FACULDADE DE TECNOLOGIA DE SÃO PAULO

MATEUS DE OLIVEIRA ANDRADE

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE TESTE AUTOMATIZADO EM DESENVOLVIMENTO MOBILE

SÃO PAULO

2024

FACULDADE DE TECNOLOGIA DE SÃO PAULO

MATEUS DE OLIVEIRA ANDRADE

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE TESTE AUTOMATIZADO EM DESENVOLVIMENTO MOBILE

Trabalho submetido como exigência parcial
para a obtenção do Grau de Tecnólogo em
Análise e Desenvolvimento de Sistemas
Orientador: Professor Sergio Luiz Banin

SÃO PAULO
2024

Dedico este trabalho aos meus pais que me deram todo apoio e suporte necessário durante o período em que estive no curso.

RESUMO

Com o uso de smartphones tão presente no nosso dia a dia, o mal funcionamento dos aplicativos pode gerar uma série de problemas, tanto para os usuários quanto para as empresas. Este trabalho aborda a importância da implementação de estratégias de teste de software automatizado para garantir a qualidade de aplicativos móveis. Esta pesquisa destaca a evolução dos dispositivos e as tecnologias de desenvolvimento, diferenciando as abordagens nativa e hibrida. Este trabalho também explora as metodologias ágeis e ferramentas de integração contínua, além disso discorre sobre as principais ferramentas de teste automatizado levando em consideração o tipo e o contexto do teste. Implementar essas estratégias é essencial para garantir um aplicativo seguro e que proporcione uma boa experiencia para o usuário além de performar de maneira satisfatória em diferentes dispositivos e sistemas operacionais.

Palavras-chave: Teste de software automatizado, Qualidade, Aplicativos móveis, Metodologias ágeis, Experiência do usuário.

ABSTRACT

With the use of smartphones so prevalent in our daily lives, the malfunctioning of applications can cause a series of problems for both users and companies. This thesis addresses the importance of implementing automated software testing strategies to ensure the quality of mobile applications. This research highlights the evolution of devices and development technologies, differentiating between native and hybrid approaches. This work also explores agile methodologies and continuous integration tools. Additionally, it discusses the main automated testing tools, taking into account the type and context of the test. Implementing these strategies is essential to ensure a secure application that provides a good user experience and performs satisfactorily on different devices and operating systems.

Keywords: Automated software testing, Quality, Mobile applications, Agile methodologies, User experience.

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia, os dispositivos móveis tornaram-se elementos indispensáveis no cotidiano das pessoas, facilitando a comunicação, o acesso à informação e o entretenimento. Segundo a pesquisa do IBGE, PNAD Contínua TIC, no ano de 2022 o acesso à internet em domicílios brasileiros foi de 91,5% e o principal meio de acesso foi o telefone móvel, em 98,9% dos casos.

Assim sendo, há uma evidente utilização de smartphones pela sociedade brasileira, de maneira que o uso de aplicativos é cada vez mais presente no dia a dia dos cidadãos. É por meio de aplicativos que as pessoas se comunicam via redes sociais, consomem conteúdos em *streaming*, seja música, podcast ou filmes e séries, realizam compras variadas, gerenciam contas bancárias, enfim, realizam uma gama gigante de atividades.

Porém, a crescente complexidade dessas aplicações e a diversidade de dispositivos e sistemas operacionais representam desafios significativos para garantir a qualidade do software. Dessa maneira é importante adotar estratégias eficazes para testar os aplicativos, visto que falhas nessas aplicações podem resultar em experiências negativas para o usuário, afetando a reputação da empresa e comprometendo a fidelidade do cliente.

Portanto, é necessário que abordagens para verificar a integridade e o desempenho desses aplicativos em diferentes cenários. É fundamental que essas abordagens sejam aplicadas em todo processo de desenvolvimento, considerando o mercado dinâmico em que vivemos.

Este trabalho reúne um conjunto de práticas e ferramentas essenciais para a qualidade no desenvolvimento de aplicativos, sendo que os principais tópicos são: Desenvolvimento *Mobile*, Qualidade de software, Metodologias de Desenvolvimento de Software e Ferramentas de automação de teste para aplicativos móveis.

1.1 Objetivos

Este trabalho visa atingir uma série de objetivos que vão desde estudar a evolução do desenvolvimento *mobile*, explorar tecnologias e ferramentas de desenvolvimento e de automação de teste e a implementação de práticas de qualidade e desenvolvimento ágil no desenvolvimento de aplicativos. Principais objetivos:

1. Estudar a Evolução do Desenvolvimento *Mobile*: Discorrer sobre a evolução dos aparelhos desde seu surgimento e analisar suas características.
2. Explorar Tecnologias e Ferramentas de Desenvolvimento *Mobile*: Examinar as principais tecnologias e ferramentas utilizadas no desenvolvimento de aplicativos móveis, tanto para plataformas Android quanto iOS, incluindo linguagens de programação e ambientes de desenvolvimento.
3. Implementar Estratégias de Teste Automatizado: Identificar estratégias de teste automatizado em aplicativos móveis levando em consideração diferentes tipos de teste, como testes unitários, funcionais e de desempenho, a plataforma (Android ou iOS) e as ferramentas utilizadas no desenvolvimento.
4. Comparar Abordagens de Desenvolvimento Nativo e Híbrido: Comparar as vantagens e desvantagens das abordagens de desenvolvimento nativo e híbrido, avaliando seu impacto na performance, experiência do usuário e viabilidade do projeto.
5. Avaliar a Qualidade de Software em Aplicativos Móveis: Investigar as práticas para garantir a qualidade do software, com destaque para a importância de testes durante o desenvolvimento.
6. Integrar Metodologias Ágeis e Práticas de Integração Contínua: Explorar a integração de metodologias ágeis e práticas de integração contínua no desenvolvimento de aplicativos com a finalidade de reduzir riscos e entregar um software moderno que atenda aos requisitos e necessidades dos usuários.

2 DESENVOLVIMENTO MOBILE

Com o lançamento do Motorola DynaTAC em 1983, o mundo foi apresentado ao primeiro telefone móvel. O aparelho custava cerca de US$ 4.000,00 e era considerado um símbolo de *status* na época. Apenas dois anos depois foi feita a primeira chamada telefônica, no Reino Unido, com o Sir Ernest Harrison.

A Motorola seguiu dando continuidade ao projeto e em 1989 lançou o modelo 9800X ou MicroTAC, que possuía uma tampa dobrável e estabeleceu o padrão *flip phone*. Com o passar dos anos diversos modelos e padrões surgiram e os aparelhos foram se tornando cada vez menores e mais populares.

Mas foi em 2007 com o lançamento do primeiro iphone que uma verdadeira revolução no mercado teve início. Antes os aparelhos eram grandes, robustos e geralmente tinham um teclado físico, entretanto o aparelho da Apple tinha uma proposta diferente, se apresentava em peça inteiriça ocupada por uma tela sensível ao toque na frente, um botão de função logo abaixo da tela, botões nas laterais para ajuste de volume e um botão superior, além das entradas de energia e fone de ouvido. O aparelho também possuía uma câmera.

Foi a criação da companhia de Steve Jobs que estabeleceu o padrão visto até hoje nos smartphones. A proposta de usabilidade do Iphone possibilitou a difusão do uso de aplicativos, que por si só é um mercado.

Com o impacto do Iphone no mercado, os concorrentes viram a necessidade de criar outros dispositivos e em 2008 foi lançado o primeiro dispositivo com o sistema operacional Android, o HTC Dream.

Os Iphones da Apple usam como sistema operacional o IOS, que é proprietário da Apple, não podendo ser usado em outros dispositivos. Já Android surgiu como um sistema operacional que pudesse ser usado em diversos dispositivos.

Com o passar dos anos, tanto a Apple quanto seus concorrentes Android se popularizaram e hoje existe um mercado gigantesco, tanto de aparelhos, mas também de aplicativos.

2.1 Importância do Desenvolvimento para dispositivos mobile

Na atualidade os celulares e smartphones fazem parte do nosso dia a dia e é por meio de aplicativos que nossa rotina fica mais prática. Podemos realizar ligações

e trocar mensagens de texto, que é o que se propôs quanto ao uso desses aparelhos com o passar dos anos, porém mais do que isso podemos checar e-mails, realizar compras, organizar finanças, realizar investimentos, navegar em sites e mais uma infinidade de atividades, ao ponto que os celulares podem ser considerados computadores de bolso.

Segundo a PNAD Contínua TIC, pesquisa do IBGE, do ano de 2022, 91,5% dos domicílios brasileiros possuíam acesso à internet sendo que o principal equipamento mais utilizado para acessar a internet foi o telefone móvel (98,9%). A seguir, com uma grande diferença, vinha a TV com (47,5%). Além disso a pesquisa também mostrou uma diminuição no uso de microcomputadores e o aumento no uso de tablets.

Embora televisores não sejam móveis e o tamanho avantajado dos tablets prejudiquem sua portabilidade, boa parte desses dispositivos usa como sistema operacional o Android. Dessa forma, há um mercado gigantesco de aparelhos e uma necessidade de desenvolver aplicativos para toda essa gama de dispositivos, pois além de resolver problemas do cotidiano dos usuários, também pode ser lucrativo.

