Escalabilidade Tecnológica com ênfase em Testes de *Software*

José Antônio Caramuru Miranda Kogure, Wesley Victor Souza da Silva, Djalma Domingos da Silva (Orientador)

e-mail: josemirandak02@gmail.com; wesley.victor@icloud.com; djalma.silva15@fatec.sp.gov.br

Faculdade de Tecnologia de São José do Rio Preto

Resumo: Os testes de *software* são uma parte essencial do processo de desenvolvimento de *software*. Eles são usados para avaliar a qualidade do *software* e garantir que atendam aos requisitos e expectativas dos usuários finais. Considerando que muitas empresas já sofreram prejuízos absurdos em decorrência de erros de *software*, decidiu-se mostrar que os testes de *software* podem ajudar a diminuir danos e custos. Para este estudo, uma pesquisa exploratória foi realizada, foram analisados em livros, *sites* da internet e artigos já publicados a fim de mostrar o quanto a escalabilidade de uma empresa pode aumentar quando são adotados testes de *software*. Os resultados mostram que os testes são um dos processos mais importantes de uma empresa, podendo ser utilizados em todas as etapas do desenvolvimento. Ao elaborar este artigo, conclui-se que para uma empresa crescer e se adaptar ao aumento da demanda sem perder a qualidade do *software*, ela deve ter uma fase de testes bem criteriosa.

Palavras-chave: testes de *software*; escalabilidade.

Abstract: *Software testings is an essential part of the software development process. They are used to assess the quality of software and ensure that they attend requirements and expectations to end users. Considering that many companies have already suffered absurd losses as a result of software errors, it was decided how software testings may help decrease damage and costs. For this study, exploratory research was carried out, they were analyzed in books, internet sites and articles already published in order to show how much the scalability of a company can increase when software tests are adopted. The results display that testing is one of the most important processes of a company, they can be used in all stages of development. Elaborating this paper, it was concluded that for a company to grow and adapt to the increase in demand without losing the quality of product, it must have a testing stage very judicious.*

Keywords: *software testing; scalability.*

1. INTRODUÇÃO

A busca de conhecimento para solucionar impasses pela vida humana sempre foi uma necessidade, erros foram inevitáveis ao longo do avanço tecnológico. Em 1996, a Agência Espacial Europeia (ESA) lançava o seu mais recente foguete, o Ariane 5. No entanto, tudo parecia ocorrer como planejado, porém, 40 segundos após o seu lançamento, o sistema interno da aeronave detectou uma falha, fazendo com que o foguete explodisse no ar, resultando em um dos erros de *softwares* mais caros da história, de aproximadamente 500 milhões de dólares. (É LIVRE E ABERTO, 2019)

Tendo em vista não só esse exemplo, mas muitos outros casos similares, que mostram que um erro de *software* pode resultar em um prejuízo e causar diversos problemas, tanto para os desenvolvedores quanto para os usuários, motivou-se a mostrar que os testes de *software* são fundamentais para garantir a qualidade de um produto de *software* e aumentar a confiabilidade.

O artigo tem como objetivo geral contribuir para os estudos de testes de *software*, fornecendo uma análise atualizada dos principais conceitos, técnicas e abordagens utilizadas nos processos de testes. Especificamente, avaliar o impacto do teste na qualidade do *software* e mostrar também, como esses processos são fundamentais para melhorar a saúde do desenvolvedor.

Desse modo, uma empresa não pode tratar os testes como apenas uma mera formalidade, pois eles se tornaram um processo de extrema importância, por meio de práticas recomendadas e estratégias bem definidas. Os testes vão ajudar a identificar e corrigir ações inesperadas no *software*. Com isso, poderemos ter uma redução de custos, aumentando a eficiência operacional e a empresa, consequentemente, obter uma boa reputação com os clientes, o que pode atrair mais negócios e consequentemente aumentará a sua escalabilidade. (HOSTINGER, 2023)

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Engenharia de Software

A Engenharia de *Software* surgiu como uma disciplina da ciência da computação no final da década de 1960 e início da década de 1970. Naquela época, o desenvolvimento de software estava se tornando cada vez mais complexo e as empresas estavam enfrentando desafios na criação de soluções de alta qualidade e confiabilidade (SMITH, 2010).

Para resolver esses desafios, um grupo de especialistas em computação começou a estudar e pesquisar métodos e técnicas para gerenciar e desenvolver esses projetos de maneira

sistemática e eficiente (JONES, 2008). Em 1968, a OTAN (Organização do Tratado do Atlântico Norte) organizou uma conferência sobre Engenharia de *Software*, que marcou o início formal dessa disciplina (BROWN, 2002).

A partir desse momento, a introdução de novos métodos e técnicas, incluindo o modelo de ciclo de vida do projeto e o processo de levantamento de requisitos, possibilitou a sua progressão na computação (JOHNSON, 1995).

Hoje em dia, a matéria é visionária. A Engenharia de *Software* vai além da criação, manutenção e desenvolvimento de sistemas, atuando principalmente na qualidade do produto e na redução de custo e tempo (GARCIA, 2018). A escalabilidade é uma peça fundamental para os princípios práticos e teóricos dessa área.

2.1.1. Processos

Os processos são fundamentais em todas as áreas, e não seria diferente na projeção de um software.

Capazes de fornecer uma estrutura de atividades definidas, embora certamente menos mecânica e mais dependente de esforço intelectual. Com isso, processos são responsáveis por apoiar toda a equipe, tornando-a mais automatizada nas soluções (JOHNSON, 2014).

2.1.2. Levantamento e Análise de Requisitos

Essa etapa consiste em identificar as necessidades que devem ser atendidas pelo sistema, para que ele possa funcionar de acordo com as expectativas dos usuários e agregar valor aos negócios, obtendo valores significativos **na garantia da qualidade, eficiência e consistência do *software*** (SOMMERVILLE, 2016).

Eles também podem ser adaptados e personalizados para atender às necessidades específicas de cada projeto, garantindo que o processo de desenvolvimento seja adaptado às necessidades do projeto em questão, embora certamente menos mecânico e mais dependente de esforço intelectual (PRESSMAN, 2014).

2.2. Desenvolvimento do Software

O trabalho do desenvolvedor começa ao analisar a estrutura do levantamento de requisitos, sendo responsável pela codificação, testes, implantação e manutenção (MCCONNELL, 2004).

