelos entrevistados.

2.4 COLETA DE DADOS

O questionário foi elaborado mediante um formulário criado por meio da ferramenta *Google Docs*. Essa ferramenta possibilita a criação, compartilhamento e realização da pesquisa *online*, sendo possível acessar o questionário em qualquer local e em qualquer computador.

A coleta de dados da pesquisa durou um mês. O questionário foi disponibilizado no dia 28 de setembro de 2015 às 00h00min, ficando disponível até o dia 28 de outubro de 2015 às 00h00min, neste período foram coletadas 85 respostas.

2.5 TRATAMENTO DE DADOS

As respostas coletas pelo formulário foram armazenadas automaticamente em uma planilha criada pelo *Google Docs,* o que auxiliou a tabulação dos dados. Através da planilha é possível visualizar cada reposta de cada participante separadamente. Para a manipulação dos dados a planilha foi exportada para edição no *Microsoft Excel*, onde os dados foram organizados de acordo com o perfil do respondente,

possibilitando a análise das respostas e a exibição dos resultados através de gráficos com a porcentagem das respostas, para proporcionar uma melhor compreensão dos resultados que serão apresentados na seção seguinte.

3 RESULTADOS

Nesta seção serão apresentados os resultados obtidos por meio das respostas dos entrevistados, um total de 85 pessoas responderam o questionário. Durante o processo de análise, duas das entrevistas realizadas foram avaliadas como duplicidade, e uma outra pessoa respondeu o questionário de forma aleatória, essas três respostas foram desconsideradas no tratamento dos dados a seguir, portanto, todos os itens analisados não contêm as respostas dos entrevistados em questão. Assim sendo, os resultados apresentados nessa seção são referentes aos 82 participantes restantes.

3.1 ANÁLISE DAS RESPOSTAS COLETADAS

As respostas exibidas a seguir serão organizadas pelos sete grupos do questionário, cada um em uma seção, sendo: Perfil do respondente, Processo de testes, Processo de testes automatizados, Desenvolvimento orientado a testes, Testes de unidade, *Framework* PHPUnit e *Zend Framework*.

3.1.1 Perfil do Respondente

A pesquisa contou com as respostas de 82 pessoas, de 12 estados brasileiros e um estado norte americano (Califórnia), 46,34% afirmou residir em Minas Gerais na cidade de Caratinga, 82,93% dos respondentes são do sexo masculino e 17,07% do sexo feminino. A maioria dos entrevistados forma um público jovem, entre eles 48,78% têm idade entre 15 e 25 anos, 34,15% com idade entre 26 e 35 anos, 15,85% com idade entre 36 e 45 anos e a minoria representada por 1,22% dos entrevistados com a idade superior aos 46 anos.

Quando perguntados sobre a formação acadêmica, 53,66% dos entrevistados declararam formação de graduação em andamento ou finalizada, 23,17% são pós-graduados, 12,20% são técnicos, 7,32% são mestres, 1,22% cursaram doutorado e 2,44% declararam ter outros tipos de formação.

Em relação às funções desempenhadas profissionalmente, apenas 6,17% disseram desempenhar funções ligadas exclusivamente à análise de testes, 11,11% responderam ser gerentes de projeto, 17,28% são analistas de sistemas e 34,57% disseram desempenhar funções ligadas ao desenvolvimento de software. Os demais respondentes, cerca de 30,86%, disseram desempenhar outras funções na área de Tecnologia da Informação (TI), como estudante, professor, suporte, administrador de redes, supervisor de TI e arquiteto de softwares.

Aproximadamente 36,59% dos entrevistados disseram que trabalham ou prestam serviços para empresas locais e outros 24,39% para empresas regionais, 14,63% trabalham para empresas de outros estados, 6,10% para empresas de outros países e 18,29% declaram outros tipos de clientela.

3.1.2 Processo de Testes

As questões dessa seção têm por objetivo indagar os entrevistados a respeito da necessidade de utilização dos testes de software, sobre os motivos pelos quais eles não seguiriam um processo de testes, se têm conhecimento sobre testes, qual tempo de experiência, como avaliam o processo de testes, como os testes são realizados, quais testes, critérios e ferramentas são utilizadas, e qual a contribuição que os testes realizados apresentam no projeto.

A qualidade dos softwares interfere diretamente no sucesso das aplicações. Como você avalia a necessidade de utilização dos Testes de Software visando essa qualidade? Essa pergunta foi feita aos entrevistados com o intuito de descobrir qual o ponto de vista deles em relação aos testes realizados nos softwares para a garantia da qualidade.

O GRAF. 1 mostra a necessidade dos testes na perspectiva dos respondentes.

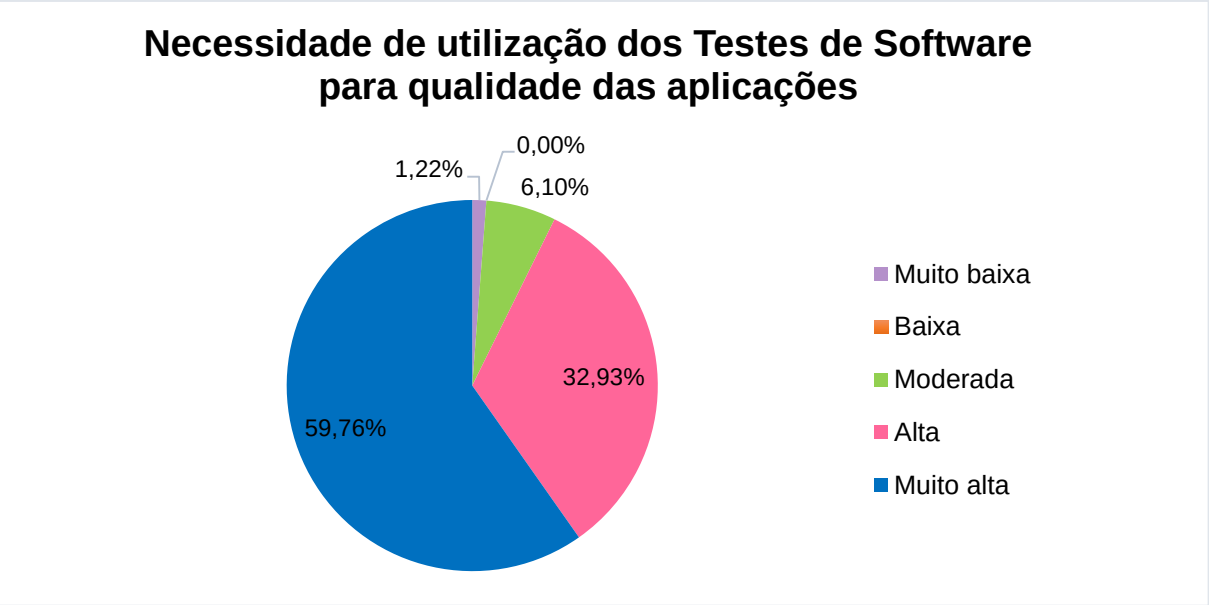

GRÁFICO 1 - A necessidade de utilização dos Testes de Software visando a qualidade e sucesso das aplicações
Fonte: Próprio autor

Aproximadamente 59,76% dos entrevistados declararam que a necessidade dos testes é muito alta, 32,93% disseram que a necessidade é alta, 6,10% consideram a necessidade moderada dos testes e 1,22% afirmaram ser muito baixa.

Foi perguntado aos entrevistados os motivos que os levariam a não seguir um processo de testes de software. A maioria respondeu que a causa seria falta de tempo (32,93% aproximadamente), a falta de controle e gestão do processo foi assinalada por 18,29% dos respondentes, 14,63% disseram que a falta de treinamento é a principal causa, 12,20% responderam ser a falta de conhecimento e 2,44% a falta de confiança no processo. Dos demais respondentes, 14,63% não souberam opinar e outros 4,88% atribuíram as causas ao custo, à necessidade do cliente e ao tempo reduzido para o desenvolvimento dos sistemas.

Do total de entrevistados, 74,39% declararam ter conhecimento sobre testes de software, e, portanto, foram selecionados para responder as demais questões sobre o processo de testes. Os outros 25,61% que responderam não possuir conhecimento, foram encaminhados para uma última questão onde puderam transcrever os motivos pelos quais não conhecem e não utilizam testes de software.

Diversas foram as razões citadas pelos entrevistados que declararam não ter conhecimento sobre testes de software, dentre elas a falta de experiência e de treinamento, a falta de conhecimento e capacitação, a falta de prioridade e por conta dos projetos atrasados na empresa, e alguns disseram que não estão atuando na

área. Contudo, todos concordaram que um software realizado sem testes pode não corresponder às expectativas desejadas, sendo de extrema importância a realização de testes para a qualidade dos sistemas desenvolvidos.

As questões descritas a seguir contam apenas com as respostas dos entrevistados que declararam conhecer sobre testes de software (um total de 61 respondentes).

Para saber um pouco mais sobre o entrevistado, foi perguntado qual o tempo de experiência ele possui com um processo de testes (GRAF. 2).

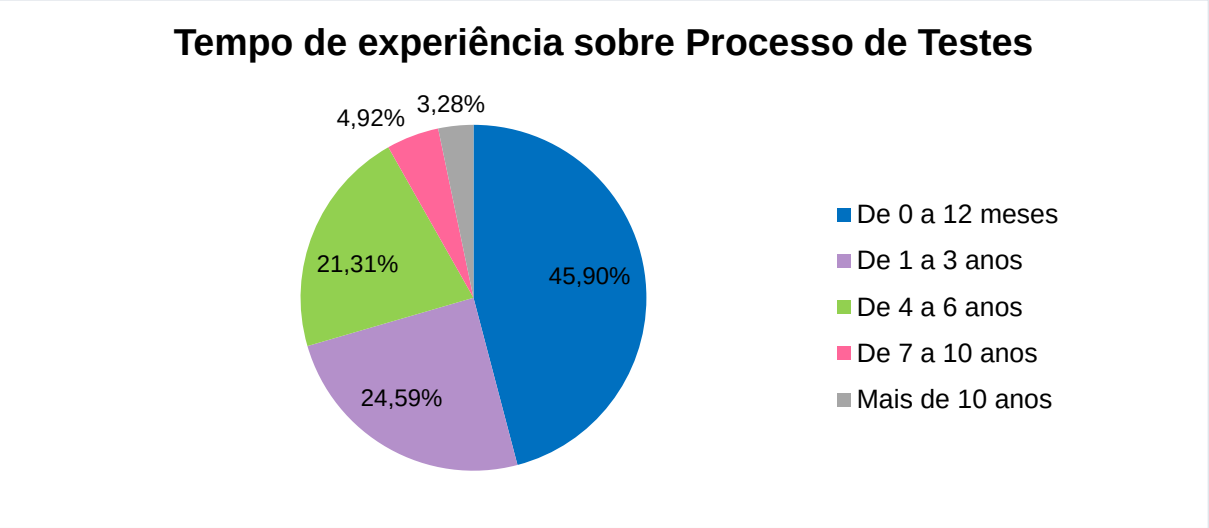

GRÁFICO 2 - Tempo de experiência sobre Processo de Testes
Fonte: Próprio autor

Dentre os entrevistados, 45,9% responderam ter experiência com testes num período de 0 (zero) a 12 meses, 24,59% assinalaram ter experiência de 1 a 3 anos, 21,31% de 4 a 6 anos, 4,92% de 7 a 10 anos, e 3,28% tem experiência com testes há mais de 10 anos, isso mostra que o envolvimento com testes de software ainda é muito recente para a maioria dos entrevistados.

