TESTES DE SOFTWARE COMO GARANTIA DE QUALIDADE E EFICIÊNCIA: ESTUDO DE CASO COM USO DO SELENIUM

ESTUDO DE CASO

BLASCKE, Francis Fumagalli Megda[1], FARINA, Renata Mirella[2], FLORIAN, Fabiana[3]

BLASCKE, Francis Fumagalli Megda. FARINA, Renata Mirella. FLORIAN, Fabiana. **Testes de software como garantia de qualidade e eficiência: estudo de caso com uso do selenium**. Revista Científica Multidisciplinar Núcleo do Conhecimento. Ano. 08, Ed. 09, Vol. 03, pp. 26-55. Setembro de 2023. ISSN: 2448-0959, Link de acesso: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software, DOI: 10.32749/nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

RESUMO

Muitos produtos e bens de serviço, antes de serem lançados no mercado, passam por infinitos testes, como forma de assegurar a qualidade deles. Com os softwares não é diferente, existem vários tipos de abordagens de processos, os quais os testes são realizados, de acordo com a realidade da empresa, funcionários e a necessidade do produto. Dessa forma cada modelo tende a se adequar ao negócio ao qual está destinado, ficando a cargo dos envolvidos verificar a necessidade de mudanças ou não, de acordo com seus planejamentos, projetos e necessidades. O objetivo deste trabalho é apresentar a importância da qualidade de softwares, destacando alguns modelos e sua eficiência. Foi realizada pesquisa bibliográfica e foi realizado um estudo de caso em que foi apresentado os desafios na qualidade dos testes automatizados de interface de usuário usando Selenium. Foram implementadas soluções como tempo de espera explícito, aumento da cobertura de testes e paralelização de testes, o que resultou em melhorias na estabilidade, qualidade e velocidade dos testes. Isso permitiu a entrega de um software de melhor qualidade em menos tempo.

Palavras-chave: Testes de Software, Eficiência de Software, Selenium.

1. INTRODUÇÃO

A evolução tecnológica tem atingido todos os setores da sociedade. A informatização tem sido um aspecto de extrema necessidade para as organizações,

fator importante para tornar o trabalho mais ágil e mais seguro. Pressman (2011), frente a necessidade de informatizar seus negócios, as empresas constantemente têm buscado adquirir softwares adequados para as suas necessidades e assim como para todos os produtos e serviços, os clientes têm buscado a qualidade como fator primário.

Desikan (2009), dentro deste contexto a garantia da qualidade dos softwares produzidos tem sido motivo de grande preocupação e para que essa qualidade seja realmente mantida, os softwares passam por testes antes de chegar ao mercado. São esses testes que mostram a capacidade de desempenho dos softwares e qual a adequação de cada um deles. Como, por exemplo, o teste de caixa-branca que é desenvolvido a partir da análise do código fonte do programa cobrindo todas as possibilidades, tanto corretas, quanto possíveis erros.

Melo (2015) esclarece que dentro desta categoria de teste de software o desenvolvedor pode construir códigos para efetuar a ligação de bibliotecas e componentes, uma vez que tem acesso ao código.

O objetivo deste trabalho é apresentar a importância da qualidade de softwares, destacando alguns modelos e sua eficiência.

As empresas têm adquirido cada vez mais softwares que se adequam a sua proposta de trabalho, e buscam como fator inicial a qualidade, pois é ela quem vai garantir a qualidade do produto ou serviço oferecido ao cliente.

Sommerville (2011) a Engenharia de Software surgiu com o objetivo de amenizar os problemas oriundos na concepção do produto de software, vividos na década de 70 a qual ficou conhecida como “crise do software”.

Sommerville (2011) esclarece que essa crise surgiu devido ao aumento da demanda por sistemas e a pela complexidade dos mesmos que acabava gerando produtos de baixa qualidade, com prazos e custos muito maiores do que o pensado inicialmente. Como os produtos muitas vezes não atendiam às expectativas dos clientes, esses

acabavam sendo levados para correção ou mesmo troca o resultava em elevados custos.

Garvin (2002) o conceito de qualidade foi associado ao fato de estar em conformidade com as especificações de um padrão esperado seja na construção de um objeto ou na prestação de um serviço. Com o passar do tempo, o conceito de qualidade passou a ser associado ao fator satisfação do cliente.

Foi realizada pesquisa bibliográfica e o desenvolvimento de testes para um site. Foram desenvolvidos testes em Selenium para verificar a estabilidade do site, alguns exemplos de casos de testes e uma função que demonstra a paralelização dos testes para uma maior agilidade, reduzindo assim, o tempo dos testes.

2 REVISÃO BIBLIOGRÁFICA

Antes de conceituar a engenharia de software, é importante entender o que se define como software e quais os tipos dele.

Sommerville (2003) o software é o conjunto de vários artefatos e não apenas o código fonte. Braga (2011) o software é muito mais do que programas-fonte e executáveis, o mesmo compreende a documentação associada a ele, dados e configuração.

Pressman (2011), o software apresenta esta característica especial em relação a outros tipos de produtos, ou seja: ele não é fabricado no sentido clássico, mas desenvolvido, passando por um processo de engenharia.

Pressman (2011) os softwares estão categorizados através dos seguintes tipos, os quais são:

- **Software de sistema.**São programas que apoiam outros programas, como o software que realiza a comunicação com o hardware (sistema operacional) e software que ajuda na construção de outro software (compiladores).
- **Software de aplicação.**São programas que são desenvolvidos para executar no negócio de uma empresa determinada.

- **Software científico e de engenharia.**São algoritmos que processam números.
- **Software embutido.**São programas construídos para executarem dentro de um produto específico como as teclas digitais de um forno micro-ondas.
- **Software para linhas de produtos.**São os softwares conhecidos como software de prateleiras.
- **Software de web.**São aplicativos que são executados via Internet.
- **Software de inteligência artificial.**São softwares que fazem os usos de algoritmos não numéricos. Estes tipos software se encaixam na robótica.
- **Computação ubíqua.** São softwares que realiza a verdadeira computação distribuída.
- **Software aberto.** São software que disponibiliza a visualização do código fonte da aplicação para o engenheiro de software modifica da maneira que deseja.

Visto alguns tipos de software e o conceito do mesmo é importante entender sobre a engenharia de software e o trabalho realizado por esse departamento.

Sommerville (2011) a Engenharia de Software surgiu com o objetivo de amenizar os problemas oriundos na concepção do produto de software, na década de 70 a qual ficou conhecida como “crise do software”.

O autor esclarece que essa crise surge devido ao aumento da demanda por sistemas, e a complexidade dos mesmos acabava pela fabricação de produtos de baixa qualidade, prazos e custos muito maiores do que o pensado inicialmente. Como os produtos muitas vezes não atendiam às expectativas dos clientes acabavam sendo levados para concertos ou mesmo troca. A correção desses produtos resultava em elevados custos.

Braga (2011) a Engenharia de Software pode ser entendida como um conjunto de processos, métodos, técnicas e ferramentas, que ajudam a produzir o software com maior qualidade e menor custo.

Engholm Júnior (2010) o trabalho da Engenharia de Software se concentra nos aspectos práticos das atividades de elaboração, construção e manutenção do produto de software em todo o seu ciclo de desenvolvimento.

