RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

MANUTENIBILIDADE DE SOFTWARE: FERRAMENTAS QUE AUXILIAM E GARANTEM A QUALIDADE

SOFTWARE MAINTENANCE: TOOLS THAT HELP AND GUARANTEE QUALITY

MANTENIMIENTO DE SOFTWARE: HERRAMIENTAS QUE AYUDAN Y GARANTIZAN LA CALIDAD

Gabriel Pereira Escareli[1], Rodrigo Daniel Malara[2]

e3122450
https://doi.org/10.47820/recima21.v3i12.2450

PUBLICADO: 12/2022

RESUMO

Os princípios da manutenção de *software* são um dos pilares no que se diz respeito a qualidade. Este trabalho tem como objetivo apresentar algumas ferramentas disponíveis no cenário de desenvolvimento de *software* (Super-linter e SpotlessCheck), para auxiliar o desenvolvedor a escrever da melhor forma um bom código e consequentemente garantir a qualidade final do projeto. A escolha e o uso de um *pipeline* se dão pela possibilidade de criar uma série de etapas a serem realizadas, o que torna o processo dinâmico. Foi utilizada a metodologia CI/CD que facilita as constantes entregas realizadas pelos desenvolvedores. Pelo fato de o projeto teste estar hospedado em um repositório do GitHub, o GitHub Actions foi a plataforma escolhida para a *pipeline*, onde é possível a criação de fluxos de trabalho utilizando CI/CD. Ferramentas de *lint* foram usadas nesse processo com o intuito de realizar uma varredura no código a fim de sinalizar erros estilísticos e construções suspeitas, bem como o uso do SpotlessCheck, para aprofundar a varredura do código, com o objetivo de emitir relatórios a respeito de *bugs*, duplicações e mais alguns indicativos de qualidade. A hipótese é que todo o processo executado no *pipeline* ajuda a garantir a maior manutenibilidade do projeto.

PALAVRAS-CHAVE: Pipeline. Qualidade. Ferramentas. Código.

ABSTRACT

The principles of software maintenance are one of the pillars when it comes to quality. This work aims to present some tools available in the software development scenario (Super-linter and SpotlessCheck), to help the developer to write a good code in the best way and consequently guarantee the final quality of the project. The choice and use of a pipeline is due to the possibility of creating a series of steps to be carried out, which makes the process dynamic. The CI/CD methodology was used, which facilitates the constant deliveries made by the developers. Because the test project is hosted in a GitHub repository, GitHub Actions was the platform chosen for the pipeline, where it is possible to create workflows using CI/CD. Lint tools were used in this process in order to scan the code in order to flag stylistic errors and suspicious constructions, as well as the use of SpotlessCheck, to deepen the code scan, with the objective of issuing reports about bugs, duplications and a few more quality indicators. The hypothesis is that the entire process executed in the pipeline helps to ensure greater project maintainability.

KEYWORDS: *Pipeline. Quality. Tools. Code.*

RESUMEN

Los principios del mantenimiento de software son uno de los pilares cuando se trata de calidad. Este trabajo tiene como objetivo presentar algunas herramientas disponibles en el escenario de desarrollo de software (Super-linter y SpotlessCheck), para ayudar al desarrollador a escribir buen código de la mejor manera y, en consecuencia, garantizar la calidad final del proyecto. La elección y uso de una

[1] Universidade de Araraquara - UNIARA
[2] Universidade de Araraquara - UNIARA

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

tubería es la posibilidad de crear una serie de pasos a realizar, lo que hace que el proceso sea dinámico. Se utilizó la metodología CI/CD, que facilita las constantes entregas realizadas por los desarrolladores. Debido a que el proyecto de prueba está alojado en un repositorio de GitHub, GitHub Actions fue la plataforma elegida para la canalización, donde es posible crear flujos de trabajo mediante CI/CD. Las herramientas Lint se utilizaron en este proceso para realizar un escaneo del código con el fin de señalar errores estilísticos y construcciones sospechosas, así como el uso de SpotlessCheck, para profundizar el escaneo del código, con el fin de emitir informes sobre errores, duplicaciones y algunos indicadores de calidad más. La hipótesis es que todo el proceso realizado en la tubería ayuda a garantizar una mayor capacidad de mantenimiento del proyecto.

