THOMAZ FLANKLIN DE SOUZA JORGE

RELATÓRIO TÉCNICO DE ESTÁGIO FUNDECC:

RESULTADOS ALCANÇADOS COM A IMPLEMENTAÇÃO DA ESTEIRA DE INTEGRAÇÃO E ENTREGA CONTÍNUA DURANTE O DESENVOLVIMENTO DE UM SOFTWARE WEB.

LAVRAS - MG
2022

THOMAZ FLANKLIN DE SOUZA JORGE

RELATÓRIO TÉCNICO DE ESTÁGIO FUNDECC: RESULTADOS ALCANÇADOS COM A IMPLEMENTAÇÃO DA ESTEIRA DE INTEGRAÇÃO E ENTREGA CONTÍNUA DURANTE O DESENVOLVIMENTO DE UM SOFTWARE WEB.

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para a obtenção do título de Bacharel.

Prof. Dr. Antônio Maria Pereira de Resende
Orientador

LAVRAS – MG
2022

THOMAZ FLANKLIN DE SOUZA JORGE

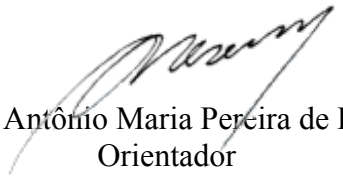
RELATÓRIO TÉCNICO DE ESTÁGIO FUNDECC: RESULTADOS ALCANÇADOS COM A IMPLEMENTAÇÃO DA ESTEIRA DE INTEGRAÇÃO E ENTREGA CONTÍNUA DURANTE O DESENVOLVIMENTO DE UM SOFTWARE WEB.

TECHNICAL REPORT ON INTERNSHIP FUNDECC: RESULTS ACHIEVED WITH THE IMPLEMENTATION OF THE INTEGRATION AND CONTINUOUS DELIVERY TREADMILL DURING THE DEVELOPMENT OF A WEB SOFTWARE.

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para a obtenção do título de Bacharel.

APROVADO em 12 de setembro de 2022.

Prof. Dr. Paulo Afonso Parreira Júnior UFLA
Gerente de Projetos T.I. Édney Pereira Pinto FUNDECC



Prof. Dr. Antônio Maria Pereira de Resende
Orientador

LAVRAS – MG
2022

À minha esposa Karina pelo apoio, carinho, paciência e incentivo em todas as etapas.
À minha avó D. Rosa pelo exemplo de amor e honestidade.
Dedico

AGRADECIMENTOS

Agradeço a Deus primeiramente, por sempre me dar forças, não me desamparar e tornar tudo mais fácil.

Ao meu filho, Paulo César, que é o motivo do meu esforço e por entender alguns momentos em que não pude estar presente.

À minha esposa Karina, que foi a pessoa que possibilitou essa conquista, por estar sempre do meu lado apoiando incondicionalmente e por ser o meu refúgio em momentos difíceis.

À minha avó, por sempre me incentivar com palavras de sabedoria.

Ao meu sogro Hélio, minha sogra Rosângela e minhas cunhadas Ingrid e Isabela, pelo apoio e confiança.

À minha mãe, meu pai e toda minha família pelo apoio e incentivo.

Ao meu orientador, Professor Doutor Antônio Maria Pereira de Rezende, por me proporcionar grandes e importantes aprendizados, que servirão de alicerce para meu futuro profissional, e por aceitar me orientar e conduzir o meu estágio e esse trabalho.

À Tribo IG, equipe que me acolheu durante minha experiência profissional e me mostrou como é ter o ser humano como prioridade em uma equipe de trabalho, onde os colegas se tornam amigos.

À FUNDECC, pela oportunidade de estágio, pela minha posterior contratação e confiança em mim depositada.

A todos que de alguma forma contribuíram

OBRIGADO!

RESUMO

Este relatório apresenta o registro das experiências desenvolvidas durante as atividades de Estágio supervisionado não obrigatório. O principal objetivo do presente trabalho é compilar uma série de apontamentos decorrentes das vivências e aprendizados, que foram oportunizados nas práticas realizadas na Fundecc - Fundação de Desenvolvimento Científico e Cultural - no período de dezembro de 2020 a dezembro de 2021. Para estruturar as discussões apresentadas neste relatório foram realizadas uma pesquisa na documentação das ferramentas utilizadas e consultas bibliográficas, abordando ferramentas, plataformas e processos empregados na elaboração, desenvolvimento e monitoramento da implementação e implantação da esteira CI/CD, em um projeto desenvolvido pela Fundecc. A esteira CI/CD é um produto resultante de uma cultura chamada DevOps, a qual mantém seu foco na criação de uma ponte entre as equipes de desenvolvimento e operações de um projeto de software. Durante o período deste estágio, foi automatizado todo um ciclo de implantação de novas funcionalidades e correções do projeto, tornando todo o processo de desenvolvimento mais rápido e com menos erros, garantindo a qualidade do produto final e levando mais valor ao cliente.

Palavras-chave: Esteira CI/CD. DevOps. Automação de processos. Qualidade de software.

ABSTRACT

This report presents the record of the experiences developed during the non-mandatory supervised internship activities. The main objective of the present work is to compile a series of notes resulting from the experiences and learning, which were provided in the practices carried out at Fundecc - Fundação de Desenvolvimento Científico e Cultural - from December 2020 to December 2021. To structure the discussions presented in this report, a research was carried out in the documentation of the tools used and bibliographic consultations, covering tools, platforms and processes used in the elaboration, development and monitoring of the implementation and implantation of the CI/CD belt, in a project developed by Fundecc. The CI/CD treadmill is a product resulting from a culture called DevOps, which focuses on creating a bridge between the development and operations teams of a software project. During this internship period, an entire cycle of implementation of new features and project corrections was automated, making the entire development process faster and with fewer errors, guaranteeing the quality of the final product and bringing more value to the customer.

Keywords: CI/CD mat. DevOps. Process automation. Software quality.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O relatório técnico de estágio apresentado neste documento é elaborado no âmbito da disciplina PRG 414 – Trabalho de Conclusão de Curso II, visando à conclusão do curso de graduação de Sistemas de Informação na Universidade Federal de Lavras – UFLA.

O estágio foi desenvolvido na Fundecc (Fundação de Desenvolvimento Científico e Cultural) vinculada à UFLA, localizada na Av. Norte UFLA - Aquenta Sol, Lavras - MG, 37200-000, com duração total de 1008 horas.

A Fundecc, através do seu setor de tecnologia da informação, propicia aos seus colaboradores, bolsistas e estagiários a oportunidade de conhecer projetos de software de grande porte voltados para o desenvolvimento ambiental, e esse foi o motivo pela escolha dessa empresa.

Este estágio foi uma oportunidade de entrar em contato com o mercado de trabalho, complementando e aperfeiçoando as competências adquiridas durante a graduação, bem como contribuir positivamente com a equipe.

1.1 Motivação

Como motivação para a realização deste relatório técnico é importante salientar a necessidade de uma experiência profissional para quem deseja seguir uma carreira no mercado de trabalho.

Um Estágio Supervisionado é mais do que apenas atender a requisitos acadêmicos. É uma oportunidade de crescimento profissional e pessoal, bem como uma importante ferramenta que integra escolas, universidades e comunidades.

Além disso, a aprendizagem é muito mais eficiente quando adquirida através da experiência. Na prática, o conhecimento é absorvido de forma muito mais eficaz, de modo que os estagiários podem relembrar atividades durante seus estágios com muito mais frequência do que faziam em sala de aula como alunos. Na prática efetiva, os estagiários têm a oportunidade de compreender alguns conceitos que foram ensinados apenas na teoria. Os

estudantes devem, portanto, perceber o estágio como uma oportunidade única e realizá-lo de forma orientada para os objetivos, dedicada e responsável.

1.2 Objetivos

O objetivo deste relatório é descrever as atividades desenvolvidas durante o estágio realizado na Fundecc, no período de Dezembro/2020 a Dezembro/2021, focando na introdução da cultura DevOps em uma equipe de desenvolvimento de software através da implantação de uma esteira conhecida como CI/CD (*Continuous Integration e Continuous Delivery*). Neste contexto, o estagiário, discente do curso de bacharelado em Sistemas de Informação da UFLA, atuou como analista DevOps no projeto Siout-SC (Sistemas de Outorga de Água de Santa Catarina). O objetivo geral para a realização deste estágio foi aumentar o valor entregue ao cliente por meio de automação de processos.

Para contribuir com o alcance do objetivo geral, foram definidos os seguintes objetivos específicos:

- Conhecer e implantar ferramentas para automação de processos;
- Aumentar a qualidade do software através de realização de testes automatizados;
- Adquirir experiência com o processo de desenvolvimento de software em conjunto com práticas DevOps;
- Ter vantagem no mercado de trabalho, pela vivência na prática de um estágio;
- Desenvolver o trabalho em equipe.

1.3 Organização do Trabalho

Além deste capítulo introdutório, o presente documento conta com a seguinte estrutura: No capítulo 2 é apresentado o referencial teórico, contendo uma base sobre a importância do Estágio Supervisionado para o discente, além de apresentar a empresa em que foi realizado o estágio, sua estrutura organizacional e seu processo de desenvolvimento de software. Ainda no capítulo 2 são apresentada s as principais tecnologias utilizadas para a realização do estágio, com uma introdução sobre Engenharia de Software Moderna e

Metodologias Ágeis, além de introduzir o conceito de Integração Contínua e Entrega Contínua, bem como uma breve descrição sobre o que é DevOps, seguindo com a descrição das ferramentas utilizadas para a realização das tarefas aqui descritas. O capítulo 3 apresenta as atividades desenvolvidas no período do estágio, juntamente com exemplos do que foi desenvolvido. Por fim, no capítulo 4 são apresentadas as considerações finais relatando os resultados obtidos.

2 REFERENCIAL TEÓRICO

Este capítulo aborda a fundamentação teórica que embasa as atividades exercidas durante o Estágio Supervisionado, apresentando uma conceitualização sobre engenharia de software, utilizando a Metodologia Ágil conhecida como *Scrum*, e como a esteira de Integração e Entrega Contínuas contribuem para seu funcionamento, propiciando velocidade e qualidade. Em seguida são apresentadas a empresa escolhida e as tecnologias, ferramentas e métodos utilizados durante a execução da atividade principal deste estágio.

2.1 Estágio Supervisionado

O Estágio Supervisionado é o primeiro contato que o estudante tem com seu futuro campo de atuação, tornando-se um momento de suma importância para o desenvolvimento da carreira profissional.

Ao se comprometer com as atividades, o futuro profissional coloca em prática o que foi estudado ao longo de sua graduação.

> o Estágio Curricular Supervisionado é aquele em que o futuro profissional toma o campo de atuação como objeto de estudo, de investigação, de análise e de interpretação crítica, embasando-se no que é estudado nas disciplinas do curso, indo além do chamado Estágio Profissional, aquele que busca inserir o futuro profissional no campo de trabalho de modo que este treine as rotinas de atuação. (PASSERINI, 2007, p. 30)

Oliveira e Cunha (2006, p. 6) afirmam que

> Podemos conceituar Estágio Supervisionado, portanto, como qualquer atividade que propicie ao aluno adquirir experiência profissional específica e que contribua, de forma eficaz, para sua absorção pelo mercado de trabalho.

Durante o estágio, o acadêmico está envolvido simultaneamente com os conceitos e a prática em um universo real, algumas vezes confrontando a teoria com a realidade vivenciada, desenvolvendo uma visão crítica dos aspectos profissionais.

2.2 A empresa

A Fundecc está localizada dentro da Universidade Federal de Lavras e tem por finalidade apoiar o desenvolvimento de atividades de ensino, pesquisa e extensão bem como os desenvolvimentos institucionais, científicos e tecnológicos da Universidade Federal de Lavras, mediante assessoramento à elaboração de projetos e administração dos recursos financeiros auferidos, etc (FUNDECC, 2022).

Segundo a própria Fundecc, ela é reconhecida como entidade cuja atuação serve de base para que as ideias desenvolvidas na Universidade Federal de Lavras possam se transformar em projetos com resultados imediatos, produtivos, levando a Universidade, além da sua função primordial, à produção de conhecimento e inteligência.

A Fundação tem atuação em todo o país, tendo assessorado projetos diversos em vários Estados da Federação, podendo associar-se a instituições nacionais e estrangeiras.

2.2.1 Negócio

Gestão estratégica de recursos em prol do Ensino, Pesquisa, Extensão e Inovação (FUNDECC, 2020).

2.2.2 Missão

Gerar soluções e oportunidades para a otimização dos propósitos dos coordenadores, pesquisadores e empreendedores acadêmicos para o desenvolvimento do ecossistema de Pesquisa, Ensino e Inovação (FUNDECC, 2020).

2.2.3 Visão

A Fundação, de acordo com sua política de Integridade (FUNDECC, 2020), definiu uma visão para cada persona envolvida em seu funcionamento:

a) Para os nossos clientes:

- Ser referência como principal provedor de soluções para coordenadores, pesquisadores e empreendedores acadêmicos;

b) Para os nossos colaboradores:
   - Ser ambiente de colaboração e proatividade, movido pelo propósito de transformar a sociedade pela Pesquisa, Ensino e Inovação;

c) Para a UFLA:
   - Contribuir para que a UFLA seja mundialmente reconhecida como instituição de excelência no âmbito de sua missão;

d) Para a sociedade:
   - Ser admirada nacionalmente.

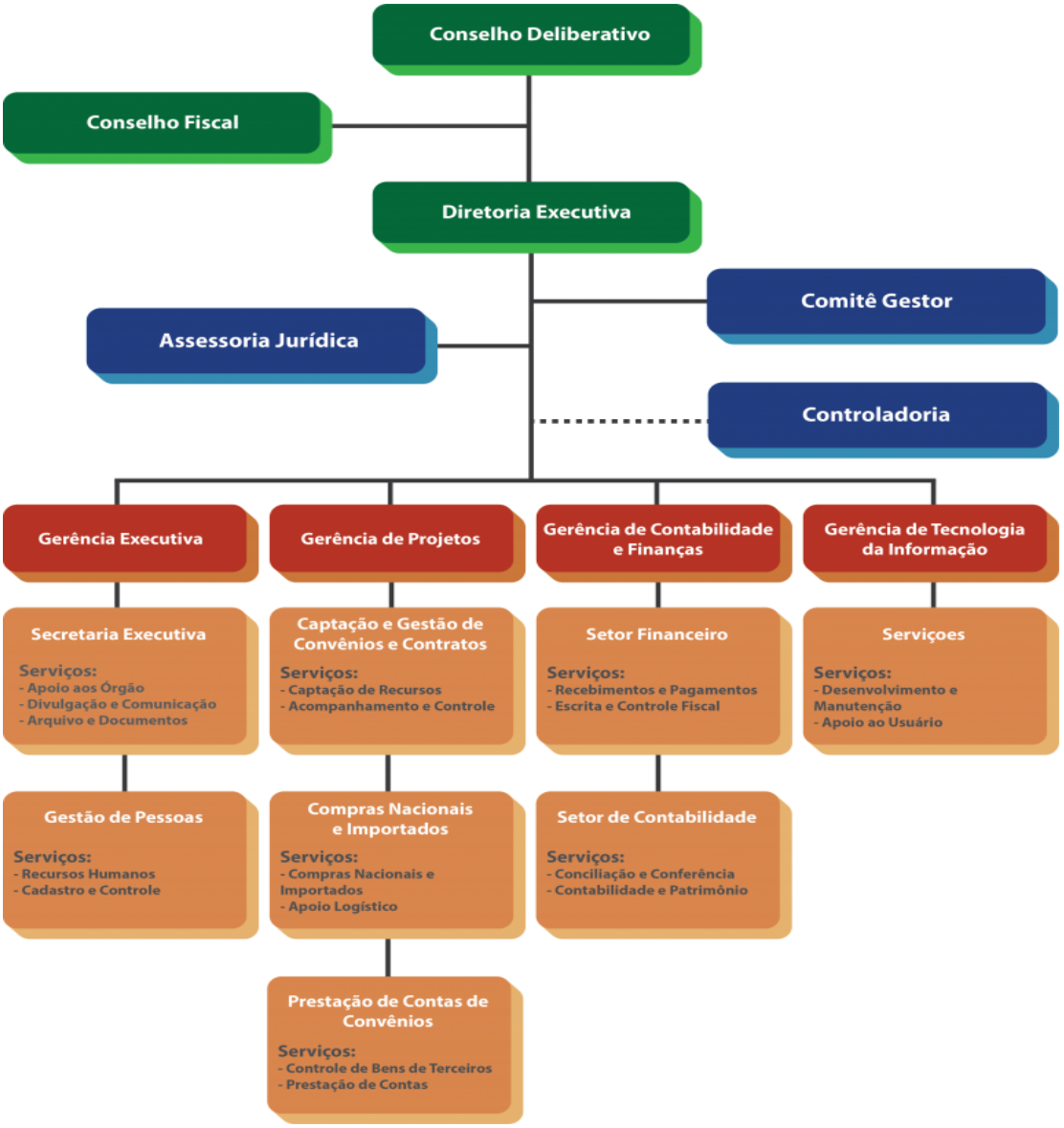
2.3.4 Valores

Excelência, integridade, sustentabilidade, transformação, diversidade e criatividade. (FUNDECC, 2020).

2.3.5 Processo da Organização

Seguindo uma padronização com base em seu organograma, mostrado na Figura 2, a Fundecc é composta por um Conselho Deliberativo, um Conselho Fiscal uma Diretoria Executiva, um Comitê Gestor, uma Assessoria Jurídica, uma Controladoria, quatro gerências sendo Gerência Executiva, Gerência de Projetos, Gerência de Contabilidade e Finanças e Gerência de Tecnologia da Informação.

Figura 1 - Organograma da Empresa



Fonte: Fundecc (2022)

Parte dos colaboradores da Fundecc estão alocados no seu setor de suporte TI (Tecnologia da Informação), tendo como responsabilidade, o desenvolvimento de soluções providas por recursos de computação que visam produção, armazenamento, transmissão, acesso, segurança e o uso de informações. Os projetos desenvolvidos por esse setor são, em

sua maioria, voltados ao meio ambiente, com tratamento de informações geográficas, ou de dados georreferenciados, por meio de softwares específicos e cálculos.