Paula Filho (2003) a Engenharia de Software foca no software como produto e descarta dentro deste contexto os softwares construídos apenas para o passatempo dos programadores.

Visto o que é software e o trabalho da Engenharia de Software é importante entender como se dá o processo de software.

2.1 PROCESSO DE SOFTWARE

Matté (2011) o produto gerado através da Engenharia de Software, é resultado da somatória de diversas atividades executadas uma após as outras, como o objetivo da produção final. Tais atividades realizadas de forma coordenada sob uma metodologia é conhecida como processo de software.

Larman (2000, p. 40) sobre o processo de software define: "um processo [sistematizado] de desenvolvimento de software é um método para organizar as atividades relacionadas com a criação, entrega e manutenção de sistemas de software."

Sommerville (2011) dentre os processos de produção de software os processos técnicos não são as únicas preocupações da Engenharia de Software, mas também as atividades que complementam o gerenciamento de projeto, desenvolvimento de ferramentas, métodos e teorias que apoiam a produção do software.

Sobre o processo de software Jalote (2005, p. 566):

> É um conjunto de atividades, ligadas por padrões de relacionamento entre ela, pelas quais se as atividades operarem corretamente e de acordo com os padrões requeridos, o resultado desejado é produzido. O resultado desejado é um software de alta qualidade e baixo custo. Obviamente, um processo que não aumenta a produção (não suporta projetos de

> software grandes) ou não pode produzir software com boa qualidade não é um processo adequado (JALOTE 2005, p. 566).

Pressman (2011, p. 53) “o processo de software é composto de várias atividades metodológicas, e em cada atividade, existem ações específicas compostas por um conjunto de tarefas, artefatos, fatores de garantia de qualidade e pontos de controle do projeto”.

Paulk *et al*., (1993, p.3) define sobre o processo de software:

> Um processo de software pode ser definido como um conjunto de atividades, métodos, práticas e transformações que as pessoas utilizam para desenvolver e manter software e seus produtos associados (por exemplo, planos de projeto, documentos de projeto do sistema, código, casos de teste e manuais do usuário) (PAULK *et al*., 1993, p.3).

Sommerville (2011) existem alguns modelos de processo de software. A seguir são apresentados os modelos: Sequencial Linear (também denominado de modelo cascata - Figura 1); modelo de prototipação (Figura 2); o modelo RAD (*Rapid Application Development* – Figura 3) e modelo espiral (Figura 4).

2.1.1 MODELO CASCATA

Leite (2000) o Modelo de Cascata o qual tornou-se conhecido na década de 70 e por esse modelo as atividades do processo de desenvolvimento são estruturadas no formato de uma cascata, onde a saída de uma é a entrada para a próxima.

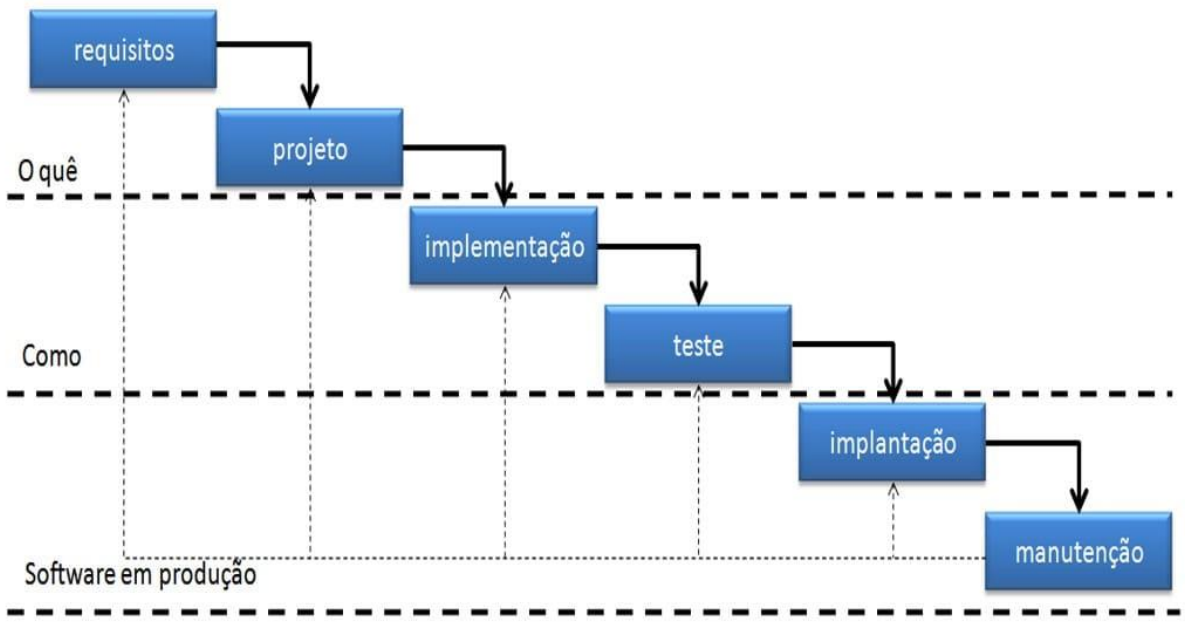
Vasconcelos (2006) o modelo cascata segue vários passos ordenados, e ao final de cada fase, a equipe responsável finaliza uma revisão.

Sommerville (2006) o desenvolvimento desse processo continua até que o cliente fique satisfeito com os resultados.

De acordo com o autor, as principais atividades do modelo cascata são: Análise dos requisitos; projeto; codificação; testes e manutenção.

- Análise de Requisitos - Etapa de comunicação entre usuários e analistas de sistemas, nessa etapa ocorrem reuniões para definir o que deve ser feito, para que se compreenda o objetivo do projeto, suas características e necessidades, além da funcionalidade do sistema (PRESSMAN (2011);
- Projeto - tem como finalidade representar os requisitos por meio de um conjunto de representações gráficas ou descritas (PRESSMAN (2011);
- Codificação: etapa em que o projeto é implementado, ou seja, o código fonte é gerado através de uma linguagem de programação e colocado em operação nos computadores (PRESSMAN (2011);
- Testes- após a etapa de codificação é necessário testar o software ou o sistema, esse procedimento tem como finalidade de verificar os possíveis erros no sistema (SOMMERVILLE, 2003);
- Manutenção: nessa etapa de manutenção ocorrem as correções dos erros e/ou anomalias observados no sistema (SOMMERVILLE, 2003).

Figura 1: Modelo Cascata



Fonte: FABRI (2012).

Leite (2000) existem muitas variantes deste modelo, as quais são propostas por diferentes pesquisadores ou empresas de desenvolvimento. A característica do

modelo cascata é sempre um fluxo linear e sequencial das atividades semelhantes ao descrito anteriormente.

2.1.2 MODELO DE PROTOTIPAÇÃO

Rocha, Isotani e Toda (2014) esse modelo é baseado no desenvolvimento de um protótipo com base no conhecimento dos requisitos iniciais para que seja elaborado o sistema. O desenvolvimento deste modelo é feito através de uma sequência obedecida de diferentes etapas de análise dos requisitos, o projeto, a codificação e os testes.