2.2.1. Waterfall

O modelo Waterfall, também conhecido como modelo em cascata, é uma metodologia tradicional de desenvolvimento de software que segue uma abordagem sequencial e linear. Essa abordagem é caracterizada por uma progressão descendente, na qual cada fase do projeto é concluída antes de avançar para a próxima.

As principais etapas do modelo *Waterfall* incluem o **levantamento de requisitos, análise, design, implementação, testes, implantação e manutenção.**

Como pode-se observar abaixo, na representação do Método Cascata *(The Waterfall Method), esses* processos são sequenciais de cima para baixo, onde o processo de **Levamento de Requisitos** *(Requirements)* é definido como o início, essa etapa é importante para a coleta das necessidades do seu cliente. Em seguida, podemos observar o ***design*** *e* a **Implementação** *(Implementation)* que prioriza desenvolver toda a estrutura dos requisitos.

Pelas fases finais, a **Verificação ou Testes** *(Verification or Testing)* e a **Implantação & Manutenção** *(Deployment & Maintenance)* são importantes para conferir se está atendendo o resultado esperado, caso as validações estiverem certas, será lançada na próxima implantação, porém, ao final dos testes, se necessário correções, será destinada à uma série de manutenções.

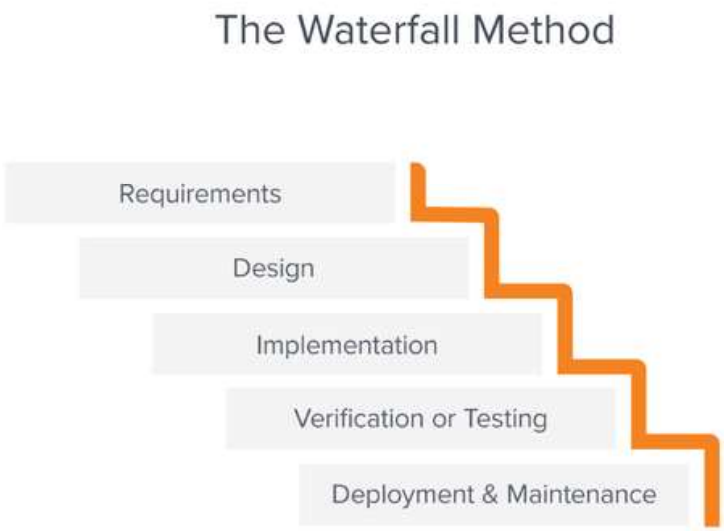

Figura 1: **Waterfall Methodology: A Complete Guide.** (Fonte: Adobe, 2022)

Uma referência importante para esse modelo é o trabalho de Winston W. Royce, em seu artigo seminal "*Managing the Development of Large Software Systems: Concepts and Techniques"* apresentado na Conferência IEEE Wescon em 1970, onde Royce descreveu um modelo de desenvolvimento de software em cascata que envolve etapas sequenciais, desde a definição dos requisitos até a manutenção do sistema. Apesar de algumas críticas posteriores

ao modelo *Waterfall*, o artigo de Royce é considerado uma das primeiras referências sobre a metodologia.

Embora o modelo *Waterfall* tenha sido criticado por sua falta de flexibilidade e capacidade de lidar com mudanças, suas ideias e conceitos continuam sendo referências importantes no campo do desenvolvimento de software. As contribuições de escritores, como Royce, Barry Boehm em seu artigo *"A Spiral Model of Software Development and Enhancement"* de 1986, entre outros pesquisadores ajudaram a moldar o pensamento sobre as metodologias de desenvolvimento e influenciaram o surgimento de abordagens mais ágeis e adaptativas, como por exemplo, o Scrum e o Kanban.

a) **Scrum** é um framework ágil que enfatiza a colaboração, a autogestão da equipe e a entrega contínua de valor ao cliente. Ele divide o trabalho em incrementos menores chamados de "sprints" e promove a transparência, a inspeção e a adaptação (ATLASSIAN, 2023).

b) **Kanban** é uma abordagem visual para o gerenciamento ágil de projetos, que utiliza um quadro Kanban para visualizar o fluxo de trabalho e limitar o trabalho em andamento. Ele promove a melhoria contínua do processo e a adaptação às mudanças (KANBANIZE, 2023).

Em outras palavras, Scrum é uma abordagem com uma estrutura mais rígida, com papéis e reuniões específicas, dividindo o trabalho em sprints definidos. O planejamento é feito no início de cada sprint, e há um foco na entrega de incrementos de valor ao cliente. Já o Kanban é uma abordagem mais flexível, sem papéis definidos, que enfatiza o fluxo contínuo de trabalho. Não há sprints fixos, e o planejamento é contínuo, adicionando tarefas ao quadro Kanban conforme a capacidade da equipe (ATLASSIAN, 2023).

Ambas as abordagens têm o objetivo de serem ágeis e adaptativas, mas diferem em suas práticas e estrutura. A escolha entre Scrum e Kanban depende das necessidades e contexto da equipe e do projeto em questão.

2.2.2. DevOps

DevOps é uma cultura, filosofia ou conjunto de práticas que une os departamentos de desenvolvimento e operações de TI em uma única equipe, trabalhando juntos ao longo de todo o ciclo de vida do *software* para obter mais rapidez e confiabilidade. Com objetivo de projetar, implementar ferramentas de automação, manter a infraestrutura e integração, e também uma

entrega contínua, bem como monitorar a produção, identificar e corrigir problemas em tempo hábil.

No exemplo abaixo, podemos conferir um looping do processo de *DevOps,* passando por processos como: **Criação** *(Create),* **Verificar** *(Verify),* **Embalar** *(Package),* **Liberação** *(Release),* **Configurar** *(Configure),* **Monitorar** *(Monitor),* **Planejar** *(Plan).* Vale lembrar que o processo não importa por onde começa e sim, o processo posterior a ser executado.