No período deste estágio, o setor suporte TI da Fundação contava com 106 colaboradores, divididos em grupos chamados de tribos. Cada tribo é liderada por um Gerente de Projetos, com a responsabilidade de gestão de 2 a 4 projetos, onde cada projeto tem sua equipe.

O estágio, aqui relatado, foi realizado no setor de suporte TI da Fundecc, em uma de suas tribos nomeada de Tribo IG. No projeto onde a atividade principal deste estágio ocorreu, era utilizado o *framework Scrum* como modelo de trabalho, com a equipe composta por 1 Gerente de Projetos, 1 *Product Owner*, 1 Administrador de Banco de Dados, 4 Desenvolvedores e 2 Analistas de Qualidade, com as seguintes atribuições:

a) Gerente de Projetos: profissional responsável pela realização dos objetivos dos projetos da tribo, sendo suas funções identificar as necessidades, estabelecer objetivos claros e alcançáveis, balancear as demandas conflitantes de qualidade, escopo, tempo e custo e adaptar as especificações, dos planos e da abordagem às diferentes preocupações e expectativas das diversas partes interessadas;
b) *Product Owner*: responsável por direcionar o projeto de acordo com a necessidade do cliente e das demais partes interessadas, além de definir o produto, incrementando-o de acordo com o andamento do projeto, bem como criar a documentação do projeto.
c) Analista de banco de dados: profissional responsável por administrar os bancos de dados referentes ao projeto;
d) Desenvolvedor: responsável por toda a documentação e manutenção do código-fonte, desenvolvimento de novas funcionalidades, correção de "*Bugs"*, implantação do projeto em ambientes para teste, homologação e produção;
e) Analista de Qualidade: responsável por monitorar cada fase do desenvolvimento de um software, com o intuito de garantir que os resultados esperados sejam cumpridos, bem como criar planos de testes, rastrear bugs, desenvolver padrões de qualidade e identificar potenciais problemas para o usuário.

2.3 Engenharia de Software

No mundo moderno, é impossível viver sem utilizar algum software. Empresas dos mais variados tamanhos e setores dependem dos mais diversos sistemas de informação para automatizar seus processos, governos e cidadãos interagem por meio de sistemas computacionais, comércio eletrônico, redes sociais, internet das coisas e uma infinidade de interações, dependem de algum software.

Portanto, pela relevância do software na sociedade, faz-se necessária uma área da computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software (VALENTE, 2020).

A IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) *Computer Society* define que a Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir software. Logo, é comum dizer que é a área da computação que aplica conceitos de engenharia na construção do software.

Para Pressman e Maxim (2016, p.14) “A engenharia de software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade”.

Valente (2020) apresenta as seguintes propostas para modernizar a engenharia de software:

a) Métodos ágeis, como *Scrum*, XP e Kanban;
b) Levantamento ágil de requisitos, incluindo histórias de usuários, MVPs e testes A/B;
c) Projeto de Software, tratando de propriedades de projeto, princípios e padrões de projeto;
d) Arquitetura de Software, incluindo padrões como MVC, microsserviços e *publish/subscribe*;
e) Testes de Software, com ênfase em testes de unidade, testabilidade, cobertura e TDD;
f) *Refactoring*, com exemplos reais de *refactorings* e *code smells*;
g) DevOps, incluindo controle de versões, integração e *deployment* contínuo.

A atividade principal deste estágio consiste em implementar uma parte do DevOps conhecida como CI/CD (*Continuous Integration and Continuous Delivery*) em um software, apresentando os resultados de sua implementação, bem como os impactos nas demais áreas citadas por Valente (2020).

2.4 Metodologias Ágeis

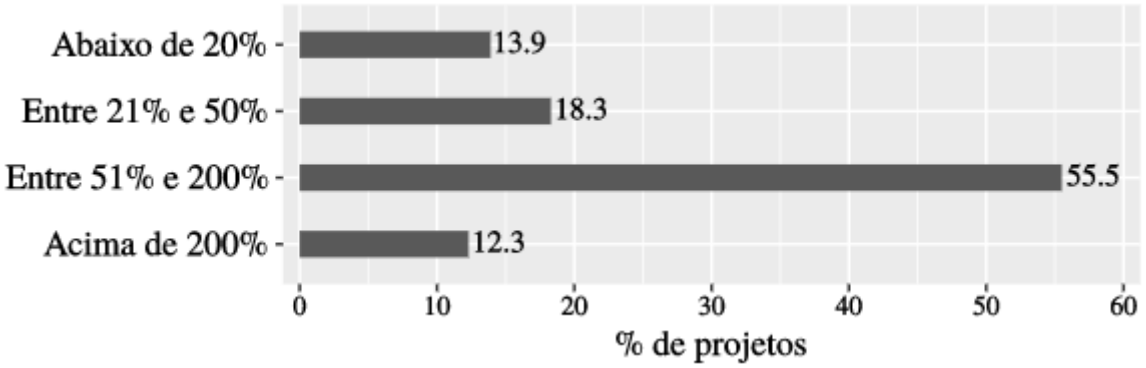
Para trabalhar com metodologias Ágeis, primeiramente devemos compreender que "Os primeiros processos de desenvolvimento de software, propostos na década de 70, eram estritamente sequenciais, começando com uma fase de especificação de requisitos até chegar às fases finais de implementação, testes e manutenção do sistema" (VALENTE, 2020, n. p.).

Valente ainda afirma que, após uma década, começou-se a perceber o quão diferente era o software de outros produtos de Engenharia, pois os cronogramas e orçamentos não eram obedecidos, e muitos softwares sequer eram entregues aos clientes, após anos de desenvolvimento.

No ano de 1994, a empresa de consultoria *Standish Group* publicou um relatório que ficou conhecido como *CHAOS Report*[1], mostrando que mais de 55% dos projetos estouraram os prazos planejados entre 51% e 200%; pelo menos 12% estouraram os prazos acima de 200%, conforme mostra o gráfico da Figura 1.

[1] Relatório produzido pela empresa de consultoria Standish Group no ano de 1994, sobre os projetos de software da época. Disponível em https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf. Acesso em 6 ago. 2022.

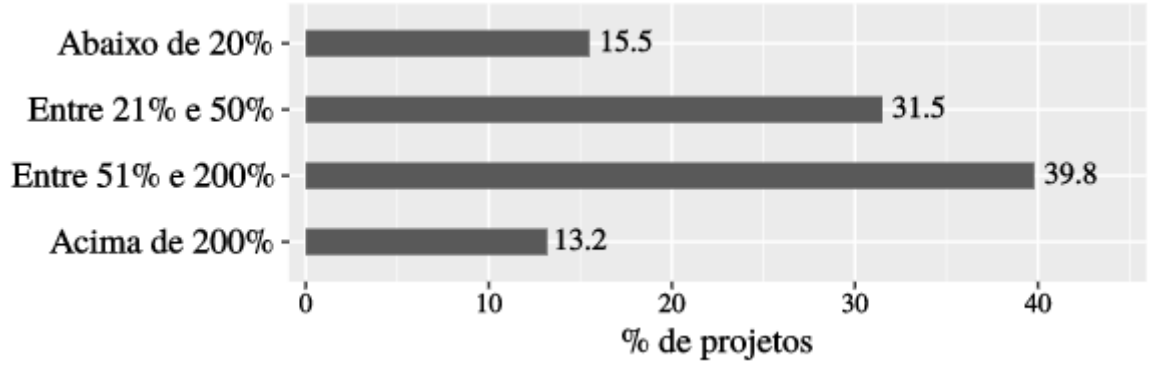
Figura 2 - Gráfico de projetos que estouram o prazo em porcentagem em 1994



Fonte: Valente (2020)

Os custos também não animavam, como mostrado na Figura 3, quase 40% dos projetos estouraram o orçamento entre 51% e 200%.

Figura 3 - Gráfico de projetos que estouram o orçamento em porcentagem em 1994



Fonte: Valente (2020)

A IEEE COMPUTER SOCIETY, publicou em 3 de junho de 2022, um artigo com um levantamento sobre as publicações do CHAOS *Report,* que acontece em média de dois em dois anos, e mostrou em uma tabela como foi a evolução dos projetos entregues ao longo dessas duas décadas (DEFRANCO ; VOAS , 2022).

Tabela 1 - Dados do projeto ao longo dos anos.

	1994	2004	2009	2019	2020
Falha ao concluir	31%	18%	24%	31,1%	19%
Falha parcial	53%	53%	44%	52,7%	50%
Bem sucedidos	16%	29%	32%	16,2%	31%

Fonte: Adaptada de DeFranco e Voas (2022, p. 13).

Na Tabela 1 é possível observar que ao longo de 26 anos houve melhorias na experiência em gerenciamento de projetos, pois observa-se que a taxa de projetos que falharam no orçamento e no tempo, caiu de 31% para 19%. Pode-se perceber que os projetos que falharam parcialmente, no orçamento ou no prazo, referente à segunda linha da tabela e, reduziram em apenas 3%, se mantendo em 50%, mostrando que ainda há muito a melhorar, pois a metade dos projetos continuam estourando o tempo ou o prazo. Ao observar os projetos bem sucedidos, nota-se uma melhora passando de 16% para 31%, no entanto, é inadmissível que em 2020 a taxa de sucesso no desenvolvimento de software ainda é menor que um terço, e uma possível razão para isso "pode ser porque as complexidades do projeto e do sistema aumentaram, enquanto o tempo de entrega foi reduzido", afirmam os autores do artigo da IEEE, DeFranco e Voas (2022).

Em 2001, em uma reunião na cidade norte-americana de *Snowbird, Utah*, um grupo de profissionais propuseram um novo conceito de processo de software, e registraram em um documento chamado Manifesto Ágil[2], com o seguinte texto:

> Por meio deste trabalho, passamos a valorizar:
> Indivíduos e interações, mais do que processos e ferramentas
> Software em funcionamento, mais do que documentação abrangente
> Colaboração com o cliente, mais do que negociação de contratos
> Resposta a mudanças, mais do que seguir um plano.

[2]O Manifesto Ágil é uma declaração de valores e princípios essenciais para o desenvolvimento de software. Disponível em https://agilemanifesto.org/iso/ptbr/manifesto.html. Acesso em 07 Ago. 2022

Com isso, as metodologias ágeis se tornaram uma forma de acelerar entregas de um determinado projeto. Consiste no fracionamento de entregas para o cliente final em ciclos menores.

A partir do Manifesto Ágil, que era genérico e abrangente, foram desenvolvidos vários métodos para auxiliar desenvolvedores a adotarem os princípios ágeis de forma mais concreta e, dentre os quais destaco:

a) *Extreme Programming* (XP): proposto por Kent Beck, em um livro lançado em 1999. Uma segunda edição do livro, incluindo uma grande revisão, foi lançada em 2004;
b) Kanban: possivelmente um sistema de controle de produção que começou a ser usado nas fábricas da Toyota, ainda na década de 50.
c) *Scrum*: proposto por Jeffrey Sutherland e Ken Schwaber, em um artigo publicado em 1995.

Durante o período deste estágio, a equipe onde o estagiário estava inserido seguia o modelo *Scrum*.

2.4.1 Scrum Framework

De acordo com o guia PMBOK (2017), uma metodologia é definida como um sistema de práticas, técnicas, procedimentos e regras usadas por aqueles que trabalham no projeto. Logo, um conjunto de formulários, diretrizes, templates e lista de verificações que podem ser aplicadas a um projeto ou situação específica (KERZNER, 2020, p. 24).

Podemos destacar o *Scrum*, criado por Ken Schwaber e Jeff Sutherland com base no Manifesto para o desenvolvimento de software Ágil, no início da década de 1990. Em 2010, a primeira versão do Guia do Scrum foi aperfeiçoada com pequenas atualizações funcionais (SCHWABER; SUTHERLAND, 2020). O *Scrum* é uma das mais bem sucedidas metodologias, sendo utilizado por mais da metade dos projetos ágeis no mundo (Version One, 2017).

Cavalcanti (2020) define *scrum* como “um *framework* com o qual as pessoas podem lidar com problemas adaptativos complexos, ao mesmo tempo que entregam produtos com o

mais alto valor possível de forma produtiva e criativa". Já Schwaber e Sutherland (2020), os autores do *Scrum*, o definem como um "*framework* leve que ajuda pessoas, times e organizações a gerar valor por meio de soluções adaptativas para problemas complexos". Schwaber e Sutherland (2020) apresentam a teoria do *Scrum* como:

> *Scrum* é baseado no empirismo e lean thinking. O empirismo afirma que o conhecimento vem da experiência e da tomada de decisões com base no que é observado. O lean thinking reduz o desperdício e se concentra no essencial.

Ainda segundo os autores do *Scrum*, afirmam que seus cinco valores: Compromisso, Foco, Abertura, Respeito e Coragem são a base para seu sucesso. O *Time Scrum* deve se comprometer a atingir seus objetivos suportando uns aos outros. Seu foco principal é o trabalho da *Sprint* para fazer o melhor progresso possível em direção às metas. O *Time Scrum* é composto por indivíduos, com os seguintes papéis:

a) *Product Owner*: gerir o *Product backlog*, maximizar o valor do produto e do time de desenvolvimento;
b) *Development Team*: responsáveis diretos pelo desenvolvimento e entrega do produto em versões utilizáveis a cada ciclo de *Sprint*;
c) *Scrum Master*: garantir que o *Scrum* seja entendido e aplicado e ajudar aqueles que estão fora do time a entender quais de suas ações são úteis, ou não.

O *Scrum* possui alguns eventos, conhecidos também como *Time-Boxes*. São eventos com um tempo limite definido, podendo ser encerrados antes, caso ele já tenha atingido seu objetivo. É de responsabilidade do *Scrum Master* garantir que os eventos ocorram durante o projeto. (DUARTE, 2016)

O principal evento do *Scrum* é conhecido como *Sprint*, período limitado entre duas e quatro semanas, onde ocorrem todos os outros eventos, e no fim deve ser entregue um incremento utilizável do produto pela equipe. A *Sprint* inicia-se com a *Sprint Planning*, onde em um prazo de duração de até 8 horas, o *Product Owner* apresenta as necessidades do produto, e baseando nisso e na capacidade produtiva da equipe, todas as ações da *Sprint* são planejadas e deliberadas.

Outro evento importante do Scrum é o *Daily Scrum*, uma reunião diária, com duração máxima de 15 minutos, com o objetivo de alinhar o status das atividades entre os membros da equipe. Durante essa reunião, todos os membros da equipe devem, de forma breve, responder a 3 perguntas:

a) O que fiz desde o último *Daily Scrum*?
b) O que pretendo fazer até o próximo *Daily Scrum*?
c) Quais impedimentos me impedem de realizar minhas atividades?

Respondendo a essas perguntas, toda a equipe fica ciente do desempenho das atividades, podendo direcionar o foco onde mais precisa.

Ainda existem o *Sprint Review,* com duração de 4 horas, apresentando o incremento do produto para os *stakeholders*[3], e por último ocorre a *Sprint Retrospective*, com duração de 3 horas, com o intuito de discutir os resultados da *Sprint* finalizada, apontando o que pode ser melhorado para as próximas *Sprints*.

Para analisar as particularidades em utilizar o *Scrum*, podemos compará-lo com a abordagem de desenvolvimento de software no modelo cascata[4] tradicional, com as etapas do desenvolvimento sendo concluídas em sucessão, onde uma etapa só inicia após a conclusão e aceitação da etapa anterior.

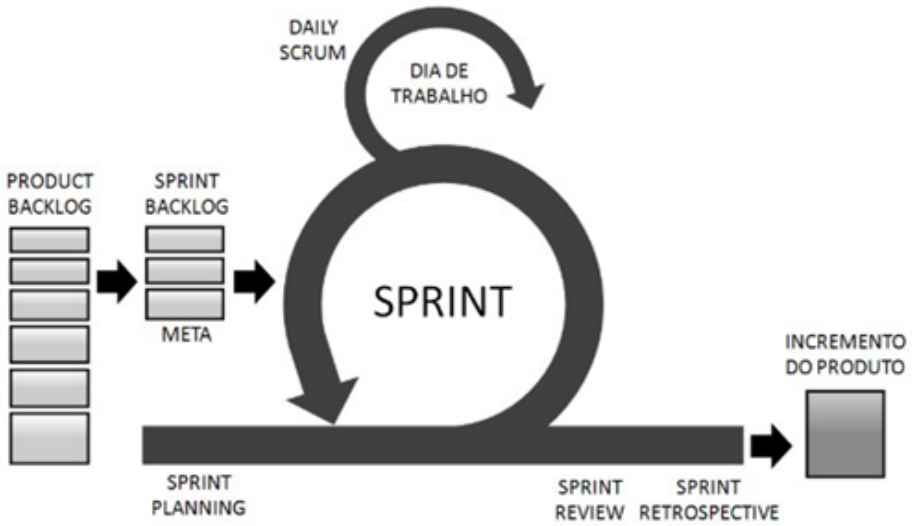
Já na metodologia *Scrum*, é enfatizada a entrega rápida com recursos completos, no período de uma *Sprint*, conforme apresentado na Figura 4. A equipe trabalha seguindo um nível de prioridade definido e avaliado pelos *stakeholders*. Todas as tarefas, conhecidas como *User Stories*, estão em uma lista, chamada *Product Backlog*. As tarefas que serão realizadas na *sprint* estarão em uma lista conhecida por *Sprint Backlog*.

---

[3] De forma prática, o *stakeholder* é qualquer pessoa física ou jurídica, que de alguma forma é impactado pelas ações do projeto, direta ou indiretamente.

[4] O modelo cascata é uma abordagem linear de gerenciamento de projetos, na qual os requisitos das partes interessadas e dos clientes são coletados no início do projeto e, em seguida, um plano sequencial é criado para acomodar esses requisitos. Veja mais em https://www.fm2s.com.br/modelo-cascata/. Acesso: 7 Ago. 2022.