2.2 Tecnologias para desenvolvimento mobile

No mercado *mobile* existem basicamente dois principais sistemas operacionais, Android e IOS. Embora o Android pertença ao Google, se trata de um projeto de código aberto, podendo ser utilizado em diversos tipos de dispositivos, de tal maneira que sua usabilidade varie de acordo com a customização feita no sistema por parte dos fabricantes dos aparelhos. Já na Apple, apenas os iPhones, iPads e iPods touch utilizam o IOS.

Isso influencia na maneira com que os aplicativos são desenvolvidos, possuindo uma gama de ferramentas para realizar os projetos. No mercado existem duas principais abordagens de desenvolvimento: o nativo e o híbrido. Cada uma possui vantagens e desvantagens, características distintas que influenciam diretamente na experiência do usuário, do desenvolvedor e no desempenho do app e viabilidade do projeto.

O desenvolvimento nativo se caracteriza pela criação de aplicativos sob medida para cada sistema operacional, utilizando linguagens de programação específicas como Swift para iOS e Java ou Kotlin para Android. Essa abordagem oferece uma série de benefícios:

- Desempenho Superior: Aplicativos nativos tiram proveito máximo dos recursos dos dispositivos, resultando em interfaces mais fluidas, animações impecáveis e um tempo de resposta rápido.
- **Experiência do Usuário Aprimorada:** A interação com o aplicativo se torna mais intuitiva e agradável, com recursos personalizados para cada plataforma e aderência às diretrizes de design de cada sistema operacional.
- Acesso Completo aos Recursos do Dispositivo: Câmera, GPS, acelerômetro e outros recursos são facilmente acessíveis, abrindo um leque de possibilidades para funcionalidades inovadoras.
- Segurança Robusta: O desenvolvimento nativo geralmente proporciona maior segurança contra vulnerabilidades, pois os aplicativos são compilados em código de máquina específico para cada plataforma.

Já na abordagem de desenvolvimento híbrido, se tivermos como exemplo React Native, as tecnologias web como HTML, CSS e JavaScript são utilizadas para criar um app único que funciona em diversas plataformas. Essa flexibilidade traz consigo algumas vantagens:

- Redução de Tempo e Custo: O desenvolvimento híbrido geralmente exige menos tempo e recursos, pois a base de código é única e compartilhada entre as plataformas.
- Manutenção Simplificada: Atualizações e manutenções são mais fáceis de gerenciar, pois a mesma base de código serve para todas as plataformas.
- Alcance Ampliado: Um único aplicativo atinge um público maior, compatível com diversos sistemas operacionais como iOS, Android e Windows Phone.
- Desenvolvimento Multiplataforma: Desenvolvedores com conhecimento em tecnologias web podem facilmente se adaptar ao desenvolvimento híbrido, expandindo as possibilidades de criação de aplicativos.

Escolher entre o desenvolvimento nativo e híbrido depende de diversos fatores, como os objetivos do aplicativo, o público-alvo, o orçamento e a expertise da equipe de desenvolvedores.

O nativo pode ser a melhor escolha se o objetivo for priorizar o desempenho e a experiência de usuário, visto que nesta abordagem é possível ter maior controle

sobre o consumo de recursos do aplicativo, dessa forma entregando uma experiência de usuário superior em aplicativos que necessitam de muitos recursos, como jogos 3D, editores de vídeo e aplicativos de realidade aumentada.

Enquanto o híbrido pode ser uma opção mais viável se o objetivo for atingir um público maior e desenvolver com menores custos, visto que a partir de um código é possível gerar dois apps, um para cada sistema operacional.

2.3 Desenvolvimento nativo para Android

O Android foi criado pela Android Inc em 2003 na Califórnia, o sistema operacional foi baseado no kernel do Linux e, portanto, se tratava de um projeto *open source*. Em 2005 a Android Inc foi comprada pela Google, então uma nova divisão dedicada ao *mobile* surgiu dentro da estrutura da empresa. Após o lançamento do iPhone, em 2007, a Google decidiu firmar aliança com diversas empresas de hardware e software para implantar o primeiro dispositivo Android no mercado. Nesta mesma época, a Google realizou diversos investimentos para que os desenvolvedores melhorassem os aplicativos e o próprio SO.

Pode-se dizer que os esforços do Google valeram a pena, visto que em 2023 o Android se consolidou como o sistema operacional *mobile* mais popular do mundo, atingindo um total de 71,6% da participação total de mercado.

Sendo assim, com uma grande quantidade de dispositivos no mercado usando Android, é notável o mercado de dispositivos e aplicativos para este sistema operacional.

No desenvolvimento de aplicativos nativos Android, Java e Kotlin são as linguagens utilizadas, sendo que desde 2017 o Kotlin foi anunciado oficialmente pelo time de Android da Google como a principal linguagem de programação para o desenvolvimento de aplicações Android.

Independentemente da linguagem escolhida, a IDE utilizada é o Android Studio, uma plataforma da empresa JetBrains mantida em parceira com o Google, que pode ser baixada e instalada em um computador compatível de maneira gratuita. O Android Studio é um programa que necessita de recursos de hardware bem consideráveis, como no mínimo 8 Gigas de memória RAM, mas o recomendado sendo 16, e processadores Intel ou AMD recentes, compatíveis com recursos de virtualização.

Durante o desenvolvimento é possível emular um aparelho na ferramenta para ter noção de como o aplicativo está ficando e realizar testes e isso é um dos recursos do Android Studio que mais impacta no uso de memória RAM e CPU dos computadores e que pode ser uma barreira, pois além de uma quantidade considerável de RAM e uma boa capacidade de CPU também é necessário que o hardware seja compatível funções de Virtual Machine ativadas em BIOS, visto que a emulação “torna” uma parte dos recursos do computador em um dispositivo Android. Uma forma de contornar este gargalo é utilizar um aparelho compatível conectado ao computador via USB para visualizar o aplicativo desenvolvido e realizar testes, economizando assim recursos de hardware da máquina que se está usando para desenvolver. Também é possível realizar testes utilizando o aparelho externo conectado.

Java é uma linguagem de programação que foi criada na década de 90 pela empresa Sun MicroSystems. Em 2008 acabou sendo comprada pela Oracle. Se trata de uma linguagem orientada a objetos e, para o desenvolvimento *mobile*, uma de suas principais vantagens é que ela foi utilizada por muitos anos e é muito utilizada até a atualidade, pelo fato de ser compatível com inúmeros sistemas, seja o Android, sistemas embarcados, computadores, aparelhos inteligentes, entre diversos outros. A alta compatibilidade com uma grande variedade de dispositivos se deve ao modo de funcionamento da linguagem, o Java é executado em uma JVM (Java virtual machine) um ambiente virtual criado para realizar a execução do código Java, o que permite que qualquer dispositivo que suporte a JVM possa executar um código Java, dessa maneira as aplicações podem ser migradas para outros sistemas e podem ser executadas sem muito esforço.

Kotlin também é uma linguagem de programação orientada a objetos, foi criada pela JetBrainss, a linguagem pode ser compliada na JVM (Java virtual machine), sendo retrocompatível com Java, de maneira que parte do código Kotlin possa ser escrita em Java sem prejudicar a etapa de compilar o código. Um dos objetivos da linguagem é resolver alguns problemas e limitações do Java e servir como um substituto atualizado. Por conta dos seus objetivos e de compilar na JVM é tão versátil quanto Java.

2.4 Desenvolvimento nativo para IOS

O sistema operacional da Apple é baseado no Darwin, kernel de código aberto lançado pela Apple em 2000, esse sistema tem como base o XNU, outro kernel (núcleo) de código aberto.

A Apple é mais rigorosa quanto aos requisitos para publicação de um aplicativo na App Store, as diretrizes para disponibilizar um app na sua loja de aplicativos estão descritas no “App Store Review Guideliness).

Para ser publicado na loja de aplicativos da Apple, o aplicativo deve atender a alguns requisitos técnicos e de conteúdo.

Alguns dos requisitos técnicos dizem respeito a:

- **Performance**: O aplicativo deve funcionar de forma eficiente, sem bugs ou falhas que afetem a experiência do usuário.
- **Segurança**: O aplicativo deve ser seguro e não conter vulnerabilidades que possam comprometer os dados dos usuários.
- **Privacidade**: O app deve ter uma política de privacidade clara e garantir a proteção dos dados dos usuários.

Já as diretrizes de conteúdo:

- **Originalidade e Qualidade**: O conteúdo do aplicativo deve ser original, útil e de alta qualidade.
- **Propriedade Intelectual**: O app não deve infringir direitos autorais, marcas registradas ou outras propriedades intelectuais.
- **Respeito às Normas Legais**: O aplicativo deve cumprir todas as leis e regulamentos aplicáveis, incluindo aqueles relacionados a jogos de azar, saúde, privacidade e segurança.

Além de satisfazer esses requisitos, os aplicativos disponibilizados na loja da Apple devem respeitar diretrizes de interface e usabilidade, garantindo uma experiencia intuitiva e consistente e acessibilidade a todos os usuários. Os aplicativos também devem ser testados e inclusive revisados pela própria Apple.