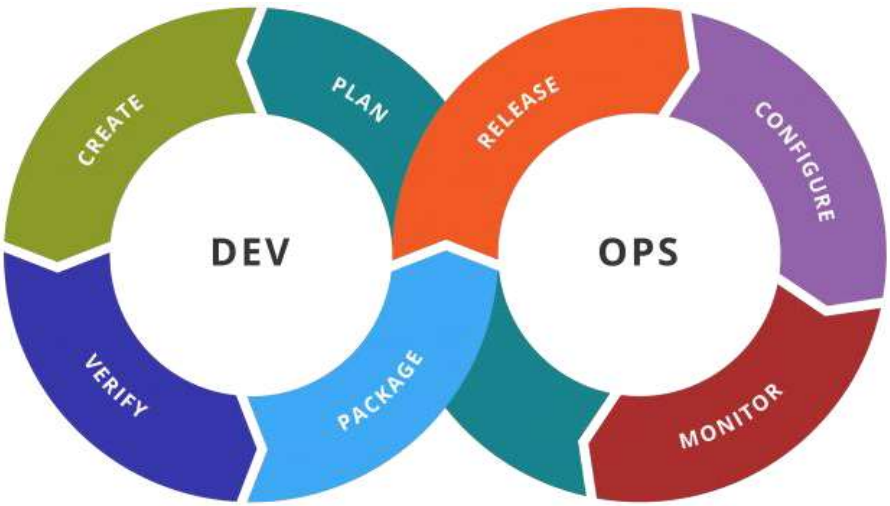

Figura 2: **Principais dúvidas sobre DevOps respondidas por um profissional da área.** (Fonte: JobHub, 2019)

Seu principal objetivo é melhorar a colaboração entre as equipes de desenvolvimento e operações, reduzindo o tempo e o custo de entrega do software, aumentando a eficiência operacional e fornecendo maior valor para os clientes e usuários finais (KIM, BEHR & SPAFFORD, 2016).

2.2.3. DevOps Testing

Segundo Mamede (2018), *DevOps Testing* é uma prática de teste de software que está alinhada com os princípios e valores do *DevOps*. É uma abordagem que se concentra em testar o software de forma colaborativa e contínua em todos os estágios do ciclo de vida do desenvolvimento de software.

Os testes são executados continuamente em cada estágio de entregas, permitindo que os problemas sejam identificados e corrigidos rapidamente. Isso garante a qualidade do software e reduz o risco de problemas de produção.

A prática também envolve a colaboração entre os membros da equipe de desenvolvimento, operações e testes. Isso ajuda a criar uma cultura de responsabilidade

compartilhada e ajuda a garantir que a qualidade seja construída desde o início do processo de desenvolvimento. Além disso, os testes são geralmente criados usando ferramentas de automação para garantir que sejam executados de maneira consistente e repetível em todos os ambientes (AZURE, 2023).

2.3. Engenharia de Testes

Conforme o tempo passa, erros que antes eram vistos como imprevisíveis, passaram a ser materiais preventivos para ampliar a qualidade do ciclo de vida do produto, neste caso, do *software*.

Segundo Valente (2020), erros, defeitos, falhas e *bugs* são conceitos bastante utilizados para descrever diferentes tipos de problemas que podem ocorrer em um sistema. Embora sejam frequentemente usados de forma intercambiável.

- Erro é uma ação humana que resulta em uma saída incorreta ou inesperada do software
- Defeito é gerado no sistema, quando desenvolvido de forma incorreta.
- Uma falha é uma condição em que o software não executa uma ação esperada, ou executa uma ação inesperada ou não desejada. As falhas geralmente são causadas por um erro no software.
- Bug é quando uma grande quantidade de erros, defeitos ou falhas são encontrados no software.

Uma curiosidade é que o uso do termo "*bug*" para se referir a um problema em um sistema eletrônico ou de computador tenha sido iniciado por Grace Hopper, uma pioneira na programação de computadores.

Em 1947, Hopper estava trabalhando no computador Harvard Mark II quando encontrou um problema em um dos painéis de controle. Depois de investigar, ela descobriu que havia um inseto (*bug* em inglês) preso no painel que estava causando o problema.

Ela removeu o inseto e o problema foi resolvido. Hopper, então, registrou o incidente em seu diário, incluindo o comentário "primeiro caso de *bug* encontrado" (REMAY, 2020)

2.3.1. Pilares de Testes

Podemos observar abaixo como os pilares de testes são compostos, divididos entre três grandes grupos, sendo eles: testes de **unidade** *(Unit Tests),* **integração** *(Integration Tests)* **e de ponta a ponta** *(End-to-End Tests).*

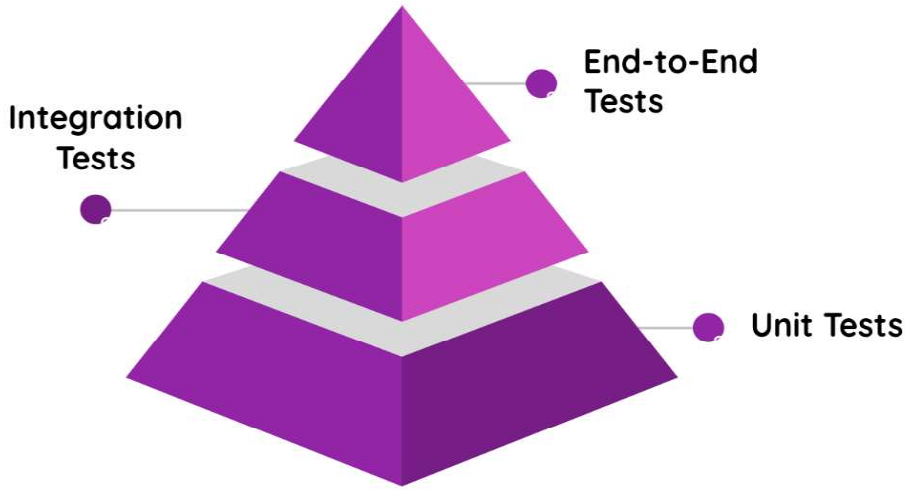

Figura 3: **Como os testes automatizados possibilitam o DevOps.** Fonte: ATLASSIAN, 2023.

Segundo Valente (2020), começando da classificação de maior granularidade para a menor, como o nome já sugere, o teste de ponta a ponta valida se o aplicativo atende às expectativas do usuário, simulando-o em um ambiente de testes, similar do real em que o software será executado. O objetivo principal é validar a funcionalidade, desempenho, segurança e confiabilidade do software em um ambiente real.