Figura 4 - Interações do Scrum



Fonte: Oliveira (2019)

Como é necessário o incremento constante do software, e a equipe de desenvolvimento está sempre integrando novas funcionalidades e correções ao código, é necessário ter ferramentas e métodos para minimizar e solucionar os problemas que essa integração simultânea pode causar. Uma alternativa para essa solução é o CI/CD.

2.5 Integração e Entrega Contínua (CI/CD)

O CI/CD é um método de entrega frequente de aplicativos para as partes interessadas e essa automação[5] pode ser aplicada durante o estágio de desenvolvimento para resolver problemas de integração de código.

Todo o ciclo de vida do aplicativo deve ser monitorado, incluindo teste, integração e implantação. Coletivamente, essas práticas são chamadas de "*pipelines CI/CD*" e devem ser compatíveis com as metodologias ágeis (RED HAT, 2018).

[5] segundo a RedHat, A automação de TI, às vezes chamada de automação de infraestrutura, é o uso de softwares para criar instruções e processos reproduzíveis para substituir ou reduzir a interação humana com os sistemas de TI.

O CI (*Continuous Integration*) ou integração contínua, se refere às modificações realizadas pela equipe de desenvolvimento no código da aplicação. Essas modificações devem ser automaticamente testadas e carregadas regularmente em um repositório remoto compartilhado, que no caso deste estágio, é usado o Gitlab. Após todos os testes serem realizados sem erros ou falhas, as modificações que estão no repositório online devem ser implantadas em um ambiente de produção, para serem usadas pelos clientes. Essa etapa é o chamado CD (*Continuous Delivery*) ou entrega contínua.

O *pipeline* de CI/CD sólido é a implantação contínua, complementando a entrega contínua, automatizando o lançamento de compilações prontas para produção. A implantação contínua depende muito da automação otimizada dos testes.

No mercado, existem diversas ferramentas para realizar a automação do pipeline de CI/CD, como por exemplo GitLab CI/CD, Bitbucket Pipelines, CircleCI, Jenkins entre outros.

A execução do *pipeline* apresentado neste estágio é criado usando a ferramenta Jenkins, utilizando o GitLab como repositório online compartilhado, uma ferramenta para análise estática de código conhecida como Sonarqube, o Cypress como ferramenta para testes automatizados de interface e o aplicativo de chat online Discord[6].

2.6 DevOps

O nome DevOps é uma combinação dos termos “desenvolvimento” e “operações”, e pode ser entendido como uma ponte entre essas duas equipes. Aborda cultura, automação e design de plataforma, agregando mais valor ao negócio e aumentando a capacidade de resposta às mudanças, entregando serviço de alta qualidade de forma rápida (REDHAT, 2018).

Apesar de sua definição, a metodologia DevOps ultrapassa a barreira de seu significado, incluindo questões de segurança, maneiras colaborativas de trabalhar, análise de dados entre outros conceitos e práticas.

[6] Discord é um aplicativo de voz sobre IP proprietário e gratuito, projetado inicialmente para comunidades de jogos.

O movimento DevOps começou entre 2007 e 2008, quando as equipes de operações e de desenvolvimento eram separadas, dificultando toda comunicação. As duas equipes tinham objetivos separados, e muitas vezes concorrentes, uma equipe escrevia o código e a outra implantava e dava suporte. A ideia de juntar os dois times começou em fóruns online, com conversas dirigidas por Patrick Dubois, Gene Kim e John Willis (RED HAT, 2018).

> As equipes que trabalham em silos geralmente não aderem ao 'pensamento sistêmico' do DevOps. 'Sistema pensado' é estar ciente de como suas ações não afetam apenas sua equipe, mas todas as outras equipes envolvidas no processo de lançamento (LOUZADA, 2019).

Para o funcionamento do DevOps é necessário uma cultura colaborativa alinhada aos princípios *open source* e abordagens transparentes e ágeis (REDHAT, 2018). O fluxo DevOps é simbolizado pelo símbolo do infinito, fazendo uma alusão de que o processo não tem um fim (FIGURA 5).

Figura 5 - DevOps

Fonte: blog 4linux[7]

A Tabela 1, mostrada anteriormente no capítulo 2.4, apresenta uma taxa de sucesso nos projetos de software de apenas 31%, e isso pode ser melhorado utilizando a cultura DevOps, através de automação de processos manuais. A redução do tempo é adquirida ao automatizar o controle de versões e configurações, integração de código, *builds* e *deploys*,

[7] Disponível em: https://blog.4linux.com.br/beneficios-do-devops/. Acesso em: 06 ago. 2022.

testes e verificação de qualidade de código fonte. Além disso, o cálculo do orçamento se torna mais confiável e a previsibilidade do projeto aumenta, devido a diminuição de falhas humanas em processos repetitivos. A redução dos erros no processo de desenvolvimento, possibilita um aumento no tempo disponível para desenvolvimento, reduz os conflitos e o sentimento de cobrança, o que naturalmente é refletido na quantidade e qualidade de funcionalidades realizadas no mesmo espaço de tempo.

Para que tal ganho ocorra, a empresa e equipe tem o desafio de mudar sua cultura e se esforçar para reduzir a rotatividade de colaboradores.

2.7 Tecnologias e Ferramentas utilizadas

Este Capítulo tem por objetivo apresentar as tecnologias e *frameworks* utilizados pelo estagiário no desenvolvimento das atividades vinculadas à sua área de atuação.

2.7.1 Jenkins

O Jenkins é um servidor de automação de código aberto. Gerencia e controla os processos de entrega de software, construindo, documentando, testando, empacotando e implantando, além de ser possível integrar com diversas ferramentas externas e *plugins*. Com ele é possível observar qualquer mudança em repositórios online, como Github, GitLab ou Bitbucket, disparando uma construção e execução da aplicação com essas alterações (JENKINS, 2021).

Originalmente chamado de Hudson, o Jenkins foi iniciado em 2004 pelo desenvolvedor da Sun Microsystems, Kosuke Kawaguchi, que estava cansado de receber notificações de que seu código estava constantemente quebrando a compilação. Kohsuke então criou o Jenkins para testar o código antes de enviar em um repositório. Jenkins rapidamente se tornou um sucesso entre os colegas de trabalho, até que abriu seu código, espalhando por todo o mundo. (GOCACHE, 2021).

> Fazer builds e deploys manuais leva um precioso tempo e, quando tudo fica automatizado com o Jenkins, parece inacreditável o tempo que era gasto fazendo essas tarefas. E você se questiona como não tinha automatizado isso antes. (BOAGLIO, 2016, p. 7)

Ao configurar corretamente o Jenkins, a equipe de desenvolvimento automatiza a maioria dos processos de criação do software, focando seu tempo no que realmente interessa, a Figura 6 mostra como é o ciclo de desenvolvimento manual e com o Jenkins.

Figura 6 - Comparativo uso do Jenkins

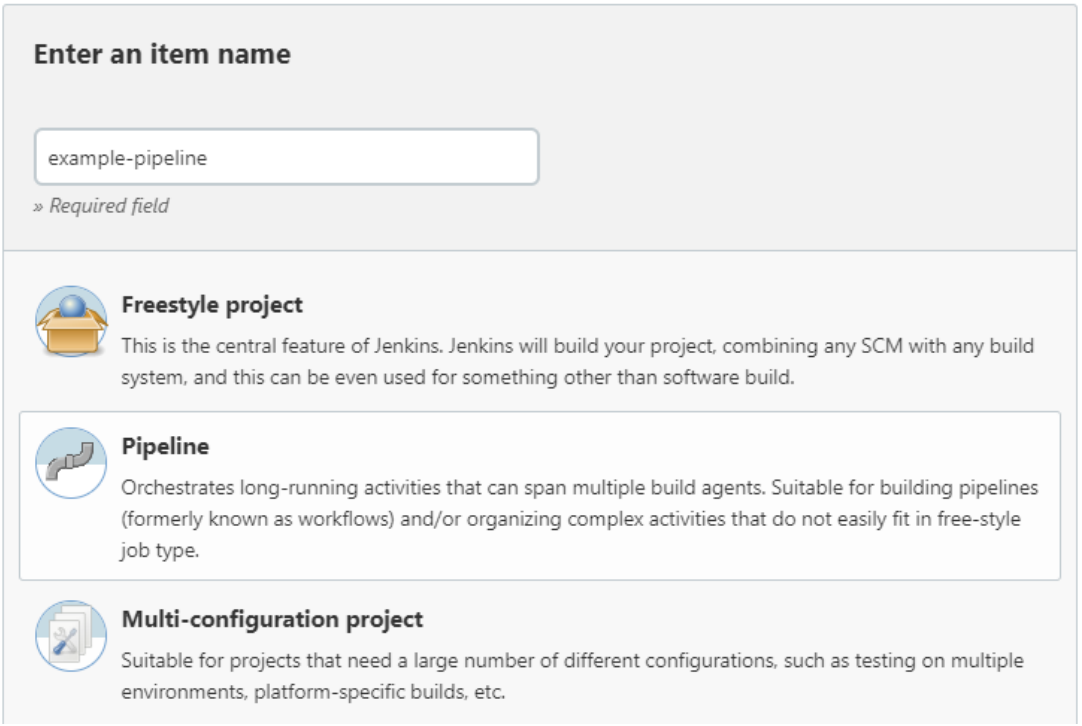
Fonte: Boaglio (2016, págs. 14-16)

O Jenkins permite orquestrar todo o pipeline de entrega de software, juntamente com a cultura DevOps, acelerando potencialmente a entrega de software (GOCACHE, 2021). A solução foi adotada mais ampla para entrega contínua no mundo, segundo publicação do site Gocache (2021). A comunidade do Jenkins oferece mais de 1700 *plugins*, integrando o Jenkins a muitas ferramentas. Seu repositório no Github possui, na data em que este documento foi produzido, 1,7 mil *forks*[8] e 19,2 mil estrelas (JENKINS, 2021).

[8] Fork é uma cópia de um repositório. Essa cópia vira um clone do estado atual do repositório, fazendo assim com que você possa experimentar mudanças e funcionalidades novas sem precisar utilizar o repositório principal.

Você pode configurar o Jenkins de algumas formas distintas, pela interface, ou por *script* escrito em uma sintaxe *Groovy*[9] conforme mostrado nas Figura 7 e Figura 8 respectivamente.

Figura 7 - Interface gráfica Jenkins



Fonte: Jenkins CD (2021)

[9] Linguagem de programação também conhecida como Apache Groovy, é compatível com a sintaxe Java, orientado a objetos e desenvolvido para a máquina virtual Java (JVM)

Figura 8 - Script Jenkinsfile

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Fonte: Jenkins (2021)

Alguns conceitos são fundamentais para melhor entender o Jenkins. Devido à clareza na construção das definições, houve a opção pelos conceitos explicitados no livro *Jenkins - Automatize tudo sem complicações* de Fernando Boaglio (2016, págs. 15-17).

> *Job*: É uma tarefa que o Jenkins deve fazer. Normalmente tem alguns parâmetros de execução, juntamente com alguns procedimentos que podem ser feitos antes ou depois de sua execução. [...]
> *Build*: É a construção, uma execução de um job (tarefa), por exemplo, o procedimento de montar um pacote, que pode envolver download, compilação ou testes. O *build* pode ser considerado como uma instância de um *Job* de *deploy*. O *build* pode ter diversos status: Falha no *build*, *build* instável, *build* feito com sucesso, *build* pendente, *build* cancelado e *build* desligado. [...]
> *Pipeline*: Um *job* pode ter dependência com outro *job*; se um for acionado, ao terminar, pode chamar o outro automaticamente, criando uma cadeia de *jobs* enfileirados. [...]
> *Plugins*: A integração do Jenkins, o nosso super executor de tarefas, com os diversos tipos de servidores e sistemas é feita por meio de plugins, não só para melhorar um funcionamento existente, mas também para criar um que não existe. [...]
> *Artifact* (artefato): é o pacote resultante de um build executado com sucesso. Ele pode ser um arquivo pom.xml, JAR, WAR, EAR etc. [...]

> *Dashboard* (painel principal): é o local em que temos uma visão completa da execução de todos os jobs do Jenkins. Ele pode ser dividido em *views* específicos de alguns *jobs*, mas tem como objetivo dar uma visão como um todo.

Essas foram as definições mais relevantes para o trabalho, entretanto o livro *Jenkins - Automatize tudo sem complicações* de Fernando Boaglio (2016), bem como a documentação da própria ferramenta, possibilita um maior entendimento sobre o assunto.

2.7.2 Git e GitLab

De acordo com a documentação oficial, "Git é um sistema de controle de versão distribuído de código aberto e gratuito, projetado para lidar com tudo, de projetos pequenos a grandes" (GIT, 2022).

Foi desenvolvido inicialmente em 2005 pelo mesmo criador do Kernel do Linux[10], Linus Torvalds, e logo se tornou o gerenciador de códigos-fonte mais popular do mundo.

A principal função do Git é a possibilidade de fazer o controle de versões de modo colaborativo, possibilitando que o mesmo arquivo seja modificado ao mesmo tempo por duas pessoas sem que nenhum código seja sobrescrito por outra alteração.

O Git utiliza o conceito de ramificação ou *branch*, onde cada *branch* é uma linha do tempo que possui marcos ou *commits*[11], e nesse *branch* os arquivos podem ser alterados livremente sem impactar outras ramificações, logo é possível trabalhar em ramos separadamente, desenvolvendo novas funcionalidades ou fazendo correções de *bugs*, e então quando decidirem juntar os dois ramos, o Git utiliza o conceito de mesclagem ou *merge*[12] para isso. (GIT, 2022)

---

[10] Linux é um Sistema Operacional, assim como o Windows e o Mac OS, que possibilita a execução de programas em um computador e outros dispositivos.

[11] um commit é a realização de um conjunto de mudanças provisórias permanentes, marcando o fim de uma transação

[12] O *comando git merge* permite que você pegue as linhas de desenvolvimento independentes criadas pelo *git branch* e as integre em uma ramificação única

A Figura 9 mostra quatro ramos sendo atualizados paralelamente ao longo do tempo através de *commits*, e então o *branch feature* sendo mesclado no *branch develop* através do processo de *merge.*

Figura 9 - Representação de *branches git*

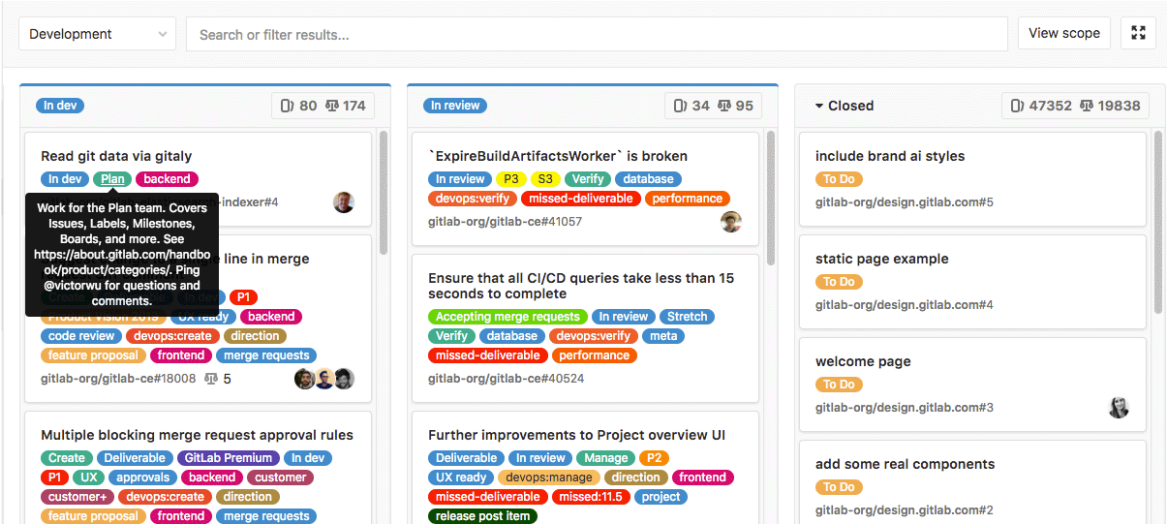
Fonte: Anônimo

O *branch Main,* representado pela cor verde, reflete o sistema em produção, onde são definidas as versões 0.1, 0.2 e 1.0. Para o desenvolvimento do projeto, é criada uma ramificação da *Main,* nomeada como Develop e representada pela cor laranja, partindo do ponto onde foi definida a primeira versão. Todas as funcionalidades a ser desenvolvidas são ramificações da *branch Develop*, chamadas de *Feature* e na simbolizadas pela cor azul, que ao ser finalizada a modificação essa funcionalidade agregada a *branch Develop*, que futuramente será juntada (mergeada) no *branch Main.*

O GitLab é uma plataforma de código aberto, com a função de hospedagem de código-fonte permitindo que várias pessoas contribuam para um mesmo projeto. O GitLab é muito mais que um gerenciador de repositórios, pois além de poder ser instalado e gerenciado em servidores próprios, ainda oferece uma solução completa em DevOps, gerenciamento de projeto, com ferramentas que auxiliam o acompanhamento do projeto em em metodologias

ágeis, como o Kanban[13] e sistema de Issues (GITLAB, 2022), que é definida como "um ponto ou assunto em discussão ou disputa ou um ponto ou assunto que não está resolvido e está sob discussão ou sobre o qual existem pontos de vistas opostos ou desacordos" (PMBOK, 2017), a Figura 10 é um exemplo desse quadro.

Figura 10 - Quadro Kanban com Issues do GitLab.com



Fonte: GitLab (2022)

2.7.3 Sonarqube

Existem várias maneiras de determinar e garantir a qualidade do software em desenvolvimento. No entanto, a maioria só é executada após a entrega pelo menos parcial do aplicativo por meio de vários testes, como testes exploratórios. Garantir a qualidade do software é essencial, razão pela qual aplicativos conhecidos podem ser associados a ferramentas como o SonarQube (LUCANIA, 2020).