PALABRAS CLAVE: *Pipeline. Calidad. Herramientas. Código.*

INTRODUÇÃO

Manutenção de *software* é definida como o processo de modificação de um software, componente ou sistema após a sua instalação (CORDEIRO, [20--]). Os conceitos aplicados na manutenibilidade de *software*, que tem a qualidade como princípio, refere-se as facilidades de correção, melhorias ou adaptações em seu ambiente. Parte vital da vida útil de um software, a manutenção é um processo que requer análises, testes e documentações, tornando-se um processo delicado e trabalhoso. O principal problema encontrado em todos os processos de manutenção é o tempo gasto arrumando códigos antigos, deixando de lado a parte mais valiosa do desenvolvimento de *software*: escrever novos códigos. Segundo VISSER (2016), as manutenções de código fonte levam, pelo menos, o dobro de tempo quando o processo é deixado em segundo plano, comparado quando a manutenção é levada em primeiro plano.

> A manutenção de *software* é um fator de extrema importância, pois, é considerada como a fase onde a organização desprende maior esforço, consumindo em torno de 70% da atenção do pessoal de TI, e, este percentual continua aumentando gradativamente (PRESSMAN, 1995, p. 876).

De acordo com Engholm Junior (2010) as empresas tendem a enfrentar sérios problemas quando há um investimento em desenvolvimento de sistemas sem a utilização da engenharia de *software*, seja por desenvolvimento interno ou de terceiros. O autor lista uma série de consequências negativas, entre elas, a dificuldade em dar manutenção tanto corretiva quanto evolutiva, *softwares* difíceis de serem implementados, baixa qualidade e desempenho.

Quando se pensa em adquirir um produto, espera-se dentre outras, a qualidade. Fazer *software* normalmente é caro e com isso não há muita margem para descuidos e erros. A proposta de trazer processos que vão ajudar o desenvolvedor a entregar o código que ele escreveu, é importante. Tempo é algo crucial no desenvolvimento de um *software*, e quanto mais tempo se leva para realizar uma tarefa mais caro o produto fica, afinal é necessário levar em conta os custos por trás das operações. Processos mais automatizados facilitam ganhar tempo, consequentemente barateando o produto final. Visando facilitar o dia a dia de alguns desenvolvedores dentre os 26,9

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

milhões espalhados pelo mundo (EVANS DATA CORPORATION, 2019), há várias ferramentas que ajudam o desenvolvedor de *software.*

Os princípios da manutenção de *software* são um dos pilares no que se diz respeito a qualidade. Este trabalho tem como objetivo apresentar algumas ferramentas disponíveis no cenário de desenvolvimento de *software* (Super-linter e SpotlessCheck), para auxiliar o desenvolvedor a escrever da melhor forma um bom código e consequentemente garantir a qualidade final do projeto. A escolha e o uso de um *pipeline* se dão pela possibilidade de criar uma série de etapas a serem realizadas, o que torna o processo dinâmico. Foi utilizada a metodologia CI/CD que facilita as constantes entregas realizadas pelos desenvolvedores. Pelo fato de o projeto teste estar hospedado em um repositório do GitHub, o GitHub Actions foi a plataforma escolhida para a *pipeline*, onde é possível a criação de fluxos de trabalho utilizando CI/CD. Ferramentas de *lint* foram usadas nesse processo com o intuito de realizar uma varredura no código a fim de sinalizar erros estilísticos e construções suspeitas, bem como o uso do SpotlessCheck, para aprofundar a varredura do código, com o objetivo de emitir relatórios a respeito de *bugs*, duplicações e mais alguns indicativos de qualidade. A hipótese é que todo o processo executado no *pipeline* ajuda a garantir a maior manutenibilidade do projeto.