Já no teste de integração, possibilita a verificação de que várias partes do código possam funcionar juntas sem consequências não intencionais.

Sendo um pouco mais restritivo e próximo ao código-fonte, o teste unitário se concentra na validação individual de cada componente ou unidade do software, geralmente é realizado pelos desenvolvedores e consiste em verificar se uma função ou método funciona corretamente, para que ela possa ser integrada futuramente.

Quando mais se aproxima do topo da pirâmide, mais limitados ficam os testes. Trata-se de um ambiente de menor granularidade, onde não tem tantos detalhes, fazendo com que cada passo os torne mais lentos, gerando ainda mais custos e consumo de tempo (VALENTE, 2020).

2.3.2. Teste em Caixa Preta e Caixa Branca

O teste de caixa branca é uma técnica de teste em que a equipe de qualidade tem acesso ao código-fonte do *software*. Eles usam esse conhecimento para projetar testes que visam cobrir todas as partes do código e identificar possíveis erros ou problemas lógicos. É uma técnica que pode ser aplicada em diferentes níveis de testes, como teste unitário, teste de integração e teste de sistema (VALENTE, 2020).

Por outro lado, continua a ressaltar que o teste de caixa preta é uma técnica de teste em que os testadores não têm acesso ao código-fonte do software. Eles projetam testes com base na funcionalidade e nos requisitos do software, sem se preocupar com a lógica subjacente ou a implementação. O objetivo é testar o software como um usuário final faria, simulando diferentes entradas e observando as saídas correspondentes (VALENTE, 2020).

Basicamente, o teste de caixa branca é mais orientado para o desenvolvedor e o teste de caixa preta é mais orientado para o usuário final.

2.3.3. *Dogfooding*

De todos os pilares e estrutura de testes, uma técnica que fortalecesse essa ligação de usuário final e desenvolvedor é chamada de dogfooding. (THE ASTROLOGY PAGE, 2023).

De acordo com Valente (2020), essa tática é muito comum em empresas de software, onde os desenvolvedores e funcionários usam os produtos e serviços que estão desenvolvendo com o intuito de identificar problemas, melhorar a experiência do usuário e incentivar feedback entre a equipe.

Garantindo também, que atenda às necessidades dos usuários, elevando a confiança do cliente e melhorias constantes do sistema, já que o próprio responsável pela solução é o usuário.

2.4. Cobertura de Testes

Segundo Valente (2020), a cobertura de testes é uma métrica que auxilia a identificação da eficácia dos testes automatizados em encontrar defeitos no software. **Quanto maior a cobertura de testes, maior a probabilidade de encontrar defeitos antes que o software seja liberado para produção**. Por outro lado, uma baixa cobertura de testes indica que pode haver partes do código que não foram testadas, deixando aberta a possibilidade de defeitos não detectados.

Com isso, a cobertura de testes é uma medida que indica a eficácia dos testes automatizados em encontrar defeitos no software. É uma métrica importante que ajuda a avaliar a qualidade dos testes e a identificar áreas de código que não foram testadas (VALENTE, 2020)

Essa métrica é feita através do **Número de Comandos executados pelos Testes** dividido pelo **Total de Comandos do Programa.**

Existem ferramentas para cálculo de cobertura de testes. Na figura abaixo, mostramos um exemplo de uso da ferramenta que acompanha a IDE Eclipse. As linhas com fundo verde — coloridas automaticamente por essa ferramenta — indicam as linhas cobertas pelos cinco testes implementados em *StackTest*. As únicas linhas não coloridas de verde são responsáveis pela assinatura dos métodos de Stack e, portanto, não correspondem a comandos executáveis. Assim, a cobertura dos testes do nosso primeiro exemplo é de 100%, pois a execução dos métodos de testes resulta na execução de todos os comandos da classe *Stack* (VALENTE, 2020).

```
public class Stack<T> {
  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty(){
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
     if (isEmpty())
        throw new EmptyStackException();
     T elem = elements.get(size-1);
     size--;
     return elem;
  }
}
```

Figura 4: **Classe Stack.** Fonte: Engenharia de Software Moderna, 2020.

Suponha agora que não tivéssemos implementado *testEmptyStackException*. Isto é, não iríamos testar o levantamento de uma exceção pelo método *pop(),* quando chamado com uma pilha vazia. Nesse caso, a cobertura dos testes cairia para 92.9%, como ilustrado a seguir:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Stack	97.1 %	132	4	136
src	97.1 %	132	4	136
(default package)	97.1 %	132	4	136
Stack.java	92.9 %	52	4	56
StackTest.java	100.0 %	80	0	80

Figura 5: **Nível de Cobertura de Testes.** Fonte: Engenharia de Software Moderna, 2020.

Nesse caso, a ferramenta de cálculo de cobertura de testes marcaria as linhas da classe *Stack* da seguinte forma:

```
public class Stack<T> {
  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty(){
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
    if (isEmpty())
      throw new EmptyStackException();
    T elem = elements.get(size-1);
    size--;
    return elem;
  }
}
```

Figura 6: **Marcação da Cobertura de Testes**. Fonte: Engenharia de Software Moderna, 2020.

Como afirmamos, as linhas verdes são cobertas pela execução dos testes. Porém, existe um comando marcado de amarelo. Essa cor indica que o comando é um desvio (no caso, um if) e que apenas um dos caminhos possíveis do desvio (no caso, o caminho false) foi exercitado pelos testes de unidade. Por fim, o leitor já deve ter observado que existe uma linha em vermelho. Essa cor indica linhas que não foram cobertas pelos testes de unidade (VALENTE, 2020).

Em Java, ferramentas de cobertura de testes trabalham instrumentando os *bytecodes* gerados pelo compilador da linguagem. Como mostrado na figura com as estatísticas de cobertura, o programa anterior, após compilado, possui 52 instruções cobertas por testes de unidade, de um total de 56 instruções. Portanto, sua cobertura é 52 / 56 = 92.9% (VALENTE, 2020).

2.4. Trabalhos Similares

Segundo Souza e Gasparotto (2013), que escreveram um artigo sobre a importância da atividade de teste no desenvolvimento de software, a atividade de teste é uma das etapas do ciclo de desenvolvimento de software e tem o objetivo de relatar possíveis defeitos existentes no sistema para que estes sejam solucionados. Nesta fase verifica se o comportamento do sistema está de acordo com o especificado nos requisitos levantados junto ao cliente.