SonarQube é uma ferramenta para garantir a qualidade do código-fonte em desenvolvimento. Ele realiza diversas análises durante o processo de compilação da aplicação, detectando por exemplo: Trechos de código-fonte que possam gerar bugs, duplicidade de linha, prevenindo a repetição de instruções desnecessárias e segurança. Essas

[13] Kanban é uso de *cards* para sinalizar o andamento das atividades pelo fluxo de trabalho

análises atendem a métricas de qualidade configuradas na própria ferramenta seguindo alguns padrões predefinidos, mas que são facilmente customizáveis (SONARQUBE, 2022).

A integração dos times é um dos principais pilares da cultura DevOps, ela permite que em conjunto sejam definidas práticas, processos e a adoção de ferramentas como o SonarQube.

Se um trecho de código violar as regras de codificação durante a execução de uma análise, o SonarQube apontará problemas. O conjunto de regras de codificação é definido pelo perfil de qualidade associado a cada linguagem do projeto.

Quando um trecho de código não está em conformidade com uma regra, um problema também chamado de Issue, é registrado em um arquivo de origem ou um arquivo de teste de unidade. Segundo a documentação SonarQube (2022), esses problemas podem ser de três tipos:

a) *Bug* - Um erro de codificação que quebrará seu código e precisa ser corrigido imediatamente;
b) Vulnerabilidade - um ponto em seu código que está aberto a ataques;
c) *Code Smell*[14] - Um problema de manutenção que torna seu código confuso e difícil de manter.

Além dos tipos, os problemas são classificados através de seu nível de severidade:

a) *Blocker*: Bug com alta probabilidade de afetar o comportamento do aplicativo em produção: vazamento de memória, conexão JDBC não fechada. O código deve ser corrigido imediatamente;
b) *Critical*: Um bug com baixa probabilidade de impactar o comportamento do aplicativo em produção ou um problema que representa uma falha de segurança: bloco catch vazio, injeção de SQL, ... O código deve ser revisado imediatamente;

[14] *Code smell* é qualquer característica no código-fonte de um programa que possivelmente indica um problema mais profundo

c) *Major*: falha de qualidade que pode impactar fortemente a produtividade do desenvolvedor: pedaço descoberto de código, blocos duplicados, parâmetros não utilizados;
d) *Minor*: falha de qualidade que pode impactar levemente a produtividade do desenvolvedor: as linhas não devem ser muito longas, variáveis não utilizadas;
e) *Info*: Nem um bug nem uma falha de qualidade, apenas um achado.

Outras análises importantes realizadas pelo SonarQube são “complexidade ciclomática” e “complexidade cognitiva”.

> Complexidade ciclomática é uma métrica do campo da engenharia de software, desenvolvida por Thomas J. McCabe em 1976, e serve para mensurar a complexidade de um determinado módulo (uma classe, um método, uma função etc), a partir da contagem do número de caminhos independentes que ele pode executar até o seu fim. Um caminho independente é aquele que apresenta pelo menos uma nova condição (possibilidade de desvio de fluxo) ou um novo conjunto de comandos a serem executados. (TEDESCO, 2016)

De forma simples e objetiva, as métricas são fórmulas e métodos matemáticos que geram estatísticas e estimativas a partir de insumos submetidos ao modelo, bem como descrevem formas de aferir os resultados gerados (JÂNIO, 2019).

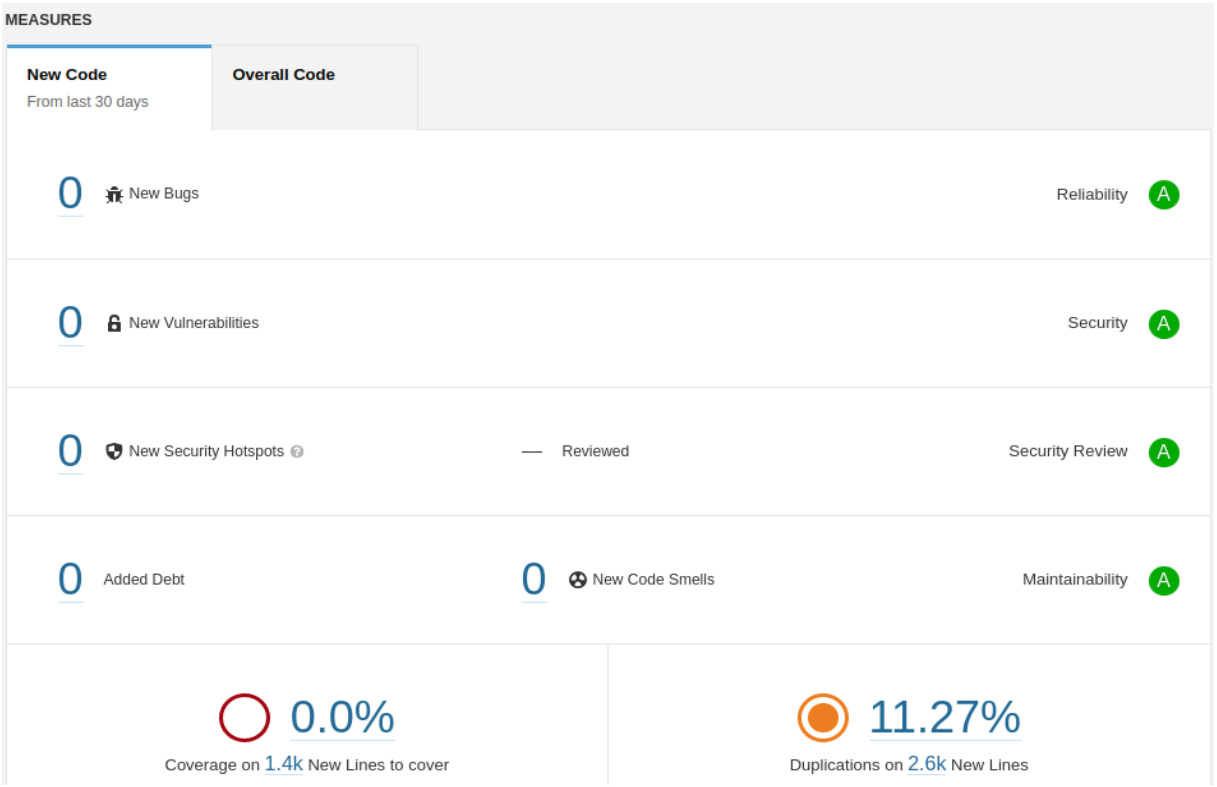
O resultado da complexidade ciclomática indica quantos testes (pelo menos) precisam ser executados para que se verifique todos os fluxos possíveis que o código pode tomar, a fim de garantir uma completa cobertura de testes (TEDESCO, 2016).

A complexidade cognitiva é um modelo matemático desenvolvido pela SonarSource, criadora do SonarQube, para avaliar a "manutenibilidade"[15] do software. A complexidade cognitiva produz pontuações de complexidade de método que se alinham bem com a forma

[15] Manutenibilidade, ou manutibilidade, é uma característica inerente a um projeto de sistema ou produto, e se refere à facilidade, precisão, segurança e economia na execução de ações de manutenção nesse sistema ou produto (BLANCHARD, Benjamin. Logistics engineering and management. 4th ed. Englewwod Cliffs: Prentice Hall, 1992. p. 15).

como os desenvolvedores percebem a manutenção (CAMPBELL; SONARSOURCE SA, 2021).

Figura 11 - Dashboard Sonarqube



Fonte: Do autor (2022)

Os resultados da análise do Sonaqube são apresentados nesta tela, mostrando a quantidade de bugs, vulnerabilidades, *hotspots* de segurança, dívida técnica, cobertura de linhas de código por testes e duplicações, bem como escalas que variam de A a E para reusabilidade, segurança, revisão de segurança e manutenibilidade, onde A representa a melhor situação e E representa a pior situação, conforme mostrado na Figura 11.

2.7.4 Cypress

Atualmente, o trabalho de um bom analista de qualidade é repleto de desafios, devido à complexidade em que os sistemas são desenvolvidos, e ao acrescentar esse contexto ao

desenvolvimento ágil, torna-se cansativo e repetitivo o trabalho de um analista de qualidade, realizando uma bateria de testes sempre que alguma funcionalidade nova é desenvolvida, ou um bug é corrigido. Com isso, é extremamente importante a automatização dos testes, possibilitando a execução de testes de regressão[16], de forma simples e resultados imediatos (DEV BLOG, QUALITY ASSURANCE, 2021).

O Cypress é uma ferramenta gratuita de código aberto, que auxilia e facilita a automação de testes, principalmente testes de interface, criado para garantir que todos os itens de uma tela funcionem adequadamente, e testes E2E (*end to end*), que garante que um fluxo da aplicação esteja correto do início ao fim, replicando cenários reais e garantindo que o sistema funcione como o esperado (CYPRESS, 2022).

Com a ferramenta é possível criar casos de teste de forma rápida, fácil e confiáveis para qualquer coisa que seja executada em um navegador. Cypress foi feito especificamente para desenvolvedores e engenheiros de controle de qualidade, para ajudá-los a fazer mais (CYPRESS, 2022).

Segundo a definição do próprio site, primeiramente, Cypress ajuda você a configurar e começar a escrever testes todos os dias enquanto você cria seu aplicativo localmente. TDD[17] no seu melhor! e depois de criar um conjunto de testes e integrar o Cypress ao seu provedor de CI, nosso serviço de painel pode registrar suas execuções de teste.

É possível salvar capturas de telas e vídeos no momento da falha caso ela ocorra, e ao utilizar algum gerador de relatórios, é possível anexar essa mídia em um relatório.

---

[16] O teste de regressão é uma técnica do teste de software que consiste na aplicação de versões mais recentes do software, para garantir que não surgiram novos defeitos em componentes já analisados (MAIA et al, 2009).

[17] *Test Driven Development* (Desenvolvimento guiado por testes) é uma técnica de desenvolvimento de software que se relaciona com o conceito de verificação e validação e se baseia em um ciclo curto de repetições, onde o desenvolvedor escreve um caso de teste para validar uma funcionalidade que será escrita posteriormente.

3 ATIVIDADES DESENVOLVIDAS

Neste capítulo, são apresentadas as atividades desenvolvidas durante o período de estágio na Fundecc. Também é apresentada uma relação entre conceitos aprendidos em sala de aula e o que foi utilizado no estágio.

3.1 Cenário inicial

Neste capítulo é descrito como foi o cenário inicial em que o estagiário foi inserido, contextualizando o projeto e relatando como era o processo de desenvolvimento.

3.1.1 O Projeto

O projeto relatado aqui, em que o estagiário atuou durante alguns meses, se trata do Sistema de Outorga de Água do Estado de Santa Catarina, O Siout-SC, que tem o intuito de modernizar e aperfeiçoar o gerenciamento das concessões e administração de atos inerentes às outorgas de uso de água, por meio de ferramentas inteligentes que explorem as informações relativas ao uso dos recursos hídricos de forma eletrônica. Desse modo, o Siout-SC torna o processo de outorga mais transparente e ágil para a sociedade (SIOUT-SC, 2022).

O projeto foi desenvolvido utilizando a linguagem de programação C# (lê-se "cê sharp") com *.NET framework*[18], em seu *backend* e *Angularjs* no *frontend*. Devido ao uso do *framework*, o sistema após construído, permite implantação somente em um sistema operacional Windows, e nesse contexto foi utilizado o Windows Server 2012 R2.

3.1.2 Como era

O processo de desenvolvimento, durante o período deste estágio, foi guiado pelo kanban disponível no próprio Gitlab, onde o *Product Owner* cadastra todas as tarefas originárias dos requisitos em uma raia inicial, com o nome de *Open*, que é referente ao *project*

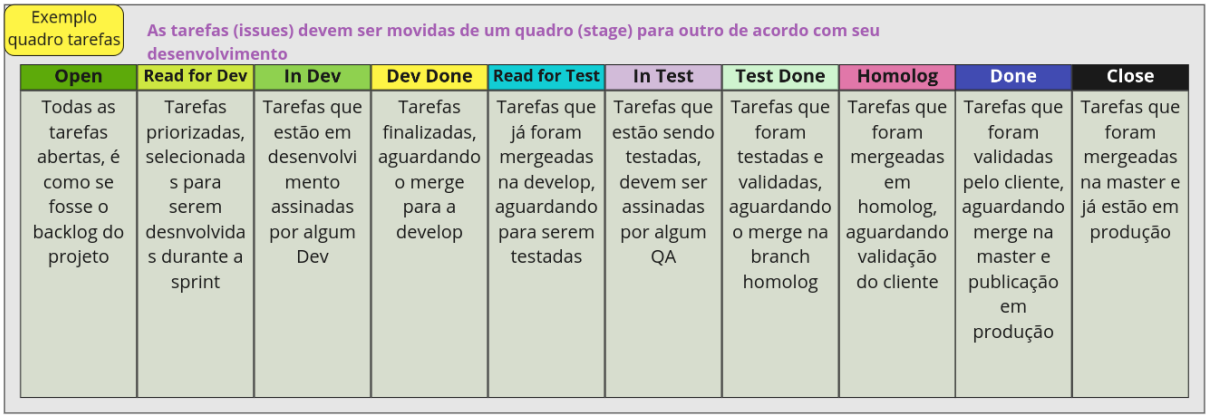
[18] O *.NET Framework* é um ambiente de execução gerenciado para o Windows que oferece uma variedade de serviços aos aplicativos em execução

backlog. Durante a *Sprint Planning*, ao definir e mensurar quais tarefas serão executadas durante a próxima *Sprint*, estas são movidas para uma próxima raia, chamada *Read For Dev*, referente à *Sprint backlog*, aguardando algum desenvolvedor assiná-la, movendo-a para a raia *In Dev*, mostrando a todos da equipe que a tarefa está em desenvolvimento.

Com o desenvolvimento finalizado, a tarefa é movida para a raia *Dev Done*, aguardando sua implantação em um ambiente interno, para que testes possam ser realizados. Após a implantação no ambiente de teste, que será detalhada posteriormente, a tarefa é movida para *Read For Test* aguardando algum analista de qualidade testá-la; durante o teste, a tarefa deve ser colocada em *In Test*, e após o teste finalizado, caso falhe, a tarefa deve voltar para *Read For Dev*; caso tenha sucesso no teste a tarefa deve seguir para *Test Done*.

A tarefa só é removida de *Test Done* e colocada em *Homolog*, quando é realizada sua implantação em ambiente de homologação. Após homologada a tarefa chega em uma raia como o nome de Done, que é quando a tarefa pode ir para o ambiente de produção e deve ser fechada no kanban. A Figura 12 mostra como esse fluxo deve ocorrer.

Figura 12 - Fluxo Kanban



Fonte: Do autor (2022)

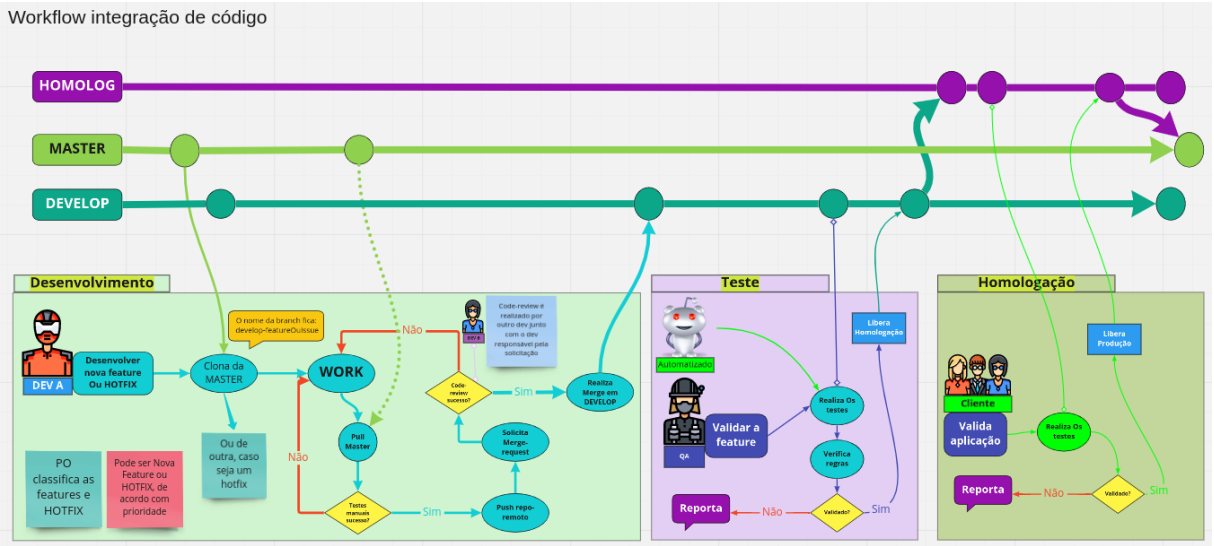
Ao assinar uma tarefa para a sua resolução, o desenvolvedor deve seguir um fluxo de desenvolvimento conhecido como *Gitflow*[19] com a variação *trunk based*, que é o

[19] O Gitflow é um modelo alternativo de ramificação do Git que consiste no uso de ramificações de recursos e várias ramificações primárias.

> Desenvolvimento baseado em tronco, em que os desenvolvedores fazem o merge de atualizações pequenas e frequentes em um 'tronco' ou ramificação principais. É uma prática comum entre as equipes de DevOps e faz parte do ciclo de vida de DevOps porque simplifica as fases de merge e integração. Na verdade, o desenvolvimento baseado em tronco é prática obrigatória de CI/CD. Os desenvolvedores podem criar ramificações de curta duração com alguns pequenos commits se comparadas a outras estratégias de ramificação de recursos de longa duração. À medida em que crescem a complexidade da base de código e o tamanho da equipe, o desenvolvimento baseado em tronco ajuda a manter o fluxo de versões de produção (ZETTLER, 2020).