Mesmo satisfazendo os requisitos de publicação, os apps também devem atender ao "iOS Design Themes”, que estabelece princípios de design afim de garantir que o usuário tenha uma experiencia harmoniosa e intuitiva em todos os aplicativos iOS. Os principais temas incluem:

1. Clareza: O design deve comunicar de forma clara, com elementos gráficos nítidos, tipografia legível e interface intuitiva. A clareza facilita a navegação e a compreensão das funções do aplicativo.
2. Deferência: O design deve ser deferente ao conteúdo, utilizando uma interface fluida e descomplicada que não distraia o usuário. Elementos interativos devem ser intuitivos e os conteúdos, prioritários.
3. Profundidade: O uso de camadas visuais e movimentos realistas que dão a sensação de profundidade, proporcionando uma experiência mais envolvente. As transições suaves e animações ajudam a orientar o usuário através da interface.

Atender aos requisitos de design do iOS oferece uma série de benefícios como maiores chances de aprovação do aplicativo na plataforma, bem como aprovação mais rápida, consistência visto que os aplicativos da loja ficam em harmonia e melhor experiencia do usuário devido a familiaridade.

Um requisito imprescindível para desenvolver um aplicativo para o iOS é utilizar um computador da Apple. Somente os computadores ou laptops que possuem o sistema operacional Mac OS podem utilizar o Xcode (Ambiente de desenvolvimento integrado) da Apple de maneira nativa. Existem alternativas não oficiais para executar o Xcode em máquinas que não são da Apple, mas não são recomendadas. Sendo esse uma das maiores barreiras de entrada para desenvolvimento iOS, devido ao custo elevado dos dispositivos Apple. Além de um computador da empresa da maçã, também é necessária uma conta de desenvolvedor cadastrada no Apple Developer Program.

Uma vez, estando ciente dos requisitos de publicação de aplicativos na app store da Appe e possuindo um Mac para desenvolver, é necessário escolher para qual versão do sistema operacional iOS o app será feito. Essa etapa é de suma importância, visto que ao publicar um aplicativo, o mais interessante é que ele esteja disponível para o maior número de pessoas possível, caso seja escolhido o sistema mais recente para publicação, será possível utilizar os recursos mais atuais da plataforma, porém muitos usuários só terão acesso ao app depois que atualizarem o SO do seu aparelho. Assim, sendo, a escolha mais coerente deve ser de disponibilizar o aplicativo para a versão do sistema com a maior base de usuários.

Em maio de 2023 a Apple divulgou que 81% de todos seus iPhones estavam executando o iOS 16.

Após decidir a versão do iOS desejada para desenvolver o aplicativo, basta escolher a linguagem de programação, as principais são Swift e Objective-C.

Objective-C é uma linguagem de programação que nasceu diretamente a partir da linguagem C, como uma evolução capaz de utilizar conceitos de orientação a objetos, a linguagem também usa o mesmo compilador de código de sua antecessora.

Em 2014, na WWDC (Worldwide Developers Conference) a Apple anunciou o Swift como linguagem oficial para desenvolvimento de Apps iOS, substituindo a Objective-C. A Swift rapidamente se popularizou por ter algumas facilidades se comparada com Objcetive-C. Embora tivesse mantido alguma similaridade com a sintaxe de sua antecessora, a nova linguagem oficial passou a utilizar também sintaxe reduzida, que lembram outras linguagens como Python e Ruby.

Swift é uma linguagem *open source* (de código aberto) e a própria Apple disponibiliza uma comunidade na qual desenvolvedores de todo o mundo possam colaborar com melhorias no código da plataforma.

2.5 Desenvolvimento Híbrido

A abordagem hibrida de desenvolvimento de aplicativos móveis se popularizou pois consegue mitigar algumas desvantagens do desenvolvimento nativo. Existem diversas linguagens e frameworks que foram criados para solucionar esses problemas.

2.5.1 React Native

O React Native é um framework criado pelo Facebook (atual Meta) para a construção de interfaces de aplicativos móveis. A ferramenta foi lançada em 2015 e rapidamente ganhou popularidade no mercado de desenvolvimento de aplicativos multiplataforma.

A empresa já utilizava o React, uma biblioteca para JavaScript usada para criar interfaces web, e aproveitou a oportunidade de adaptar a tecnologia para o desenvolvimento *mobile*. O React Native surgiu da necessidade de criar aplicativos eficientes que pudessem ser executados em diferentes plataformas.

Como foi criado a partir do React, o React Native se utiliza de vários recursos comuns do desenvolvimento web e é implementado em JavaScript, uma linguagem muito popular, o que facilita quando o time de desenvolvimento é formado, pois diversos profissionais têm conhecimento na linguagem.

Embora seja uma ferramenta de desenvolvimento hibrida, é possível utilizar componentes nativos de cada plataforma (Android ou iOS) com o React Native, o que garante que os aplicativos tenham a mesma aparência e que os aplicativos desenvolvidos com as ferramentas nativas de cada plataforma sem comprometer o desempenho.

Mas uma das principais qualidades do framework é a possibilidade de gerar um app nativo para cada plataforma a partir de um único código, sendo necessário trabalhar apenas nas partes nativas de cada executável se necessário.

2.5.2 Flutter

O Flutter é um framework de desenvolvimento criado pelo Google para a construção de interfaces de aplicativos *mobile* multiplataforma nativas usando a linguagem de programação Dart. Ele foi lançado em 2017 e rapidamente se popularizou, tornando-se uma das ferramentas mais populares para o desenvolvimento de aplicativos móveis.

O Google já utilizava o Dart, uma linguagem de programação moderna e orientada a objetos e viu a oportunidade de adaptá-la ao desenvolvimento *mobile*.

O Flutter tem a capacidade de compilar o código Dart em código nativo para cada plataforma, de maneira semelhante ao React Native, proporcionando assim a mesma aparência de aplicativos nativos sem que o desempenho seja prejudicado.

Embora o Flutter possua boas características em comum com outras ferramentas de desenvolvimento de aplicativos multiplataforma, a utilização do Dart pode oferecer certa escassez de profissionais qualificados com experiência nessa linguagem, mesmo que sua sintaxe se assemelhe a do JavaScript.

2.5.3 Ionic

O Ionic é outro framework de desenvolvimento de código, utilizado para a construção de aplicativos móveis híbridos e de aplicativos da web progressivos (PWAs). Assim como o React Native, utiliza recursos de desenvolvimento web como HTML, CSS e JavaScript. Foi lançado em 2013 e se tornou uma escolha popular

para o desenvolvimento de aplicativos multiplataforma mantendo uma única base de código.

Seus criadores, Max Lynch e Ben Reed, buscavam uma maneira de desenvolver aplicativos móveis multiplataforma sem a necessidade de aprender várias linguagens de programação nativas.

Os aplicativos Ionic são executados em um navegador web dentro de um contêiner nativo, dessa maneira, o aplicativo é compatível com uma ampla gama de dispositivos, porém podem apresentar um desempenho relativamente menor se comparado a aplicativos nativos.

A ferramenta possui uma grande variedade de plugins que podem ser usados para adicionar funcionalidades aos aplicativos. Além disso utiliza ferramentas como HTML, CSS e JavaScript, o que o torna mais fácil de aprender para desenvolvedores que já tenham experiência com desenvolvimento web.

2.5.4 Kotlin Multplataform

O Kotlin é atualmente a linguagem de programação mais recomendada para desenvolvimento de aplicativos Android nativos, porém em 2018 a Google anunciou o Kotlin Multiplataform (KMM) e vem trabalhando na ferramenta ao longo dos anos visando atender também ao desenvolvimento híbrido de aplicativos de maneira que ao desenvolver um app Android nativo seja possível reutilizar a mesma base de código e gerar um aplicativo compatível com iOS.

3 QUALIDADE DE SOFTWARE

A Qualidade de Software se refere ao conjunto de características que um software deve possuir para atender às necessidades e expectativas dos seus usuários. Isso significa que o software deve funcionar de forma consistente e previsível, sem falhas ou erros frequentes, além de ser fácil de aprender e usar, com interface intuitiva e amigável. Tais aspectos são perceptíveis ao usuário final, entretanto existem processos de qualidade que devem ser aplicados ao desenvolvimento do software. A implementação desses processos e requisitos devem ser feitas, visto que contribuem para a qualidade como um todo.
Sobre a qualidade de software e sua importância:

> Os sistemas de software são parte integrante de nossa vida cotidiana. A maioria das pessoas já teve experiência com software que não funcionou como esperado. Um software que não funciona corretamente pode causar muitos problemas, inclusive perda de dinheiro, tempo ou reputação comercial e, em casos extremos, até mesmo lesões ou morte. O teste de software avalia a qualidade do software e ajuda a reduzir o risco de falha do software em operação. (Certified Tester Foundation Level Syllabus, 2024, v4.0, p. 14).

3.1 Objetivos da qualidade de Software

Melhorar aspectos do desenvolvimento de software, de maneira que os projetos sejam bem planejados, entregues respeitando os prazos e mantendo os custos sobre controle.

Melhorar a satisfação dos usuários, atendendo as necessidades na execução de suas tarefas. Esse é um dos principais objetivos, visto que uma base de usuários satisfeitos tem a capacidade de tornar o aplicativo popular, além de chamar a atenção de novos usuários também reter aqueles que já experimentaram o sistema, destacando o app no mercado e ampliando a imagem positiva da empresa.

Tornar o produto e a empresa competitivos no mercado, não só por ter um aplicativo satisfatório do ponto de vista do usuário final, mas também trabalhar na evolução do sistema, tornando-o adaptável ao passar do tempo, ou seja, capaz de acompanhar as tendencias de mercado e melhor satisfazer os usuários.