Além de colaborar para obtenção de um produto com qualidade, a atividade de teste tem o papel fundamental na redução de custos com possíveis reparos ao sistema. Para que os possíveis defeitos inseridos durante o andamento do projeto sejam encontrados o quanto antes, é importante que atividades de verificação e validação sejam executadas. O objetivo da verificação é garantir que o sistema está sendo desenvolvido da maneira correta, dentro das normas e metodologia adotadas para o projeto. Nesta atividade, normalmente, são realizadas revisões e inspeções dos artefatos e código-fonte, ou seja, uma verificação estática.

3. METODOLOGIA

Propondo-se a descrever e analisar como o teste de software pode ter impacto em uma empresa, foi empregado o método descritivo. Visa mostrar de maneira clara o que está sendo descrito, com uma contextualização do tema, posteriormente mostrando aspectos relevantes e, por fim, resumindo as informações apresentadas.

Empresas de tecnologia não podem tratar testes como uma mera formalidade, pois é um processo de extrema importância. **Dados apontam que 71% das empresas acham o uso de**

teste de software contínuo essencial ou importante, porém apenas 22% adotam essa prática (INFORCHANNEL, 2017).

Podemos dizer que os dados foram coletados por meio de pesquisa bibliográfica, que é uma técnica de coleta de dados que envolve a revisão e análise de artigos, livros e outras fontes de informação disponíveis na literatura científica ou na internet. A pesquisa bibliográfica é uma forma de coletar dados secundários, ou seja, dados que já foram coletados e publicados por outras pessoas ou organizações.

Depois de coletados, os dados foram tratados e analisados para ter um resultado significativo através deles, foram removidos dados duplicados e ausentes. Além disso, foram realizados testes de hipóteses e análise de correlação para mostrar os dados da forma mais apropriada possível.

Algumas métricas de testabilidade e boas práticas foram encontradas através das pesquisas, métodos como: Desenvolvimento Dirigido por Testes (*Test-Driven Development*), Refatoração (*Refatoring*), *DevOps* e *DevOps Testing* são práticas bem comuns de testes e são usadas tanto para melhorar a qualidade do código quanto a eficiência do Desenvolvimento, trazendo em comparação métodos anteriores, como *Waterfall* (GRUPO MULT, 2023).

4. DESENVOLVIMENTO

Segundo Dev Media (2023), o desenvolvimento de *software* é o processo de criar, implementar e manter programas de computador. Envolve a concepção, codificação, teste e documentação de software para atender às necessidades específicas de usuários e empresas. A seguir, alguns objetivos do processo de desenvolvimento:

- Definição das atividades a serem executadas;
- Quando determinada atividade deve ser executada;
- Pessoa ou grupo a executar tais atividades;
- Padronização no processo de desenvolvimento.

Existem várias abordagens e metodologias para o desenvolvimento de software, como o modelo cascata, desenvolvimento ágil (por exemplo, Scrum e Kanban) e DevOps. Essas metodologias fornecem estruturas para gerenciar o ciclo de vida do software de forma eficiente, promovendo a colaboração entre equipes e permitindo entregas mais rápidas e de alta qualidade (DEV MEDIA, 2023)

No desenvolvimento de software, são utilizadas várias linguagens de programação, como Java, Python, C++, JavaScript, entre outras, bem como frameworks e bibliotecas específicas para facilitar o processo de criação de software.

4.1. Testabilidade

A testabilidade é um processo importante para as empresas de tecnologia, no qual permite a detecção mais precoce de defeitos e uma entrega mais rápida de software de alta qualidade. As empresas podem aumentar a testabilidade do software adotando práticas de desenvolvimento que o torna mais fácil de testar e mantê-lo (TANNOURI, 2023).

4.1.1. *Refactoring*

Conhecido também como Refatoração, é um processo de modificar o código-fonte de um software existente para melhorar sua qualidade, sem alterar seu comportamento externo. A refatoração geralmente é feita para melhorar a estrutura do código, tornando-o mais fácil de entender, manter e estender, bem como para eliminar problemas de código, como código duplicado, complexidade desnecessária ou falta de coesão (VALENTE, 2020).

Sendo uma prática importante na Engenharia de *Software*, pois ajuda a melhorar a qualidade do código e diminuindo redundâncias e custos. Entre os custos principais, estão a manutenção e evolução do *software*, priorizando que seu objetivo externo não seja alterado e seu código seja cada vez mais otimizado, tornando o mais fácil de ler (AWARI, 2023).

4.1.2. *Test-Driven Development* (TDD)

Conhecido também como Desenvolvimento Guiado por Teste, Test-Driven Development (TDD) é uma técnica de desenvolvimento de software em que os testes automatizados são escritos antes do código de produção. Segundo Valente (2020), o processo começa com a criação de um teste automatizado para uma funcionalidade específica que ainda não foi implementada, em seguida o código é escrito para passar no teste e, por fim, o código é refatorado para melhorar sua qualidade. Esse ciclo é repetido várias vezes para cada funcionalidade, de acordo com o Acervo Lima (2023), dentre vários benefícios do TDD, alguns incluem:

a) Reduzir margens de defeitos e falhas;
b) Melhorar a qualidade do código, tornando-o mais limpo;
c) Aumentar a confiança na entrega de *software;*

d) Automação na comunicação entre os membros da equipe de desenvolvimento e a equipe de testes, tornando mais fácil para todos entenderem as funcionalidades a serem implementadas e as expectativas de comportamento do software;

e) Redução de custos e aproveitamento melhor do tempo a ser utilizado.

4.2. Resultados obtidos

Um estudo publicado no Comunidade DEV (2020) discute a implementação do Desenvolvimento Orientado a Testes (TDD) pela Microsoft e IBM. Destacando a experiência da Microsoft com o Windows NT, onde o TDD ajudou a garantir a qualidade do software, e também, menciona o caso da IBM e seu projeto *Clean Room Software Engineering*, no qual o TDD resultou em código mais limpo e menos propenso a erros.

O artigo enfatiza a importância de escrever testes antes do código e ressalta os benefícios do TDD a longo prazo. Encoraja os desenvolvedores a adotarem essa prática em seus projetos, aprendendo com as experiências das empresas mencionadas (COMUNIDADE DEV, 2020).