A Figura 13 descreve esse fluxo de desenvolvimento mergeada

Figura 13 - Workflow projeto



Fonte: Do autor (2022)

Existia, na época do estágio, três ambientes, ou lugares, onde a implantação do sistema deveria ser feita:

a) Ambiente de Teste: Era o sistema rodando refletindo exatamente o que estava na *branch develop*, implantado na rede local da empresa, usado para toda a equipe de desenvolvimento, e principalmente, os analistas de qualidade, para a verificação e validação das modificações efetuadas pela equipe. No ambiente de teste foram realizados todos os testes de interface, e testes *end to end;*

b) Ambiente de Homologação: Refletia a versão do código fonte presente na *branch homolog*, e era implantado em um servidor fornecido pelo cliente, onde somente quem estava na rede interna do cliente tinha acesso. Esse ambiente era onde o cliente verificava e validava o sistema, de acordo com os requisitos;
c) Ambiente de Produção: Refletia a *branch master*, e era o sistema de produção, em sua versão mais estável, onde todas as modificações foram previamente testadas pelos analistas de qualidade no ambiente de teste, validadas por uma equipe interna do cliente no ambiente de homologação, e caso estivesse de acordo com os requisitos, essa modificação era publicada nesse ambiente.

No início deste estágio, o estagiário foi inserido em um projeto onde todo o processo de implantação era realizado de forma manual, onde havia a necessidade do trabalho ser realizado por um desenvolvedor que já estava há algum tempo no projeto, tomando cerca de 30 minutos de seu tempo durante cada implantação, se não houvesse nenhum problema. Como o projeto possuía, no tempo do estágio, 4 desenvolvedores, o processo de implantação no ambiente de teste era algo comum, ocorrendo pelo menos uma vez por dia; então toda vez que era necessário integrar novo código ao projeto, algum desenvolvedor com mais experiência precisava parar o que estava fazendo e realizar tal tarefa, começando com a atualização do código fonte presente em seu computador, com as modificações realizadas. Em seguida era necessário realizar o *build* da aplicação em sua máquina, prosseguindo com uma conexão via ssh no servidor do ambiente de teste, realizando um *backup* com sistema que estava funcionando. Após isso, todo o conteúdo que foi compilado anteriormente, era copiado para o servidor substituindo o conteúdo antigo.

Após todo esse processo, a equipe recebia a comunicação da nova implantação no ambiente de teste, e começavam os testes, que eram realizados de forma manual pelos analistas de qualidade. Depois de tudo verificado, todo esse processo de implantação era realizado novamente para o ambiente de homologação e posteriormente produção.

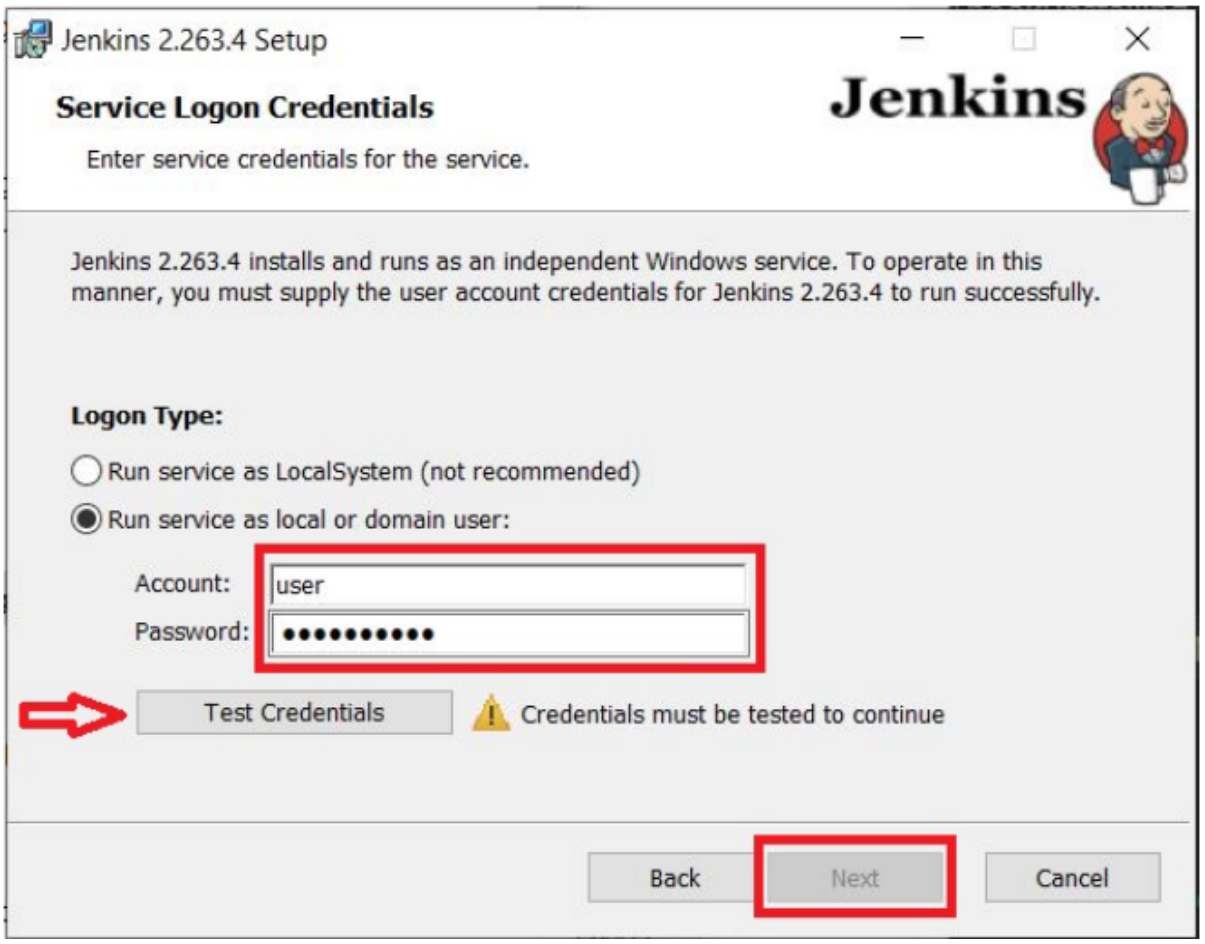
3.2 Desenvolvimento

Este capítulo descreve como foi a implantação da esteira de CI/CD no projeto utilizando a ferramenta Jenkins, que foi escolhida anteriormente ao estágio, por outro colaborador da empresa, porém outras ferramentas foram analisadas, como por exemplo o GitLab CI/CD, que foi descartado pelo motivo do projeto em questão utilizar a tecnologia .Net Framework em uma versão que não é compatível com Linux. Este capítulo também apresenta a implementação de análise estática de código e implementação de testes automatizados. Por fim serão relatados os resultados alcançados e os impactos no projeto e na equipe de desenvolvimento.

3.2.1 Instalação

A instalação do Jenkins foi realizada no mesmo servidor do ambiente de testes, utilizando o Windows Server 2012 R2 como sistema operacional. A instalação ocorreu seguindo a documentação, que exige como requisitos mínimos, de 4 GB de memória RAM e 50 GB de armazenamento, além de outros softwares necessários para seu funcionamento, como Java 8 ou Java 11, navegador da Web compatível com o Jenkins, Google Chrome, Mozilla Firefox, Microsoft Edge ou Apple Safari. Foi preciso baixar o executável de instalação, que ao executá-lo aparece um assistente de configuração de instalação, que ao seguir os passos, em um determinado momento, é necessário definir um usuário e senha para acesso ao servidor Jenkins, conforme mostrado na Figura 14.

Figura 14 - Instalação Jenkins, definindo login

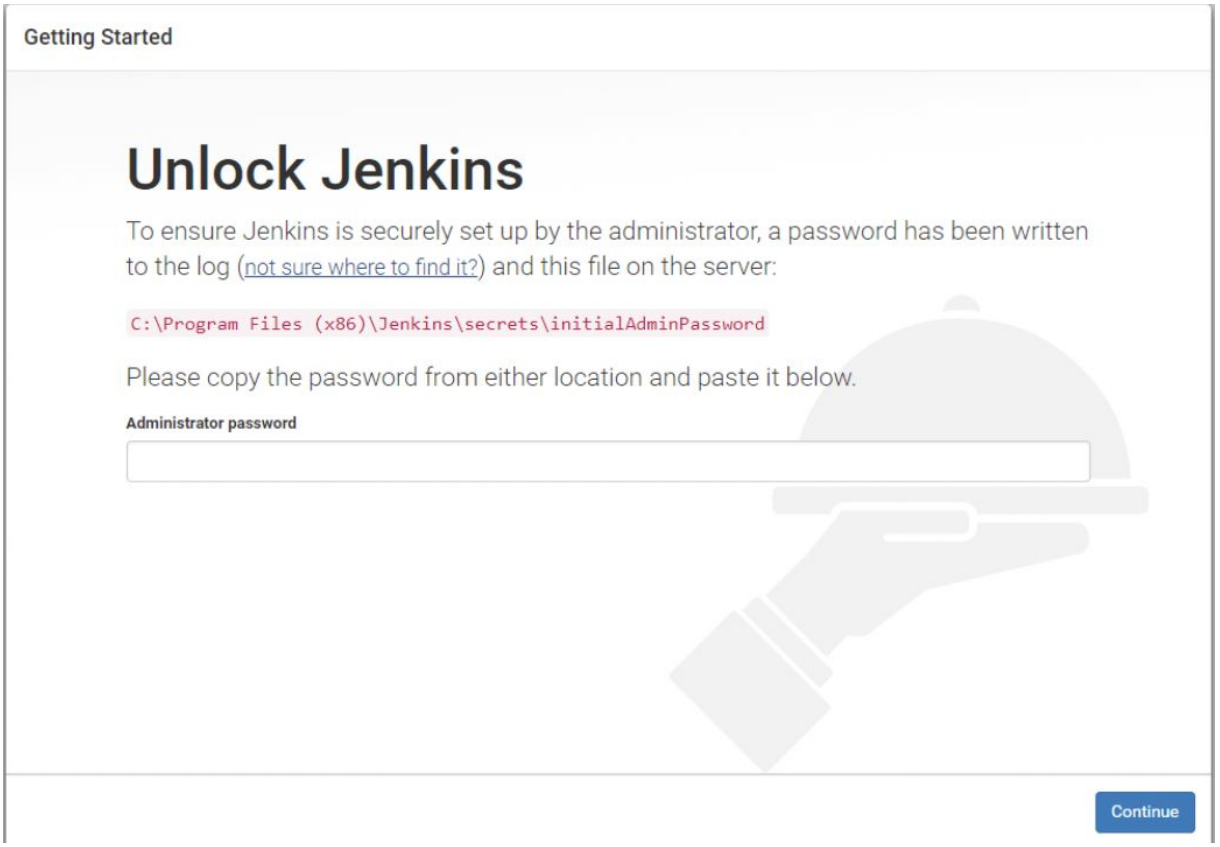


Fonte: Jenkins (2022)

Após baixar, instalar e executar o Jenkins, o assistente de configuração pós-instalação é iniciado levando a algumas etapas rápidas para desbloquear o Jenkins, personalizá-lo com plugins e criar o primeiro usuário administrador por meio do qual você pode continuar acessando o Jenkins.

Ao acessar uma nova instância do Jenkins pela primeira vez, é solicitado que você a desbloqueie usando uma senha gerada automaticamente. Navegue até http://localhost:8080 (ou a porta definida durante a instalação) e espere até que a página Unlock Jenkins apareça, conforme a Figura 15.

Figura 15 - Desbloqueie o Jenkins



Fonte: Jenkins (2022)

Essa senha para desbloqueio pode ser encontrada em dois locais diferentes, na pasta indicada na imagem ou no log de instalação.

A próxima etapa é para escolha dos plugins, exibindo duas opções, uma para instalar plugins sugeridos e outra para selecionar plugins para instalação. Para essa instalação do Jenkins foi escolhida a segunda opção selecionando os seguintes plugins além dos recomendados:

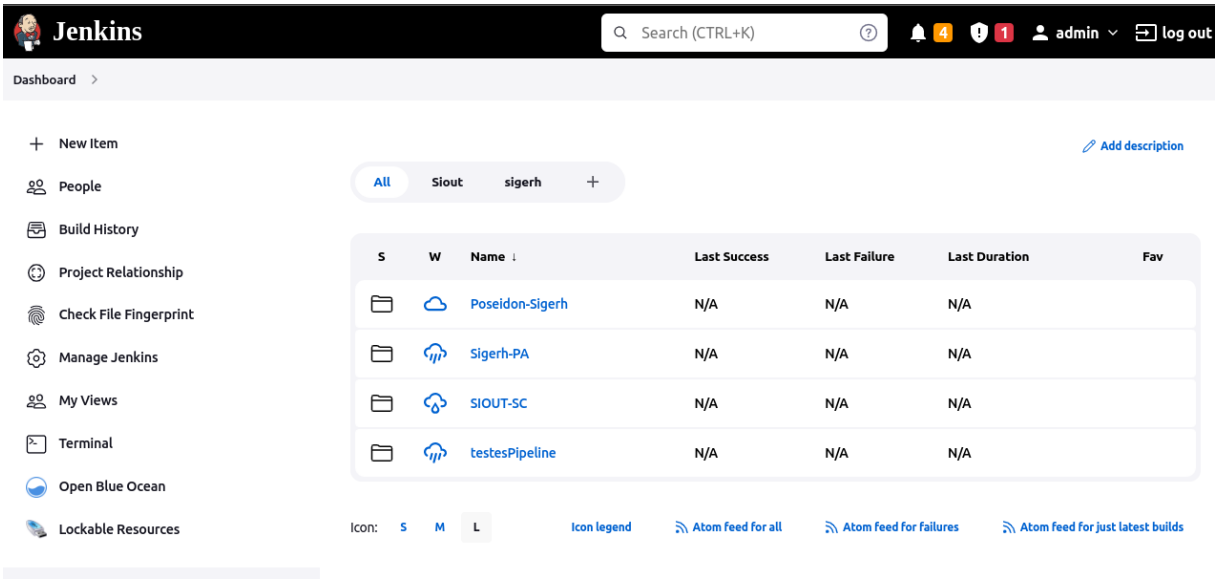
a) *Allure*: para gerar o relatório dos testes com o Cypress;
b) *Email Extension*: permite configurar todos os aspectos das notificações por e-mail;
c) *EnvInject API*: encapsula a Biblioteca Jenkins EnvInject e fornece API adicional para gerenciamento de variáveis de ambiente nos plug-ins do Jenkins;

d) *Git Parameter*: adiciona a capacidade de escolher ramificações, *tags* ou revisões de repositórios Git configurados no projeto.
e) *GitLab*: permite que o GitLab acione compilações do Jenkins e exiba seus resultados na interface do usuário do GitLab;
f) *MSBuild*: possibilita construir um projeto do Visual Studio (.proj) e arquivos de solução (.sln), utilizados para projetos *.NET Framework*;
g) *NodeJS*: executa o script Node JS como uma etapa de construção;
h) *Pipeline*: conjunto de plugins que permite orquestrar a automação, simples ou complexa;
i) *PowerShell*: permite que o Jenkins invoque o *PowerShell* como scripts de compilação;
j) *Sonar Quality Gates Plugin*: falha na compilação sempre que os critérios do *Quality Gates* na análise do Sonar;
k) *SSH plugin*: executa comandos *shell* remotamente usando o protocolo SSH;
l) *Timestamper*: adiciona carimbos de data/hora à saída do console;
m) *Discord Notifier*: permite que você envie incorporações do Discord sobre suas compilações por meio dos webhooks do Discord.

3.2.2 Pipelines

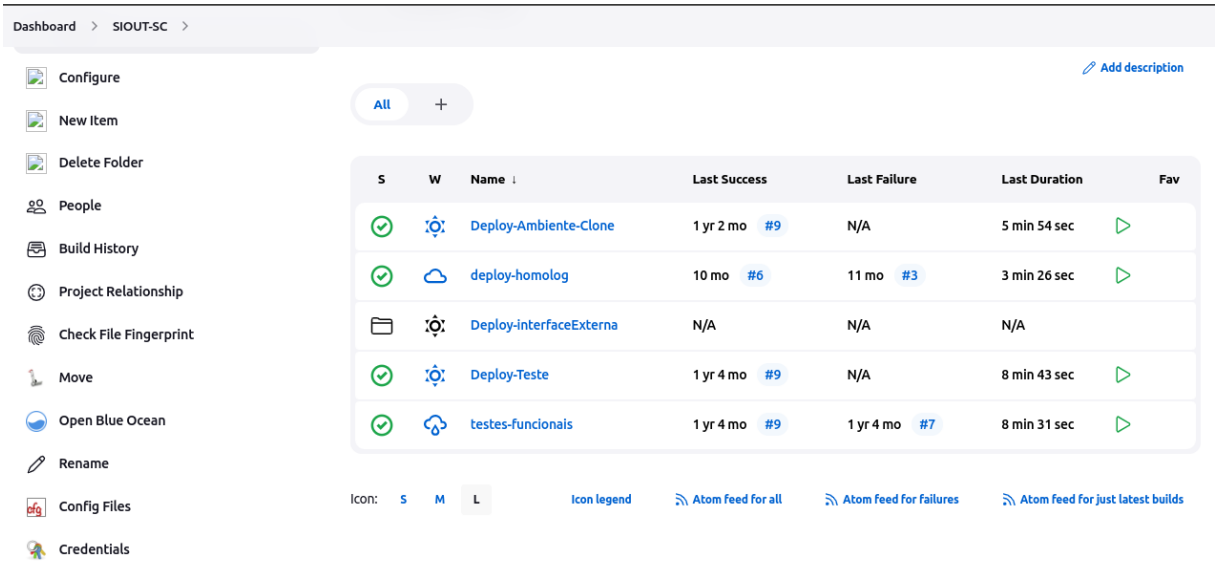
Após a configuração completa do Jenkins e instalação dos plugins, foi necessário criar pastas para diferentes projetos, conforme a Figura 16, logo foi criada uma pasta para o projeto do Siout-SC, e nessa pasta foram criados alguns pipelines, conforme a Figura 17.