Melhorar o custo-benefício do projeto, a medida em que o aplicativo tem uma boa base de usuários e é tolerante a falhas, a empresa mantenedora pode evitar gastos com processos judiciais por conta de falhas-críticas ou com o retrabalho no projeto.

3.2 Melhores práticas da qualidade de software

A qualidade de software engloba uma série de atividades, processos e metodologias a fim de assegurar um software satisfatório para o usuário final, mitigar riscos evitando bugs e falhas e, assim, aumentar as chances de sucesso do sistema quando publicado. Boa parte dessas atividades são dão por meio de testes, que validam diversos aspectos do software.

3.2.1 Testes de software

O teste de software engloba todo um conjunto de atividades para revelar defeitos e validar a qualidade dos objetos de teste. Por meio de testes é possível validar se o software respeita os requisitos previamente estabelecidos, assegurando que o que está sendo desenvolvido está de acordo com o planejamento. É testando o software que é possível detectar defeitos e fornecer dados para que esses defeitos sejam corrigidos antes da entrega do produto.

3.2.1.1 Objetivos gerais do teste

Os principais objetivos do teste de software são:

- Examinar documentos de requisitos, histórias de usuários, projetos e código para assegurar que atendam aos padrões e critérios estabelecidos.
- Identificar erros e problemas no software que precisam ser corrigidos antes do lançamento.
- Assegurar que todas as partes importantes do software sejam testadas adequadamente, abrangendo todos os requisitos e funcionalidades.
- Diminuir a chance de problemas de qualidade no software, identificando e corrigindo defeitos cedo no processo de desenvolvimento.
- Confirmar que o software atende a todos os requisitos especificados, tanto funcionais quanto não funcionais.
- Assegurar que o software cumpra todas as obrigações contratuais, legais e normativas relevantes.
- Prover dados detalhados sobre a qualidade e o estado do software, auxiliando os stakeholders na tomada de decisões fundamentadas sobre o projeto.

- Aumentar a confiança dos stakeholders na qualidade e confiabilidade do software por meio de teste;
- Confirmar que o software está completo e funciona de acordo com as expectativas e necessidades dos stakeholders, garantindo sua satisfação.

É importante ressaltar que os objetivos variam de acordo com o contexto, o que pode se referir ao produto de trabalho que está sendo validado, os riscos, o nível de teste, o ciclo de vida de desenvolvimento de software que está sendo seguido e aspectos relacionados ao contexto do negócio, como por exemplo, estrutura corporativa, área de atuação, considerações competitivas ou tempo de comercialização.

3.2.1.2 Os sete princípios do teste

Os sete princípios de teste são diretrizes fundamentais que ajudam a realizar testes de software de maneira eficaz e eficiente. Eles foram desenvolvidos para melhorar a qualidade do software e garantir que os testes sejam realizados de forma sistemática e com propósito claro

1. Teste mostra a presença de defeitos, não a ausência**:** Os testes podem revelar a presença de defeitos, porém, mesmo após testes extensivos, não é possível garantir que o software esteja completamente livre de defeitos. Testar pode reduzir a chance de defeitos, mas nunca provar a total ausência deles.
2. Testes exaustivos são impossíveis: É inviável testar tudo (todas as combinações de entradas e condições de execução). Ao invés disso, é mais vantajoso utilizar métodos de amostragem e análise de risco afim de concentrar os esforços de teste em áreas mais críticas.
3. Testes antecipados: A atividade de teste deve iniciar o mais cedo possível no ciclo de vida do desenvolvimento e deve estar focada nos objetivos definidos. Quanto antes identificar os defeitos, mais fácil e menos custoso será para corrigi-los.
4. Agrupamento de defeitos: Geralmente, a maior parte dos defeitos encontra-se em um pequeno número de módulos do sistema. Tal fenômeno é conhecido como "Princípio de Pareto", que também pode ser chamado de "Regra 80/20", na qual 80% dos problemas se concentram em apenas 20% dos módulos.

5. Degradação dos testes: Caso os mesmos testes sejam executados continuamente, de maneira repetitiva, eventualmente esses testes deixarão de encontrar novos defeitos. Para manter a eficácia dos testes, é necessário revisar e modificar regularmente os casos de teste, e adicionar novos para cobrir diferentes partes do software ou funcionalidades.
6. Testes dependem do contexto: Diferentes tipos de software requerem diferentes abordagens de teste. Por exemplo, software crítico para segurança é testado de maneira diferente de um aplicativo de e-commerce. Adaptar a estratégia de teste ao contexto específico do projeto é crucial.
7. Ausência de erros é uma ilusão: De nada adianta encontrar e corrigir defeitos se o sistema construído não atender às necessidades e expectativas dos usuários. Testes devem estar alinhados com os requisitos de negócio e necessidades dos usuários para garantir que o software entregue seja funcionalmente útil e de qualidade.

3.2.1.3 Processos de Teste

Embora não exista uma cartilha para testar todos os softwares, o processo de testes apresenta alguns grupos de atividades que são comuns para realizar as validações no sistema. De certa maneira, ao analisar os grupos de atividades, pode-se ter a impressão de haver uma sequência lógica a ser seguida, entretanto essas atividades podem ser realizadas de maneira iterativa ou paralela. Além disso é normal que essas atividades precisem ser adaptadas de acordo com o projeto.

- O planejamento de testes envolve definir a estratégia e a abordagem para as atividades de teste, o que inclui a definição do escopo, objetivos, critérios de entrada e saída, recursos necessários, cronograma e orçamentos. Além disso é nessa atividade que os riscos associados ao teste são identificados e estratégias de mitigação são propostas.
- O Monitoramento e Controle de Teste consistem em acompanhar o progresso das atividades de teste em relação ao plano, usando métricas e relatórios de *status* para tomar ações corretivas conforme necessário, assegurando que os objetivos de teste sejam cumpridos.
- A análise de teste é a etapa de revisão de artefatos de teste para identificar e priorizar condições de teste baseadas em risco e importância, determinando a

cobertura necessária para abordar adequadamente as áreas mais críticas do software.

- A modelagem de teste significa criar e aprimorar casos de teste com base nas condições identificadas, definindo entradas, ações, saídas esperadas e critérios de aceitação; utilizando técnicas de modelagem para uma representação clara dos casos de teste.
- A implementação do teste inclui a criação de procedimentos de teste, scripts de automação, configuração do ambiente de teste e preparação de dados, garantindo que todas as ferramentas e recursos estejam disponíveis para a execução dos testes.
- A execução do teste envolve realizar os casos de teste, registrar os resultados, reportar defeitos e reexecutar testes após correções, assegurando que os defeitos sejam resolvidos adequadamente.
- As atividades de conclusão de teste incluem a coleta e análise dos dados de teste, documentação dos resultados, preparação de relatórios finais, condução de retrospectivas para identificar lições aprendidas e melhorias, e arquivamento de todos os artefatos de teste para referência futura.

3.2.1.4 Níveis de Teste

O Teste de Componente, ou teste unitário, foca em testar componentes individuais do software (como funções, classes ou módulos) de maneira isolada. Geralmente, são utilizados frameworks de teste de unidade e são realizados pelos próprios desenvolvedores.

No nível de teste integrado, as validações estão voltadas para a interação entre os componentes, verificando se eles se comunicam e funcionam em harmonia para atender aos objetivos do sistema. Existem dois tipos principais de testes de integração:

No Teste de Integração de Componentes o foco principal está nas interfaces entre os componentes, garantindo que a troca de dados e informações aconteça de forma precisa e sem falhas. Diferentes abordagens podem ser utilizadas, como *bottom-up* (integração gradual, começando pelos componentes menores), *top-down* (integração a partir do componente principal) ou *big-bang* (integração de todos os componentes de uma vez).

O Teste de Integração de Sistema possui escopo ampliado e concentra-se no teste das interfaces do sistema e validando a integração com outros sistemas e serviços externos. Para realizar esses testes, são necessários ambientes de teste adequados, de preferência semelhantes ao ambiente operacional.

O Teste de Sistema foca no comportamento geral do sistema, avaliando se ele atende aos requisitos funcionais e não funcionais especificados. O teste funcional avalia se o sistema realiza as tarefas de ponta a ponta conforme o esperado, enquanto o teste não funcional avalia outras características como usabilidade, segurança e performance. Para valida alguns aspectos no que diz respeito a requisitos não funcionais, o ideal é que os testes sejam realizados em um sistema completo configurado com o ambiente de teste e que essa etapa seja realizada por uma equipe independente.

O Teste de Aceite tem por objetivo validar que o sistema está pronto para ser implantado e que atende às necessidades dos usuários. Por meio de diferentes tipos de testes de aceite, como por exemplo Teste de Aceite do Usuário (UAT), Teste de Aceite Operacional, Teste Alfa e Teste Beta, os stakeholders e os usuários finais avaliam o sistema em cenários reais de uso. Essa etapa crucial garante que o software atenda às expectativas do negócio e esteja pronto para a entrega final.

Para evitar que as atividades de teste se sobreponham, existe uma diferenciação dos níveis de teste, que leva em consideração diversos aspectos do teste de software, como por exemplo: o objeto de teste, os objetivos, a base de teste, responsabilidades etc.

3.2.1.5 Tipos de Teste

Dentre os tipos de teste, podemos ainda separá-los em duas categorias: os que são feitos durante o desenvolvimento e os que são feitos após a disponibilização do software (seja o sistema em si ou a entrega de uma nova funcionalidade).