Os autores esclarecem que é uma tarefa difícil definir inicialmente todos os requisitos necessários ao sistema. Não é possível prever de que forma o sistema irá afetar o funcionamento das práticas de trabalho e como acontecerá a interação com os outros sistemas.

Dessa forma para testar os requisitos de uma forma mais eficiente seria necessária a utilização de um protótipo de sistema.

De acordo com o site Engenharia de Software (2012) um protótipo é uma versão inicial de um sistema de software, o qual é utilizado para mostrar conceitos, experimentar ações de projeto e de forma geral conhecer com mais amplitude os problemas e soluções.

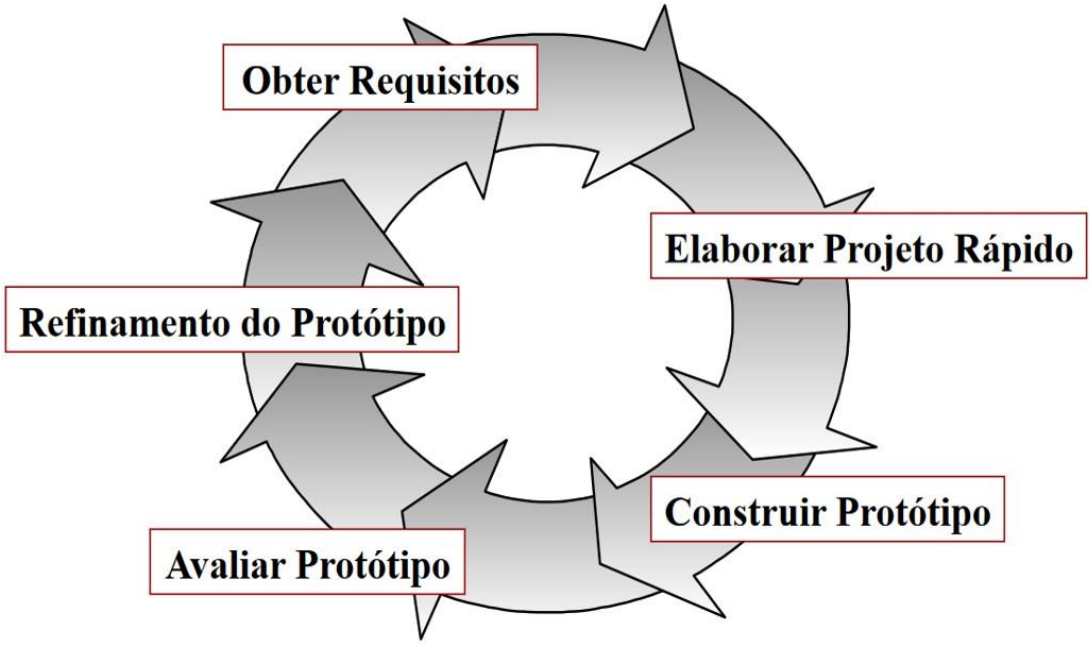
O desenvolvimento rápido de um protótipo é essencial e necessário para controlar os custos e realizar as experiências com o protótipo no início do processo de software.

As vantagens da prototipação são caracterizadas pelo fato do modelo ser interessante; possibilidade de demonstrar realizabilidade; possibilidade de obtenção da versão, mesmo que de forma básica, mas com a capacidade de entendimento do que será o sistema e sua necessidade de investimento e a experiência adquirida na elaboração da prototipação.

RC: 148546
Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

Já como desvantagem está a necessidade de reconstrução do produto, o que pode muitas vezes desagradar o cliente.

Figura 2: Modelo de Prototipação



Fonte: Profa. Dra. Elisa Yumi Nakagawa (2017).

2.1.3 MODELO *RAPID APPLICATION DEVELOPMENT* - RAD

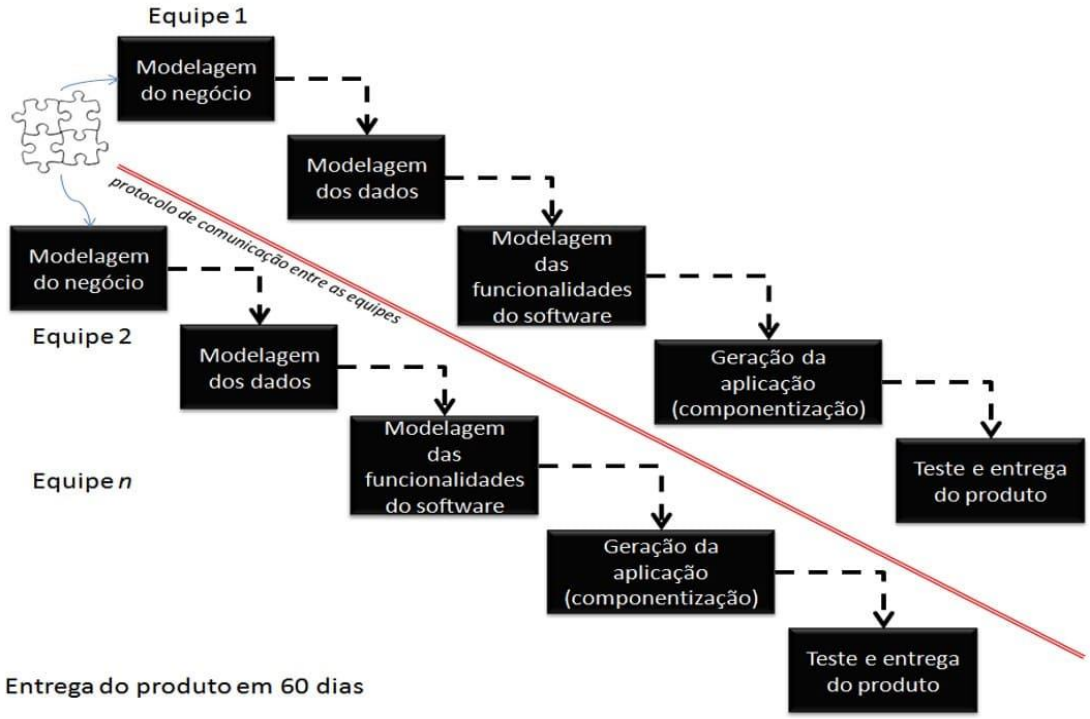
Fabri (2012) esse modelo é caracterizado como um processo sequencial e linear o qual enfoca o ciclo de um desenvolvimento de projeto curto. Nesse projeto os requisitos devem e precisam estar estabilizados e esse projeto é composto de subprojetos, os quais são direcionados para equipes de desenvolvimento, as quais tem a finalidade de agilizar a construção do produto.

O autor esclarece que para esse modelo é necessário a aplicação de recursos humanos suficientes para todas as equipes e tanto os clientes como os desenvolvedores devem estar comprometidos, pois, o projeto deve ser construído a curto prazo.

Rocha, Isotani e Toda (2014) explicam que nem todos os tipos de aplicação são apropriados para o RAD, assim esclarecem que para esse tipo de modelo deve ser possível a modularização efetiva, o alto desempenho deve ser uma característica.