Figura 16 - Dashboard geral



Fonte: Do autor (2022)

Figura 17 - Dashboard Siout-SC



Fonte: Do autor (2022)

Como mostrado na Figura 17, existem diferentes pipelines para o projeto em questão, descritos na Tabela 2, na qual se evidencia o nome do pipeline, sua descrição, onde realiza o

monitoramento de modificações, podendo ser um branch específico ou uma pasta específica em um branch específico, e por fim informa qual o tipo de configuração está utilizando.

Tabela 2- Pipelines Siout-SC

Nome	**Descrição**	**Branch monitoramento**	**Configuração**
Deploy-Ambiente-Clone	responsável por realizar a implantação em ambiente Clone de produção (Utilizado para replicar *bugs* reportados diretamente de produção)	master	Configuração e Jenkinsfile
Deploy-homolog	responsável por realizar o *deploy* no ambiente de homologação	homolog	Pipeline parametrizado
Deploy-interfaceExterna	Responsável por realizar a implantação somente da interface gráfica pública do projeto	develop - somente pasta interfaceExterna	Configuração e Jenkinsfile
Deploy-Teste	Responsável por realizar o *deploy* em ambiente de teste	develop	Configuração e Jenkinsfile
Testes-funcionais	Realiza o monitoramento do projeto de testes automatizados utilizando o Cypres, realizando as atualizações dos scripts no servidor.	master	Configuração e Jenkinsfile

Fonte: Do autor (2022)

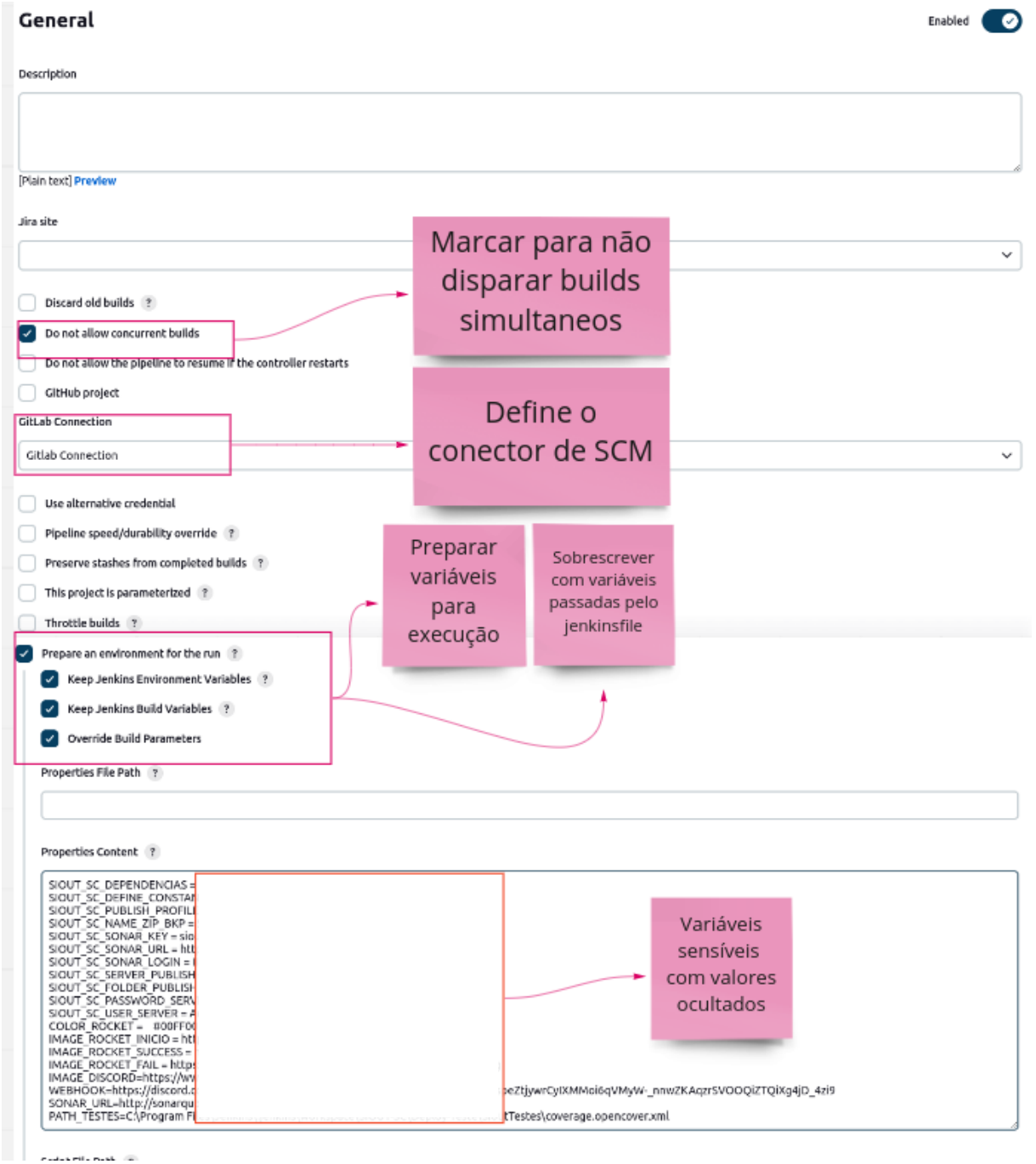
A Figura 18 mostra um print screen do *dashboard* relacionado ao projeto Deploy-Teste, onde cada linha representa um build, cada coluna representa um estágio desse build.

Figura 18 - Dashboard de projeto Jenkins

Fonte: Do autor (2022)

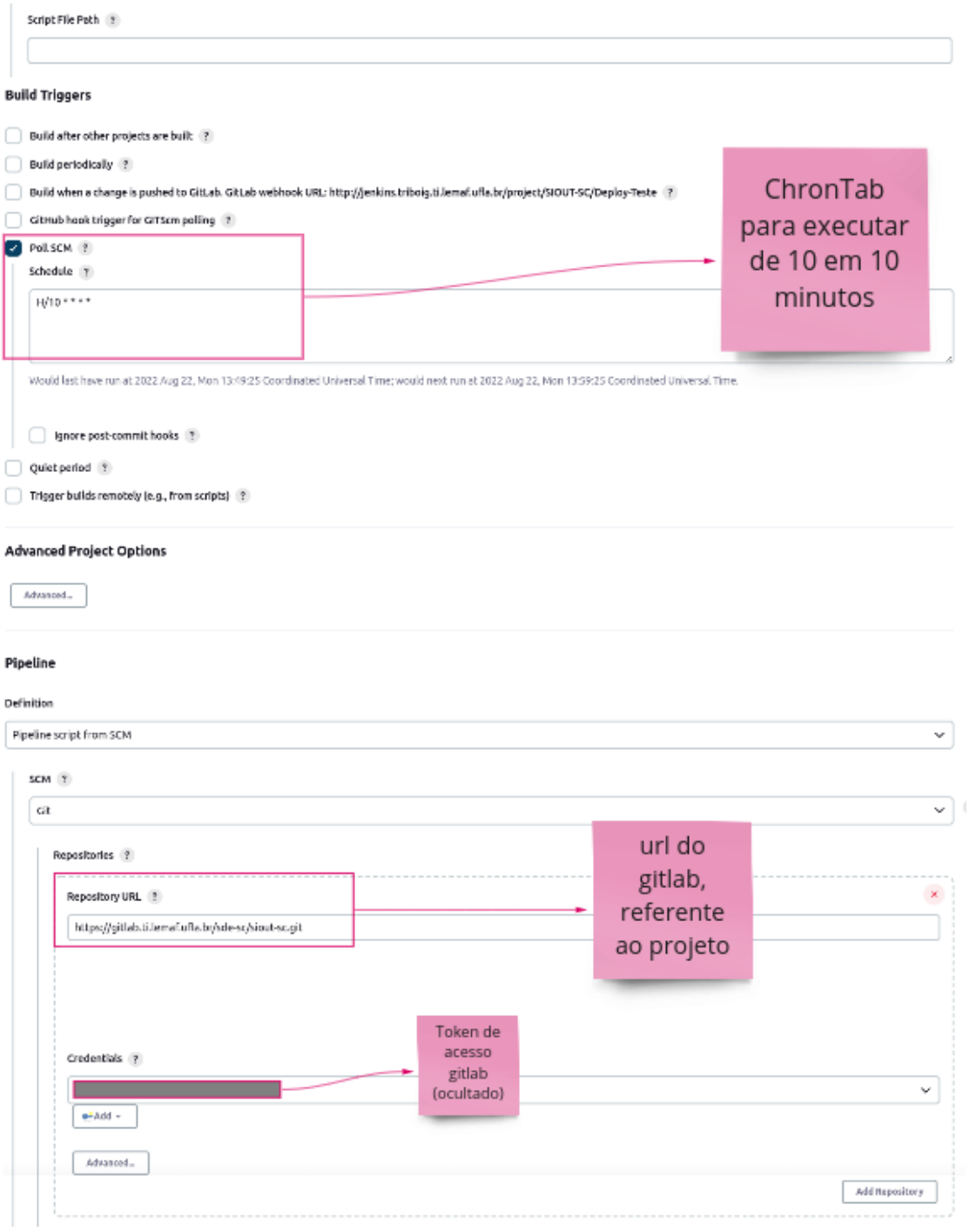
Todos os *pipelines* apresentados precisam de configuração pela interface gráfica do Jenkins, com o objetivo de configurar como os builds serão executados, a Figura 19, mostra como foi feita essa configuração.

Figura 19 - Página de configuração do *pipeline* (Continua)



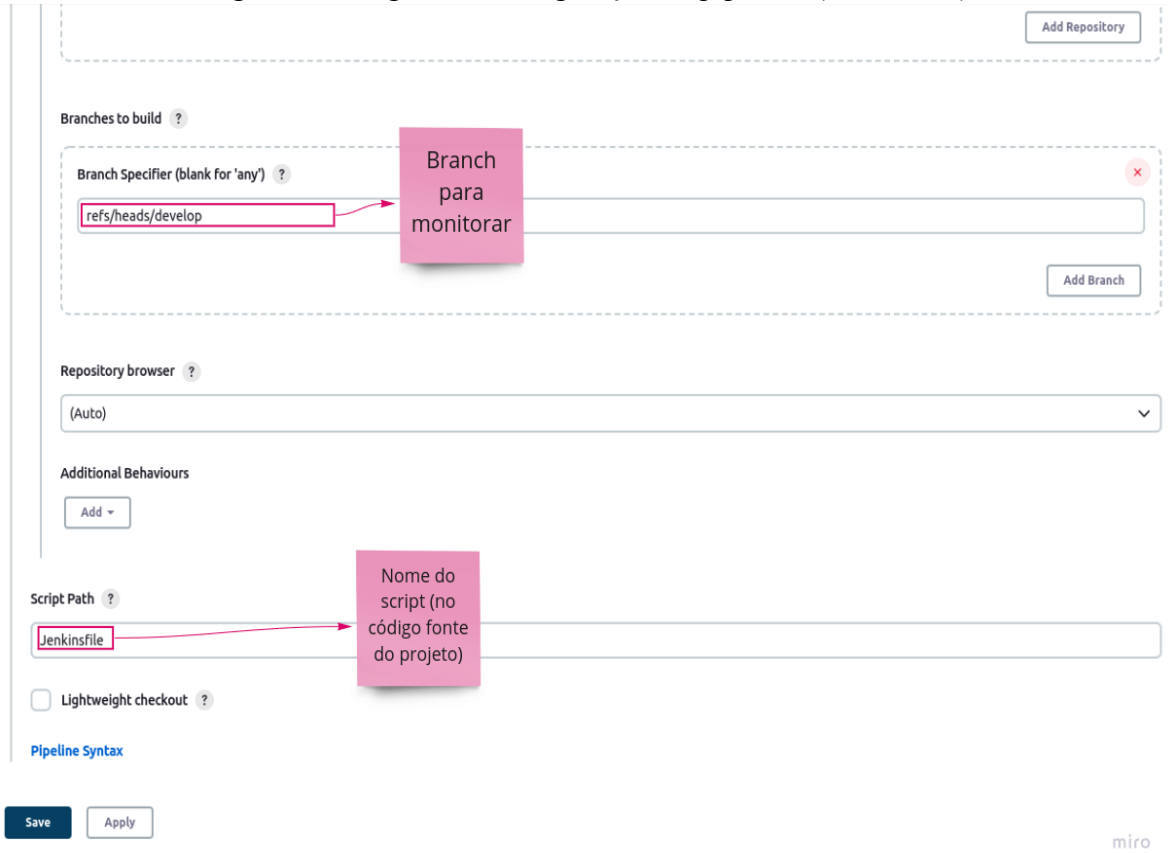
Fonte: Do autor (2022)

Figura 19 - Página de configuração do *pipeline* (Continua)



Fonte: Do autor (2022)

Figura 19 - Página de configuração do *pipeline* (Conclusão)



Fonte: Do autor (2022)

A Figura 19 mostra que não é permitido executar builds simultâneos, além de especificar a conexão com o repositório hospedado no Gitlab. Também é possível notar o uso de variáveis de ambientes definidas para o pipeline, sobrescrevendo as variáveis passadas pelo Jenkinsfile. Na configuração referente aos gatilhos de execução (*Build Triggers*) é possível notar que foi selecionado a opção "*Poll SCM*", responsável por disparar a execução de um job assim que notar alguma alteração no branch definido, que nesse caso é o "*develop*", a verificação ocorre constantemente, com intervalo definido no quadro apresentado como

"*Schedule*" que segue a sintaxe do *cron*[20], e aqui está definido para realizar uma verificação a cada 10 minutos.

Na seção nomeada como *Pipeline*, tem-se a definição do *script* de *pipeline*, que pode ser de duas formas, escrito diretamente nessa página ou fornecido com o código fonte da aplicação, a segunda opção foi escolhida, sendo necessário informar a url do Gitlab, o *token* de acesso e a especificação do *branch* que será monitorado.

O *script* Jenkinsfile deve estar na raiz do projeto, no *branch develop*, seguindo a sintaxe *Groovy*. O *script* deste projeto está completo no Apêndice A. O Jenkinsfile está dividido em estágios, nomeados como "*stage*", onde cada estágio é responsável por uma parte do *build*, em cada estágio existem passos ("*steps*"), onde cada linha dentro do bloco "*steps*" representa um passo. A Figura 20 mostra o início do *script* juntamente com o primeiro estágio.

Algumas cópias utilizam o comando *robocopy*, que é uma ferramenta de linha de comando que realiza cópia de dados de uma forma segura e eficiente, com os parâmetros:

a) /mt: cria cópias com vários *threads* com n *threads*. n deve ser um inteiro entre 1 e 128. O valor padrão para n é 8;
b) /z: copia arquivos no modo reinicializável. No modo reinicializável, se uma cópia de arquivo for interrompida, o *Robocopy* poderá continuar de onde parou em vez de recopiar todo o arquivo;
c) /s: copia arquivos com segurança;
d) /e: copia subdiretórios. Essa opção inclui automaticamente diretórios vazios;

Já o comando *'EXIT /b 0*', é responsável por sair do interpretador de comando ou do *script* em lote atual com o código 0.

[20] um arquivo crontab contém 5 campos – cada um é representado por um asterisco – que determinam a data e a hora que certa tarefa é programada para desempenhar repetidamente, veja mais em https://www.hostinger.com.br/tutoriais/cron-job-guia#:~:text=Como%20Escrever%20uma%20Sintaxe%20Cron,%C3%A9%20programada%20para%20desempenhar%20repetidamente. Acesso em 20 Ago. 2022.

Figura 20 - Início Jenkinsfile Deploy-Teste (1º estágio)

```
Jenkinsfile
1   pipeline{
2     agent any
3     options { timestamps() }
4     tools {nodejs "nodejs12.18.2"}
5     stages{
6       stage('Preparando o ambiente'){
7         steps{
8           discordSend description: "Deploy iniciando\n
9                                     Ambiente: ${SIOUT_SC_DEFINE_CONSTANTS}\n
10                                    [console](${env.BUILD_URL}console)",
11            link: "${JOB_URL}",
12            thumbnail: "${IMAGE_DISCORD}",
13            result: "UNSTABLE",
14            title: "DEPLOY ${SIOUT_SC_DEFINE_CONSTANTS}",
15            webhookURL: "${WEBHOOK}",
16            footer: "build numero ${BUILD_ID}"
17
18          dir('Interface'){
19            nodejs(nodeJSInstallationName: 'nodejs12.18.2'){
20              bat 'copy %SIOUT_SC_DEPENDENCIAS%\\token'
21              bat 'robocopy %SIOUT_SC_DEPENDENCIAS%\\node_modules ./node_modules /mt /z /s /e & EXIT /B 0 '
22              bat 'robocopy %SIOUT_SC_DEPENDENCIAS%\\bower_components ./bower_components /mt /z /s /e & EXIT /B 0 '
23              bat 'nvm use 12.18.2'
24              bat 'node --version'
25            }
26          }
27        }
28      }
```

Fonte: Do autor (2022)

A linha 1 inicia o *script*, definindo como um *Pipeline*. As linhas 2, 3 e 4, informam que o agente de execução, a utilização do *Timestamps* e o uso do Node em determinada versão, respectivamente. A linha 5 inicia o bloco de estágios, seguido pelo primeiro estágio, nomeado de “Preparando o ambiente”, com os seguintes passos:

a) *DiscordSend*: cria uma mensagem para notificar a equipe utilizando o canal de mensagens instantâneas Discord. A mensagem em questão é escrita substituindo as variáveis pelos valores definidos na configuração do pipeline;
b) dir: define o diretório “Interface” para executar os comandos;
c) nodejs: define a versão do node que será utilizada e abre um bloco da linha 13 até a linha 17, com instruções para o powershell executar; esses comandos são responsáveis por copiar alguns diretórios e um token necessários para realizar o build da aplicação.