Os testes funcionais e não funcionais geralmente são executados após o desenvolvimento do software, ou após a disponibilização de uma entrega iterativa ao sistema, assim como os testes de mudança (regressivo e de confirmação) e os testes A/B que necessitam de alguma versão do sistema disponibilizada. Esses testes executam em uma abordagem “caixa-preta” na qual não há a necessidade de acesso ao código do software.

Os testes durante o desenvolvimento possuem uma abordagem ao estilo “caixa-branca” e validam aspectos do código do sistema. O teste mais comum nessa abordagem é o teste unitário.

3.2.1.5.1 Testes Funcionais

Os testes funcionais têm como objetivo validar se os recursos do aplicativo funcionam de acordo com os requisitos estabelecidos em etapas anteriores ao desenvolvimento. Esse tipo de teste pode ser executado no modelo conhecido como caixa-preta (em que os testes são executados em uma aplicação disponibilizada, mas sem acesso ao código fonte), também podendo ser executado para validar componentes ou funcionalidades específicas. Como esse tipo de teste valida se a funcionalidade do aplicativo está de acordo com o que foi planejado anteriormente, os cenários de teste funcional são criados a partir da documentação do projeto, tendo como base principalmente os requisitos funcionais, fluxos de negócio e critérios de aceite.

3.2.1.5.2 Testes não funcionais

Os testes não funcionais validam o comportamento do sistema e diferentes aspectos. Esses testes são executados no final do ciclo de desenvolvimento de software e usam como base os casos de teste funcionais, porém visando avaliar se os requisitos funcionais estão sendo atendidos. O teste não funcional valida as seguintes características:

- Eficiência de Performance: responsável por validar que o sistema funciona de acordo com o esperado em relação ao desempenho e se funciona bem até mesmo sob stress.
- Compatibilidade: que valida o comportamento, por exemplo, em diferentes aparelhos ou em diferentes sistemas operacionais.
- Usabilidade: Se o sistema é fácil e intuitivo de ser usado.
- Confiabilidade: capacidade do sistema de resistir e se recuperar a falhas.
- Segurança: como o sistema se comporta em casos de ataque, se atende a requisitos da LGPD

- Capacidade de manutenção: Se o código está bem documentado e se é organizado, modular e testável.
- Portabilidade: válida se o software seja executado em diferentes plataformas de hardware e software.

Alguns aspectos validados nos testes não funcionais (ex. segurança), são de suma importância e, caso não atendidos, podem oferecer um alto risco ao projeto. Assim sendo, uma boa estratégia é fazer validações durante o desenvolvimento do software, em diferentes níveis de teste.

3.2.1.5.3 Testes de caixa-branca

Os testes de caixa branca são realizados principalmente durante o desenvolvimento de software. A principal premissa desse tipo de teste é ter acesso ao código-fonte do software. Esses testes focam em verificar a estrutura interna do software, incluindo módulos, métodos e classes, de maneira técnica e lógica.

A eficácia desses testes pode ser avaliada por meio de ferramentas voltadas para o desenvolvimento, sendo o SonarQube a mais conhecida. Esta ferramenta mapeia toda a aplicação, apontando não apenas a cobertura dos testes unitários, mas também sugerindo melhorias de código, identificando possíveis vulnerabilidades e outras áreas de aperfeiçoamento.

Para a modelagem desses testes, podem ser considerados tanto os requisitos funcionais quanto o mapeamento de métodos. Na execução, além dos testes unitários, podem ser realizados testes durante o *code review* (processo de revisão de código, geralmente realizado por um membro mais experiente da equipe) em um *pull request* (processo onde se solicita a autorização para a fusão de dois arquivos, sendo que um deles geralmente está em uma versão mais atualizada).

3.2.1.5.4 Testes de Confirmação e Regressão

Esses testes são executados quando há alguma mudança no sistema, como novas funcionalidades, melhorias ou a correção de um bug.

Os testes de confirmação validam se defeitos identificados em versões anteriores do software foram resolvidos com sucesso. Após a correção de um bug, esses testes são executados especificamente para garantir que o problema não persista e que a solução funcione como o esperado.

Já os testes regressivos são mais complexos e visam validar se o software como um todo está funcionando após uma mudança, ou seja, os testes validam que até as partes não alteradas do sistema continuam em funcionamento.

3.2.1.5.5 Testes durante o desenvolvimento

A fim de aumentar a qualidade do software e se comprometer a garantir uma alta cobertura de testes, existem metodologias de desenvolvimento que auxiliam na implementação de testes durante o desenvolvimento. Os testes implementados durante o desenvolvimento geralmente são do tipo caixa-branca e são feitos pelos próprios desenvolvedores. Os mais comuns são os testes unitários e testes de integração.

Esse tipo de teste pode ser feito sem metodologia, mas nesse caso é comum que os testes sejam deixados de lado e feitos somente após o desenvolvimento do software ou do método. As metodologias TDD (Test Driven Development) e BDD (BehaviorDriven Development) ajudam a manter os testes em vista e auxiliam no ciclo de desenvolvimento afim de entregar um software de qualidade.

O TDD, em português, Desenvolvimento Orientado a Testes, é uma metodologia de desenvolvimento de software na qual o desenvolvimento das funcionalidades inicia a partir da escrita de testes.

Essas práticas consistem em um ciclo de 3 estágio:

1. Primeiro, antes de escrever qualquer código funcional, o desenvolvedor escreve um teste automatizado que descreve uma nova funcionalidade ou melhoria. Quando executado este teste deve falhar, afinal a funcionalidade ainda não foi implementada.
2. Com a falha do teste, o desenvolvedor então escreve o código mínimo necessário para fazê-lo passar. A ideia é implementar apenas o suficiente para que o teste falho agora passe, sem se preocupar com a perfeição ou otimização do código.
3. Enfim, com o teste passando, o desenvolvedor melhora e refatora o código, limpando duplicações, melhorando a estrutura e a eficiência sem alterar o comportamento externo. A refatoração deve manter todos os testes passando.

Este ciclo é repetido para cada nova funcionalidade ou melhoria, promovendo um desenvolvimento incremental e garantindo que o software seja continuamente testado e refatorado para manter alta qualidade e facilidade de manutenção.

Já o BDD, em português Desenvolvimento Orientado a Comportamento, surgiu como uma evolução do TDD e consiste em uma técnica de desenvolvimento ágil que tem como objetivo integrar as regras de negócios com linguagem de programação e foca no comportamento do software.

No BDD, assim como no TDD, os testes ainda são escritos antes do código, porém tem um foco maior na linguagem e nas interações usadas no processo de desenvolvimento. Essa metodologia propõe o uso da combinação da linguagem nativa com a linguagem ubíqua, o que permite que os casos de teste sejam escritos com foco em porque o código deve ser criado e possibilita uma melhor comunicação entre as equipes de desenvolvimento e de teste.

Os testes escritos com o BDD são criados a partir das estórias de usuário ou especificações definidas pelo cliente na etapa de elicitação de requisitos. Existem diversas ferramentas que podem ser utilizadas nesse tipo de escrita, algumas permitem até automatizar os cenários de teste a partir dessa escrita.

No desenvolvimento ágil a escrita das histórias costuma ter uma estrutura básica que ajudam a identificar as necessidades do usuário e definir a finalidade da funcionalidade a ser implementada. A proposta é basicamente definir um perfil, uma ação e o objetivo.

Tabela 1 - Exemplo de guia para escrita de história de usuário

Funcionalidade	**Título**
Como	Papel/Perfil
Posso	Ação
Para	Benefício/Razão

Fonte: CAROLI, Paulo (2023)

Após definir a história de usuário, é necessário criar os cenários de teste, que funcionam como se fossem critérios de aceite, que validam se o que foi desenvolvido está de acordo com o que foi requisitado.

Tabela 2 - Exemplo de modelo para escrita de cenário de teste

Cenário	**Título**
Dado	Pré-condição
Quando	Ação
Então	Resultado

Fonte: CAROLI, Paulo

Além de promover a comunicação e mantê-la ativa, os artefatos do BDD podem servir de documentação das funcionalidades desenvolvidas. Assim sendo, a documentação acaba sendo feita de maneira orgânica.

4 METODOLOGIAS DE DESENVOLVIMENTO E INTEGRAÇÃO CONTÍNUA

No desenvolvimento de software existem metodologias que servem de referência para todo o processo, desde a ideia inicial até a entrega do produto. A escolha de uma metodologia e um processo bem definido são cruciais para a qualidade do projeto, durante e após o desenvolvimento.

Em meados dos anos 70 o modelo cascata (ou *waterfall*) surgiu como uma das primeiras metodologias de desenvolvimento de software conhecidas. O modelo foi proposto como uma forma de tornar os projetos mais previsíveis e se baseia em uma hierarquia de etapas sequenciais.

As etapas que marcam o modelo cascata são: levantamento de requisitos, planejamento, modelagem, desenvolvimento, testes e implantação.

Essas etapas são executadas na ordem descrita e, com isso, podemos ver um dos grandes problemas do modelo cascata atualmente, já que as etapas de teste e implantação estão apenas no final. Somente no final do projeto que problemas como bugs são identificados e, como os clientes só são envolvidos na etapa de levantamento de requisitos, é muito provável que o resultado não seja satisfatório e não atenda aos requisitos dos usuários pois não há participação dos clientes no decorrer do processo

O modelo cascata possui mais algumas desvantagens, visto que exige que cada etapa do processo seja documentada de maneira extensa e há grande rigidez nas etapas, o que torna o processo burocrático e lento. A necessidade de gerar documentos sobre cada etapa proporciona um gargalo para a produtividade.