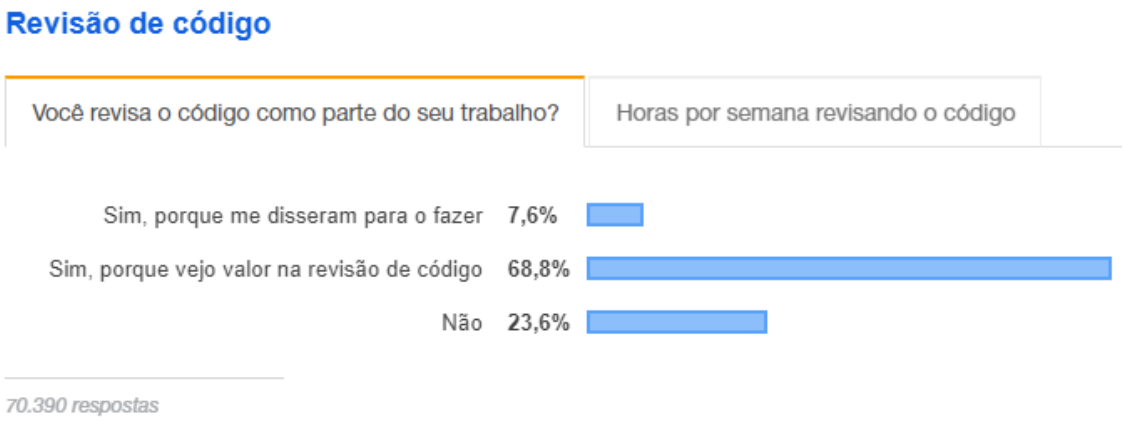
1 REVISÃO BIBIOGRÁFICA

Ao longo do tempo são criadas e aplicadas algumas rotinas e situações que tornam o desenvolvimento das atividades mais prático. Algumas são tão essenciais que se tornaram parte vital no desenvolvimento de *software* para garantir um código de qualidade. Nesta seção encontra-se algumas ferramentas disponíveis no ambiente de desenvolvimento de *software.*

REVISÃO DE CÓDIGO

Segundo uma pesquisa do *Stack Overflow*, um site de perguntas e respostas de entusiastas de programação, apenas 23% dos desenvolvedores dentre de mais de 70 mil que participaram da pesquisa não usavam revisão de código (STACK OVERFLOW, 2019).

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218



Fonte: Stack Overflow

Para a garantia de qualidade de *software,* algumas estratégias podem ser adotadas. Para uma produção de *software* moderna, diversas organizações adotam a revisão de código como prática de qualidade, isto é, um código feito por um desenvolvedor só entrará em produção assim que ele for revisado por outro desenvolvedor. Tem como objetivo a detecção de *bugs*, garantia de manutenibilidade e outros parâmetros que disseminam as boas práticas de engenharia de *software*. (VALENTE, 2013)

ANÁLISE ESTÁTICA

Sommerville (2011) diz que a análise estática é um método de verificação de código fonte antes que haja a execução do *software*. Essa verificação tem como parâmetros um conjunto de regras pré-estabelecidas, que geralmente já estão inseridas nas ferramentas utilizadas, com o objetivo de ir o mais fundo possível em uma análise, entregando relatórios de erros e correções.

Muitas vezes as inspeções de códigos são guiadas por *checklists* de verificação de erros e heurísticas. Para isso, é possível automatizar os processos de verificação onde, frequentemente, nenhuma entrada adicional é requerida fazendo com que uma análise estática automatizada seja mais fácil de ser introduzida em um processo. (SOMERVILLE 2011)

METODOLOGIAS DE TRABALHO

CONTINUOS INTEGRATION/CONTINUOUS DELIVERY (CI/CD)

CI/CD é um método para entregar aplicações com frequência aos clientes. Para isso, é aplicada a automação nas etapas do desenvolvimento. Os principais conceitos atribuídos a esse método são a integração, entrega e implantação contínua. Com o CI/CD, é possível solucionar os problemas que a integração de novos códigos pode causar para as equipes de operações e desenvolvimento.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Especificamente, o CI/CD aplica o monitoramento e automação contínua em todo o ciclo de vida das aplicações, incluindo as etapas de teste e integração, além da entrega e implantação. Juntas, essas práticas relacionadas são muitas vezes chamadas de “pipeline de CI/CD” e são compatíveis com o trabalho conjunto das equipes de operações e desenvolvimento com métodos ágeis. (REDHAT, s. d.).

PIPELINE

Pipelines consistem em uma série de etapas a serem realizadas para a disponibilização de uma nova versão de um *software.* O pipeline é uma prática que tem como objetivo acelerar a disponibilização de *software*, adotando a abordagem DevOps.

O pipeline inclui monitoramento e automação para melhor o processo de desenvolvimento de aplicações principalmente dos estágios de integração e teste, mas também na entrega e na implantação. É possível executar manualmente cada etapa do pipeline, mas o real valor dele está na automação. (REDHAT, s. d.)

DEVOPS

A metodologia DevOps é uma abordagem de cultura, automação e design de plataforma que tem como objetivo agregar mais valor aos negócios e aumentar sua capacidade de resposta às mudanças por meio de entregas de serviços rápidas e de alta qualidade. Isso tudo é possível por meio da disponibilização de serviços de TI iterativa e rápida. (REDHAT, s. d.)

FERRAMENTAS DE TRABALHO

SPOTLESS

O *SpotlessCheck* é uma *plugin de formatação* que auxilia na inspeção da qualidade de códigos. Neste momento mais de 4 mil projetos no GitHub usam esse *plugin* para inspeção. *Spotless* é um projeto *open-source* que está disponível gratuitamente em um repositório no GitHub onde a comunidade pode fazer uso e aprimoramentos. O *plugin* oferece todos os recursos de desempenho integrados ao Gradle e também os recursos de correções automáticas. Ele tem suporte para diversas linguagens de programação como Java, Kotlin, C/C++ e Python. (DIFFPLUG, s. d.)