IBM

1 Equipe trabalhando em:	Drivers de Dispositivo

Figura 7: **O projeto e as equipes: IBM** (Fonte: Comunidade DEV)

A IBM adotou o TDD em 2002, quando escolheu uma nova plataforma e uma arquitetura totalmente nova para seus drivers de dispositivo. Logo abaixo, observamos que os drivers de dispositivos foram utilizados nos testes de TDD tanto no sistema legado com o novo sistema revelando mais adiante desse artigo, as diferenças significativas, mas também demonstrando o valor do TDD na abordagem de desenvolvimento (COMUNIDADE DEV, 2020).

Microsoft

3 Equipes trabalhando em:	Windows, MSN, Visual Studio

Figura 8: **O projeto e as equipes: Microsoft** (Fonte: Comunidade DEV)

Na Microsoft, tanto os projetos TDD quanto os não-TDD são liderados por equipes com níveis de responsabilidades semelhantes, reportando ao mesmo gerente de alto nível. Isso

proporciona um equilíbrio ideal para comparar os resultados e determinar a eficácia da abordagem TDD nas equipes (COMUNIDADE DEV, 2020).

Contextos variados no caso dessas 4 equipes	
Localização Geográfica	- Co-localizado e distribuído
	- Estados Unidos e México
Domínio de aplicativo de software	- Driver do dispositivo
	- Software de sistemas
	- Aplicativo para serviço Web
	- Software de Aplicativs
Experiência do desenvolvedor	- De baixo para o alto
Linguagem de programação	- Java, C++, .Net
Especialização do domínio	- De baixo para o alto

Figura 9: **Resumo dos fatores da equipe** (Fonte: Comunidade DEV)

Observamos os **Contextos Variados no Caso dessas 4 Equipes** *(Varying contexts in case of these 4 teams)* na tabela abaixo, ressaltando alguns fatores, como:

- **Localização geográfica** *(Geographical Location)*: co-localizado e distribuído *(co-located and distributed).*
- **Domínio de Aplicativo de Software** *(Software Application Domain):* diversas áreas foram utilizadas, como Software de driver de Dispositivo (*device driver software),* Software de Sistema (*system software),* Aplicação de Serviços Web (*web service application) e* Software de Aplicativos *(application software).*
- **Experiência do Desenvolvedor** *(Developer's Experience):* de baixo para o alto *(From Low to High).*
- **Linguagem de Programação** *(Programming Language):* Java, C++, .NET
- **Conhecimento do Domínio** *(Domain Expertise):* de baixo para o alto *(From Low to High).*

Com a aplicação dos métodos apresentados no Processo de Desenvolvimento de Software, podemos observar que várias empresas conseguiram obter valores significativos na sua escalabilidade, que se trata de uma expansão das suas atividades sem comprometer a qualidade e sem gerar um aumento expressivo dos gastos, mantendo um bom custo-benefício a longo prazo.

Descrição da Métrica	IBM: Drivers	Microsoft: Windows	Microsoft: VS	Microsoft: MSN
Densidade de defeito sem o uso do TDD	W	X	Y	Z
Densidade de defeito com o uso do TDD	0.61W	0.38X	0.24Y	0.09Z
Aumento no tempo gasto para codificar o recurso por causa do TDD (%)	15-20 %	25-35 %	15%	25-20 %

Figura 10: **Qual o resultado do TDD em qualidade e produtividade para as equipes?** (Fonte: Comunidade DEV)

Na tabela acima, podemos observar dois fatores importantíssimos: a Densidade de Defeito da Equipe usando TDD *(Defect Density of Team using TDD)* e o Aumento no tempo gasto para codificar o recurso por causa do TDD em porcentagem *(Increase in time taken to code the feature because of TDD).*

As letras (W, X, Y e Z) equivalem a Densidade de Defeitos de (IBM: Drivers, Microsoft: Windows, Microsoft: VS e Microsoft: MSN) respectivamente, antes da Implementação do Desenvolvimento Dirigido por Testes.

E na linha de baixo é possível identificar o quanto os defeitos diminuíram após a Implementação do TDD, por exemplo, o W abaixou para 61% do que era antes, o X para 38%, o Y para 24% e o Z para 9%, nos mostrando um aumento significativo na eficiência de desenvolvimento do *software*.

Nesse caso acima, podemos analisar o estudo realizado pelas equipes de produto da Microsoft e IBM constatou que a adoção do Desenvolvimento Orientado a Testes (TDD) pode **levar a um aumento no tempo de desenvolvimento, variando de 15% a 35%.** No entanto, essa maior demanda de tempo **é compensada pelos benefícios de redução de custos de manutenção** devido à melhoria na qualidade do software. Essa observação reforça a importância do TDD como uma prática que resulta em software de maior qualidade e menor necessidade de correções e atualizações futuras (COMUNIDADE DEV, 2023).

Segundo o InfoQ (2023), estes resultados podem ser comparados com aqueles encontrados no artigo ***Making Software: What Really Works, and Why We Believe It*** publicado em 2006 por Maria Siniaalto. Esse artigo resumiu os resultados de outros 13 estudos

em *Test-Driven Development*, incluindo pesquisa conduzida em contextos industrial, semi-industrial e acadêmico. Entre as conclusões do artigo, o a autora escreveu:

> "Baseado nas descobertas dos estudos existentes, pode-se concluir que TDD parece melhorar a qualidade de software, especialmente quando empregado em um contexto industrial. As descobertas não foram tão óbvias nos contextos semi-industrial ou acadêmico, mas nenhum desses estudos reportou queda na qualidade. Os efeitos na produtividade do TDD não foram muito óbvios, e os resultados variam independentemente do contexto do estudo. Porém, houve indícios de que TDD não necessariamente diminui a produtividade do desenvolvedor ou aumenta o tempo de desenvolvimento do projeto: Em alguns casos, melhorias significantes de produtividade foram alcançadas com TDD enquanto que apenas dois dos treze estudos reportaram queda de produtividade. Porém, em ambos estudos a qualidade foi melhorada."