Uma das características marcantes do modelo cascata é sua rigidez, sendo que o que foi definido, deverá ser entregue, sem revisitar etapas e realizar alterações conforme o processo. Isso se deve ao fato de que na época em que o modelo foi criado, ele teve como base principalmente conceitos de outras áreas de engenharia, como civil. Assim sendo, é difícil utilizar o modelo cascata atualmente, visto que hoje os softwares necessitam se adaptar rapidamente e apresentar constante evolução.

4.1 Metodologias Ágeis

A fim de atender a projetos em um mundo dinâmico, em que o mercado pode mudar a todo instante, surgiram as metodologias de desenvolvimento de software ágeis, que aproximam os clientes dos desenvolvimentos e buscam melhorias a cada etapa do processo.

Nas metodologias ágeis há ênfase na colaboração, flexibilidade e melhoria contínua. Nessa maneira mais moderna de trabalhar os projetos são desenvolvidos em pequenos incrementos entregáveis em curtos períodos, chamados de sprints ou iterações.

4.1.1 Scrum

Scrum é uma metodologia de gerenciamento de projetos que foi criada por Mike Beedle, Ken Schwaber e Jeff Sutherland. Segundo o livro Scrum: The Art of Doing Twice the Work in Half the Time (“Scrum: a arte de fazer o dobro do trabalho na metade do tempo”), o framework tem como objetivo principal tornar equipes mais produtivas, permitindo a entrega de produtos complexos de maneira iterativa e incremental.

O Scrum possui uma divisão em ciclos, que são chamados de sprint e o tempo de cada sprint pode variar de acordo com o projeto, porém o mais comum são ciclos de duas ou três semanas.

No início de um projeto Scrum, é criado um *Product Backlog*, que é como uma lista de tudo que é necessário para o produto. O Backlog é constantemente atualizado conforme o avanço do projeto. É durante a sprint que as equipes de desenvolvimento se concentram em completar um conjunto específico de tarefas selecionadas do Backlog.

Nessa metodologia existem eventos que são realizados a cada ciclo, são elas: *Plannig*, *Daily*, *Review* e Retrospectiva.

- Planning: Reunião inicial de planejamento para definir o que será feito na sprint.
- Daily Scrum: Reunião diária de curta duração para sincronizar as atividades, ajustar o plano conforme necessário e sinalizar problemas e impedimentos.
- Review: Reunião para inspecionar o incremento e adaptar o backlog do produto.
- Retrospectiva: Reunião para refletir sobre a sprint e identificar melhorias para o próximo ciclo.

Existem também alguns papeis específicos do time Scrum, cada um deles tem papel crucial para as entregas.

- Product Owner: Responsável por maximizar o valor do produto e gerenciar o backlog do produto. O Product Owner (ou simplesmente PO) é a voz do cliente no projeto. O PO define e prioriza os itens do backlog que estarão em uma sprint.
- Scrum Master: Facilita o processo Scrum e ajuda a remover impedimentos para a equipe.
- Equipe de Desenvolvimento: Grupo multifuncional que trabalha para entregar incrementos do produto.

4.1.2 Kanban

O Kanban é outra metodologia ágil que utiliza um sistema visual para gerenciar o trabalho conforme o progresso. Tem origem japonesa, surgindo na Toyota como uma forma de melhorar a eficiência na fabricação de automóveis e foi adaptado para o desenvolvimento de software e outros tipos de trabalho.

Esta metodologia possui uma estrutura muito simples e eficiente:

- Quadro kanban: No quadro kanban (ou Kanban Board) o fluxo de trabalho é definido de maneira visual, de maneira que é possível acompanhar o trabalho em progresso. O quadro é divido em colunas que representam diferentes estados do trabalho. Um exemplo simples, mas bastante

comum, para as colunas é o seguinte: “A fazer”, “Em progresso”, “Concluído”.

- Cartões Kanban: São os itens de trabalho. Esses cartões (cards) são posicionados no quadro e movimentados conforme o avanço do trabalho.
- Limites de Trabalho em Progresso (Work-In-Porgress ou WIP): Restrições no número de itens que podem estar em progresso simultaneamente, ajudando a identificar gargalos e melhorar o fluxo.

A ênfase no Kanban está na melhoria contínua, flexibilidade e eficiência operacional, permitindo que as equipes ajustem rapidamente seu processo de trabalho em resposta às mudanças nas prioridades e demandas.

4.1.3 Devsecops

DevSecOps é uma prática que integra segurança ao longo do ciclo de vida de desenvolvimento de software (SDLC), combinando desenvolvimento (Dev), operações (Ops) e segurança (Sec). O objetivo do DevSecOps é criar um ambiente de desenvolvimento colaborativo e ágil que incorpora a segurança desde o início, em vez de tratá-la como uma preocupação posterior.

Principais Componentes do DevSecOps:

- Integração Contínua/Entrega Contínua (CI/CD): Automatiza a construção, teste e implantação do software, incluindo verificações de segurança em cada etapa.
- Segurança Automatizada: Ferramentas automatizadas para análise de código estático, análise de vulnerabilidades, testes de segurança dinâmicos e verificação de conformidade.
- Cultura de Segurança: Promove a colaboração entre desenvolvedores, operações e profissionais de segurança para garantir que a segurança seja uma responsabilidade compartilhada.
- Feedback Contínuo: A segurança é constantemente monitorada e avaliada, permitindo respostas rápidas a novas ameaças.

DevSecOps garante que a segurança seja uma parte integral do processo de desenvolvimento, proporcionando uma entrega mais rápida de software seguro e confiável.

Essas metodologias, Scrum, Kanban e DevSecOps, representam abordagens modernas e eficientes para o desenvolvimento de software, cada uma com suas próprias características e benefícios sendo que ambas podem ser utilizadas em conjunto. É possível que, por exemplo, Scrum seja utilizada para definir todo o processo de desenvolvimento, Kanban para gerenciar tarefas e atividades do dia a dia e ainda assim implementar a cultura de DevSecOps. Juntas, essas metodologias ajudam as equipes a enfrentar os desafios de um mercado dinâmico, onde a capacidade de adaptação, a entrega contínua de valor e a segurança são cruciais para o sucesso.

5 FERRAMENTAS PARA TESTE AUTOMATIZADO MOBILE

Assim como existem várias ferramentas e linguagens de programação para desenvolver os aplicativos, também existe toda uma gama de ferramentas que podem ser utilizadas para automatizar os testes.

A escolha de qual ferramenta utilizar pode variar de acordo com o contexto do teste, levando em consideração principalmente o tipo de teste, o momento em que o teste será executado e algumas vezes as ferramentas em que o aplicativo foi desenvolvido.

5.1 Testes Unitários

O teste unitário, ou teste de componente, valida o comportamento de um componente específico do código. Geralmente esses testes são executados em esteiras de integração contínua e são feitos pelos próprios desenvolvedores antes de subir o código para algum ambiente de desenvolvimento, homologação ou produção.

As ferramentas utilizadas no desenvolvimento geralmente possuem a capacidade de desenvolver e executar esse tipo de teste, mas é muito comum a utilização de bibliotecas com essa finalidade.

5.1.1 Testes unitários no Android com JUnit

O Junit é um framework de testes unitários, muito utilizado no desenvolvimento móvel com linguagens Java e Kotlin. Ele fornece uma estrutura robusta para criar e executar testes automatizados, garantindo a confiabilidade e robustez do seu código.

O framework nos permite escrever testes com clareza. É possível criar classes de teste com o método @Test para cada cenário que deseja validar. O JUnit possui uma série de asserções para fazer as validações do teste, como *assertEquals* e *assertTrue* para verificar o comportamento esperado.

O JUnit possui muitas funcionalidades interessantes, entre elas, é destacável sua capacidade de testes em Mock, permitindo simular dependências externas para isolar o código em teste, a possibilidade de executar testes em diferentes ambientes, como emuladores ou o dispositivo real e os relatórios detalhados e abrangentes, contendo falhas e tempos de execução.

O Android Studio oferece suporte nativo ao JUnit, o que simplifica a criação e execução de testes.

5.1.2 Testes unitários no iOS com XCTest

O XCTest é o framework nativo do Xcode para testes unitários em iOS. Com ele é possível criar testes automatizados que verificam tanto o comportamento individual de unidades do código (como classes, métodos e funções) quanto o comportamento da interface do aplicativo.

Assim como no JUnit, o XCTest também permite a criação de classes de teste para cada funcionalidade que deseja testar, porém no framework de testes para iOS é por meio do método testSomething. Já as asserções podem ser feitas com os métodos *XCTAssertEqual* e *XCTAssertTrue*.

O XCTest também permite que testes *Mock* sejam feitos, porém também permite que testes de interface validem a UI do aplicativo. Além disso é possível fazer testes de performance com o framework.

5.1.3 Testes unitários em apps híbridos com Jest e Mocha

No desenvolvimento em JavaScript, Jest e Mocha são as ferramentas mais comuns para desenvolver testes unitários, assim sendo, são essas mesmas

ferramentas utilizadas para criar os testes unitários de aplicativos desenvolvidos com React Native.