Os participantes responderam também questões sobre há quanto tempo o processo de testes foi implantado no projeto de desenvolvimento (GRAF. 3) e como ele avalia o processo de testes utilizado (GRAF. 4).

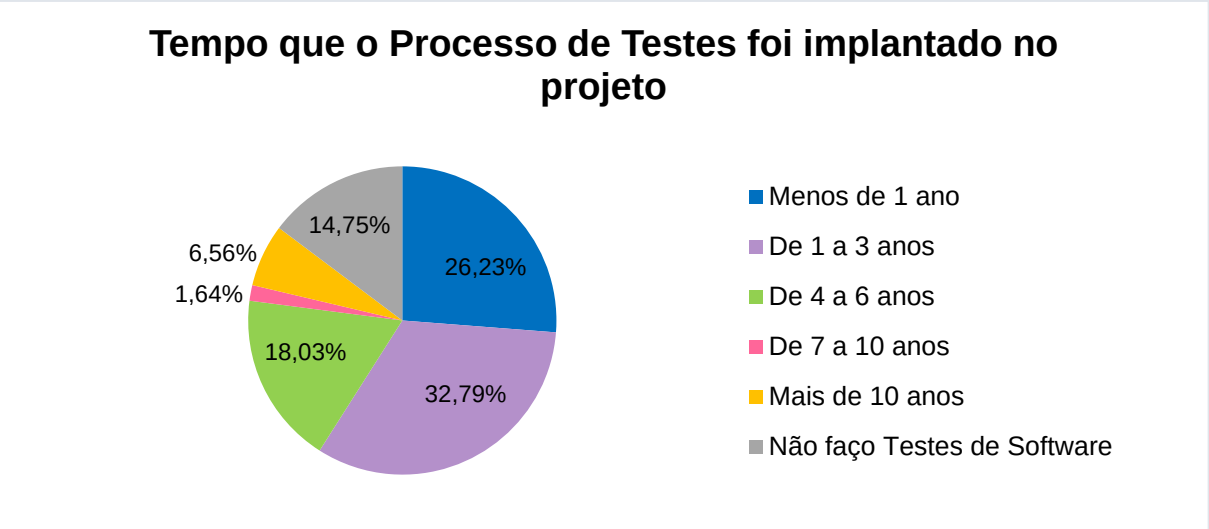

GRÁFICO 3 - Tempo que o Processo de Testes foi implantado no projeto
Fonte: Próprio autor

Para 26,23% dos projetos, o processo de testes foi implantado há menos de 1 ano, em 32,79% foi implantado entre 1 a 3 anos, 18,03% entre 4 a 6 anos, 1,64% de 7 a 10 anos e 6,56% dos projetos tem um processo de testes há mais de 10 anos. 14,75% dos respondentes disseram que não fazem testes de software, isso porque o processo de testes, em muitos projetos, não é bem definido.

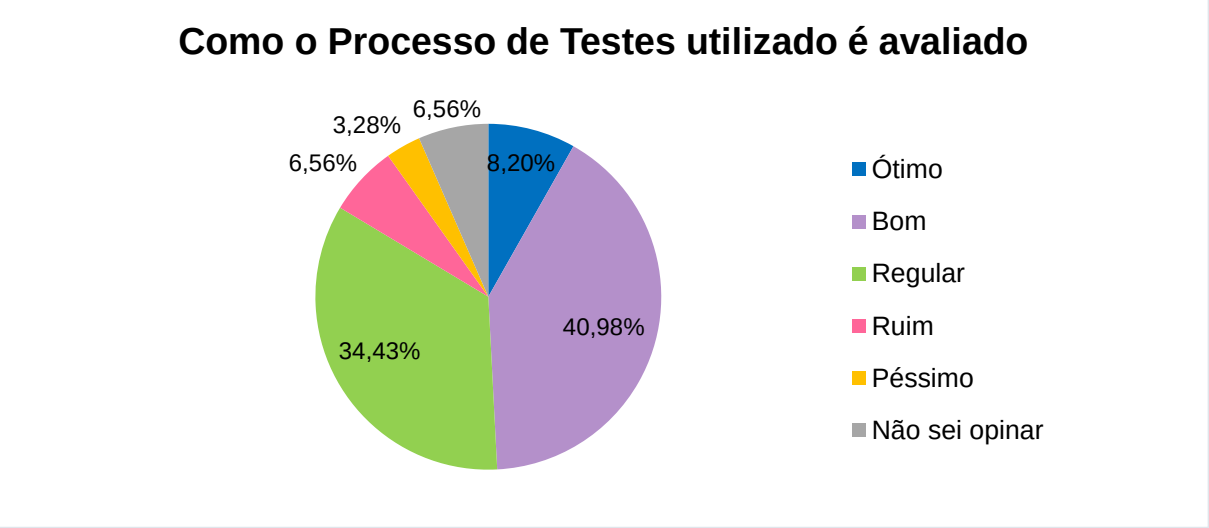

GRÁFICO 4 - Como o Processo de Testes utilizado é avaliado
Fonte: Próprio autor

Aproximadamente 8,2% dos entrevistados avaliam o processo de testes utilizado como ótimo, 40,98% avaliam como sendo um processo bom, 34,43% disseram ser regular, 6,56% avaliaram como ruim e 3,28% avaliaram como péssimo, 6,56% dos respondentes não souberam opinar.

Algumas questões desse grupo tinham o intuito de saber como os testes são realizados e quais testes, critérios e ferramentas são utilizadas pelos entrevistados nos seus projetos.

O GRAF. 5 mostra as respostas dos entrevistados sobre quais testes de software são realizados no projeto, é importante frisar que nessa questão era possível assinalar mais de uma opção.

GRÁFICO 5 - Testes de software que são realizados no projeto
Fonte: Próprio autor

De acordo com as respostas é possível observar que os testes mais realizados são: teste funcional (37), teste de usabilidade (32), teste de unidade e teste de sistema (ambos com 30), teste de aceitação (27), teste de integração (26), teste estrutural (25) e teste de desempenho (23), os demais, teste de segurança (19), teste de carga (18), teste de configuração (16), teste de regressão e teste de instalação (ambos com 12), teste de recuperação e teste de volume (ambos com 6), e teste paralelo (2), são os menos utilizados nos projetos.

Sobre os critérios de teste utilizados foram feitas duas questões, sobre os testes de caixa preta e caixa branca, essas questões não eram obrigatórias. Em relação aos critérios de teste caixa preta, 20% utiliza análise de valor limite, 16,67% utiliza grafo de causa e efeito, 11,67% utiliza particionamento de equivalência, e 51,67% desconhece os critérios indicados. Já em relação aos critérios de teste caixa

branca, 30% utiliza critérios baseados em fluxo de dados, 20% utiliza critérios baseados em fluxo de controle, 15,71% utiliza critérios baseados na complexidade, e 34,29% desconhece os critérios indicados. O fato da grande maioria dos respondentes desconhecer os critérios de teste utilizados, comprova que o processo de testes não é bem definido e os colaboradores do projeto não tem conhecimento adequado sobre os testes que são executados.

Os entrevistados foram questionados também sobre quando é iniciada a execução dos testes (GRAF. 7) e quem são os responsáveis por realizar esses testes, nesta questão era possível assinalar mais de uma opção (GRAF. 6).

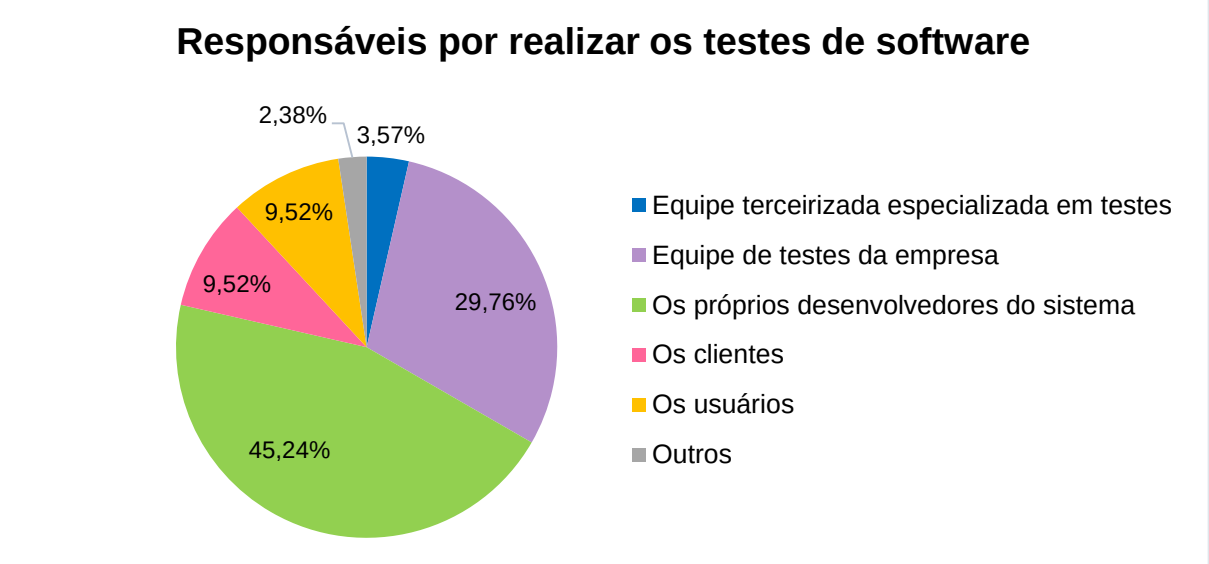

GRÁFICO 6 - Responsáveis por realizar os testes de software
Fonte: Próprio autor

Na maioria dos casos (45,24%), os próprios desenvolvedores do sistema são os responsáveis por realizar os testes, 29,76% dos testes são realizados pela equipe de testes da empresa, 9,52% dos testes são realizados pelos clientes e pelos usuários, 3,57% são realizados por equipes terceirizadas especializadas em testes, os outros 2,38% representam os demais responsáveis. Esses dados mostram que apesar de serem realizados, os testes não são feitos por profissionais especializados, mas sim pelos próprios desenvolvedores do sistema, que não são capacitados para executar os diversos tipos de testes.