SUPER-LINTER

Super-linter é um repositório do GitHub onde há uma combinação de vários *linter*s para ajudar na validação de código. Tem como objetivo prevenir que códigos quebrados sejam levados a *branch master* do repositório, ajuda a estabelecer boas práticas de códigos em várias linguagens e automatizar processos para simplificar as revisões.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

O Super-linter capta e reporta os erros no console e as sugestões são apresentadas, mas não corrigidas automaticamente (GITHUB INC, s. d.)

GITHUB

GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e/ou *Open Source* de qualquer lugar do mundo. GitHub é amplamente utilizado por programadores para divulgação de seus trabalhos ou para que outros programadores contribuam com o projeto, além de promover fácil comunicação através de recursos que relatam problemas ou mesclam repositórios remotos. (GITHUB INC, s. d.)

GITHUB ACTIONS

GitHub Actions é uma plataforma de integração contínua (CI/CD) que permite automatizar a sua compilação, testar e fazer um pipeline de implementação. É possível criar fluxos de trabalho que criam e testam cada *pull request* no seu repositório, ou implantar *pull requests* mesclados em produção. GitHub Actions vai além de apenas DevOps e permite a execução de fluxos de trabalho quando outros eventos ocorrem no repositório. Por exemplo a execução de um fluxo para adicionar automaticamente as *tags* apropriadas sempre que alguém cria um novo problema no repositório. (GITHUB INC, s. d.)

GRADLE

O Gradle é uma ferramenta de compilação automática de código aberto com o propósito de flexibilizar e aumentar o desempenho do projeto. (GRADLE INC, s. d.)

Com ele é possível realizar tarefas de programação em seu arquivo de configuração, pois os arquivos Gradle são baseados em *scripts* e também existe a possibilidade do uso de *plugins* que adicionam funcionalidades ao projeto. (HNZ)

2 PIPELINE PARA CHECAGEM DE CÓDIGOS

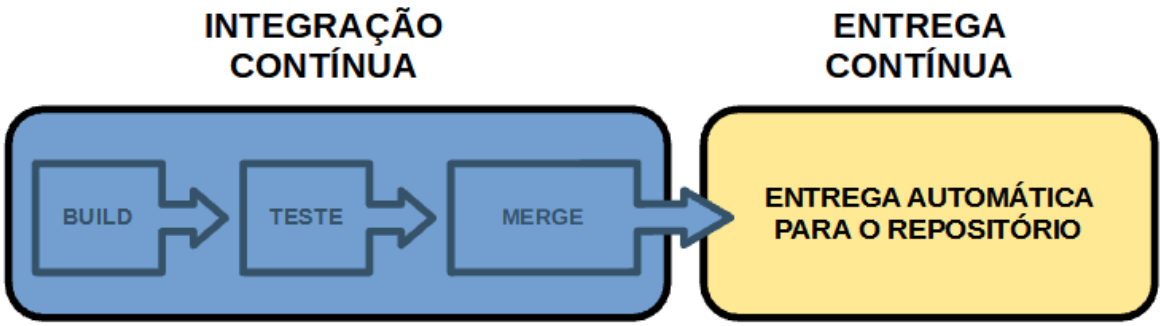
A meta primordial do processo de manutenção é facilitar a acomodação de mudanças, que são inevitáveis em produtos de *software*, reduzindo a quantidade de esforço na fase que se encontra. Este processo deve ser amplo o bastante para suportar manutenção para a correção de um erro, para a modificação de uma funcionalidade existente ou a inclusão de funcionalidade. O que importa é que haja um processo formal e documentado para que a mudança seja efetivada ou rejeitada, registrando as informações, independente do veredicto da mudança (CORDEIRO, [20--]).

Com princípio de garantir a maior qualidade em um projeto de s*oftware*, este trabalho apresenta maneiras de garantir a qualidade esperada na entrega de códigos. Usando um pipeline

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

como base, tendo início a validação pelo Super-Linter da estética do código, ou seja, se ele está de acordo com os padrões estabelecidos, por exemplo a falta de ";" ou o uso de variáveis erradas. Como segunda etapa, após a primeira validação, há uma correção automática utilizando uma task do *spotless* seguindo de uma nova validação, desta vez do *spotless*, a fim de gerar documentações claras para garantir que não há mais possíveis problemas com o código. A figura 1 apresenta um exemplo de um pipeline que está sendo usado no projeto deste artigo:

Figura 1

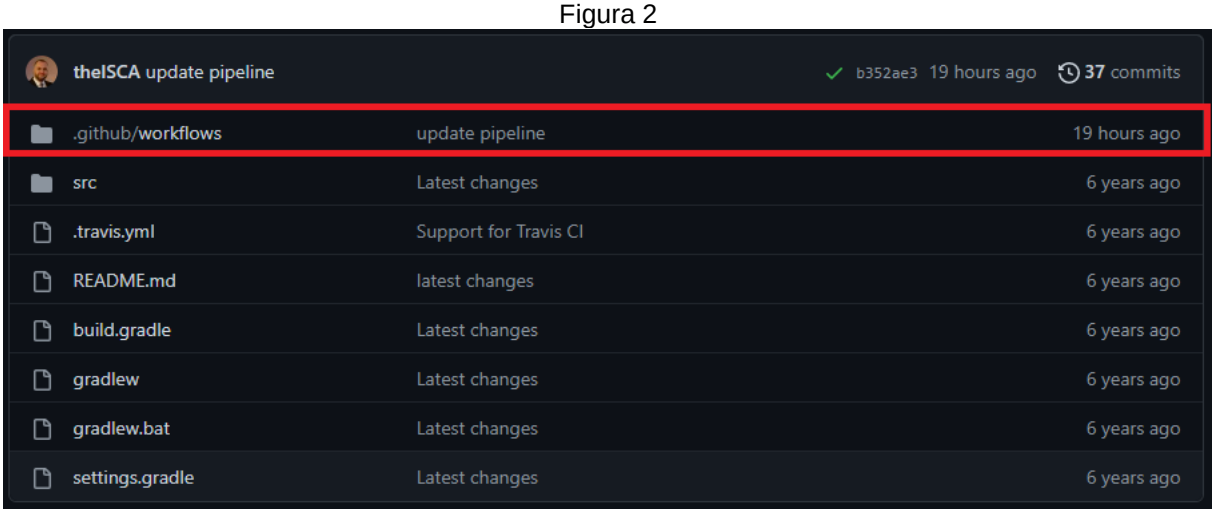


Fonte: Autor

A criação do pipeline teve como ponto de partida com a configuração do espaço de trabalho, o GitHub. Após feito o *fork* – uma cópia do repositório com possibilidades de experiências sem comprometer o original – do projeto fornecido pelo orientador, deu-se início as atividades para a criação das tarefas que são executadas para a garantia da qualidade. A primeira etapa foi a criação da pasta *workflows* no diretório do projeto no GitHub, onde nesta pasta está o arquivo para a execução do pipeline no GitHub actions. (Figura 2)

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Figura 2



Fonte: Autor

O arquivo tem como configuração as etapas que o pipeline funcionará cada vez que ele for ativado. O seguinte código se baseia em uma extensão .yml (YAML Is not Markup Language) que é um arquivo de tecnologia para documentos. (Figura 3)