Os autores citam também as desvantagens como o grande trabalho em equipe de forma rápida, o que pode não funcionar corretamente e assim o projeto pode não ser concluído dentro do prazo.

Figura 3: Modelo RAD



Fonte: FABRI (2013).

2.1.4 MODELO ESPIRAL

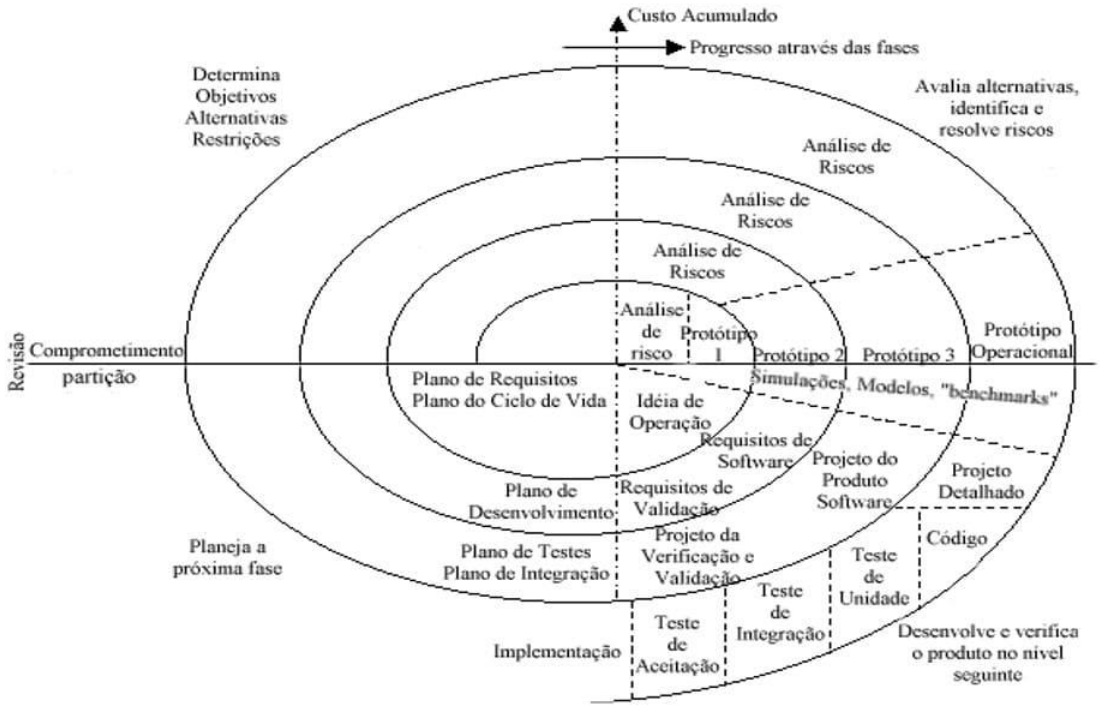
Rezende (2005) esse processo permite que haja muitas interações no processo de desenvolvimento. Cada interação implica em uma volta no modelo espiral. Dentro desse modelo os resultados podem ser obtidos a curto prazo. Leite (2000) esclarece que o objetivo do modelo espiral é promover um metamodelo capaz de acomodar diversos processos específicos. O modelo espiral segundo o autor

prevê prototipação, desenvolvimento evolutivo e cíclico, e as principais atividades do modelo cascata.

Leite (2000) o modelo espiral descreve um fluxo de atividades cíclico e evolutivo, o qual é constituído de 4 estágios. No primeiro estágio devem ficar determinados os objetivos, soluções alternativas e restrições. Já o segundo estágio incumbe-se da análise dos riscos das decisões tomadas no primeiro estágio.

O estágio 3 é a fase de desenvolvimento, a qual inclui design, especificação, codificação e verificação. A principal característica deste estágio é que as especificações surgem a cada ciclo. Já no quarto estágio acontece a revisão das etapas anteriores e o planejamento da fase posterior.

Figura 4: Modelo Espiral



Fonte: FABRI (2012).

2.2 COMPARAÇÃO ENTRE O MODELO ÁGIL E O MODELO CASCATA

CronApp (2018) os modelos de software foram criados com o objetivo da adequação específica a necessidade de cada usuário. As metodologias de desenvolvimento ágil

foram criadas frente a necessidade de flexibilizar as rotinas de criação dos softwares e dar mais agilidade para se adequarem a qualquer tipo de mudança.

De forma geral os modelos de desenvolvimento ágil são mais maleáveis do que os tradicionais e tem as bases para que as equipes entreguem o projeto rapidamente. Diferentemente dos modelos tradicionais, não há a necessidade de apresentar todos os recursos para que uma alteração seja feita.

Rocha, Isotani e Toda (2014) esclarecem que a priorização das etapas do desenvolvimento ágil é mais simples e as mudanças podem ser realizadas facilmente, além disso a priorização das etapas é algo mais simples, e em caso de necessidade da criação de uma determinada função basta utilizar um time pequeno para que a tarefa seja executada.

CronApp (2018) explica que as metodologias ágeis funcionam a partir de quatro princípios: foco no indivíduo; funcionalidade de software priorizada; colaboração do cliente e elevada capacidade de resposta a mudanças.

As vantagens dos ciclos de desenvolvimento mais curtos são as maiores vantagens da metodologia ágil, uma vez que permite maior contato com o cliente e a capacidade de mudanças sempre que necessário.

Em um intervalo menor, resultados são apresentados e, a partir do feedback obtido, correções são feitas. Como consequência, as chances de o negócio conseguir sucesso com a iniciativa são muito maiores (CRONAPP, 2018).

Em contrapartida, o modelo cascata é mais antigo que o modelo de desenvolvimento ágil. Esse modelo segundo CronApp (2018) envolve a criação de ferramentas a partir de etapas bem definidas como: análise, projeto, teste, codificação e manutenção.

Esse modelo apresenta uma estrutura com pouca mobilidade e poucas aberturas para mudanças no meio do caminho.

Rocha, Isotani e Toda (2014) explica que o modelo cascata está perdendo lugar para os modelos de desenvolvimento ágil, fator decorrente da flexibilidade e agilidade, além do que no modelo tradicional cascata a comunicação com o cliente é menor, uma vez que acontece ao final de cada etapa, ou seja, o cliente só vê o resultado final.

Segundo o CronApp (2018) o modelo cascata exige que a empresa faça uma análise completa dos requisitos antes de cada etapa, como forma de evitar riscos e garantir assim a entrega de um produto de acordo com as necessidades do cliente. Em relação ao modelo cascata as principais vantagens são: etapas bem planejadas e definidas; maior foco nos processos de planejamento e etapas que se iniciam apenas quando o cliente está de acordo com o que foi feito na etapa anterior.

Como desvantagem a CronApp (2018) esclarece que esse modelo tem apresentado alguns problemas quando comparado aos modelos ágeis. O principal impasse está na necessidade de o cliente definir todos os objetivos e requisitos no início do projeto, fator que deixou de ser uma necessidade.

A agilidade também é algo prejudicado, pois as equipes só podem trabalhar quando uma tarefa se completa, o que amplia as chances de atraso.