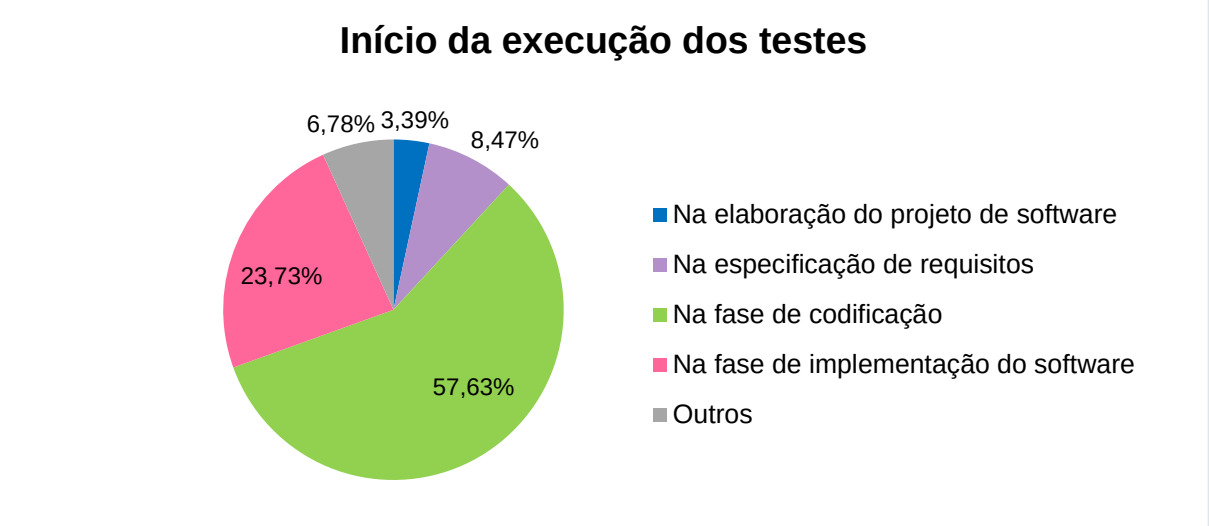

GRÁFICO 7 - Início da execução dos testes
Fonte: Próprio autor

Os entrevistados afirmaram que na maioria das vezes os testes são iniciados na fase de codificação, representado por 57,63%, outros 23,73% disseram que é na fase de implementação do software, 8,47% disseram ser na especificação dos requisitos e 3,39% apontaram a elaboração do projeto de software como fase inicial de execução dos testes. Esse gráfico mostra que não existe planejamento para a atividade de testes, haja vista que os testes realizados durante a codificação correspondem aos testes realizados pelos desenvolvedores do sistema.

Quando perguntados sobre a forma que os testes de software são realizados, 57,38% dos respondentes afirmaram fazer os testes manualmente, 4,92% através da automatização e 37,70% responderam que realizam ambas as metodologias. Além de analisar a maneira que os testes são realizados, essa pergunta serviu para selecionar os entrevistados que são considerados aptos a responder as questões de outro grupo, que serão apresentadas na seção Processo de testes automatizados.

As demais questões desse grupo buscavam entender o impacto e a contribuição dos testes de software durante o desenvolvimento.

O GRAF. 8 mostra as fases do processo de desenvolvimento onde são encontrados o maior número de erros.

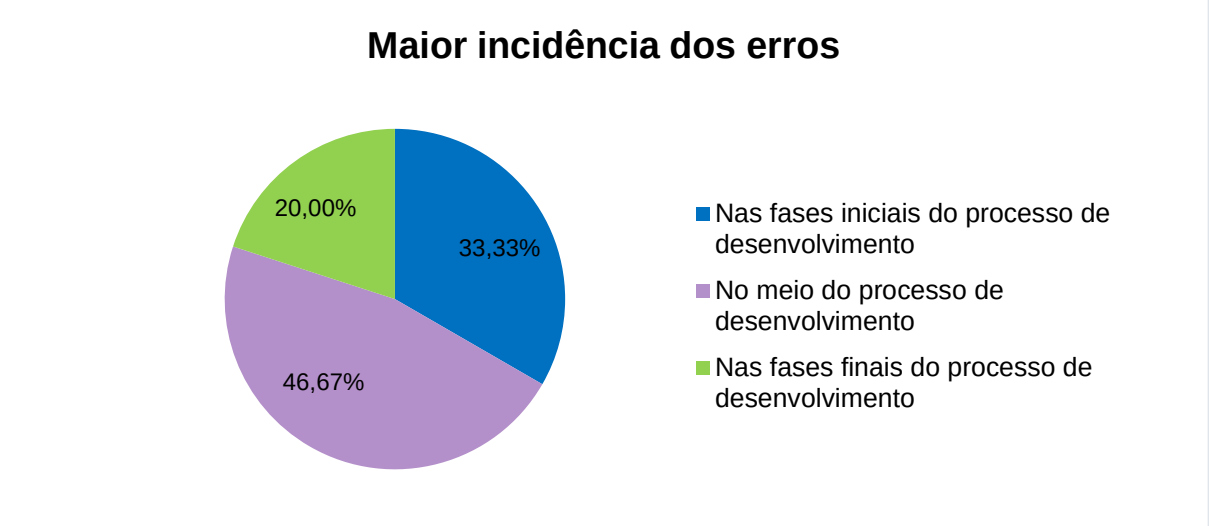

GRÁFICO 8 - Maior incidência dos erros
Fonte: Próprio autor

De acordo com 33,33% dos respondentes, as fases iniciais do processo de desenvolvimento apresentam a maior incidência dos erros, no meio do processo são encontrados 46,67% dos incidentes, e 20% são descobertos nas fases finais do processo de desenvolvimento.

Os erros mais comuns de serem encontrados nos sistemas desenvolvidos são apresentados no GRAF. 9.

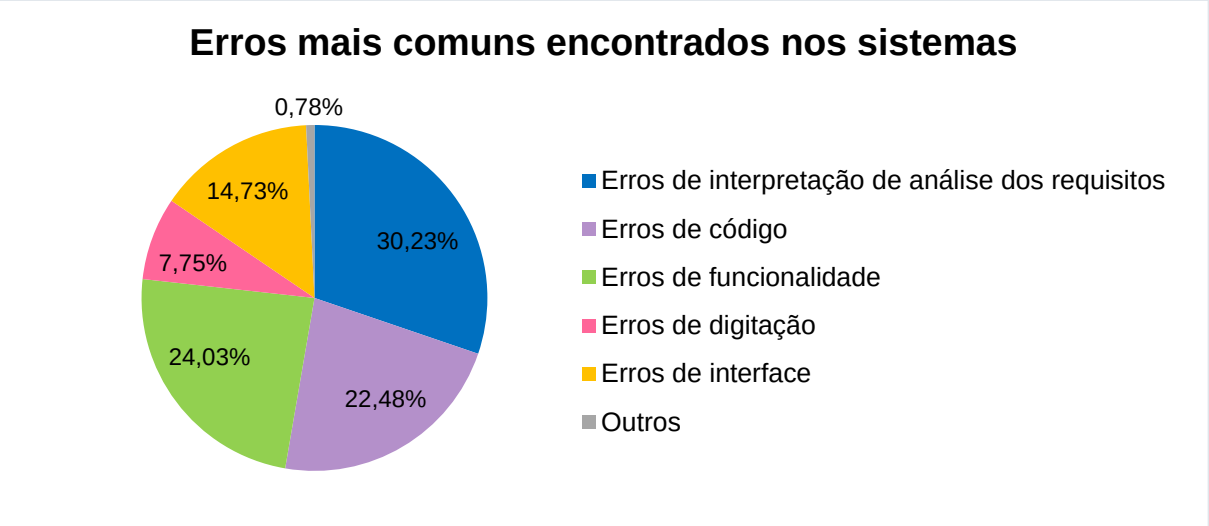

GRÁFICO 9 - Erros mais comuns encontrados nos sistemas
Fonte: Próprio autor

Os erros de interpretação de análise dos requisitos englobam 30,23% dos erros encontrados, 22,48% são erros de código, 24,03% são erros de funcionalidade, 7,75% são erros de digitação, e os erros de interface representam 14,73% dos erros encontrados, 0,78% corresponde a outros erros encontrados nos sistemas. Pode se

observar que a maioria dos erros está relacionada à interpretação dos requisitos do sistema, aplicar testes de unidade com a metodologia TDD ajudaria a reverter esse quadro.

Foi perguntado aos entrevistados sobre a frequência dos erros encontrados após a implantação da atividade de teste de software, o GRAF. 10 mostra que os erros encontrados depois da entrega do produto ao cliente, diminuíram consideravelmente.

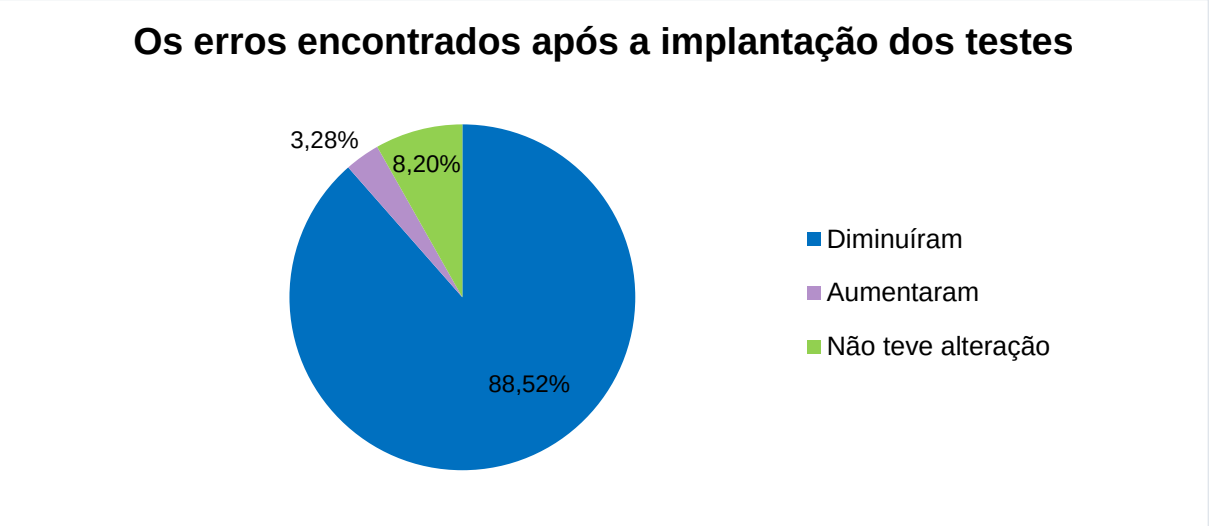

GRÁFICO 10 - Quantidade de erros encontrados após a entrega do produto ao cliente, depois da implantação dos testes de software
Fonte: Próprio autor

Em 88,52% dos casos, os erros após a entrega do produto diminuíram com o uso dos testes de software, 3,28% disseram que os erros aumentaram, e 8,2% responderam que não teve alteração.