Figura 3

```yaml
name: TCC validator

on:
  push:
    branches:
      - master
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - run: java -version

  lint:
    name: Lint Scan
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3
        with:
          fetch-depth: 0
      - name: Lint Code Base
        uses: github/super-linter@v4
        env:
          VALIDATE_ALL_CODEBASE: false
          DEFAULT_BRANCH: master
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Fonte: Autor

Como vemos na figura 3, sempre que houver um *push* na *branch master*, ou seja, um envio do repositório local para o remoto na parte principal dele, o pipeline é ativado e começa a funcionar,

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

tendo como primeiro passo um *build* para validar que tudo está correto, seguindo para o primeiro *lint* do projeto, onde nesse *Super-Linter* somente são relatados os problemas na saída do console mas não corrigidos automaticamente.

Nesta etapa seguinte é onde o SpotlessCheck cumprirá o seu papel após a validação do Super-Linter. A vantagem é que no *Spotless* existe uma opção de correção automatizada fazendo com que o processo e o desenvolvedor ganhem tempo de produção. Essa etapa se dá com o spotlessApply. Seguindo a varredura, assim que as correções são feitas, o *spotless* realiza uma nova checagem no código – spotlessCheck – para garantir que tudo foi corrigido de maneira correta. (Figura 4)

Figura 4

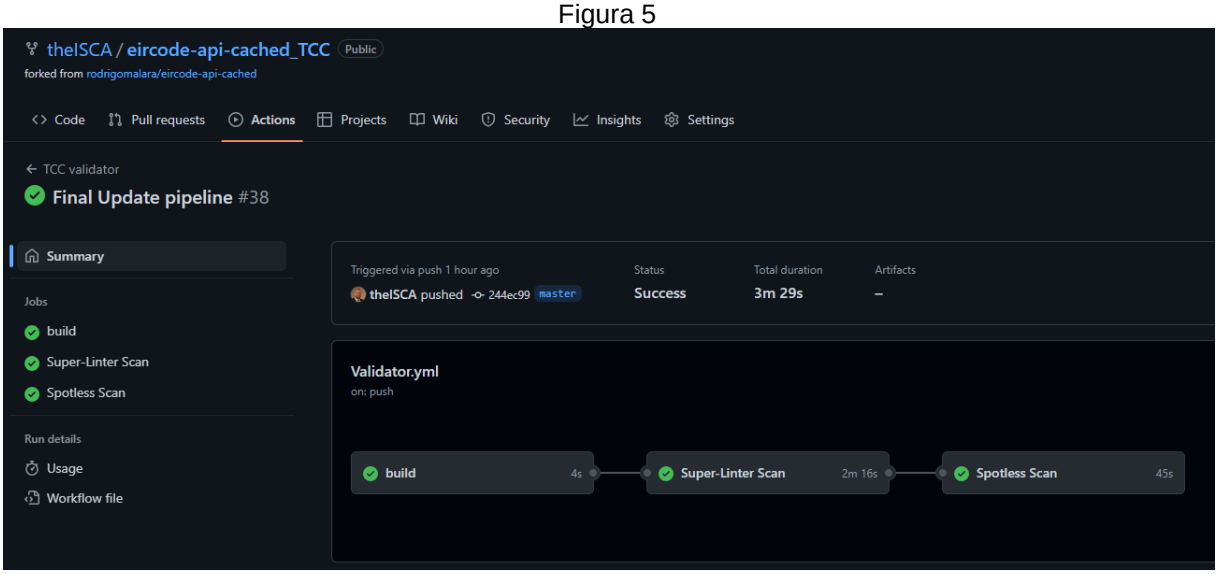
```
35    spotless:
36      name: Spotless Scan
37      runs-on: ubuntu-latest
38      needs: lint
39      steps:
40        - uses: actions/checkout@v2
41          with:
42            fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
43        - uses: gradle/wrapper-validation-action@v1
44        - name: Set up JDK 11
45          uses: actions/setup-java@v1
46          with:
47            java-version: 11
48        - name: Spotless Apply
49          run: |
50            chmod +x ./gradlew
51            ./gradlew spotlessApply
52        - name: Spotless Check
53          run: |
54            chmod +x ./gradlew
55            ./gradlew spotlessCheck
```

Fonte: Autor

Durante o processo de funcionamento do *pipeline,* é possível acompanhar as etapas que cada passo dado é percorrido. Assim que completado, é registrado se o que foi realizado está correto, sendo possível acessar uma documentação detalhada do que foi escaneado. (Figuras 5 e 6)

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Figura 5



Fonte: Autor

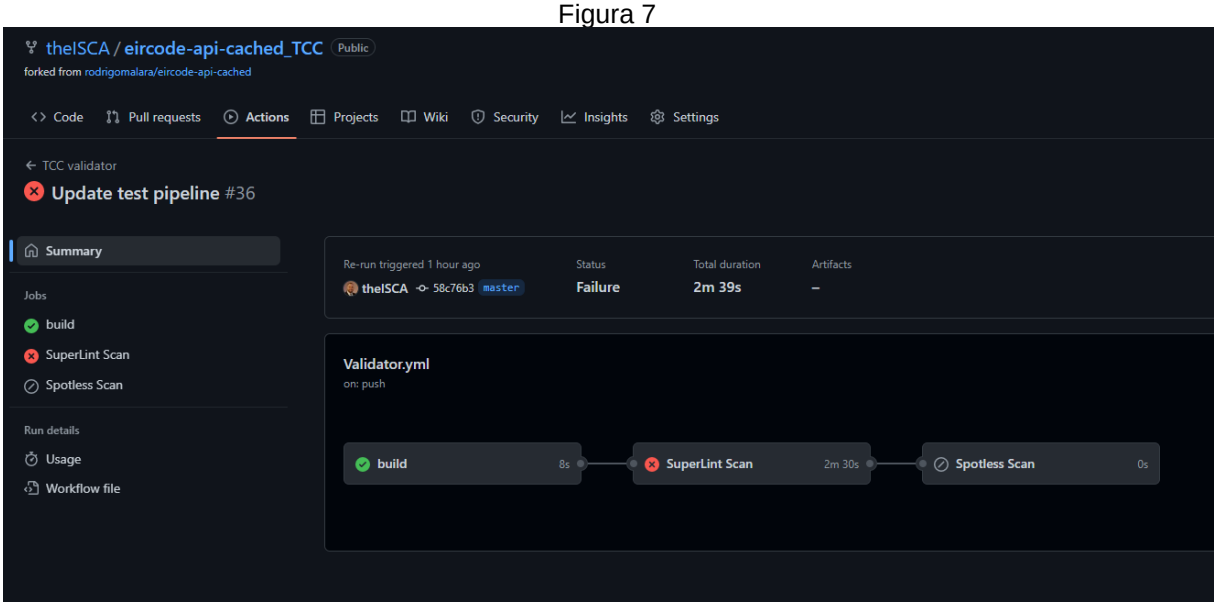
Figura 6

Fonte: Autor

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