De acordo com Rocha, Isotani e Toda (2014) a definição de cada modelo a ser usado vai depender do tipo de software que o empreendimento pretende fazer. Dentro deste contexto, o primeiro passo para que a escolha seja definida é conhecer o perfil da aplicação de seu usuário e os seus requisitos.

CronApp (2018) esclarece que em muitos casos devido a sua formalização maior o modelo cascata é o mais escolhido. Quando se contrata uma equipe terceirizada para a realização do serviço o modelo cascata é o mais preferido, fator ocasionado pela pouca abertura a mudanças e ter um escopo claro desde o início, delegar responsabilidades para terceiros torna-se mais seguro.

Já em relação a escolha pelo método ágil ocorre frente a necessidade ampla de adicionar, remover ou modificar funcionalidades, além da facilidade de comunicação.

Dentro do contexto dos modelos de software a escolha destes está relacionada a necessidade de sua utilização e a qualidade apresentada pelo mesmo.

Para algumas empresas a satisfação no serviço ou produto que apresentam ao mercado está diretamente ligada a qualidade e essa é imprescindível em qualquer setor da sociedade independente do mercado que serve.

Sobre esses testes para a qualidade do software, Oliveira (2013) os desenvolvedores de software acreditam que as metodologias ágeis são as melhores, no que se diz respeito a criação de produtos com qualidade, uma vez que o uso dessa metodologia muda a postura dos agentes do processo. Esse fator pode ser uma das principais razões pelas quais os métodos ágeis criam projetos de software com alta qualidade.

Hossain, Kashem e Sultana (2013) para comprovar a qualidade nos modelos ágeis foram considerados 14 fatores de qualidade dentre eles: exatidão; verificabilidade; eficiência; integridade; e outros.

Os resultados dos estudos feitos chegaram à conclusão que as metodologias ágeis e qualidade do software, podem trabalhar em conjunto, para a melhoria da qualidade do software.

A utilização dos métodos ágeis traz satisfação ao cliente, pelo curto tempo, orçamento, usabilidade e nível de segurança.

2.3 QUALIDADE DE SOFTWARE

Figueira (2015) a palavra qualidade deriva do latim qualitate e tem como significado “propriedade, atributo ou condição das coisas ou pessoas capaz de distingui-las das outras e de lhes determinar a natureza”

Na indústria e mercado Garvin (2002) o conceito de qualidade foi associado ao fato de estar em conformidade com as especificações de um padrão esperado seja na construção de um objeto ou na prestação de um serviço. Com o passar do tempo, o conceito de qualidade passou a ser associado ao fator satisfação do cliente.

Pressman (2011) o controle de qualidade de software é um conjunto complexo de fatores que podem sofrer variações de acordo com as diferentes aplicações e de acordo com quem os requisita.

A norma ABNT NBR ISO/IEC 25030 (2008), fornece um conjunto de diretrizes e requisitos para avaliar a qualidade do software, o que pode ser útil para empresas e organizações que desejam desenvolver e entregar software de alta qualidade aos seus clientes. Podemos entender que esta norma define qualidade de software como a capacidade do software de atender às necessidades e expectativas dos usuários finais em relação a atributos específicos de qualidade e fornece um conjunto de requisitos e orientações para avaliar e garantir a qualidade do software.

Gomes (2001) as necessidades explícitas são as condições e os objetivos propostos por aqueles que produzem o software. São fatores relativos à qualidade do processo de desenvolvimento do produto e são percebidas pelas pessoas que trabalham no desenvolvimento destes. Já as necessidades implícitas são as necessidades dos usuários, são chamadas também de fatores externos e podem ser percebidas pelos desenvolvedores e pelos usuários.

Ainda de acordo com a autora, os usuários de software analisam a qualidade do mesmo através do desempenho e dos efeitos que seu uso acarreta na organização a qual está inserido. Para os usuários os aspectos internos ou como o software foi desenvolvido não apresenta importância.

Verificando a qualidade do produto uma pergunta é feita pelos estudiosos e profissionais da área a respeito da possibilidade de medir a qualidade de um software.

A norma ABNT NBR ISO/IEC 25030 (2008), tem foco em requisitos e avaliação da qualidade de software e, portanto, aborda vários aspectos da qualidade do software. Ela define um conjunto de atributos de qualidade que podem ser usados para avaliar o software em diferentes contextos e fornece orientações sobre como avaliar a qualidade do software.

Os atributos de qualidade definidos pela norma ABNT NBR ISO/IEC 25030 (2008) incluem:

- Funcionalidade: capacidade do software de atender aos requisitos funcionais e de desempenho especificados.
- Confiabilidade: capacidade do software de manter o desempenho especificado sob condições especificadas por um período especificado.
- Usabilidade: capacidade do software de ser entendido, aprendido, utilizado e atraente para o usuário.
- Eficiência de desempenho: relação entre o desempenho do software e a quantidade de recursos utilizados sob condições especificadas.
- Compatibilidade: capacidade do software de interoperar com outros sistemas em um ambiente comum compartilhado.
- Segurança: capacidade do software de proteger informações e dados confidenciais e garantir a integridade do software.
- Manutenibilidade: capacidade do software de ser modificado e corrigido para atender às necessidades em mudança.

A norma ABNT NBR ISO/IEC 25030 (2008), também fornece orientações sobre como avaliar cada um desses atributos de qualidade do software e define um conjunto de requisitos para avaliar a qualidade do software em diferentes contextos.

Alguns pontos podem e devem ser observados como forma de verificar a qualidade do software. Os testes de qualidade são fundamentais para que haja a observância da qualidade.

2.4 TESTES DE QUALIDADE DE SOFTWARE

Melo (2015) o teste de software é uma das fases do processo de Engenharia de Software que tem como objetivo atingir um nível de qualidade de produto superior. O foco destes testes é encontrar erros para que os mesmos possam ser reparados antes da entrega do produto final.

As atividades de testes para garantir a qualidade dos softwares precisam fazer parte de todo o processo de desenvolvimento do software. Todas as fases do projeto precisam ser testadas.

Deustsch (1995) os erros podem ser encontrados nas primeiras fases do projeto, e quanto mais tarde esses problemas forem encontrados maiores serão os impactos causados.

DevMedia (2018) o teste de software é algo novo, mas de extrema importância. No entanto, algumas empresas ainda justificam que a realização destes testes apresenta altos custos.

Pressman (2011) realizar os testes na fase da engenharia de requisitos custa um real mais caro, já realizar esses testes na fase de funcionamento pode custar cem reais mais caro.

DevMedia (2018) a falta de testes para verificar a qualidade pode ser desastroso, o mais conhecido deles é o incidente que ocorreu na década de 80, o Therac-25, dispositivo computadorizado para tratamento por radiação para câncer. Esse dispositivo emitiu doses elevadas de radiação em pelo menos 6 pacientes, alguns foram mortos e outros incapacitados.

Segundo o site *Food and Drug Administration* (FDA) o problema estava em um programa mal documentado, sem especificações e nem planos de testes: uma quantidade significativa de software para sistemas críticos para a vida vem de pequenas empresas, especialmente na indústria de equipamentos médicos; empresas que enquadram no perfil daquelas que resistem aos princípios tanto da segurança de sistemas quanto da engenharia de software ou os desconhecem.