Dentre as principais dificuldades encontradas para a realização dos testes, foram assinaladas que a atividade de teste é limitada por restrições do cronograma (21,52%), que existe uma falta de profissionais especializados na área de testes (13,29%), que existe o desconhecimento de técnicas de teste adequadas e dificuldades em implantar um processo de testes (ambos com 13,29%), e ainda que há o desconhecimento de um procedimento de teste adequado e sobre como planejar a atividade de teste (ambos com 12,66%), e que o teste é um processo caro (9,49%). Realmente, a pressa para entregar o produto final e a falta de conhecimento sobre como realizar corretamente os testes, são as principais dificuldades para sua realização, entretanto, apesar de ser considerado um processo caro, o GRAF. 11 confirma que, na maioria dos casos, os testes não geram mais custos.

Quando perguntados sobre o percentual de tempo do projeto de desenvolvimento que é consumido com a atividade de testes, 8,2% dos entrevistados disseram que os testes ocupam até 5% do tempo, 24,59% disseram consumir entre 6% e 10% do tempo, 21,31% disseram consumir entre 11% e 20% do tempo, 22,95% disseram que os testes consomem de 21% a 30% do tempo, 4,92% disseram que consomem de 31% a 40% do tempo do projeto, e 1,64% assinalaram que consomem de 41% a 50% e acima de 50%, outros 14,75% não souberam opinar.

Os entrevistados afirmaram que o valor efetivo dos testes, em 50,82% dos casos apresentam grande contribuição no projeto, 42,62% disseram que os testes apresentam média contribuição, e a minoria (somente 6,56%) responderam que os testes apresentam uma pequena contribuição.

Em relação aos custos gastos com o projeto após a implantação dos testes de software, a maioria dos entrevistados disseram que não teve alteração (GRAF. 11).

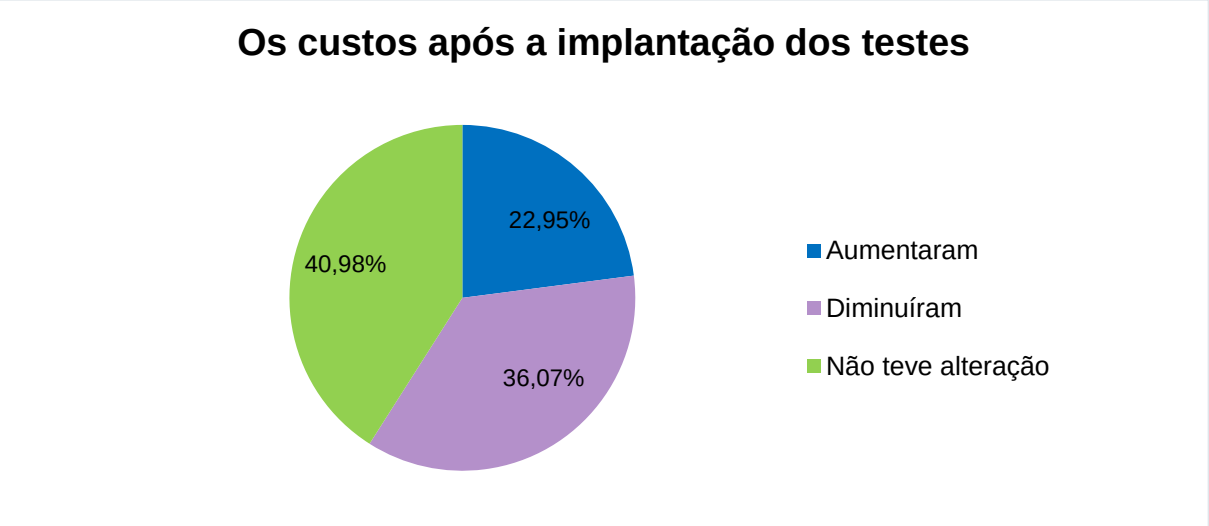

GRÁFICO 11 - Os custos após a implantação dos testes
Fonte: Próprio autor

Aproximadamente 22,95% dos entrevistados responderam que houve um aumento dos custos após a implantação dos testes de software, 36,07% responderam que os custos diminuíram, e 40,98% dos respondentes disseram que não houve alteração sobre os gastos com o projeto, depois da implantação da atividade de testes, o que mostra que os testes não causam grandes impactos nos custos do projeto.

Como é desenvolver sistemas com o *Zend Framework* utilizando o *framework* de testes PHPUnit com o apoio da técnica de Desenvolvimento Orientado a Testes (TDD)? Essa foi a última pergunta realizada aos entrevistados que declararam conhecer testes de software, e o resultado pode ser visto no GRAF. 12.

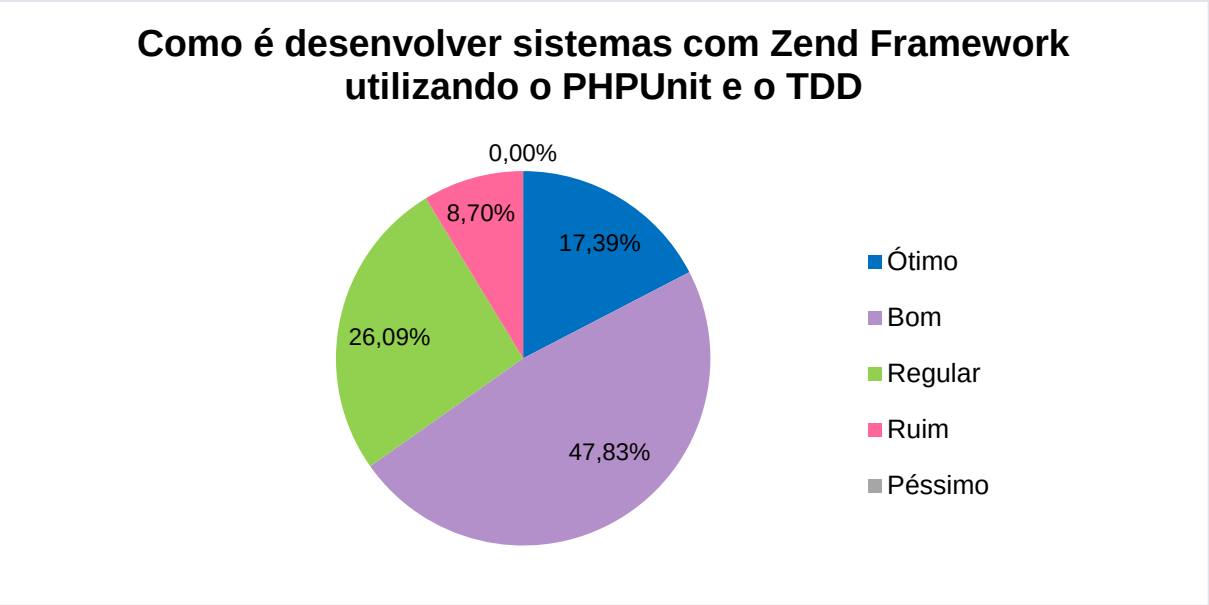

GRÁFICO 12 - Como é desenvolver sistemas com o Zend Framework utilizando o PHPUnit e o TDD
Fonte: Próprio autor

Dentre os respondentes que souberam opinar, 47,83% disseram que é bom desenvolver sistemas utilizando em conjunto o ZF, o PHPUnit e o TDD, 26,09% disseram que é regular, 17,39% disseram que é ótimo, e apenas 8,7% avaliou como ruim. Não houveram avaliações para péssimo, concluindo assim que em geral a combinação desses métodos é bom para o desenvolvimento de sistemas na linguagem PHP.

3.1.3 Processo de Testes Automatizados

Aos 23 respondentes que disseram realizar testes automatizados, foi perguntado como eles avaliam o processo de testes automatizados e quais as vantagens em relação aos testes realizados manualmente, nesta questão era possível assinalar mais de uma opção (GRAF. 13).

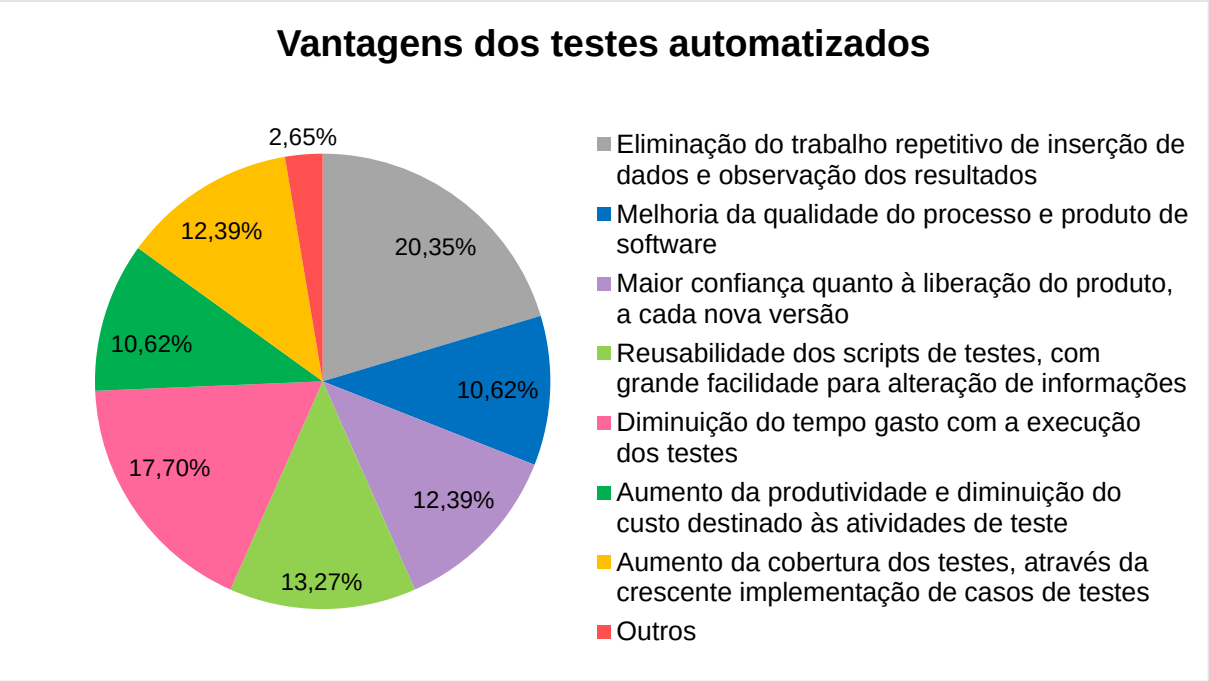

GRÁFICO 13 – Vantagens dos testes automatizados em relação aos testes feitos de forma manual
Fonte: Próprio autor

Dentre as principais vantagens dos testes automatizados em relação aos testes feitos de forma manual os mais votados foram: Eliminação do trabalho repetitivo de inserção de dados e observação dos resultados, com 20,35%; Diminuição do tempo gasto com a execução dos testes, com 17,7%; Reusabilidade dos scripts de testes, com 13,27%; Maior confiança quanto à liberação do produto e Aumento da cobertura dos testes, ambos com 12,39%; Melhoria da qualidade do processo e produto de software e Aumento da produtividade e diminuição do custo destinado às atividades de teste, ambos com 10,62%; e outros 2,65% citados como: *feedback* mais rápido, economia de dinheiro e entrega contínua com maior segurança.