Vale ressaltar que esse mercado tecnológico está em ampla expansão. De acordo a Revista Exame (2021)

> "No mundo da economia digital, os profissionais de tecnologia são cada vez mais disputados pelas grandes empresas de tecnologia, mas também por qualquer negócio, dada a importância do ambiente digital atualmente. Se há uma busca maior por um profissional capacitado na área de tecnologia, há também um verdadeiro boom de profissionais interessados em migrar ou começar a atuar nessa área Segundo os dados do Google, a pesquisa por ciência de dados e tecnologia cresceu 1170% nos últimos cinco anos no Brasil e procura pela carreira de programador aumentaram 30% em comparação com 2021".

Outro estudo publicado e traduzido por Michael Warren no **BairesDevBlog** (2021), cita que:

> "Em 2021, existem 26,9 milhões de desenvolvedores de software espalhados pelo mundo – com 4,3 milhões deles na América do Norte – um número que deve crescer para 28,7 milhões em 2024.
>
> À medida que cresce a demanda por desenvolvedores de software, cresce também o número de profissionais capacitados para esse trabalho.
>
> O Bureau of Labor Statistics dos EUA prevê que o emprego de desenvolvedores de software, analistas de garantia de qualidade e testadores deve crescer 22% de 2019 a 2029, muito mais rápido que a média de todas as ocupações".

5. CONSIDERAÇÕES FINAIS

Em conclusão, este trabalho teve como objetivo geral analisar a importância dos testes de *software* para a escalabilidade de empresas e suas contribuições para a qualidade do produto. Durante a pesquisa, foram descritas práticas recomendadas para testes de *software* em diferentes fases do ciclo de vida do desenvolvimento, identificando metodologias e ferramentas

utilizadas para testes e como elas podem ser aplicadas. Além disso, foram analisados casos de sucesso de empresas que prezam pela qualidade do *software* com rotinas de testes, assim, as suas versões futuras apresentarão baixas densidades de defeitos devido ao uso de ativos de testes, o que afeta diretamente a saúde do desenvolvedor, fazendo com que ele não trabalhe em cima de uma atividade de forma desgastante, uma vez que existirá uma equipe para apoiá-lo nessa etapa do desenvolvimento de *software.*

Portanto, conclui-se que a inclusão de testes de *software* na cultura de uma empresa é fundamental para aprimorar a escalabilidade e garantir a qualidade da solução final. As práticas recomendadas descritas neste trabalho podem ser utilizadas como um guia para a implementação de testes em diferentes fases do ciclo de vida do desenvolvimento. É importante ressaltar que, para obter sucesso na implementação de testes de *software*, é necessário um planejamento cuidadoso, a adoção de metodologias eficientes e o uso de ferramentas adequadas.

REFERÊNCIAS BIBLIOGRÁFICAS

ACERVO LIMA. **Vantagens e Desvantagens do Test Driven Development** (TDD). Disponível em: <https://acervolima.com/vantagens-e-desvantagens-do-test-driven-development-tdd/>. Acesso em: 02 jun. 2023

ADOBE, Communications Team. **Waterfall Methodology: A Complete Guide.** 2022. Disponível em: <https://business.adobe.com/blog/basics/waterfall/>. Acesso em: 15 mai. 2023

ATLASSIAN. **Como os testes automatizados possibilitam o DevOps.** Disponível em: < https://www.atlassian.com/br/devops/devops-tools/test-automation/>. Acesso em: 15 mai. 2023

ATLASSIAN. **Kanban vs. Scrum: que tipo de ágil é você?** Disponível em: < https://www.atlassian.com/br/agile/kanban/kanban-vs-scrum#:~:text=Resumo%3A%20O%20Kanban%20%C3%A9%20uma,de%20valores%2C%20princ%C3%ADpios%20e%20pr%C3%A1ticas.>. Acesso em: 27 jun. 2023

ATLASSIAN. **O que é scrum e como começar.** Disponível em: <https://www.atlassian.com/br/agile/scrum/>. Acesso em: 27 jun. 2023

AWARI. **Refatoração: Transformando Código Bagunçado em Código Limpo.** 2023. Disponível em: <https://awari.com.br/refatoracao/#:~:text=A%20refatora%C3%A7%C3%A3o%20%C3%A9%20uma%20pr%C3%A1tica,eficiente%20e%20f%C3%A1cil%20de%20entender./>. Acesso em: 28 jun. 2023

AZURE, Microsoft. **O que é o DevOps?** Disponível em:<https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops>. Acesso em: 07 mai. 2023

BAIRESDEVBLOG. **How Many Software Developers Are There in the World?** Disponível em: <https://www.bairesdev.com/blog/types-of-developers-2021/>. Acesso em: 02 jun. 2023

BARBOSA, Matteus. **Conceitos de Teste de Software**. 2021. Disponível em: <https://matteus.dev/conceitos-testes-software/ > Acesso em: 20 mar. 2023

BOEHM, Barry. William. (1986). **A Spiral Model of Software Development and Enhancement. IEEE Computer Society.** Vol. 21, No. 5, maio de 1988

BROWN, T. (2002). **History of software engineering.** In Proceedings of the 24th International Conference on Software Engineering (pp. 9-15).

CARVALHO, Ingrid. **Você sabe o que é Teste Caixa Branca e Teste Caixa Preta?** 2019. Disponível em: <https://medium.com/@ingrid.carvalho.mo/você-sabe-o-que-é-teste-caixa-branca-e-teste-caixa-preta-9a2d08fe9d0c>. Acesso em: 20 mar. 2023

COMUNIDADE DEV. **TDD - Aprenda com a experiência da Microsoft e da IBM.** Disponível em: < https://dev.to/vaibhav_arora__/tdd-learn-from-experience-of-microsoft-and-ibm-25o6>. Acesso em: 04 mai. 2023

DEV MEDIA. **Atividades básicas ao processo de desenvolvimento de *Software*.** Disponível em: <https://www.devmedia.com.br/atividades-basicas-ao-processo-de-desenvolvimento-de-software/5413>. Acesso em: 28 jun. 2023

É LIVRE E ABERTO. **Um erro de ponto flutuante que causou um dano no valor de meio bilhão.** Disponível em: <https://livreeaberto.com/um-erro-de-ponto-flutuante-que-causou-prejuizo-meio-bilhao>. Acesso em: 02 jun. 2023

EAD PUC GOIÁS. **Desenvolvimento de Software: o que é, como trabalhar na área?** 2021. Disponível em:<https://ead.pucgoias.edu.br/blog/desenvolvimento-software> Acesso em: 28 fev. 2023

VALENTE, Marco Tulio. **Engenharia de Software Moderna.** 2020. Disponível em: < https://engsoftmoderna.info/cap8.html> Acesso em: 28 jun. 2023

EXAME. **Procura por profissionais de tecnologia cresce 671% durante a pandemia.** Disponível em: <https://www.cnnbrasil.com.br/economia/procura-por-profissionais-de-tecnologia-cresce-671-durante-a-pandemia/>. Acesso em: 02 jun. 2023

GARCIA, M. (2018). **Software Engineering: Advancements and Challenges.** Journal of Software Engineering Research and Development, 6(2), 47-59.