Neste contexto é que se analisa a importância dos testes para verificar a qualidade dos softwares.

Bastos *et al*., (2007) há três dimensões de qualidade que precisam ser consideradas e define:

- Confiança - o sistema é resistente a falhas durante a execução;
- Funcionalidade - o sistema se comporta conforme o esperado e definido em seus requisitos;
- Performance - o sistema tem um tempo de resposta adequado e aceitável, mesmo quando é submetido a um volume próximo de situações reais ou de pico.

Como forma de atender essas três dimensões, o desenvolvedor e o analista, precisa ter como foco o desenvolver com qualidade, além das dimensões a equipe de análise e desenvolvimento pode fazer o uso do tão conhecido ciclo PDCA: Planejar, Executar, Verificar, Agir.

Melo (2015) ressalta em relação aos testes de software, os quais são feitos para garantia de qualidade, que existem algumas técnicas bem conhecidas e que são usadas com bastante frequência.

2.4.1 TÉCNICAS DE TESTES

Melo (2015) muitas são as maneiras de realização de testes nos softwares, algumas dessas técnicas sempre foram muito desenvolvidas por meio de linguagens estruturadas que são extremamente importantes. Dentre os tipos de testes há: caixa-branca; caixa-preta; testes alpha, Beta e gama; teste de unidade; teste de integração; teste de sistema; teste de verificação e teste de aceitação.

2.4.1.1 TESTE CAIXA-BRANCA

Moraes e Laurindo (2000) estes testes são gerados a partir da análise da estrutura do programa o qual deverá ser testado.

Melo (2015) nesta categoria de teste de software o desenvolvedor tem acesso ao código da fonte de aplicação e pode assim construir códigos para efetuar a ligação de bibliotecas e componentes.

RC: 148546
Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

O autor afirma que esse tipo de teste é desenvolvido a partir da análise do código fonte e elaborando-se casos de teste que cubram todas as possibilidades do programa, dentro deste contexto todas as variações originadas por estruturas de condições são testadas.

2.4.1.2 TESTE CAIXA-PRETA

Melo (2015) neste tipo de teste não há acesso algum ao código fonte do programa. O objetivo deste teste é efetuar as operações sobre as diferentes funcionalidades e verificar se o resultado gerado por estas está conforme o esperado.

2.4.1.3 TESTE ALPHA E BETA

No processo de desenvolvimento do software os testes devem ser realizados antes que o produto final seja entregue ao usuário, o período entre o desenvolvimento e a entrega é conhecido como período alpha (os quais são aplicados após ou em paralelo com os testes de unidade ou unitário. No início dos testes da fase alpha são utilizadas as técnicas caixa branca, posteriormente as técnicas de caixa preta. Ao final da fase alpha são lançados a grupos restritos de usuários versões de testes do sistema denominados de versões Beta (MELO 2015).

Melo (2015) na fase Beta, os usuários podem encontrar defeitos diferentes das tarefas costumeiramente realizadas por eles. Nessa fase podem ser executados testes de caixa-preta dando maior eficiência ao processo.

2.4.1.4 TESTE DE VERIFICAÇÃO

Moraes e Laurindo (2000) definem como testes de verificação as atividades conduzidas pela organização com o objetivo fundamental de verificar os resultados obtidos, e se os mesmos atendem às especificações realizadas em etapas anteriores, e assim garantir que as ações, atividades e decisões entre as etapas de desenvolvimento mantenham sua consistência.

2.4.1.5 TESTE DE ACEITAÇÃO

Melo (2015) os testes de aceitação são realizados por um grupo restrito de usuários finais do sistema. Esse grupo simula operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com o solicitado.

Moraes e Laurindo (2000) os testes de aceitação geralmente são realizados pelos clientes, para que avaliem o produto que lhe será enviado, a organização de desenvolvimento também participa destes testes. Os autores ressaltam que a forma e os critérios a serem utilizados no teste de aceitação são acordados entre as partes logo nas fases iniciais do desenvolvimento.

Muitos são os testes definidos para a análise do correto funcionamento dos softwares. Essas análises ocorrem como forma de validação do produto e garantia de qualidade para os usuários. Sendo assim estes testes são fundamentais e extremamente importantes.

3. ESTUDO DE CASO

Foi realizado um estudo de caso em que foi apresentado os desafios na qualidade dos testes automatizados de interface de usuário usando Selenium. Foram implementadas soluções como tempo de espera explícito, aumento da cobertura de testes e paralelização de testes, o que resultou em melhorias na estabilidade, qualidade e velocidade dos testes. Isso permitiu a entrega de um software de melhor qualidade em menos tempo.

Inicialmente uma equipe de desenvolvimento de software trabalhou em um projeto que requer testes automatizados de interface de usuário. Eles decidiram usar o Selenium como ferramenta de teste automatizado de interface de usuário. No entanto, eles enfrentaram muitos desafios na qualidade de seus testes. Neste estudo de caso, analisou-se os desafios enfrentados pela equipe de desenvolvimento de software e as possíveis soluções para melhorar a qualidade de seus testes em Selenium.

Selenium é uma ferramenta de automação de testes de software amplamente utilizada para automatizar testes de aplicativos web. Ele permite aos desenvolvedores criarem scripts que simulem a interação de usuários com um aplicativo web em diferentes navegadores e plataformas, incluindo computadores desktop e dispositivos móveis.

Esta ferramenta de código aberto suporta várias linguagens de programação, incluindo Java, Python, C#, Ruby, entre outras. Isso o torna uma escolha popular para equipes de desenvolvimento que desejam automatizar seus testes e integrá-los em seus processos de desenvolvimento de software.

Além disso, é altamente configurável e pode ser integrado com outras ferramentas de automação e gerenciamento de projetos, como Jenkins, JIRA e Selenium Grid, para criar testes contínuos em um processo de integração contínua. Por todas essas razões, o Selenium é uma escolha popular para equipes de desenvolvimento de software em todo o mundo.

A equipe de desenvolvimento de software estava enfrentando os seguintes desafios ao testar a interface do usuário com o Selenium:

- Testes instáveis: Os testes estavam falhando intermitentemente sem qualquer motivo aparente.
- Testes de baixa qualidade: Os testes estavam testando apenas alguns casos de teste em vez de cobrir todos os possíveis cenários de teste.

Os testes estavam levando muito tempo para serem executados, o que estava atrasando o processo de desenvolvimento. Solução: Para melhorar a qualidade dos testes em Selenium, a equipe de desenvolvimento de software implementou as seguintes soluções:

3.1 ESTABILIDADE DOS TESTES

Suponha que um teste estava falhando intermitentemente porque a página web não estava completamente carregada antes de uma ação ser executada. Para resolver

RC: 148546
Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

este problema, a equipe de desenvolvimento de software adicionou um tempo de espera explícito antes de cada ação no teste, figura 5.

Figura 5: Tempo de espera

```
WebElement usernameInput = driver.findElement(By.id("username"));
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.elementToBeClickable(usernameInput)).click();
```

Fonte: O autor (2023).

Este trecho aguarda até que o elemento de entrada do nome de usuário esteja clicável e, em seguida, executa a ação de clique.