Mais da metade dos entrevistados (57,69%) avaliaram o processo de testes automatizados como sendo bom, 26,92% avaliaram como ótimo e 11,54% avaliaram como regular, não houve nenhuma avaliação ruim ou péssima, os demais 3,85% disseram não saber opinar.

3.1.4 Desenvolvimento Orientado a Testes

As questões dessa seção tinham por objetivo descobrir se os entrevistados conhecem e utilizam o desenvolvimento orientado a testes, e como eles avaliam a aplicação dessa metodologia no processo de testes.

Do total de entrevistados, 28 pessoas declararam ter conhecimento sobre a metodologia de desenvolvimento orientado a testes, e, portanto, foram selecionadas para responder as demais questões sobre TDD. As outras pessoas que disseram não conhecer a metodologia, foram encaminhadas para a página de questões sobre Testes de Unidade.

Foi perguntado aos entrevistados como eles avaliam o processo de testes baseado no desenvolvimento orientado a testes, o GRAF. 14 mostra o resultado.

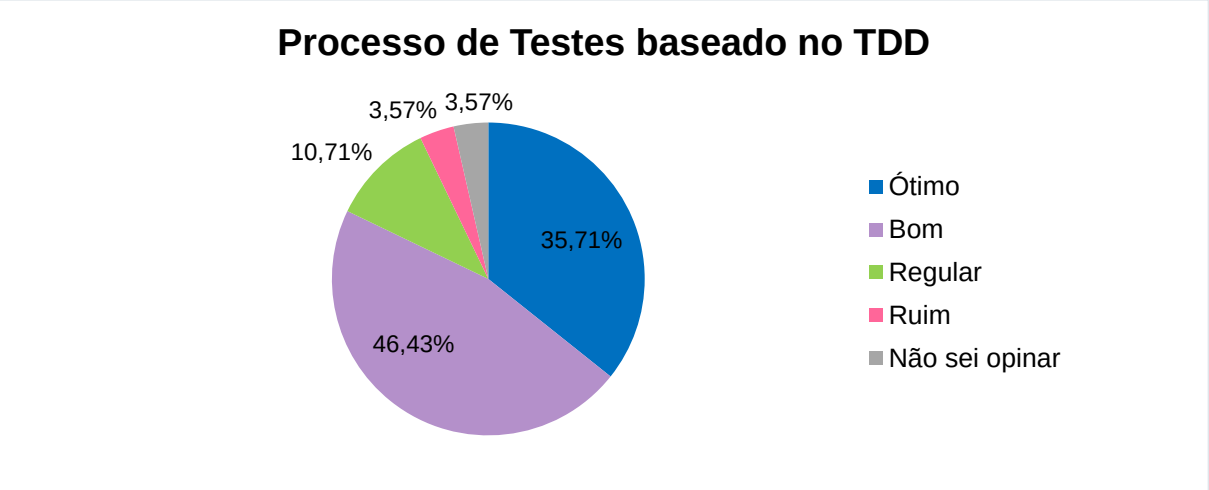

GRÁFICO 14 – Processo de Testes baseado no Desenvolvimento Orientado a Testes (TDD)
Fonte: Próprio autor

Para 82% dos entrevistados, o processo de testes baseado no TDD é bom ou ótimo e 11% acha que é regular, apenas 3% disseram que é ruim, e 4% não souberam opinar, esses dados mostram a boa credibilidade da metodologia de desenvolvimento orientado a testes, quando utilizada para ajudar o processo de desenvolvimento de softwares.

Quando perguntado numa escala de zero a cinco qual seria a aceitação do TDD no projeto por parte dos colaboradores, a maioria afirmou uma boa aderência, a escala foi convertida em gráfico para uma melhor apresentação dos resultados (GRAF. 15).

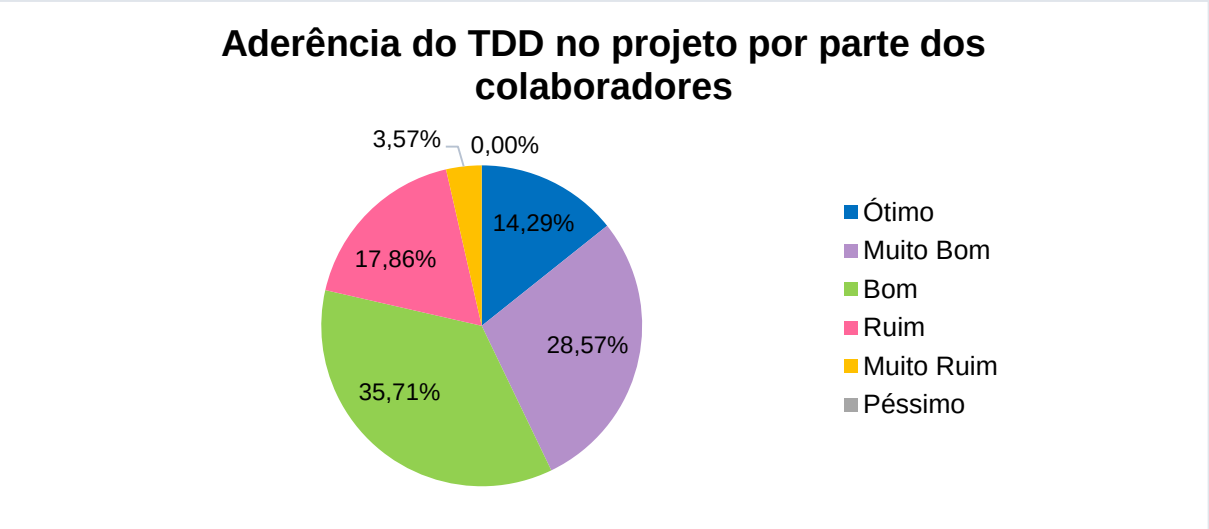

GRÁFICO 15 – Aderência do TDD no projeto por parte dos colaboradores
Fonte: Próprio autor

Aproximadamente 35,71% dos entrevistados acreditam que o nível de aderência por parte dos colaboradores é bom, 28,57% disseram ser muito bom e 14,29% afirmam ser ótimo. Não houveram avaliações para o nível péssimo, no entanto, 3,57% dos respondentes avaliaram como muito ruim, e 17,86% disseram que a aceitação do TDD por parte dos colaboradores é ruim.

Grande parte dos respondentes que avaliaram a metodologia TDD, cerca de 46,43%, afirmaram utilizar o desenvolvimento orientado a testes, e 53,57% não utilizam a metodologia. Muitas vezes os empregadores não incentivam o uso de metodologias e por isso os desenvolvedores acabam por não utilizar, mesmo conhecendo os benefícios para o processo de desenvolvimento.

3.1.5 Testes de Unidade

Nessa seção foi perguntado aos entrevistados qual o nível da capacidade de progresso e correção de falhas que os testes de unidade agregam ao processo de desenvolvimento, as respostas podem ser observadas no GRAF. 16.

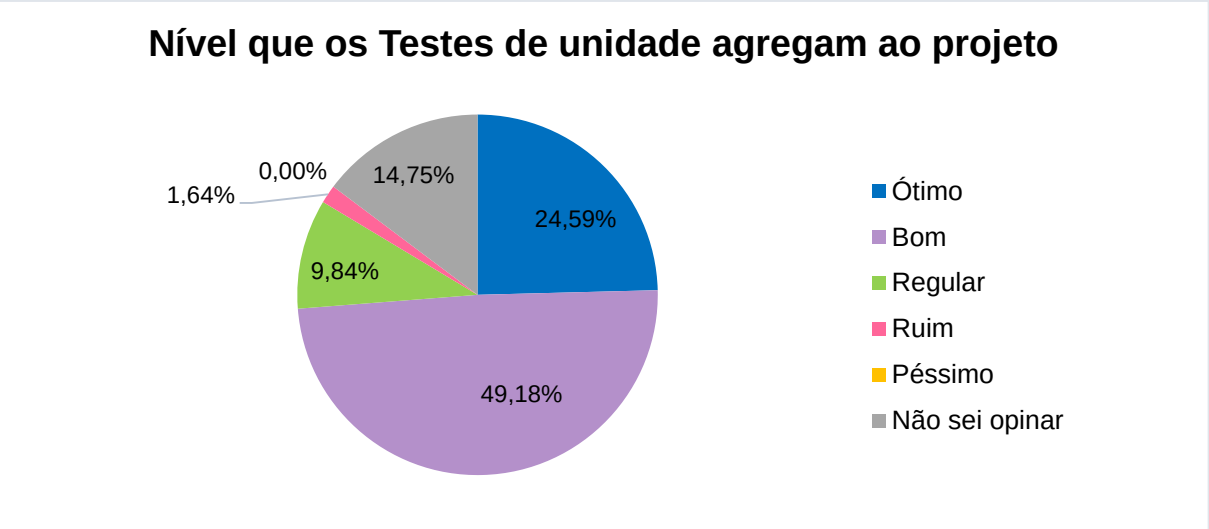

GRÁFICO 16 – Nível da capacidade de progresso e correção de falhas que os Testes de unidade agregam ao processo de desenvolvimento
Fonte: Próprio autor

Quase metade dos respondentes (49,18%) afirmaram que os testes de unidade acrescentam um nível bom à capacidade de progresso e correção de falhas do processo de desenvolvimento. Outros 24,59% consideram um nível ótimo, e 9.84% um nível regular, 1,64% responderam que é ruim. Não houveram avaliações para o nível péssimo e 14,75% não souberam opinar.