GRUPO MULT. **Desenvolvimento Orientado a Testes: Conheça as Vantagens em Implementar TDD.** Disponível em: <https://www.grupomult.com.br/desenvolvimento-orientado-a-testes-conheca-as-vantagens-em-implementar-tdd/#:~:text=O%20desenvolvimento%20orientado%20a%20testes,ap%C3%B3s%20o%20desenvolvimento%20do%20c%C3%B3digo./>. Acesso em: 28 jun. 2023

HOSTINGER. **Entenda o Que são Testes de Produção e Como Executá-Los**. 2023. Disponível em:< https://www.hostinger.com.br/tutoriais/teste-em-producao> Acesso em: 27 jun. 2023

HRISTOV, Anton. **Como os testes automatizados possibilitam o DevOps**. 2022. Disponível em:< https://www.atlassian.com/br/devops/devops-tools/test-automation> Acesso em: 05 mar. 2023

InfoQ. **Estudos Empíricos Mostram Que Test Driven Development Melhora Qualidade.** Disponível em: <https://www.infoq.com/br/news/2009/03/TDD-Improves-Quality//>. Acesso em: 15 mai. 2023

InforChannel. **De 5 empresas, apenas 1 usa automação de testes como facilitador dos negócios.** 2017. Disponível em:< https://inforchannel.com.br/2017/09/28/de-5-empresas-apenas-1-usa-automacao-de-testes-como-facilitador-dos-negocios/>. Acesso em: 12 mai. 2023

JOHNSON, R. (1995). **Software engineering: An introduction.** New York, NY: Prentice Hall.

JOHNSON, R. (2014). **Software Development: A Practitioner's Approach.** Boston, MA: Pearson.

JONES, C. (2008). **Applied software measurement.** New York, NY: McGraw-Hill.

KANBANIZE. **O que é Kanban? Explicado para Iniciantes.** 2023. Disponível em: <https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban> Acesso em: 27 jun. 2023

KIM, G., BEHR, K., & SPAFFORD, G. (2016). **The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.** Portland, OR: IT Revolution Press.

MAMEDE, Elisabeth. **Vamos falar de DevOps Testing?** 2018. Disponível em: <https://medium.com/@elisabethmamede/vamos-falar-de-devops-testing-d41e19513d12/>. Acesso em: 08 mai. 2023

MCCONNELL, S. (2004). **Code Complete: A Practical Handbook of Software Construction.** Redmond, WA: Microsoft Press.

PRECIOSO, Vitor**. DevOpsTesting, conheça essa nova tendência!** 2018. Disponível em: <https://www.cedrotech.com/blog/devopstesting-conheca-essa-nova-tendencia/> Acesso em: 23 mar. 2023

PRESSMAN, R. (2014). **Software Engineering: A Practitioner's Approach.** New York, NY: McGraw-Hill.

REMAY. **Conheça a curiosa história do primeiro bug de computador.** Disponível em: <https://remay.com.br/2020/10/26/conheca-a-curiosa-historia-do-primeiro-bug-de-computador/>. Acesso em: 02 jun. 2023

ROYCE, Winston W. (1970). **Managing the Development of Large Software Systems: Concepts and Techniques.** Conferência IEEE Wescon, agosto de 1970.

SINAALTO, Maria. **Making Software: What Really Works, and Why We Believe It.** Disponível em: <https://books.google.com.br/books?id=DxuGi5h2-HEC&lpg=PA218&ots=0XxysePdtN&dq=artigo%20publicado%20em%202006%20por%20maria%20siniaalto&hl=pt-BR&pg=PP1#v=onepage&q&f=false>. Acesso em: 02 jun. 2023

SMITH, J. (2010). **Software Engineering: A Historical Perspective.** Communications of the ACM, 53(8), 42-48.

SOMMERVILLE, I. (2016). **Software Engineering. Harlow, UK: Pearson Education.**

SOUZA, Karla Pires de; GASPAROTTO, Angelita Moutin Segoria. **A Importância da Atividade de Teste no Desenvolvimento De Software**. 2013. Disponível em:<https://abepro.org.br/biblioteca/enegep2013_TN_STO_177_007_23030.pdf>. Acesso em: 10 mar. 2023

SOUZA, Manoel. **Desenvolvedor de software: profissão que mais cresce no mundo**. 2021. Disponível em: < https://itforum.com.br/noticias/desenvolvedor-de-software-profissao-que-mais-cresce-no-mundo/>. Acesso em: 25 fev. 2023.

TANNOURI, Patrícia Aline. **O que é testabilidade?** Linha de Código. Disponível em: <http://www.linhadecodigo.com.br/artigo/923/o-que-e-testabilidade.aspx/>. Acesso em: 13 mai. 2023

THE ASTROLOGY PAGE. **O que é dogfooding?** 2023. Disponível em: <https://pt.theastrologypage.com/dogfooding>. Acesso em: 11 mai. 2023

VALENTE, Marcos Tulio. **Engenharia de Software Moderna:** Princípios e Práticas para Desenvolvimento de Software com Produtividade, Editora: Independente, 395 páginas, 2020.

WARREN, Michael. ***How Many Software Developers Are There in the World*?** 2021. Disponível em:< https://www.bairesdev.com/blog/how-many-software-developers-in-the-world/>. Acesso em: 10 mar. 2023.

WK, JobHub. **Principais dúvidas sobre DevOps respondidas por um profissional da área.** 2019. Disponível em: <https://wkrh.com.br/o-que-e-devops//>. Acesso em: 15 mai. 2023