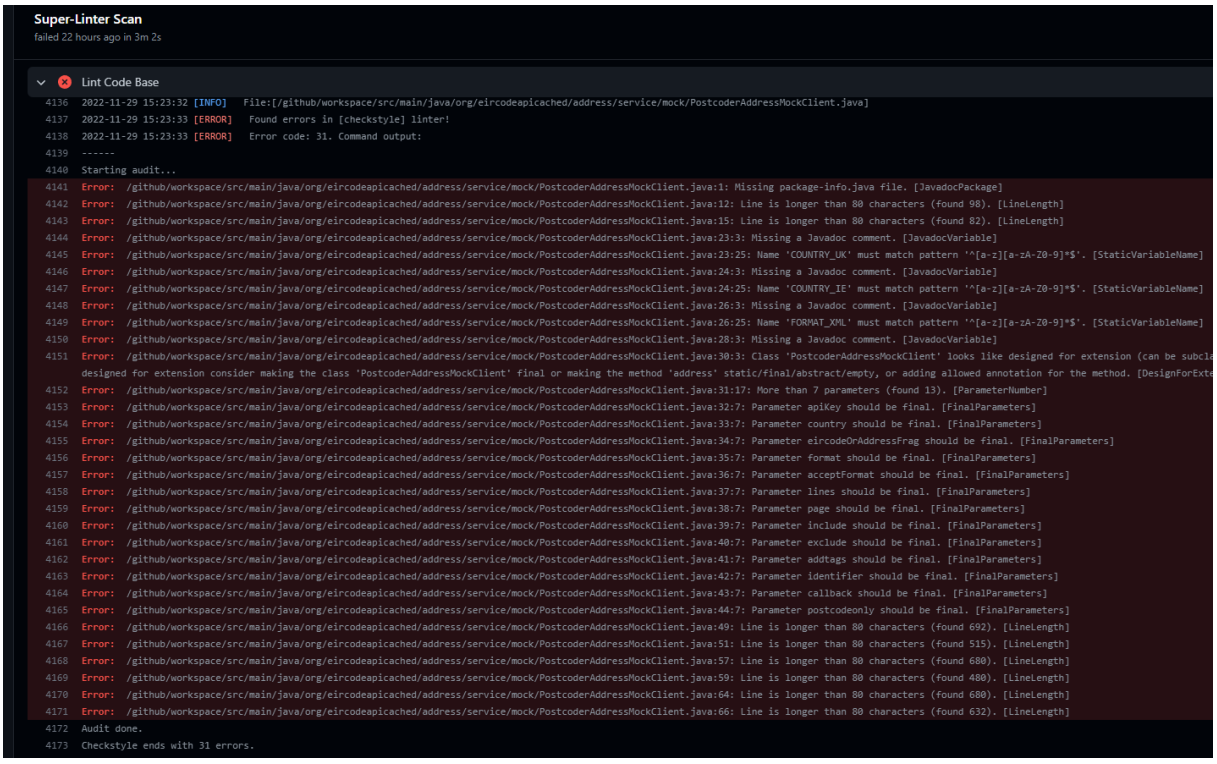
É importante relatar que quando o processo do *pipeline* não dá certo, uma documentação também é gerada a fim de mostrar os erros e bugs que foram encontrados, tanto no código quanto no próprio desenvolvimento do *pipeline.* (Figuras 7 e 8)

Figura 7



Fonte: Autor

Figura 8



Fonte: Autor

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Depois que tudo ocorreu de maneira esperada e que os resultados baseados nos aspectos da qualidade de código foram atingidos, o desenvolvedor, após realizar essa entrega, pode continuar para a próxima tarefa à qual foi designado, dando andamento no projeto.

CONCLUSÃO

A preocupação com a qualidade foi o motim do desenvolvimento deste trabalho, onde ficou claro que com a quantidade de aplicações presentes no cotidiano do mundo, é essencial a presença de um produto de qualidade. Um bom produto sempre foi visto com bons olhos em qualquer segmento, onde a qualidade sempre foi fator de relevância. *Software* não pode ser diferente. Uma boa aplicação é aquela que não trava, não demora para obter a resposta desejada, é bem otimizada para funcionar em diversos aparelhos.

Qualidade está presente nas últimas etapas de concepção de um projeto não à toa. São nessas etapas onde serão comprovadas que todos os processos foram realizados de acordo com as diretrizes de qualidade do projeto e da empresa. Uma forma de melhorar esses processos é inserir os conceitos de qualidade no decorrer do desenvolvimento das atividades, para que assim a expectativa final de qualidade do projeto seja atingida.

As principais metodologias hoje presentes no ambiente de desenvolvimento de *software* já estão bem estruturadas, mas nem sempre tão presentes na rotina do profissional de desenvolvimento. Assim, trazendo algumas das formas de se fazer uma análise e garantir que o que está sendo entregue esteja de acordo com o que foi proposto, essas etapas apresentadas no decorrer deste trabalho estão voltadas a auxiliar o cotidiano de um desenvolvedor, onde uma das preocupações foi o uso de ferramentas que sejam simples de usar, não fazendo o uso de requisitos específicos e suporte para várias linguagens de programação.

No momento em que toda a etapa de garantia da qualidade durante o processo de desenvolvimento for finalizada, a satisfação de dever cumprido esperará no final de tudo, quando a aplicação está funcionando perfeitamente. Muitas vezes esses processos podem parecer chatos ou irrelevantes, mas, com certeza, são essenciais para um bom desenvolvedor.

REFERÊNCIAS

BEHFOROOZ, A.; HUDSON, F. **Software engineering fundamentals.** New York: Oxford University Press,1996.