Foi feita também uma pergunta para saber se o entrevistado conhece o *framework* de testes de unidade PHPUnit, assim foi possível filtrar as pessoas que estariam aptas a responder as questões da seção sobre o *Framework* PHPUnit, 30 entrevistados afirmaram conhecer o *framework*. Os outros que responderam não conhecer o PHPUnit, foram encaminhados para a página de questões sobre *Zend Framework*.

3.1.6 Framework PHPUnit

As questões dessa seção tinham por objetivo descobrir se os entrevistados utilizam o *framework* PHPUnit, e saber qual é o nível de dificuldade e de eficácia que essa ferramenta apresenta, quando utilizada nos projetos.

Quando perguntados sobre como eles avaliam o *framework* de testes, 50% dos entrevistados disseram ser uma boa ferramenta, 26,67% disseram ser um ótimo *framework* e 6,67% disseram ser regular, os demais 16,67% não souberam opinar.

Foi perguntado ao entrevistado como ele classifica o nível de dificuldade em utilizar o PHPUnit (GRAF. 17).

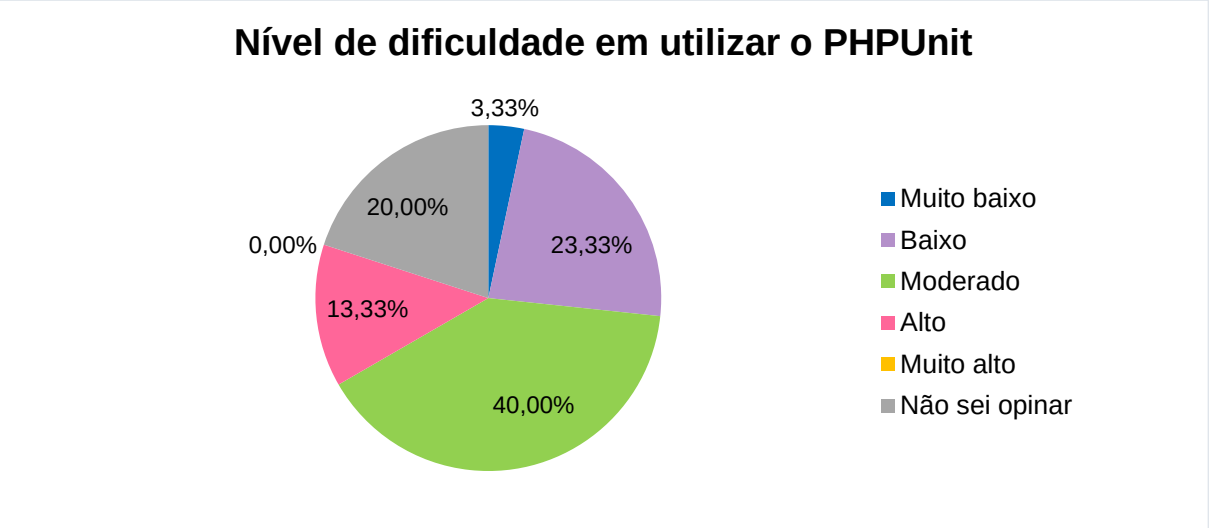

GRÁFICO 17 – Nível de dificuldade em utilizar o PHPUnit
Fonte: Próprio autor

Aproximadamente 13,33% responderam que o nível de dificuldade em utilizar o PHPUnit é alto, a maioria, 40% dos respondentes, disseram que o nível é moderado, 23,33% disseram que o nível é baixo e 3,33% disseram que é muito baixo. Não houveram avaliações para o nível de dificuldade muito alto, e 20% dos respondentes não souberam opinar. Isso quer dizer que, embora o *framework* possa parecer um pouco intimidante pela quantidade de configurações complicadas que possui, ele não possui um nível de dificuldade tão elevado.

Os entrevistados foram perguntados também sobre qual o nível de eficácia que o *framework* de testes PHPUnit apresenta no projeto (GRAF. 18).

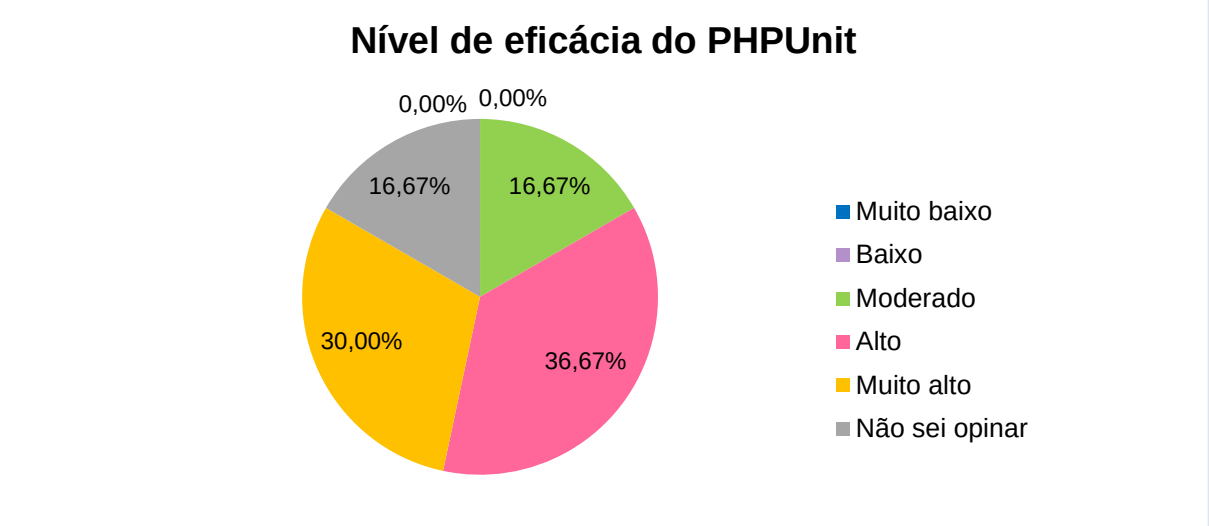

GRÁFICO 18 – Nível de eficácia que o PHPUnit apresenta no projeto
Fonte: Próprio autor

Para 16,67% dos entrevistados o rendimento do PHPUnit é moderado, 36,67% disseram que é alto, 30% disseram que é muito alto, e 16,67% não souberam opinar.

Metade dos entrevistados afirmaram que utilizam o *framework* de testes PHPUnit em algum projeto, a outra metade não utiliza.

3.1.7 Zend Framework

As questões dessa seção tinham por objetivo descobrir se os entrevistados conhecem e utilizam o *Zend Framework*, e como eles avaliam o nível de eficiência que o ZF apresenta no projeto.

Do total de entrevistados, 37 declararam ter conhecimento sobre o *framework* de desenvolvimento PHP em questão, e, portanto, foram selecionados para responder as demais questões sobre *Zend Framework*. Os demais que disseram não conhecer o *framework*, foram encaminhados para a próxima seção de questões sobre Processo de Testes.

Foi perguntado aos entrevistados como eles avaliam o *framework* de desenvolvimento PHP, *Zend Framework*, o GRAF. 19 mostra o resultado.

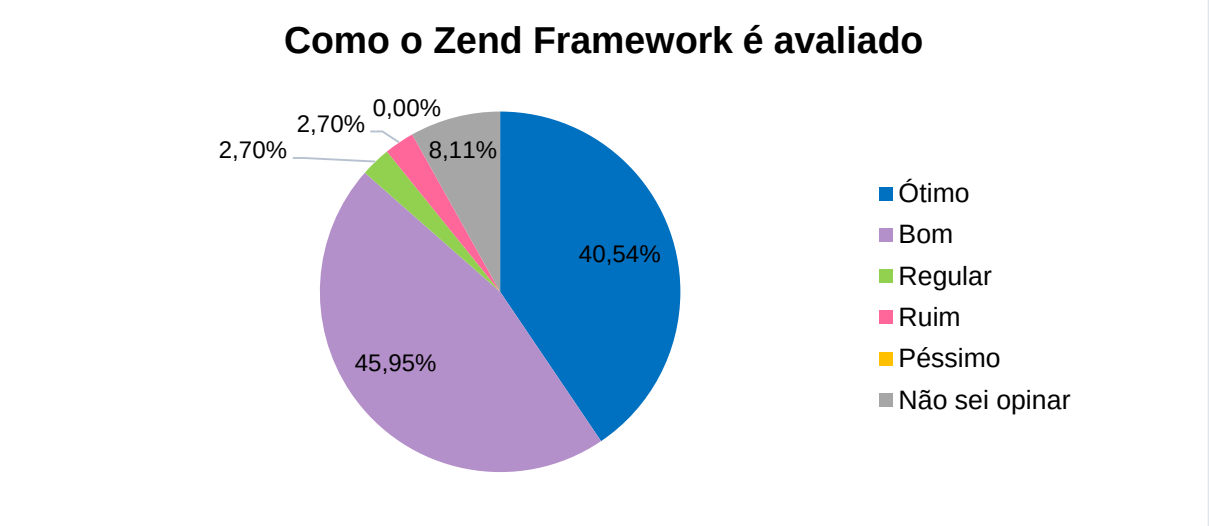

GRÁFICO 19 - Como o Zend Framework é avaliado
Fonte: Próprio autor

Aproximadamente 40,54% dos entrevistados qualificaram o ZF como ótimo, 45,95% avaliaram como bom, 2,7% avaliaram como regular e outros 2,7% ponderaram como ruim, os demais 8,11% não souberam opinar. Ou seja, o *framework* em questão foi muito bem avaliado.

Quando perguntado numa escala de zero a cinco qual seria a eficiência do ZF no projeto, a maioria afirmou um bom rendimento, a escala foi convertida em gráfico para uma melhor apresentação dos resultados (GRAF. 20).