3.2 QUALIDADE DOS TESTES

Suponha que a equipe de desenvolvimento de software estava testando um formulário de registro que exigia um nome de usuário, senha e endereço de e-mail. Eles perceberam que não estavam testando todos os possíveis cenários de teste, como senhas fracas ou endereços de e-mail inválidos. Para melhorar a qualidade dos testes, eles adicionaram mais casos de teste para cobrir mais cenários de teste, figura 6.

Figura 6: Testes de senhas e e-mail

```
@Test
public void testRegistrationForm() {
  // Test valid registration
  fillRegistrationForm("john_doe", "strongpassword", "john.doe@example.com");
  assertTrue(isRegistrationSuccessful());

  // Test registration with weak password
  fillRegistrationForm("jane_doe", "weak", "jane.doe@example.com");
  assertFalse(isRegistrationSuccessful());

  // Test registration with invalid email
  fillRegistrationForm("jim_smith", "strongpassword", "jim.smith@");
  assertFalse(isRegistrationSuccessful());
}
```

Fonte: O autor (2023).

Este teste cobre três cenários de teste: registro válido, registro com senha fraca e registro com e-mail inválido.

3.3 VELOCIDADE DOS TESTES

Suponha que a equipe de desenvolvimento de software tinha um grande número de testes a serem executados e o tempo de execução estava atrasando o processo de desenvolvimento. Para acelerar o processo de teste, eles implementaram a paralelização de teste, figura 7.

RC: 148546
Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

Figura 7: Paralelização de teste

```java
@Test
public void testLogin() throws InterruptedException {
  ExecutorService executor = Executors.newFixedThreadPool(5);
  List<Future<Boolean>> results = new ArrayList<>();

  for (int i = 0; i < 5; i++) {
    Future<Boolean> result = executor.submit(() -> {
      WebDriver driver = new ChromeDriver();
      driver.get("https://example.com");

      WebElement usernameInput = driver.findElement(By.id("username"));
      WebElement passwordInput = driver.findElement(By.id("password"));
      WebElement loginButton = driver.findElement(By.id("login-button"));

      usernameInput.sendKeys("user" + i);
      passwordInput.sendKeys("password");
      loginButton.click();

      boolean loginSuccessful = driver.getCurrentUrl().equals("https://example.com/dashboard");
      driver.quit();
      return loginSuccessful;
    });

    results.add(result);
  }

  executor.shutdown();
  executor.awaitTermination(1, TimeUnit.MINUTES);

  for (Future<Boolean> result : results) {
    assertTrue(result.get());
  }
}
```

Fonte: O autor (2023).

Este teste cria uma *pool de threads* (*As threads*, “tarefas”, são mantidas em uma *pool*, um local reservado na memória, para que possam ser reutilizadas em execuções futuras, em vez de criar novas threads toda vez que uma tarefa é executada.) e executa cinco instâncias do *Selenium* em paralelo para testar o login de usuário em uma página web. Isso acelera o processo de teste, permitindo que mais testes sejam executados em menos tempo.

4. RESULTADOS

Foi analisado um cenário em que uma equipe de desenvolvimento de software enfrentava desafios relacionados à qualidade dos testes automatizados de interface de usuário utilizando a ferramenta Selenium. Para solucionar esses problemas, foram implementadas estratégias como tempo de espera explícito, aumento da cobertura de testes e paralelização dos testes, resultando em melhorias significativas na estabilidade, qualidade e velocidade dos testes. Essas melhorias permitiram a entrega de um software de alta qualidade em um tempo reduzido.

4.1 DESAFIOS IDENTIFICADOS

A equipe de desenvolvimento de software inicialmente optou pelo uso do Selenium como ferramenta para automação dos testes de interface de usuário. No entanto, eles enfrentaram os seguintes desafios:

Testes instáveis: Os testes automatizados apresentavam falhas intermitentes sem uma razão aparente, prejudicando a confiabilidade dos resultados obtidos.

Testes de baixa qualidade: A equipe estava realizando testes apenas em casos de teste limitados, deixando de cobrir todos os possíveis cenários de teste. Isso comprometia a identificação de falhas em partes específicas do software.

Lentidão na execução dos testes: Os testes demandavam um tempo excessivo para serem concluídos, o que causava atrasos no processo de desenvolvimento como um todo.

4.2 SOLUÇÕES IMPLEMENTADAS

Estabilidade dos Testes Para melhorar a estabilidade dos testes automatizados, a equipe de desenvolvimento implementou a utilização de tempos de espera explícitos antes de cada ação realizada nos testes. Esses tempos de espera garantiam que as páginas web fossem completamente carregadas antes de qualquer ação ser

RC: 148546
Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

executada. Com isso, evitou-se falhas intermitentes decorrentes de páginas ainda não totalmente carregadas.

Qualidade dos Testes A equipe identificou que os testes estavam deixando de abranger diversos cenários de teste relevantes. Para abordar essa questão, eles ampliaram a cobertura de testes, adicionando casos de teste para contemplar uma gama mais abrangente de cenários. Por exemplo, foram incluídos testes para verificar a robustez da aplicação diante de senhas fracas ou endereços de e-mail inválidos. Essa abordagem permitiu a identificação e correção de problemas que poderiam surgir em situações reais de uso.

Velocidade dos Testes A lentidão na execução dos testes era um fator que impactava negativamente o processo de desenvolvimento. Para mitigar esse problema, a equipe adotou a estratégia de paralelização dos testes. Com isso, múltiplas instâncias do Selenium eram executadas simultaneamente, aumentando a eficiência e reduzindo o tempo necessário para concluir todos os testes. Essa abordagem permitiu que mais testes fossem executados em menos tempo, acelerando o processo de desenvolvimento.

4.3 RESULTADOS OBTIDOS

Estabilidade dos Testes A implementação dos tempos de espera explícitos contribuiu para uma maior estabilidade dos testes automatizados. As falhas intermitentes, que anteriormente prejudicavam a confiabilidade dos resultados, foram consideravelmente reduzidas. Agora, as ações executadas nos testes eram realizadas em um contexto de página web totalmente carregada, proporcionando resultados mais consistentes.

Qualidade dos Testes Com a ampliação da cobertura de testes, os casos de teste adicionais permitiram à equipe identificar e solucionar problemas que anteriormente passavam despercebidos. Os testes abrangiam agora uma maior diversidade de cenários, incluindo situações em que a aplicação poderia ser mais vulnerável. Isso resultou em um software mais robusto e confiável, capaz de atender às necessidades dos usuários finais.

Velocidade dos Testes A paralelização dos testes trouxe ganhos significativos em termos de velocidade de execução. A capacidade de executar múltiplas instâncias do Selenium simultaneamente permitiu que um maior número de testes fosse concluído em um tempo reduzido. Isso possibilitou uma maior agilidade no processo de desenvolvimento, com entregas mais rápidas sem comprometer a qualidade dos testes.

Essas melhorias permitiram à equipe entregar um software de alta qualidade em um prazo reduzido, atendendo às expectativas dos usuários e cumprindo os requisitos do projeto. O uso efetivo do Selenium como ferramenta de automação de testes de interface de usuário foi maximizado, proporcionando uma solução confiável e eficiente para a equipe de desenvolvimento de software.