O segundo e terceiro estágio são responsáveis por realizar o *build* do *frontend* e o *build* do *backend*, respectivamente, conforme mostra a Figura 21:

Figura 21 - Segundo e terceiro estágio Jenkinsfile

```
29      stage('Construindo frontend'){
30        steps{
31          dir('Interface'){
32            bat 'gulp'
33          }
34        }
35      }
36      stage('Construindo backend'){
37        steps{
38          bat "nuget restore SioutSC.sln"
39          bat "MSBuild.exe ./Siout/Siout.csproj /t:Rebuild /p:DeployOnBuild=true
40              /p:DefineConstants=%SIOUT_SC_DEFINE_CONSTANTS% /p:PublishProfile=%SIOUT_SC_PUBLISH_PROFILE%"
41        }
42      }
```

Fonte: Do autor (2022)

O segundo estágio, nomeado como “construindo o *frontend*”, executa o comando “gulp” dentro do diretório “interface”; esse comando é responsável por realizar o *build* do *frontend* da aplicação, criando seus arquivos estáticos. Já o terceiro estágio, nomeado como “Construindo o backend”, é responsável por compilar e construir o *backend* da aplicação *.NET Framework.*

O próximo estágio é responsável por realizar um *backup* da aplicação, conforme a Figura 22.

Figura 22 - Quarto estágio Jenkinsfile

```
43      stage('Construindo o Zip e Preparando o BKP'){
44        steps{
45          bat 'copy %SIOUT_SC_DEPENDENCIAS%\\versao.html .\\Interface'
46          dir('Interface'){
47            bat 'gulp gerarZip'
48          }
49          dir('Interface/dist'){
50            bat 'dir'
51
52            //Agrupar os zips
53            bat 'WinRAR.exe a -r -ag+YYYY-MM-DD- "%SIOUT_SC_NAME_ZIP_BKP%.zip" logica.zip interface.zip'
54
55            //Extraindo os arquivos gerados
56            bat 'WinRAR.exe x -y logica.zip logica/ & EXIT /B 0'
57            bat 'WinRAR.exe x -y interface.zip interface/ & EXIT /B 0'
58
59            //Remover os zips desnecessarios
60            bat 'del interface.zip'
61            bat 'del logica.zip'
62
63            //criar a pasta bkp
64            bat 'mkdir backup'
65            bat 'move *.zip backup'
66          }
67        }
68      }
```

Fonte: Do autor (2022)

Ainda de acordo com a Figura 22, linha 38, é realizada uma cópia do arquivo ‘versão.html’ para o diretório ‘Interface; em seguida, é possível identificar a instrução ‘gulp

gerarZip' responsável por compactar toda a aplicação e colocar em um diretório chamado 'dist', dentro da pasta 'Interface'. Na linha 42, define o diretório 'Interface/dist' para a execução dos próximos comandos. Já na linha 46 os arquivos compactados 'logica.zip' e 'interface.zip', são compactados em um único arquivo, que recebe o nome através de uma variável de ambiente. As próximas linhas são responsáveis por criar uma pasta com o nome de backup, e mover todos os arquivos compactados para ela.

Na Figura 23 é exibido o próximo estágio, responsável por recompilar toda a aplicação, realizando o scanner do SonarQube, e enviando para um servidor do SonarQube, configurado nas variáveis de ambiente.

Figura 23 - Quinto estágio Jenkinsfile

```
69      stage('Analise do sonarqube'){
70        steps{
71          bat "SonarScanner.MSBuild.exe begin /k:${'"%SIOUT_SC_SONAR_KEY%"'}
72              /d:sonar.host.url=${'"%SIOUT_SC_SONAR_URL%"'} /d:sonar.login=${'"%SIOUT_SC_SONAR_LOGIN%"'}"
73          bat "MSBuild.exe /t:Rebuild"
74          bat "SonarScanner.MSBuild.exe end /d:sonar.login=${'"%SIOUT_SC_SONAR_LOGIN%"'}"
75        }
76      }
```

Fonte: Do autor (2022)

O *deploy* (implantação do sistema) no ambiente de teste se dá copiando toda a lógica, interface e *backup*, gerado no quarto estágio para o servidor remoto, onde é disponibilizado o ambiente de teste, conforme mostra a Figura 24.

Figura 24 - Sexto estágio Jenkinsfile

```
77      stage('Deploy para ambiente de teste'){
78        steps{
79          dir('Interface/dist'){
80            //Entrar na pasta do servidor
81            bat 'net use \\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout
82              "%SIOUT_SC_PASSWORD_SERVER%" /USER:%SIOUT_SC_USER_SERVER%'
83
84            //Copiar o codigo gerado para as pastas
85            bat 'robocopy logica/ \\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\Logica\\
86                 /mt /z /s /e & EXIT /B 0'
87            bat 'robocopy interface/ \\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\Interface\\
88                 /mt /z /s /e & EXIT /B 0'
89            bat 'robocopy backup/ \\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\backup\\
90                 /mt /z /s /e & EXIT /B 0'
91          }
92        }
93      }
94    }
```

Fonte: Do autor (2022)

Na imagem anterior fica claro a cópia da pasta ‘logica’ para o caminho %SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\Logica onde os valores das variáveis é definido na configuração *Pipeline*, o mesmo serve para as outras duas cópias.

A Figura 25 mostra o que deve ser realizado após os estágios (stages) anteriores, e sempre será executado.

Figura 25 - Post script Jenkinsfile

```
95    post{
96      failure {
97        discordSend description: "Houve falha no deploy\n
98                                  Ambiente: ${SIOUT_SC_DEFINE_CONSTANTS}\n
99                                  [ver console](${env.BUILD_URL}console)\n
100                                 [SonarQube](${env.SONAR_UR}${env.SIOUT_SC_SONAR_KEY})",
101           link: "${JOB_URL}",
102           thumbnail: "${IMAGE_DISCORD}",
103           result: "FAILURE",
104           title: "FALHA NO DEPLOY ${SIOUT_SC_DEFINE_CONSTANTS} num ${BUILD_ID}",
105           webhookURL: "${WEBHOOK}",
106           footer: "build numero ${BUILD_ID}"
107       }
108     success {
109       discordSend description: "SUCESSO no deploy\n
110                                 Ambiente: ${SIOUT_SC_DEFINE_CONSTANTS}\n
111                                 [ver console](${env.BUILD_URL}console)\n
112                                 [SonarQube](${env.SONAR_UR}${env.SIOUT_SC_SONAR_KEY})",
113           link: "${JOB_URL}",
114           thumbnail: "${IMAGE_DISCORD}",
115           result: "SUCCESS",
116           title: "SUCESSO NO DEPLOY EM ${SIOUT_SC_DEFINE_CONSTANTS} num ${BUILD_ID}",
117           webhookURL: "${WEBHOOK}",
118           footer: "build numero ${BUILD_ID}"
119     }
120   }
121 }
122
```

Fonte: Do autor (2022)

Logo, é possível perceber que caso o *job* tenha sucesso, ou seja, todos os passos dos estágios ocorreram com sucesso, é executada a linha 96, enviando uma notificação para a plataforma de mensagens Discord, informando à equipe. Caso tenha ocorrido alguma falha em algum passo, a equipe é notificada com uma mensagem de falha.

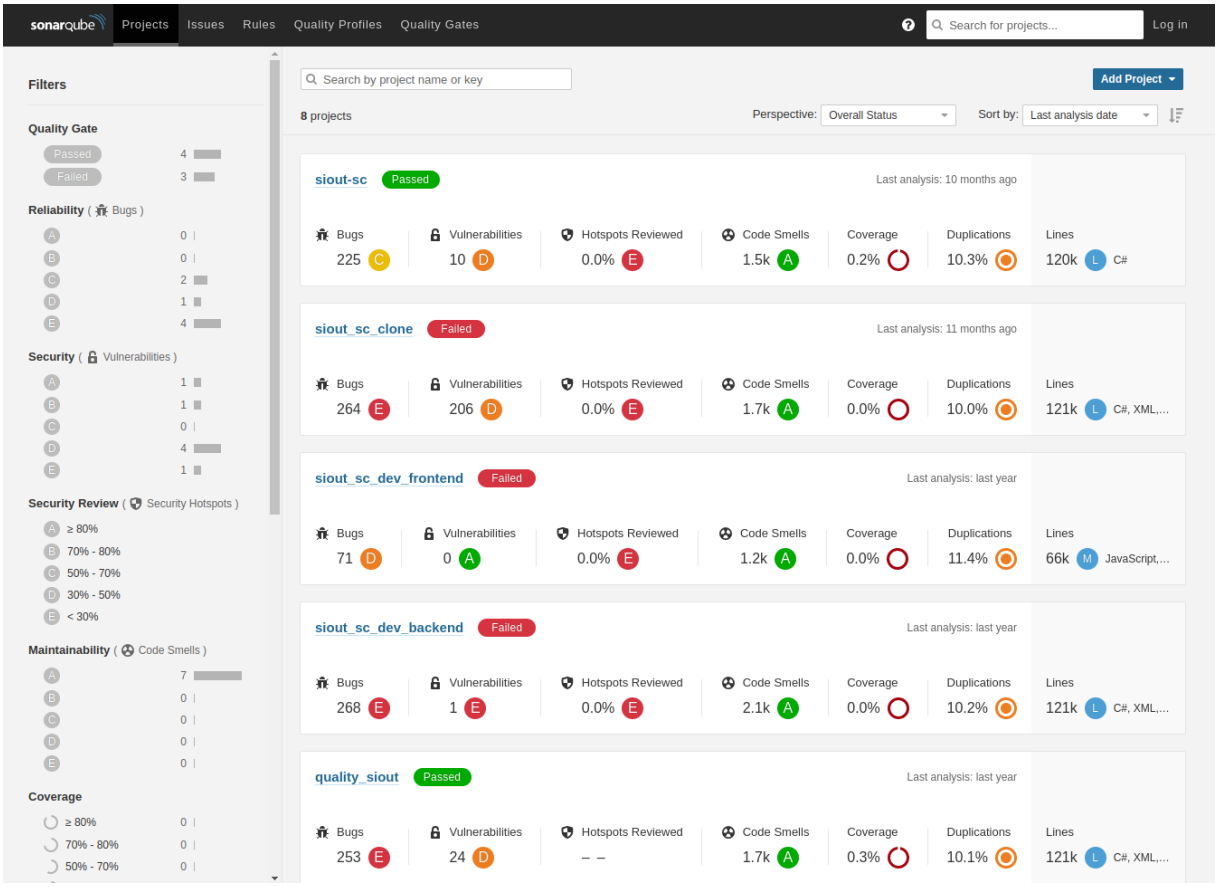
Os demais *Pipelines* implementados seguem basicamente o mesmo procedimento descrito até aqui.

Como próxima etapa, adicionamos a análise estática de código, escolhendo, junto com a equipe, a ferramenta mais conhecida do mercado, o SonarQube, que por ser open source, a grande maioria de suas funcionalidades estão na versão gratuita.

3.2.3 Adicionando Sonarqube

O SonarQube foi adicionado ao projeto com o intuito de verificar a qualidade do código, e com isso facilitar o trabalho da equipe para diminuir sua dívida técnica. Foram criados alguns projetos com diferentes padrões de métricas no SonarQube, conforme a Figura 26, o projeto utilizado no *Pipeline* de *deploy* em ambiente de teste é o nomeado Siout-sc.

Figura 26 - Projetos Sonarqube

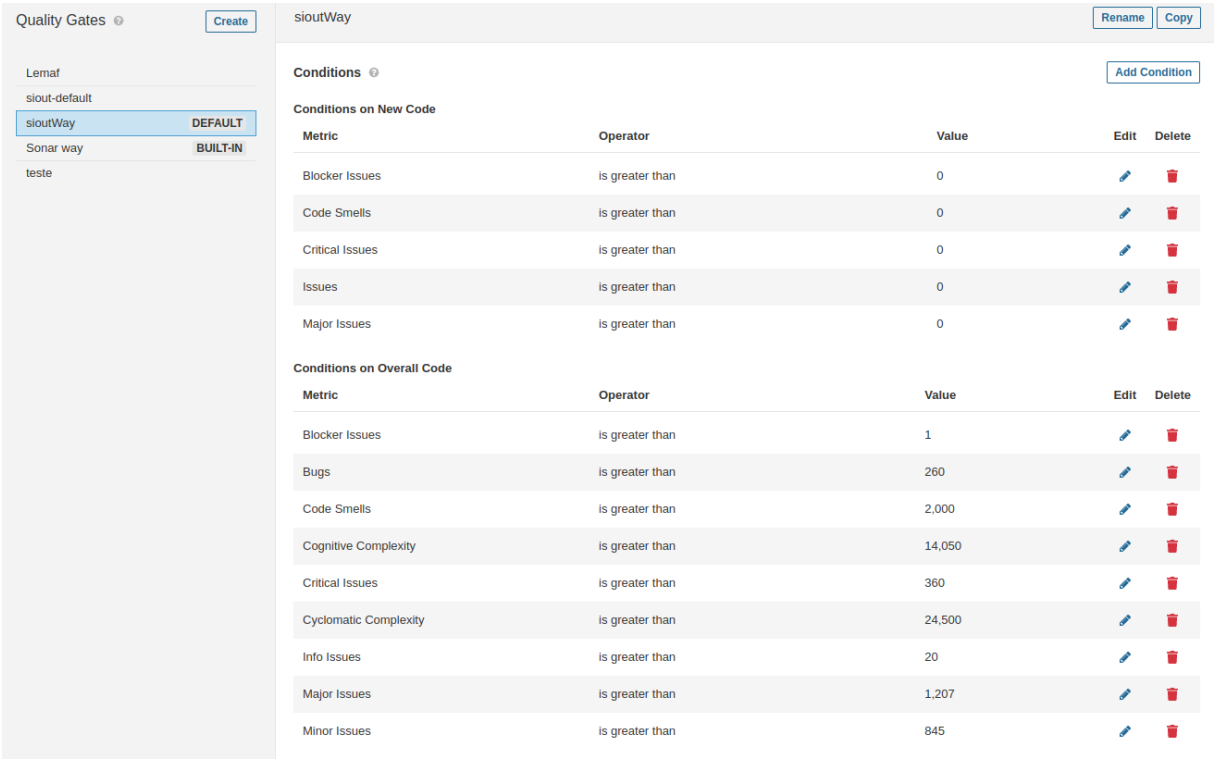


Fonte: Do autor (2022)

Esse dashboard exibe do lado esquerdo a soma dos números relacionados às análises realizadas em todos os projetos, que são exibidos à direita, com seus respectivos resultados.

Como o Sonarqube foi introduzido na esteira de CI/CD como um estágio, foi necessário definir padrões de qualidade relacionados à linguagem C#, alterando suas métricas, permitindo, no início, que alguns problemas não fossem impeditivos para a conclusão da esteira. Esses padrões são conhecidos como *"Quality Gate"*, e além de definir as métricas, também é responsável por devolver uma resposta para o Jenkins; essa resposta tem dados como o *timestamp* e a duração da execução do *Sonar scanner*. O *"Quality Gate"* definido para o projeto em questão foi nomeado de sioutWay, com suas regras (FIGURA 27).

Figura 27 - Quality gate Sonarqube



Fonte: Do autor (2022)

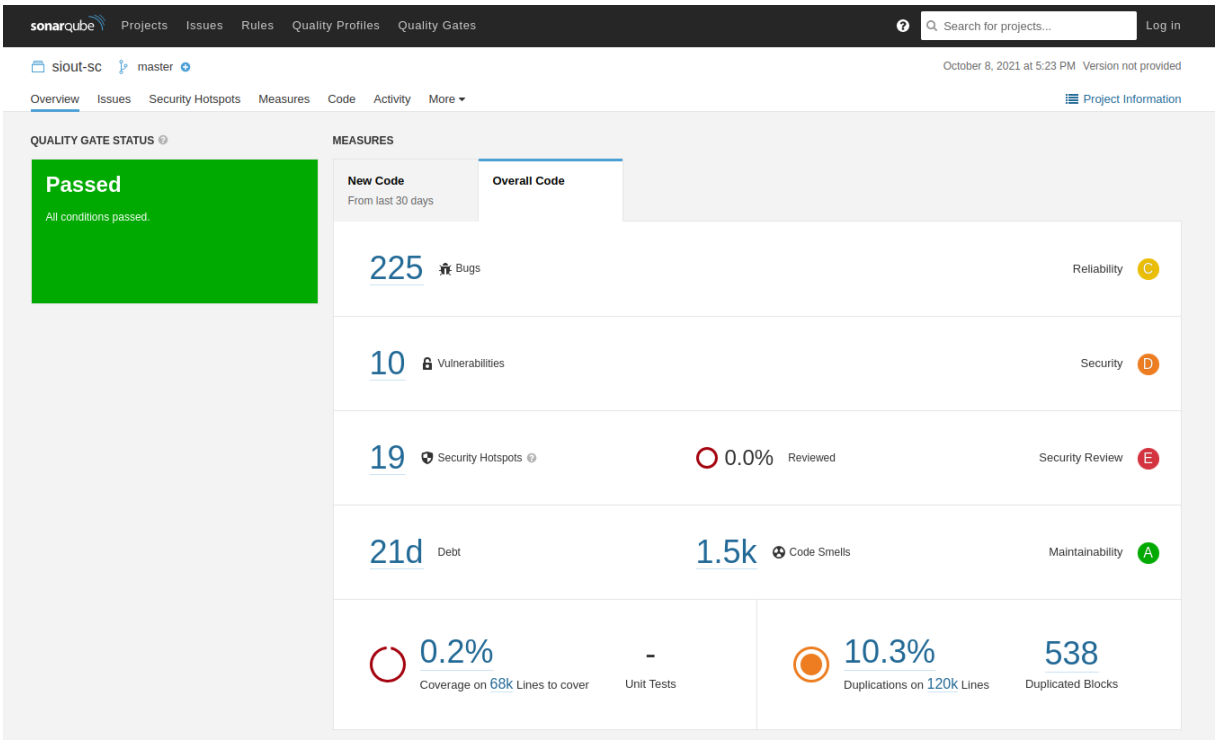
Considerando que o projeto do Siout já estava em desenvolvimento, a equipe chegou ao consenso de definir as métricas de acordo com uma primeira análise, mantendo esses valores, e diminuindo os limites em cada *Sprint*. Utilizando essa estratégia é possível continuar o desenvolvimento impedindo o aparecimento de novos problemas e diminuindo a quantidade dos problemas existentes a cada entrega.