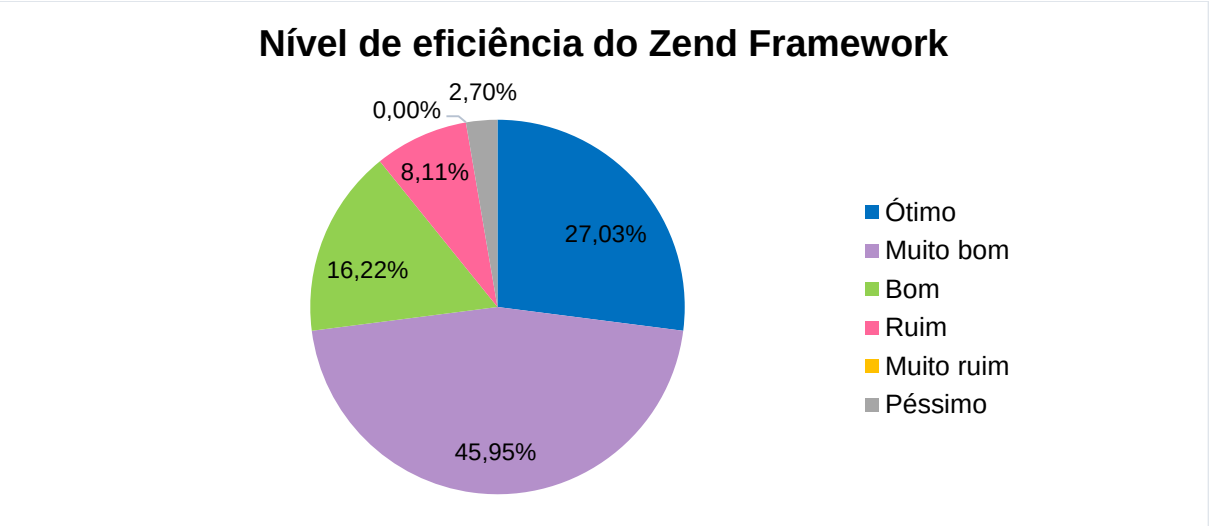

GRÁFICO 20 - Nível de eficiência que o Zend Framework apresenta no projeto
Fonte: Próprio autor

Para 27,03% dos respondentes o *Zend Framework* apresenta um ótimo nível de eficiência, 45,95% disseram ser um nível de eficiência muito bom, e outros 16,22% disseram ser um nível bom. Para 8,11% dos entrevistados a eficiência do *framework*

é ruim, e 2,7% avaliaram como péssimo, o nível de eficiência muito ruim não teve avaliações. Conclui-se, portanto, que o *Zend Framework* é um ótimo *framework* para o desenvolvimento de sistemas na linguagem PHP.

Dentre os entrevistados que avaliaram o *Zend Framework*, 56,76% afirmaram utilizar o *framework* para o desenvolvimento de softwares, os demais 43,24% declararam não utilizar o *framework*.

3.2 DISCUSSÃO DOS RESULTADOS

A partir da análise do questionário foi possível observar que a grande maioria dos entrevistados concordam com o fato de que os testes de software são necessários para a qualidade das aplicações, como disse Bartié (2002), "um processo de garantia da qualidade de software não é uma opção a ser estudada, mas parte de uma estratégia de sobrevivência".

Porém, segundo Pressman (2011), em um mundo orientado ao mercado não existem tempo e esforço necessários para produzir sistemas de alta qualidade. O processo de testes nos projetos ainda é algo recente, de acordo com a pesquisa os testes foram implantados nos projetos há mais ou menos três anos, e alguns profissionais ainda relutam em realizar os testes da maneira adequada, principalmente por considerar que eles ocupam muito tempo do processo de desenvolvimento, e ainda existe a falta de controle e gestão do processo, o que dificulta a execução de um processo de testes.

As principais dificuldades para a realização dos testes são as restrições de cronograma e a falta de profissionais qualificados e com conhecimento sobre processo de testes de software. Em sua grande maioria os testes ainda são realizados pelos próprios desenvolvedores do sistema durante a fase de codificação, mas algumas empresas possuem sua própria equipe de testes. No meio do processo de desenvolvimento é onde são encontrados o maior número de erros, estes relacionados à interpretação de requisitos e erros de código e funcionalidade.

Entretanto, os erros encontrados após a implantação dos testes diminuíram consideravelmente e a pesquisa mostra que os testes apresentam grande

contribuição para a qualidade dos projetos entregues. É importante frisar também que, no fim do projeto, os testes não saem tão caros quanto parecem, de acordo com a pesquisa em 77,05% dos casos, após implantar os testes os custos diminuíram ou não houve alterações.

A eliminação do trabalho repetitivo de inserção de dados e observação dos resultados e a diminuição do tempo gasto com a execução dos testes, foram as principais vantagens apontadas para o processo de testes automatizados, porém mais da metade dos entrevistados afirmaram que ainda realizam somente testes manuais.

Em relação ao processo de desenvolvimento orientado a testes (TDD), ao PHPUnit e ao *Zend Framework*, todos eles foram avaliados como muito bons para o desenvolvimento de softwares. De acordo com a pesquisa, o TDD tem uma boa aceitação por parte dos colaboradores, o PHPUnit apesar de ser moderadamente difícil de se utilizar, possui um nível de eficácia muito alto, e o *Zend Framework* também foi avaliado com um nível de eficiência muito bom.

Ainda de acordo com as respostas dos entrevistados, a utilização do *framework* de testes PHPUnit com o apoio da técnica de desenvolvimento orientado a testes é sim uma boa opção para desenvolvimento de sistemas com o *Zend Framework*, contribuindo assim para o desenvolvimento de softwares de alta qualidade, o que testifica que os testes de unidade são importantes para a grande maioria das situações, pois são "executados em intervalos regulares de validação e verificação sobre o correto funcionamento das unidades" (FERREIRA, 2011), aumentando a capacidade de progresso e correção de falhas não identificadas no processo de desenvolvimento de softwares.

CONCLUSÃO

Com a elaboração deste trabalho foi possível constatar que os profissionais de Tecnologia da Informação têm ciência da necessidade de um processo de testes de software, em função da complexidade dos sistemas produzidos e das exigências dos clientes que querem softwares de qualidade, confiáveis e eficientes.

Os testes de software têm grande contribuição na melhoria da qualidade dos sistemas, apresentando defeitos que serão corrigidos antes da entrega final ao cliente. Para cada sistema é importante elaborar um processo de testes com os tipos de testes que são mais adequados para o sistema em questão, porém, em muitos projetos ainda não é utilizado um processo bem definido, por questões como restrições de qualidade, custo, prazo e recursos disponíveis para o desenvolvimento do sistema.

Com este trabalho foi possível demonstrar que a utilização do *framework* de testes PHPUnit com o apoio da técnica de desenvolvimento orientado a testes é uma boa opção para o desenvolvimento de sistemas com o *Zend Framework*, e que os testes de unidade aumentam a capacidade de progresso e correção de falhas não identificadas no processo de desenvolvimento de softwares*,* concluindo assim, que o resultado do trabalho foi satisfatório, pois alcançou o objetivo proposto, sendo um trabalho de grande importância pois auxilia no conhecimento do processo de testes e na divulgação dos testes como grande facilitador do processo de desenvolvimento visando a qualidade dos softwares.

TRABALHOS FUTUROS

Como direção para possíveis trabalhos futuros tem se: um estudo de caso comparativo entre sistemas desenvolvidos com o *Zend Framework*, com e sem a aplicação da metodologia de desenvolvimento orientado a testes; a proposta de um projeto de testes com a utilização do *framework* de testes PHPUnit; e a aplicação de um questionário mais aprofundado no processo de testes, sobre todas as linguagens de programação, para descobrir mais sobre o custo/benefício dos testes no processo de desenvolvimento de softwares.

REFERÊNCIAS

BARTIÉ, Alexandre. **Garantia da Qualidade de Software**. Rio de Janeiro: Campus, 2002.

BERGMANN, Sebastian. **PHPUnit - The PHP Testing Framework**. 2015. Disponível em: <https://phpunit.de/>. Acesso em: 02 de novembro de 2015.

BERNARDO, Paulo Cheque; KON, Fabio. **A Importância dos Testes Automatizados**. Artigo publicado na Engenharia de Software Magazine, 2008.

BLANCO, Mariana Zanuzzio. **Documentação de teste baseado na Norma IEEE 829 – estudo de caso: "sistema de apoio a tomada de decisão"**. Departamento de Computação – Universidade Federal de São Carlos (UFSCar). São Carlos – SP – Brasil, 2012.

BONOTO, Rodrigo Guimarães; JUNIOR, Edson A. Oliveira. **O Padrão Model-View-Controller Apoiado pelo Framework Zend**. 2012.

CARDOSO, André. ANICHE, Mauricio. **Test-Driven Development: Teste e design no mundo real com PHP**. Casa do Código. 2015.

FERREIRA, Allan Rett. **Utilização da Metodologia TDD para desenvolvimento de software**. Trabalho de Curso Bacharelado em Ciência da Computação – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha". Marília, 2011.

HARTJES, Chris. **The Grumpy Programmer's PHPUnit Cookbook**. Leanpub: 2015.

LOPES, José Euclides da Silva. SISJOL – **Sistema de Controle Acadêmico para Escola Jovem On Line**. Curso de Sistemas de Informação - Faculdade de Balsas. Balsas – MA, 2013.

LUFT, Cristiane Carline. **Teste de software: uma necessidade das empresas**. UNIJUÍ – Universidade Regional do Noroeste do Estado do Rio Grande do Sul. DCEEng - Departamento de Ciências Exatas e Engenharias. Santa Rosa, 2012.

MINETTO, Elton Luís. **Frameworks para Desenvolvimento em PHP**. São Paulo: Novatec. 2007.

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**. 7. ed. Porto Alegre: AMGH Editora Ltda, 2011.

SEARLS, Justin. Prefácio. In: HARTJES, Chris. **The Grumpy Programmer's PHPUnit Cookbook**. Leanpub: 2015.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

TONIAZZO, José Carlos. **Extreme Programming: Uma Abordagem em Testes de Software Utilizando XUnit**. Curso de Pós-Graduação em Engenharia e Qualidade de Software. Universidade Comunitária Regional de Chapecó. Centro Tecnológico. Chapecó - SC, 2007.

ZEND FRAMEWORK. **Unit Testing**. Disponível em: <http://framework.zend.com/manual/2.0/en/user-guide/unit-testing.html>. Acesso em 02 de novembro de 2015.

APÊNDICE 1 - QUESTIONÁRIO

Testes de software

O presente questionário é elaborado no âmbito de um trabalho de conclusão de curso de Ciência da Computação, o qual pretende analisar a utilização dos testes de software para a qualidade das aplicações desenvolvidas.

Os dados recolhidos são confidenciais e serão utilizados apenas para fins acadêmicos. O tempo estimado para o preenchimento deste questionário é de aproximadamente 5 a 15 minutos.

A sua colaboração é muito importante para minha caminhada ao conhecimento. Será relevante para este estudo que responda a todas as perguntas com seriedade e honestidade.

Muito obrigada por participar!

*Obrigatório

Perfil do respondente

As questões seguintes têm por objetivo conhecer um pouco sobre seu perfil.