Diferente do que ocorre no desenvolvimento nativo, que o Android Studio e o XCode possuem suporte nativo as ferramentas de teste unitário, para utilizar Jest ou Mocha é necessário importar as bibliotecas como dependências de desenvolvimento para poder utilizar suas funcionalidades.

Assim como no desenvolvimento nativo, essas ferramentas possibilitam realizar testes em *mock*, possuem diversas asserções e relatórios claros que reportam diversas informações importantes, como a ocorrência de erros.

5.2 Testes Funcionais

O Teste funcional valida o comportamento de alguma funcionalidade do sistema, como por exemplo o cadastro de um usuário, ou a alteração do perfil. Esse tipo de teste necessita que alguma versão do aplicativo esteja publicada. Comumente nos projetos que aderem ao DevSecOps existem os ambientes de Desenvolvimento, Homologação e Produção, dessa maneira, é possível ter um aplicativo publicado em cada ambiente e o Teste unitário pode ser executado nesse aplicativo disponibilizado.

Esse tipo de teste pode ser feito de modo manual, mas automatizá-los é vantajoso, visto que reduz as chances de falha humana e facilitam um cenário em que haja a necessidade de executar um teste de regressão, permitindo que vários cenários sejam executados rapidamente em sequência.

Geralmente esses testes são feitos por pessoas que não atuaram no desenvolvimento, a fim de evitar o viés de confirmação. Além disso esses testes não dependem de maneira tão direta das ferramentas que foram utilizadas no desenvolvimento, sendo possível que um app tenha sido desenvolvido com React Native utilizando Javascript e o teste funcional automatizado seja desenvolvido com Robot utilizando python. Uma outra vantagem desse tipo de teste é escrevê-los com a linguagem Gherkin, pois o caso de teste fica alinhado com os critérios de aceite das histórias de usuário e são de fácil interpretação. Muitas linguagens de programação aceitam a linguagem Gherkin por meio de frameworks, como é o caso do Python com Robot, ou Java com Cucumber e Selenium.

5.2.1 Nativo Android com Espresso

O Espresso é uma ferramenta poderosa para testes de interface no desenvolvimento Android, integrada diretamente no Android Studio, o que facilita sua configuração e uso. Ele permite testar interações do usuário de forma automatizada e eficiente, garantindo que a interface do aplicativo funcione conforme esperado.

Para começar a usar o Espresso, é necessário adicionar a biblioteca ao seu projeto. Uma das vantagens do Espresso é sua capacidade de sincronização automática com a interface do usuário e ações de teste, facilitando a validação de ações assíncronas. Ele também oferece suporte para a validação de *intents* e pode ser utilizado para testar componentes de *WebView*.

A estrutura básica de um teste com Espresso inclui localizar uma view específica usando *ViewMatchers*, realizar uma ação nessa *view* com *ViewActions*, e então verificar o resultado com *ViewAssertions*.

O Espresso também suporta testes em dispositivos reais e emuladores, permitindo executar testes de interface em diferentes cenários.

5.2.2 Nativo iOS com (XCUITest/XCTest-WebDriver)

No iOS, o XCTest é o framework padrão para testes, e inclui o módulo XCUITest, que permite a automação de testes de interface do usuário. O XCUITest é utilizado para verificar a funcionalidade de elementos da interface, garantindo que o aplicativo responda corretamente às interações do usuário.

Para configurar testes com XCUITest, você deve criar uma nova target de teste no Xcode e adicionar os testes necessários. As classes de teste utilizam métodos como *testExample*() para definir os casos de teste, e as verificações são realizadas com asserções como *XCTAssertTrue* e XCTAssertEqual.

O XCUITest permite não apenas testar a interface, mas também realizar testes de performance e interagir com a interface de forma semelhante a um usuário real, incluindo gestos complexos e interações com múltiplos elementos.

5.2.3 Framwork Híbrido Appium

Para aplicativos híbridos, que utilizam tecnologias como React Native, o Appium é uma das ferramentas mais populares para automação de testes. Appium é uma ferramenta de código aberto que permite escrever testes em várias linguagens de programação, como Java, Python, e JavaScript, e pode ser usado tanto para Android quanto para iOS.

A configuração do Appium envolve iniciar um servidor Appium, configurar o driver para o dispositivo ou emulador em teste, e então escrever os scripts de teste.

5.3 Outros Testes

5.3.1 Testes A/B (Firebase e versões diferentes de um mesmo app)

O teste A/B é uma atividade comum no marketing de aplicativos móveis, consiste em um experimento controlado que divide de maneira aleatória uma amostra de usuários em dois grupos e cada grupo é exposto a uma versão diferente do aplicativo. Nesse tipo de experimento o objetivo é validar se a alteração de cores em botões pode melhorar a taxa de conversão, ou identificar por quais motivos os usuários podem abandonar a etapa de cadastro ou o aplicativo, e outras atividades semelhantes.

Uma ferramenta muito popular no desenvolvimento *mobile* é o Firebase do Google. Se trata de uma plataforma que oferece vários recursos diferentes, sendo possível gerenciar a infraestrutura do *back-end*, monitorar a estabilidade e o desempenho do aplicativo e realizar testes A/B.

Além de possibilitar o gerenciamento de versões, a plataforma também possui ferramentas de *Analytics*, que podem contribuir para análises realizadas nos testes A/B.

Além do Firebase, existem outras ferramentas como Optmizely e Apptimize que podem ser utilizadas para realizar esse tipo de teste.

5.3.2 Ferramentas de teste web aplicadas ao mobile

Algumas ferramentas de desenvolvimento, como React Native quando usado em conjunto com o framework Expo por exemplo, permitem que o aparelho seja simulado em algum navegador da web, dispensando assim a necessidade de emular um dispositivo ou de usar o dispositivo físico conectado ao computador em que se está desenvolvendo.

O Expo é uma ferramenta de código aberto que oferece um ecossistema completo de ferramentas para desenvolver, atualizar e monitorar aplicativos móveis. Embora o expo seja completo e forneça, diversos recursos para o desenvolvimento móvel, para este ponto do trabalho vale destacar a sua integração facilitada com um dispositivo físico para acompanhar como o aplicativo está ficando e principalmente a possibilidade de utilizá-lo para visualizar o aplicativo direto em um navegador web,

utilizando as ferramentas de inspeção e a Device Toolbar do próprio navegador, incluindo até a capacidade de atualizar a visualização sempre que uma alteração no código for feita e em seguida salva.

Dessa maneira, é possível analisar os elementos de tela do aplicativo de modo semelhante a um site da web com as ferramentas de inspeção do próprio navegador.

Isso permite que ferramentas de automação web também possa ser usadas para automatizar os testes funcionais dos aplicativos. Sendo assim, Selenium, Robot, Cypress, Puppeter entre outras ferramentas que podem ser utilizadas. Essas ferramentas são muito versáteis e compatíveis com diversas linguagens, não se restringindo a linguagem na qual o aplicativo foi desenvolvido.

6 CONCLUSÃO

Com o avanço tecnológico, o uso de dispositivos móveis se tornou parte do nosso dia a dia. Com uma variedade tão grande de dispositivos e aplicativos disponíveis no mercado se faz necessário aplicar estratégias de teste automatizado, a fim de garantir a qualidade e evitar problemas como erros e falhas que podem prejudicar tanto empresas quanto os usuários.

Tendo como base a bibliografia usada neste trabalho e ao estudar as ferramentas e tecnologias utilizadas no desenvolvimento mobile, é possível constatar que Java e Kotlin são as principais ferramentas de desenvolvimento Android, sendo que Kotlin é recomendado pela Google, empresa dona do Sistema Operacional, desde 2017. Já quando se trata do iOS, a linguagem usada para desenvolver os aplicativos é a Swift. Porém, isso diz respeito a abordagem nativa, se tratando da abordagem hibrida, na qual aplicativos são desenvolvidos para ambos os sistemas operacionais com uma única base de código, temos como principais ferramentas Flutter e React Native. Ao comparar as abordagens de desenvolvimento nativa e hibrida o desenvolvimento nativo proporciona maior desempenho e integração com recursos do dispositivo, enquanto o desenvolvimento híbrido oferece maior flexibilidade e redução de custos.

Além disso, quanto ao estudo da área de qualidade de software, é fundamental implementar estratégias para testar os aplicativos a fim de assegurar a qualidade e evitar problemas que podem atingir tanto empresas quanto usuários. É por meio do uso de práticas e estudos da área da qualidade que é possível entregar um software seguro, com boa experiencia de usuário e boa performance. No que diz respeito a automação dos testes, a escolha da abordagem de desenvolvimento, seja nativa ou hibrida, o nível do teste e o tipo de teste influenciam diretamente nas ferramentas utilizadas para automação. Todavia, automatizar os testes é essencial para assegurar a qualidade, minimizando problemas da abordagem manual e reduzindo o tempo na execução dos cenários de teste, o que implica em uma redução nos custos também.

Ainda sobre o estudo do material referente a metodologias ágeis e práticas de integração contínua, como Scrum e DevSecOps, ao adotar essas ferramentas é possível obter uma melhora significativa no processo de desenvolvimento possibilitando entregas de valor, proximidade do cliente e aprimorando a

colaboração das equipes. Além de aumentar a eficiência das equipes e permitir acompanhar as mudanças do mercado durante o desenvolvimento com metodologias ágeis, a adoção da cultura DevSecOps da ênfase para que a segurança e a qualidade sejam incorporadas desde o início do desenvolvimento.