5. CONCLUSÃO

A partir do objetivo proposto foi possível concluir que os softwares têm desempenhado um papel de extrema importância na sociedade que tem acompanhado as evoluções tecnológicas.

Devido à grande utilidade destes softwares a qualidade dos mesmos é um fator fundamental, especialmente para as organizações que muitas vezes dependem desses sistemas para o desenvolvimento de suas atividades.

Para a garantia da qualidade dos softwares, os testes em busca de possíveis defeitos são de extrema necessidade. No entanto, é preciso entender que estes testes por si só não livram totalmente os softwares de possíveis falhas.

Como os testes de software definem a qualidade dos mesmos, precisam ser realizados por equipes que dominem o conhecimento a respeito do que são os softwares e os testes que são podem e devem ser aplicados.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR ISO/IEC**

25030:2008: Engenharia de software - Requisitos e Avaliação da Qualidade de

Produto de Software (SQuaRE) - Requisitos de qualidade. Rio de Janeiro, 2008.

BASTOS, A. *et al*. **Base de conhecimento em teste de software**. São Paulo: Martins Fontes, 2007.

BRAGA, R. **Evolução Histórica da Computação e Aplicações**. História da

Engenharia de Software, 2011. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/3371072/mod_resource/content/1/Aula06-HistoriaEngenhariaDeSoftwareFinal.pdf>. Acesso em: 20 abr. 2018.

CRONAPP. **Compreenda as diferenças entre o desenvolvimento nativo e híbrido**. Cronapp Blog, 2018. Disponível em: <https://www.cronapp.io/pt-br/compreenda-as-diferencas-

entre-o-desenvolvimento-nativo-e-hibrido/>.

DESIKAN, S.; RAMESH G. **Software Testing:** Principles and Practices. 1.ed. Delhi: Pearson, 2009.

DEUSTSCH, M. Verification, and validation. In: PRESSMAN, R. **Engenharia de software**. São Paulo: Makron Books, 1995.

DEVMEDIA. **Cursos de Programação Online**. Devmedia, 2018. Disponível em: <https://www.devmedia.com.br/cursos/>.

ENGHOLM, H. J**. Engenharia de Software na prática**. 1ª. ed. São Paulo: Novatec, 2010.

FABRI, J. A. **A aplicabilidade do modelo cascata na Engenharia de Software**. Engenharia de Software, 2012. Disponível em: <https://engenhariasoftware.wordpress.com/2013/01/24/a-aplicabilidade-do-modelo-cascata-na-engenharia-de-software/>. Acesso em: 20 abr. 2018.

FIGUEIRA, A. **Avaliação da Pessoa em Situação Crítica**: Aplicação do National Early Warning Score (NEWS). Doctoral dissertation, Instituto Politécnico de Setúbal. Escola Superior de Saúde, 2015.

GARVIN, D. A. **A note on corporate venturing and new business creation**. Executive Education, 2002.

GOMES, N. S. **Qualidade de Software - uma necessidade**. Unisalesiano, 2001. Disponível em:< http://www.unisalesiano.edu.br/salaEstudo/materiais/pd6952/material1.pdf>. Aceso em: 12 mai. 2018.

HOSSAIN, A.; KASHEM, D. M. A.; SULTANA, S. **Enhancing Software Quality Using Agile Techniques**. Tese (Doutorado) - Curso de Computer Science And Engineering Department, Dhaka University Of Engineering & Technology, Bangladesh, 2013. Disponível em:<http://www.ceavi.udesc.br/arquivos/id_submenu/787/gabriel_linardi___qualidade_de_software_na_metodologia_agil.pdf>. Acesso em: 20 set. 2018.

JALOTE, P. **An Integrated Approach to Software Engineering**. 3.ed. New York: Springer, 2005.

LARMAN, C. **Utilizando UML e Padrões:** uma introdução à análise e ao projeto orientados a objetos. Tradução de Luiz Augusto Meirelles Salgado. Porto Alegre: Bookman, 2000.

LEITE, J. C. **O processo de desenvolvimento de software.** Dimap 2000. Disponível em: <https://www.dimap.ufrn.br/~jair/ES/c2.html>. Acesso em: 10 mar. 2018.

MATTÉ, M. A. **Testes de software:** uma abordagem da atividade de teste software em metodologias ágeis aplicando a técnica behavior driven development em um estudo experimental.2011. Monografia (Especialização em Engenharia de

Software) - Universidade Tecnológica Federal do Paraná – UTFPR – Medianeira, 2011. Disponível em:<http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/1111/3/MD_ENGESS_I_2012_17.pdf>.

Acesso em: 3 maio 2018.

MELO, W. **Qualidade + testes de Software = Qualidade de Software**. Tiespecialistas, 2015. Disponível em: <https://www.tiespecialistas.com.br/qualidade-testes-de-softwares-qualidade-de-software/>. Acesso em: 20 mai. 2018.

MORAES, R. O.; LAURINDO, F. J. B. **Teste de Software e Qualidade de Software:** uma visão geral. ABEPRO, 2000. Disponível em: <http://www.abepro.org.br/biblioteca/enegep1999_a0198.pdf>. Acesso em: 10 mar. 2018.

NAKAGAWA, E. Y. *et al.* **Towards a Reference Architecture for Software Testing Tools**. In: 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007), 2007, Boston. p. 157-162.

OLIVEIRA, B, H. **Qualidade de software no desenvolvimento com métodos ágeis**. 2013. Dissertação (Mestrado em Ciências da Computação e Matemática) Universidade de São Paulo, São Carlos, 2013.

PAULK, M. C. *et al***. Capability Maturity Model for Software (Version 1.1).** Software Engineering Institute, 1993. CMU/SEI-93-TR- 024. Disponível em:<https://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr24.93.pdf >. Acesso em: 2 abr. 2018.

PAULA FILHO, W. P. **Engenharia de software:** fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2003.

PRESSMAN, R. S. **Engenharia de Software - uma abordagem profissional.** 7ª ed. Porto Alegre: AMGH Bookman, v. I, 2011.

REZENDE, D. A. **Engenharia de Software e Sistemas de Informações.** Rio de Janeiro: Brasport, 2005.

ROCHA, R.; ISOTANI, S.; TODA, A. M. **Modelos de Processo de Software**. Edisciplinas-USP, 2014. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/2939594/mod_resource/content/1/Aula02_ModelosProcessos.pdf>. Acesso em: 12 jun. 2018.

SOMMERVILLE, I. **Engenharia de Software**. 6ª ed. São Paulo: Addison Wesley, 2003.

_____. **Engenharia de Software**. 9ª ed. São Paulo: PEARSON, v. I, 2011.

Enviado: 16 de junho, 2023.

Aprovado: 27 de junho, 2023.

[1] Graduando em Engenharia da Computação. ORCID: https://orcid.org/0009-0002-3025-0193.
[2] Mestre em Engenharia da Produção – USP. ORCID: https://orcid.org/0000-0001-9602-5293. Currículo Lattes: http://lattes.cnpq.br/2839375294680374.
[3] Doutora em Ciências dos Alimentos e Nutricionais. ORCID: https://orcid.org/0000-0002-9341-0417. Currículo Lattes: http://lattes.cnpq.br/7128829324567785.