1. Email: *

2. Cidade onde mora: *

3. Estado onde mora: *

4. Idade: *

- ○ De 15 a 20
- ○ De 21 a 25
- ○ De 26 a 30
- ○ De 31 a 35
- ○ De 36 a 40
- ○ De 41 a 45
- ○ De 46 a 50
- ○ De 51 a 55
- ○ De 56 a 60
- ○ Acima de 60

5. Gênero: *

- ○ Feminino
- ○ Masculino

6. Qual a sua Formação Acadêmica? *

- ○ Técnico
- ○ Graduação
- ○ Pós - Graduação
- ○ Mestrado
- ○ Doutorado
- ○ Outro: [ ]

7. Qual é o cargo que você ocupa? *

- ○ Analista de Teste

- ○ Desenvolvedor
- ○ Analista de Sistemas
- ○ Gerente de Projetos
- ○ Outro: [ ]

8. Qual a sua clientela? *

Para quem você trabalha, quem contrata seus serviços.

- ○ Empresa local
- ○ Empresa regional
- ○ Empresa de outros estados
- ○ Empresa de outros países
- ○ Outro: [ ]

Processo de Testes

9. A qualidade dos softwares interfere diretamente no sucesso das aplicações. Como você avalia a necessidade de utilização dos Testes de Software visando essa qualidade? *

- ○ **Muito baixa**
- ○ Baixa
- ○ Moderada
- ○ Alta
- ○ Muito alta
- ○ Não sei opinar

10. Qual motivo te levaria a não seguir o Processo de Testes de Software? *

- ○ Falta de treinamento
- ○ Falta de controle e gestão do processo
- ○ Falta de tempo
- ○ Falta de conhecimento
- ○ Falta de confiança no processo
- ○ Não sei opinar
- ○ Outro: [ ]

11. Você tem conhecimento sobre Testes de Software? *

Já ouviu falar sobre ou trabalhou com testes e qualidade de software.

- ○ Sim
- ○ Não (Ir para a pergunta 47)

Processo de Testes

12. Qual o seu tempo de experiência sobre Processo de Testes? *

- ○ De 0 a 12 meses
- ○ De 1 a 3 anos
- ○ De 4 a 6 anos
- ○ De 7 a 10 anos
- ○ Mais de 10 anos

13. Há quanto tempo o Processo de Testes foi implantado no projeto? *

- ○ Menos de 1 ano
- ○ De 1 a 3 anos
- ○ De 4 a 6 anos

- ○ De 7 a 10 anos
- ○ Mais de 10 anos
- ○ Não faço Testes de Software

14. Como você avalia o Processo de Testes utilizado? *

- ○ Ótimo
- ○ Bom
- ○ Regular
- ○ Ruim
- ○ Péssimo
- ○ Não sei opinar

Processo de Testes

As questões a seguir se referem à maneira que os testes são realizados.

15. Quais testes de software são feitos? *

É possível assinalar mais de uma opção.

- ☐ Teste funcional (caixa preta)
- ☐ Teste estrutural (caixa branca)
- ☐ Teste de unidade
- ☐ Teste de integração
- ☐ Teste de sistema
- ☐ Teste de aceitação
- ☐ Teste de regressão
- ☐ Teste de carga
- ☐ Teste de configuração
- ☐ Teste de desempenho/performance
- ☐ Teste de instalação

☐ Teste de recuperação

☐ Teste de segurança

☐ Teste de usabilidade

☐ Teste de volume

☐ Teste paralelo

☐ Outro:

16. Caso utilize teste de caixa preta marque os critérios utilizados: *

É possível assinalar mais de uma opção.

☐ Particionamento de equivalência

☐ Análise de valor limite

☐ Grafo de causa e efeito

☐ Desconheço os critérios indicados

☐ Outro:

17. Caso utilize teste de caixa branca marque os critérios utilizados: *

É possível assinalar mais de uma opção.

☐ Critérios baseados em fluxo de controle

☐ Critérios baseados em fluxo de dados

☐ Critérios baseados na complexidade

☐ Desconheço os critérios indicados

☐ Outro:

18. Quem são os responsáveis por realizar os testes de software no projeto que você trabalha? *

É possível assinalar mais de uma opção.

☐ Equipe terceirizada especializada em testes

☐ Equipe de testes da empresa

☐ Os próprios desenvolvedores do sistema

☐ Os clientes

☐ Os usuários

☐ Outro:

19. Como os testes de software são realizados no seu projeto? *

○ Manualmente (Após a pergunta 20, ir para a pergunta 24)

○ Através da automatização

○ Ambos

20. Quando é iniciada a execução dos testes? *

○ Na especificação de requisitos

○ Na elaboração do projeto de software

○ Na fase de codificação

○ Na fase de implementação do software

○ Outro:

Processo de Testes Automatizados

21. Como você avalia o processo de Testes Automatizados? *

○ Ótimo

○ Bom

○ Regular

○ Ruim

○ Péssimo

○ Não sei opinar

22. Na sua opinião, quais as vantagens dos testes automatizados em relação aos testes feitos de forma manual? *

É possível assinalar mais de uma opção.

- [ ] Eliminação do trabalho repetitivo de inserção de dados e observação dos resultados
- [ ] Melhoria da qualidade do processo e produto de software
- [ ] Maior confiança quanto à liberação do produto, a cada nova versão
- [ ] Reusabilidade dos scripts de testes, com grande facilidade para alteração de informações
- [ ] Diminuição do tempo gasto com a execução dos testes
- [ ] Aumento da produtividade e diminuição do custo destinado às atividades de teste
- [ ] Aumento da cobertura (abrangência) dos testes, através do número crescente de implementações de casos de testes
- [ ] Outro:

23. Quais ferramentas são utilizadas nos testes automatizados? *

Processo de Testes

As questões seguintes se referem aos efeitos que os testes de software apontam.

24. Onde está centrada a maior incidência dos erros? *

- ○ Nas fases iniciais do processo de desenvolvimento
- ○ No meio do processo de desenvolvimento
- ○ Nas fases finais do processo de desenvolvimento
- ○ Outro:

25. Quais os erros mais comuns de serem encontrados no sistema desenvolvido?

É possível assinalar mais de uma opção.

- ☐ Erros de interpretação de análise dos requisitos
- ☐ Erros de código
- ☐ Erros de funcionalidade
- ☐ Erros de digitação
- ☐ Erros de interface
- ☐ Outro: [ ]

26. Depois da implantação da atividade de teste de software, os erros encontrados após a entrega do produto ao cliente: *

- ○ Diminuíram
- ○ Aumentaram
- ○ Não teve alteração

27. Quais as dificuldades encontradas na realização dos testes de software?*

É possível assinalar mais de uma opção.

- ☐ O teste é um processo caro
- ☐ A atividade de teste é limitada por restrições de cronograma
- ☐ Há a falta de profissionais especializados na área de teste
- ☐ Existem dificuldades em implantar um processo de testes
- ☐ Há o desconhecimento de um procedimento de teste adequado
- ☐ Há o desconhecimento de técnicas de teste adequadas
- ☐ Há o desconhecimento sobre como planejar a atividade de teste
- ☐ Outro: [ ]

28. Qual percentual de tempo do projeto de desenvolvimento do software é consumido com a atividade de testes? *

- Até 5%
- De 6% a 10%
- De 11% a 20%
- De 21% a 30%
- De 31% a 40%
- De 41% a 50%
- Acima de 50%
- Não sei opinar

29. Qual o valor efetivo dos testes de software realizados no projeto? *

- Apresentam pequena contribuição
- Apresentam média contribuição
- Apresentam grande contribuição

30. Com relação aos custos, após a implantação da atividade de teste observou-se que: *

- Diminuíram
- Aumentaram
- Não teve alteração

Desenvolvimento Orientado a Testes

31. Você conhece a metodologia de Desenvolvimento Orientado a Testes (TDD)? *

- Sim

○ Não (Ir para a pergunta 35)

Desenvolvimento Orientado a Testes

32. Como você avalia o Processo de Testes baseado no Desenvolvimento Orientado a Testes (TDD)? *

○ Ótimo

○ Bom

○ Regular

○ Ruim

○ Péssimo

○ Não sei opinar

33. Numa escala de 0 (zero) a 5, quanto você considera que o TDD tem/teria uma aderência positiva no projeto por parte dos colaboradores? *

	0	1	2	3	4	5	
Péssimo	○	○	○	○	○	○	Ótimo

34. Você utiliza a metodologia de Desenvolvimento Orientado a Testes (TDD)?*

○ Sim

○ Não

Testes de Unidade

35. Qual o nível da capacidade de progresso e correção de falhas que os Testes de unidade agregam ao processo de desenvolvimento? *

○ Ótimo

○ Bom

○ Regular

○ Ruim

- Péssimo
- Não sei opinar

36. Você conhece o framework de testes PHPUnit? *

- Sim
- Não (Após a pergunta 37, ir para a pergunta 42)

37. Você utiliza alguma outra ferramenta para realizar os Testes de unidade? Se sim, qual (is)?

Framework PHPUnit

38. Como você avalia o framework de testes PHPUnit? *

- Ótimo
- Bom
- Regular
- Ruim
- Péssimo
- Não sei opinar

39. Como você classifica o nível de dificuldade em utilizar o PHPUnit? *

- Muito baixo
- Baixo
- Moderado
- Alto
- Muito Alto
- Não sei opinar

40. Qual o nível de eficácia que o framework de testes PHPUnit apresenta no projeto? *

- Muito baixo
- Baixo
- Moderado
- Alto
- Muito Alto
- Não sei opinar

41. Você utiliza o framework de testes PHPUnit em algum projeto? *

- Sim
- Não

Zend Framework

42. Você conhece o framework de desenvolvimento Zend Framework? *

- Sim
- Não (Ir para a pergunta 46)

43. Como você avalia o framework de desenvolvimento Zend Framework? *

- Ótimo
- Bom
- Regular
- Ruim
- Péssimo
- Não sei opinar

44. Numa escala de 0 (zero) a 5, quanto é o nível de eficiência que o Zend Framework apresenta no projeto? *

	0	1	2	3	4	5	
Péssimo	○	○	○	○	○	○	Ótimo

45. Você utiliza o Zend Framework em algum projeto? *

- ○ Sim
- ○ Não

Processo de Testes

46. Como é desenvolver sistemas com o Zend Framework utilizando o framework de testes PHPUnit com o apoio da técnica de Desenvolvimento Orientado a Testes (TDD)? *

- ○ Ótimo
- ○ Bom
- ○ Regular
- ○ Ruim
- ○ Péssimo
- ○ Não sei opinar
- ○ Outro: [ ]

Pare de preencher este formulário.

Processo de Testes

Considere a importância da qualidade dos softwares para o sucesso das aplicações.

47. Por que você não utiliza Testes de Software? *

[ ]