Enfim, com base na bibliografia estudada, este trabalho demonstra que a implementação de estratégias de teste automatizado, aliada a uma abordagem de desenvolvimento e metodologias ágeis, é essencial para o sucesso dos aplicativos móveis. A qualidade do software, a satisfação do usuário e a competitividade no mercado dependem diretamente da eficácia dessas práticas, que devem ser continuamente aprimoradas para acompanhar as rápidas mudanças e demandas do setor de tecnologia.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ALICE, Akemi; FERNANDES, Arthur. Linguagem Kotlin: o que é e um Guia para aprender. 2023. Disponível em: <https://www.alura.com.br/artigos/kotlin>. Acesso em: 23 mar. 2024.

Apple. App Review Guidelines, 2024. Disponível em: <https://developer.apple.com/app-store/review/guidelines/>. Acesso em: 26 mar. 2024.

Apple. Designing for iOS.2024. Disponível em: <https://developer.apple.com/design/human-interface-guidelines/designing-for-ios>. Acesso em: 27 mar. 2024.

BELANDI, Caio. 161,6 milhões de pessoas com 10 anos ou mais de idade utilizaram a Internet no país, em 2022. 2023. Disponível em: <https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/38307-161-6-milhoes-de-pessoas-com-10-anos-ou-mais-de-idade-utilizaram-a-internet-no-pais-em-2022>. Acesso em 25 mar. 2024.

BUSCAPÉ. iPhone vs Android: o que é melhor? 2022. Disponível em: <https://www.buscape.com.br/celular/conteudo/o-que-e-melhor-iphone-ou-android>. Acesso em: 24 mar. 2024.

CAROLI, Paulo. Histórias do usuário e a construção de produtos de sucesso. 2023. Disponível em: <https://caroli.org/historias-do-usuario-e-a-construcao-de-produtos-de-sucesso/>. Acesso em: 11 abr. 2024.

CARVALHO, Ingrid. Quais são as funções do Teste Alfa, Beta, e Regressão? 2019. Disponível em: <https://medium.com/@ingrid.carvalho.mo/quais-são-as-funções-do-teste-alfa-beta-e-regressão-b0db1c3bf5d0>. Acesso em: 08 abr. 2024.

CARVALHO, Ingrid. Você sabe o que é Teste Caixa Branca e Teste Caixa Preta? 2019. Disponível em: <https://medium.com/@ingrid.carvalho.mo/voc%C3%AA-sabe-o-que-%C3%A9-teste-caixa-branca-e-teste-caixa-preta-9a2d08fe9d0c>. Acesso em: 05 abr. 2024.

CERTIFIED Tester Foundation Level Syllabus, CTFL. BSTQB, Brasil, 2023. Disponível em: <https://bcr.bstqb.org.br/docs/syllabus_ctfl_4.0br.pdf>. Acesso em: 11 abr. 2024.

FELICIANO, Beatriz. O que é TDD? 2023. Disponível em: <https://dev.to/womakerscode/o-que-e-tdd-4b5f>. Acesso em: 08 abr. 2024.
FIA BUSINESS SCHOOL. Scrum: o que é e como aplicar a metodologia ágil para gestão? 2024. Disponível em: <https://fia.com.br/blog/scrum/>. Acesso em: 12 abr. 2024.

FONSECA, Heloise. GHERKIN: INTRODUZINDO SEUS CONCEITOS E BENEFÍCIOS. 2020. Disponível em: <https://blog.onedaytesting.com.br/gherkin/>. Acesso em: 10 abr. 2024.

GIDEON, Oyindamola Olarewaju. Introduction to Unit Testing in Android Kotlin. 2023. Disponível em: <https://medium.com/@deonolarewaju/introduction-to-unit-testing-in-android-kotlin-4331eb2366a9>. Acesso em: 22 mar. 2024.

GOOGLE DEVELOPERS. Fazer o download e instalar o Android Studio. 2024. Disponível em: <https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio?hl=pt-br#0>. Acesso em: 23 mar. 2024.

GOOGLE DEVELOPERS. Google I/O 2018: Novidades do Android. 2018. Disponível em: <https://developers-br.googleblog.com/2018/05/google-io-2018-novidades-do-android.html>. Acesso em: 23 mar. 2024.

GOOGLE. Criar testes de unidade locais. Disponível em: <https://developer.android.com/training/testing/local-tests?hl=pt-br>. Acesso em: 18 mar. 2024.

HAMILTON, Thomas. 10 MELHORES ferramentas e aplicativos de teste móvel. 2024. Disponível em: <https://www.guru99.com/pt/mobile-testing-tools.html>. Acesso em: 06 mar. 2024.

IBGE Educa. INFORMAÇÕES ATUALIZADAS SOBRE TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÃO. 2023. Disponível em: <https://educa.ibge.gov.br/jovens/materias-especiais/21581-informacoes-atualizadas-sobre-tecnologias-da-informacao-e-comunicacao.html>. Acesso em 25 mar. 2024.

IBM. O que é DevSecOps? Disponível em: <https://www.ibm.com/br-pt/topics/devsecops>. Acesso em: 20 abr. 2024.

IBM. O que são hipervisores? Disponível em: <https://www.ibm.com/br-pt/topics/hypervisors>. Acesso em: 26 mar. 2024.

KANAI, Shinji. Getting Started with Espresso: A Complete Guide. 2022. Disponível em: <https://www.headspin.io/blog/getting-started-with-espresso-a-complete-guide>. Acesso em: 24 mar. 2024.

KATALON. Top 10 Automated Mobile Testing Tools in 2024. 2024. Disponível em: <https://katalon.com/resources-center/blog/mobile-testing-tools>. Acesso em: 15 mar. 2024.

Lenildo. Qualidade de Software - Engenharia de Software 29. 2010 Disponível em: < https://devmedia.com.br/qualidade-de-software-engenharia-de-software-29/18209>. Acesso em: 10 abr 2024.

LEUNG, Jane. 7 Melhores Ferramentas de Teste A/B para Aplicativos Móveis. 2023. Disponível em: <https://uxcam.com/br/blog/ferramentas-de-teste-a-b-para-aplicativos-moveis>. Acesso em: 24 abr. 2024.

MAMEDE, Elisabeth. Quais os tipos de teste que devo fazer em aplicações móveis? 2018. Disponível em: <https://medium.com/@elisabethmamede/quais-os-tipos-de-teste-que-devo-fazer-em-aplica%C3%A7%C3%B5es-m%C3%B3veis-abbfb71c9aa1>. Acesso em: 05 abr. 2024.

MICROSOFT. O que é o DevSecOps? Disponível em: <https://www.microsoft.com/pt-br/security/business/security-101/what-is-devsecops>. Acesso em: 20 abr. 2024.

Ministério das Comunicações (MCom). Internet chega a 87,2% dos brasileiros com mais de 10 anos em 2022, revela IBGE. 2023. Disponível em: <https://agenciagov.ebc.com.br/noticias/202311/internet-chega-a-87-2-dos-brasileiros-com-mais-de-10-anos-em-2022-revela-ibge>. Acesso em: 25 mar. 2024.

OPENTEXT. O que é teste funcional? Disponível em: <https://www.opentext.com/pt-br/o-que-e/functional-testing>. Acesso em: 28 mar. 2024.

SAUCELABS. Appium vs. Espresso. 2022. Disponível em: <https://saucelabs.com/resources/blog/getting-started-with-espresso>. Acesso em: 27 mar. 2024.

SAUCELABS. Getting Started with Espresso. 2023. Disponível em: <https://saucelabs.com/resources/blog/getting-started-with-espresso>. Acesso em: 27 mar. 2024.

SILVA, Alan. 5 melhores ferramentas de automação para testar aplicativos Android (Android App Testing Tools). 2019. Disponível em: <https://pt.linkedin.com/pulse/5-melhores-ferramentas-de-automa%C3%A7%C3%A3o-para-testar-android-alan-silva>. Acesso em: 03 mar. 2024.

Teleco. Sistema Operacional Móvel. 2023. Disponível em: <https://www.teleco.com.br/sist_operacional.asp>. Acesso em 23 mar. 2024.

TOTVS. Kanban: conceito, como funciona, vantagens e implementação. 2023. Disponível em: <https://www.totvs.com/blog/negocios/kanban/>. Acesso em: 12 abr. 2024.

TUDO CELULAR. Apple revela que o iOS 16 estava presente em 81% de todos os iPhones em maio de 2023. 2023. Disponível em: <https://www.tudocelular.com/mercado/noticias/n206926/apple-revela-ios-16-presente-81-iphones-maio-2023.html>. Acesso em: 26 mar. 2024.

VASCONCELOS, Matheus de. XCTest em execução. 2020. Disponível em: <https://ma-vasconcelos98.medium.com/xctestalem-8d9407a2a0d1>. Acesso em: 24 mar. 2024.

Wikipédia. Qualidade de software. Disponível em: < https://pt.wikipedia.org/wiki/Qualidade_de_software>. Acesso em: 10 abr 2024.

WINDER, Davey. New Critical Security Warning For iPhone, iPad, Watch, Mac - Attacks Underway. 2023. Disponível em: <https://www.forbes.com/sites/daveywinder/2023/09/23/ios-1701-critical-security-update-warning-for-all-iphone-users/?sh=237397be4e83>. Acesso em: 28 mar. 2024.