CORDEIRO, A. M. **Manutenibilidade de Software**. Curitiba: Bayte, [20--]. Disponível em: http://www.batebyte.pr.gov.br/Pagina/Manutenibilidade-de-Software. Acesso em: 10 abr. 2022.

DIFFPLUG. **Spotless plugin for Gradle.** [S. l]: Diffplug, s. d. Disponível em: https://github.com/diffplug/spotless/blob/main/plugin-gradle/README.md. Acesso em: 18 nov. 2022.

ENGHOLM JUNIOR, H. **Engenharia de Software na Prática.** São Paulo: Novatec Editora Ltda, 2010.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

Evans Data Corporation. **Worldwide Professional Developer Population of 24 Million Projected to Grow amid Shifting Geographical Concentrations.** [S. l.]: Evans Data Corporation, 2019.Disponível em: https://evansdata.com/press/viewRelease.php?pressID=278. Acesso em: 10 maio 2022.

GITHUB ACTIONS. **Entendendo o GitHub Actions.** [S. l]: Github Actions, s. d. Disponível em: https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions. Acesso em: 15 maio 2022.

GITHUB. **Super-linter.** Disponível em: https://github.com/github/super-linter. Acesso em: 18 nov. 2022.

GRADLE. **Gradle User Manual.** Disponível em: https://docs.gradle.org/current/userguide/userguide.html. Acesso em: 15 maio 2022.

HNZ. **Gradle:** saiba o que é e como utilizar. [S. l]: HNZ, s. d. Disponível em: https://hnz.com.br/gradle-saiba-o-que-e-e-como-utilizar/. Acesso em: 15 maio 2022.

MARTIN, C. R. **Clean Code:** A handbook of Agile Software Craftsmanship. São Paulo: Pearson Education, 2009.

PRESSMAN, R. S. **Engenharia de Software.** 3. ed. São Paulo: Makron Books, 1995.

REDHAT. **CI/CD:** integração e entrega contínuas. [S. l.]: Redhat, s. d. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd. Acesso em: 22 maio 2022.

REDHAT. **Introdução ao DevOps.** [S. l.]: Redhat, s. d. Disponível em: https://www.redhat.com/pt-br/topics/devops. Acesso em: 22 maio 2022.

REDHAT. **Pipilenes de CI/CD.** [S. l.]: Redhat, s. d. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-cicd-pipeline. Acesso em: 22 maio 2022.

SOMMERVILLE, I. **Engenharia de Software.** 9. ed. São Paulo: Pearson Education Brasil, 2011.

SONARQUBE. **About Sonarqube.** [S. l.]: Sonarqube, s. d. Disponível em: https://www.sonarqube.org/about/. Acesso em: 26 abr. 2022.

STACK OVERFLOW. **Developer Survey Results.** [S. l.]: Stack Overflow, 2019. Disponível em: https://insights.stackoverflow.com/survey/2019#work-_-code-review. Acesso em: 08 nov. 2022.

VALENTE, M. T. **Engenharia de Software Moderna.** São Paulo: Independente, 2013.

VISSER, J. **Building Maintainable Software:** Ten Guidelines for Future-Proof Code. California: O'Reilly Media, Inc, 2016.