As regras são responsáveis por definir o status do código. Caso algum valor seja superior aos definidos, é devolvida uma resposta de "*failure*" para quem chamou o *Scanner*.

Ao devolver uma resposta com o status de "*failure*" ou "*error*" os estágios posteriores não são executados, que neste caso é somente o *deploy* em ambiente de teste e a execução dos testes automatizados, que será apresentado logo mais, e por fim é executado o "*post*" no bloco "*failure*". Caso tenha sucesso no *Sonar scanner*, ou seja, todas as métricas foram atendidas, o *deploy* era realizado, notificando o usuário com o bloco referente ao status de "*success*" do "*post*":

O SonarQube também apresenta um *dashboard* para acompanhamento do projeto. A Figura 28 mostra esse dashboard para o projeto em questão.

Figura 28 - Dashboard Sonarqube

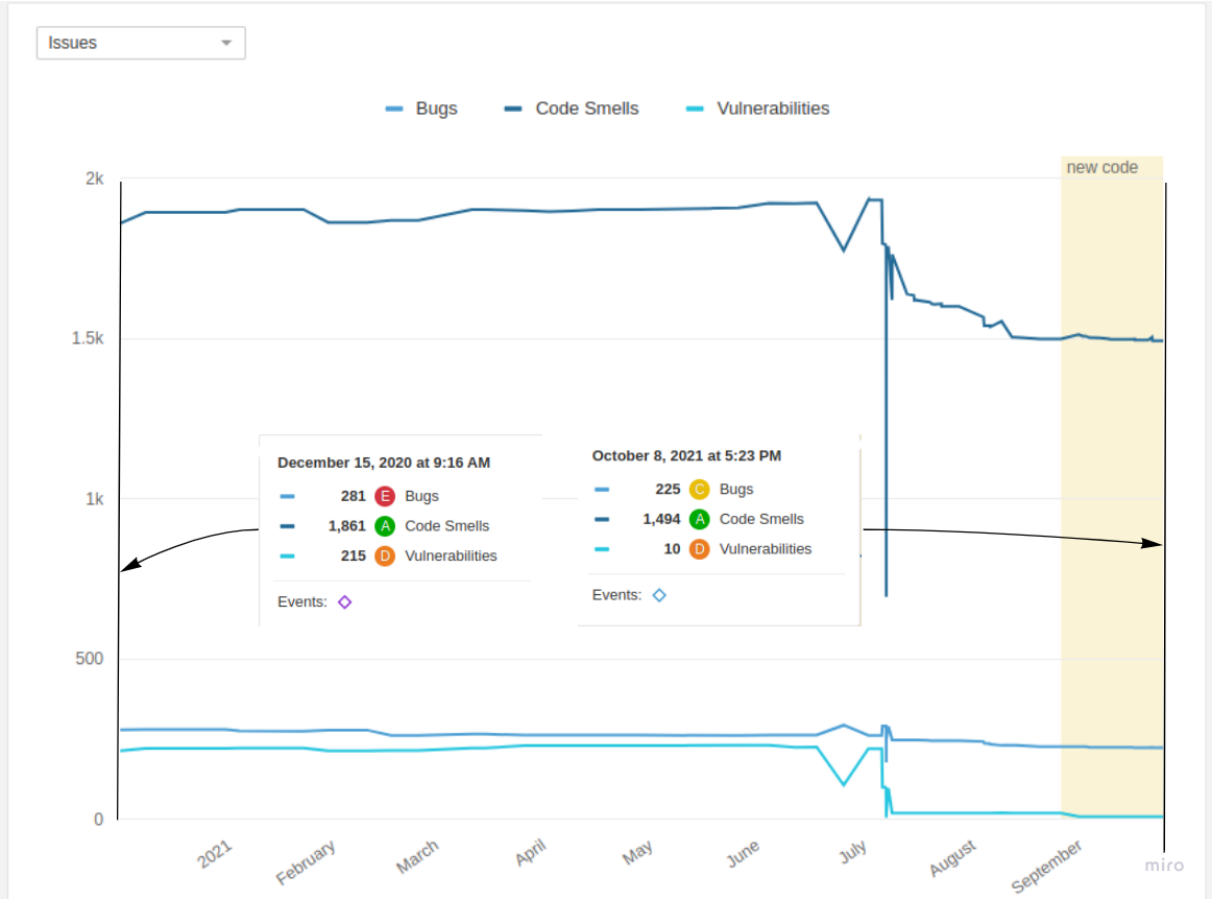


Fonte: Do autor (2022)

Na tela principal do dashboard é possível visualizar informações como o número total de Bugs, code smells, vulnerabilidades e questões de segurança, dívida técnica, etc.

Foi possível exibir alguns gráficos, com o objetivo de mostrar a melhora que o projeto analisado obteve ao longo dos meses de dezembro de 2020 a outubro do ano de 2021.

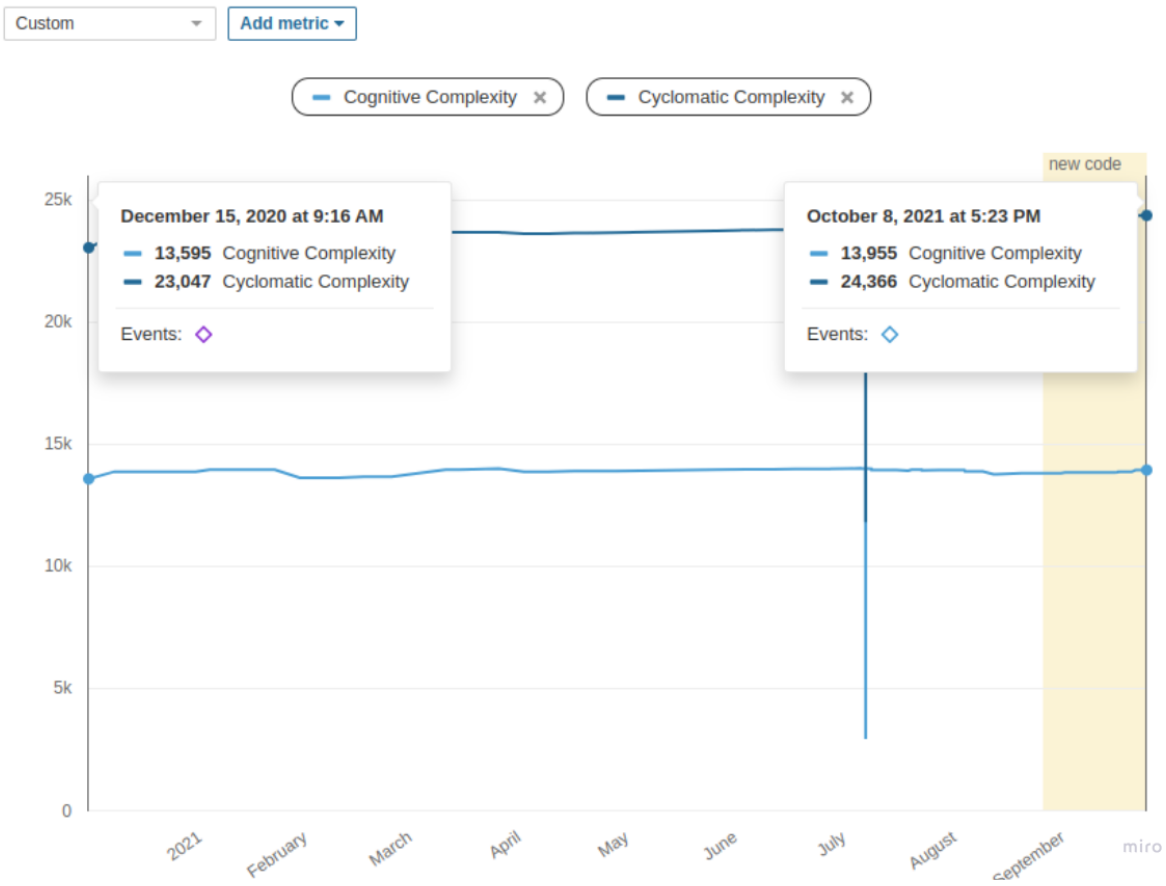
Figura 29 - Gráfico de Issues



Legenda: Gráfico de linha apresentando a evolução da quantidade de problemas descobertos pelo Sonarqube.
Fonte: Do autor (2022)

No gráfico da Figura 29 é possível observar uma redução de aproximadamente 20% na quantidade de bugs e code smells, e uma surpreendente diminuição de 95% na quantidade de vulnerabilidades; logo, conclui-se que a equipe focou em diminuir a vulnerabilidade do sistema. Outro gráfico interessante é o gráfico referente às complexidades ciclomática e cognitiva apresentado na Figura 30.

Figura 30 - Gráfico da complexidade ciclomática e cognitiva



Fonte: Do autor (2022)

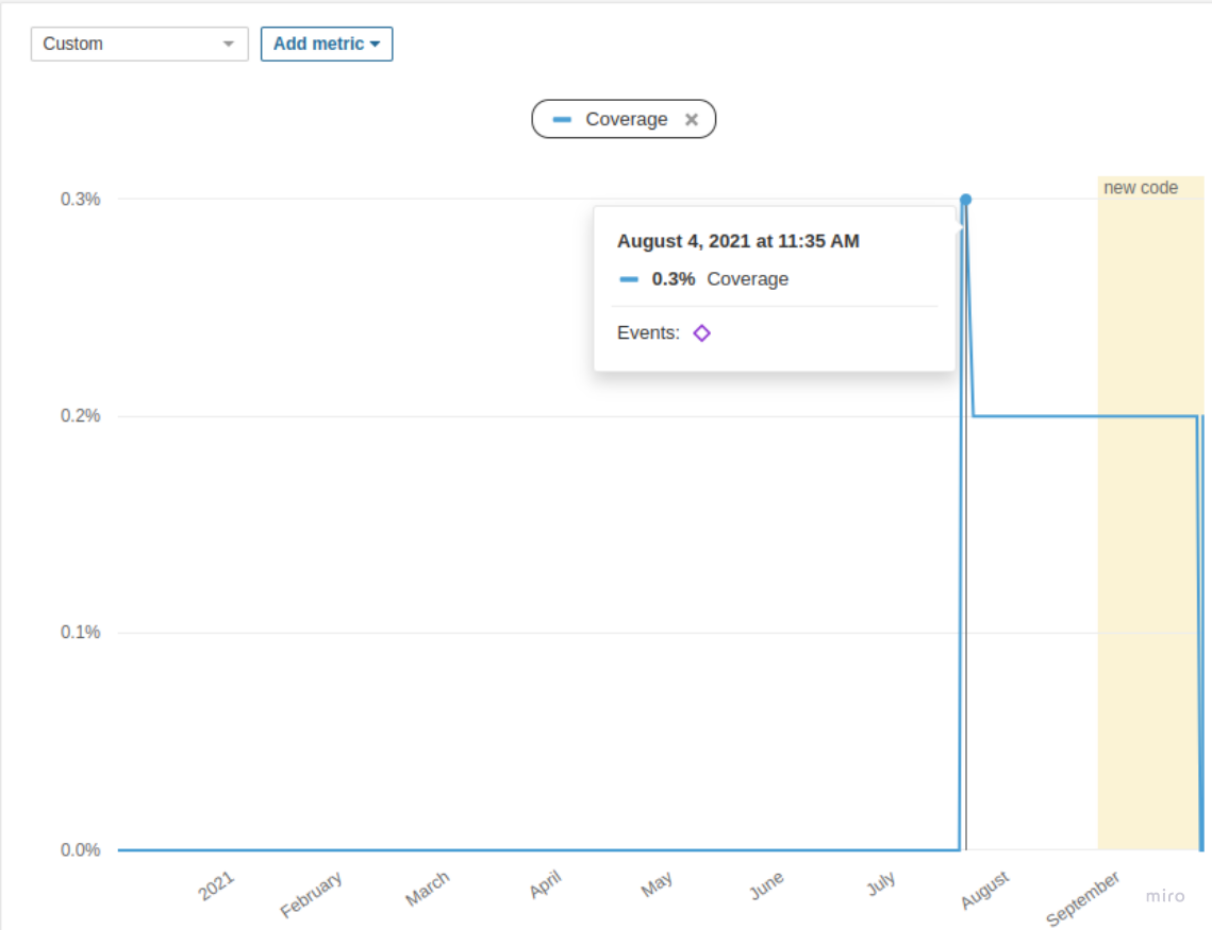
Foi observado o aumento da complexidade durante o período mensurado, porém temos que destacar que além do aumento do número de linhas de código, mostrado no gráfico da Figura 31, existe a refatoração do código fonte, o que pode contribuir para esse crescimento da complexidade temporariamente, devido às modificações estruturais e criação de novos métodos, e que, provavelmente irá diminuir com a evolução da refatoração. Salientamos que esses resultados foram constatados a partir da observação e análise do que foi desenvolvido, isto é, empiricamente.

Figura 31 - Gráfico do número de linhas de código

Fonte: Do autor (2022)

Outro fator em que houve uma melhora muito significativa foi a questão da cobertura de linhas de código por testes de unidade, que no início não havia nenhuma e no dia 4 de agosto de 2021 chegou a 0,3% do total de linhas, conforme o gráfico da Figura 32.

Figura 32 - Cobertura de linhas de código por testes

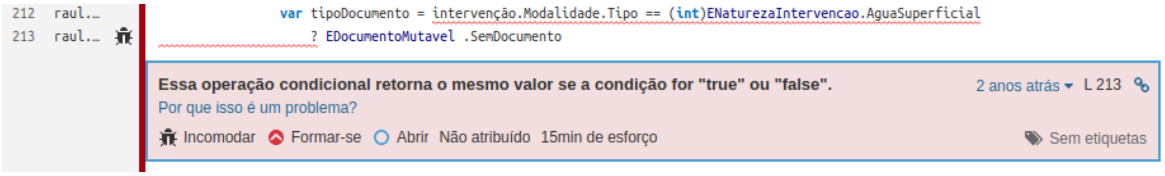


Fonte: Do autor (2022)

O acompanhamento das métricas disponibilizadas pelo SonarQube foi essencial para diminuir os custos com a manutenção do código, além de agregar mais valor no projeto, diminuindo suas vulnerabilidades de segurança.

O SonarQube, além de apresentar as métricas dos problemas encontrados, ainda exibe os problemas e indica sugestões para sua correção. A Figura 33 mostra a ocorrência de um bug.

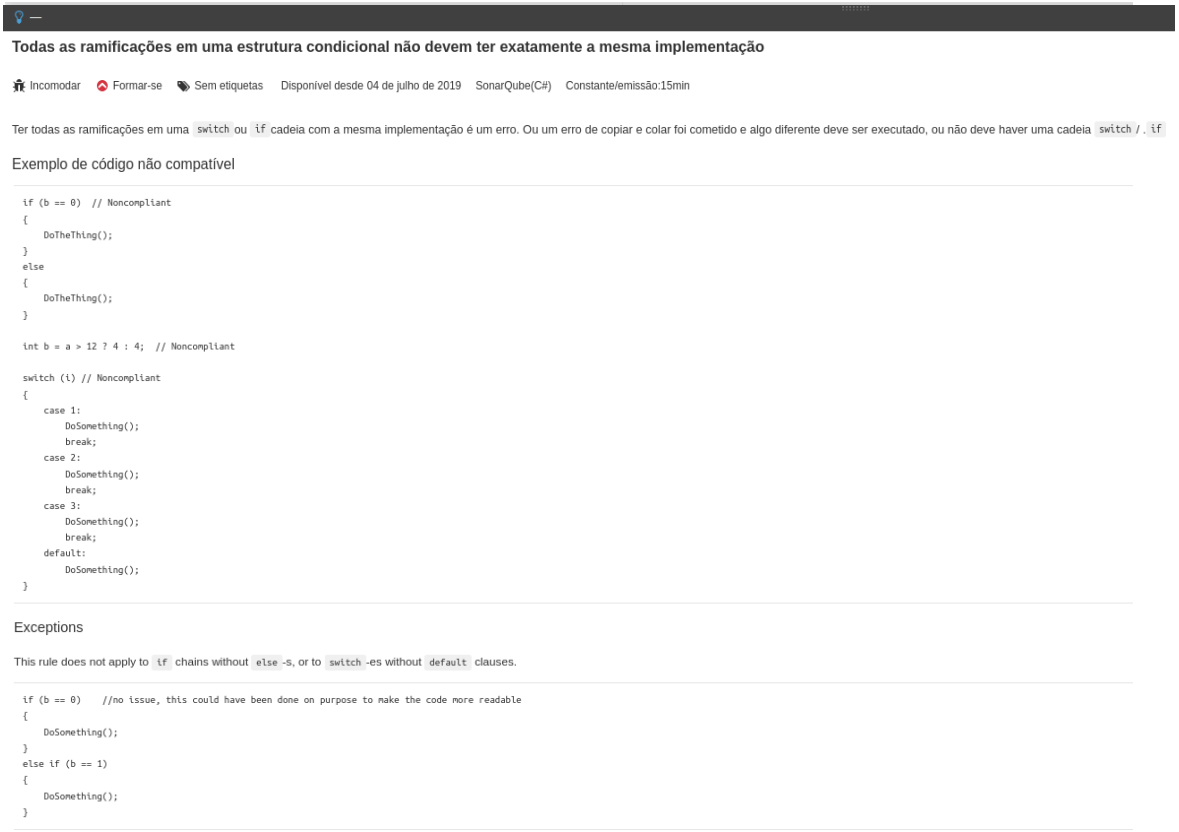
Figura 33 - Indicação de bug



Fonte: Do autor (2022)

A figura mostra em qual linha ocorre o bug, além de mostrar o criador do trecho de código (destacado em vermelho), no quadro exibido é informado o porque esse trecho é considerado um bug, e ao clicar no trecho destacado como um link “Por que isso é um problema?” é exibido uma janela explicando o motivo, e sugerindo a correção, conforme a Figura 34.

Figura 34 - Explicação do bug



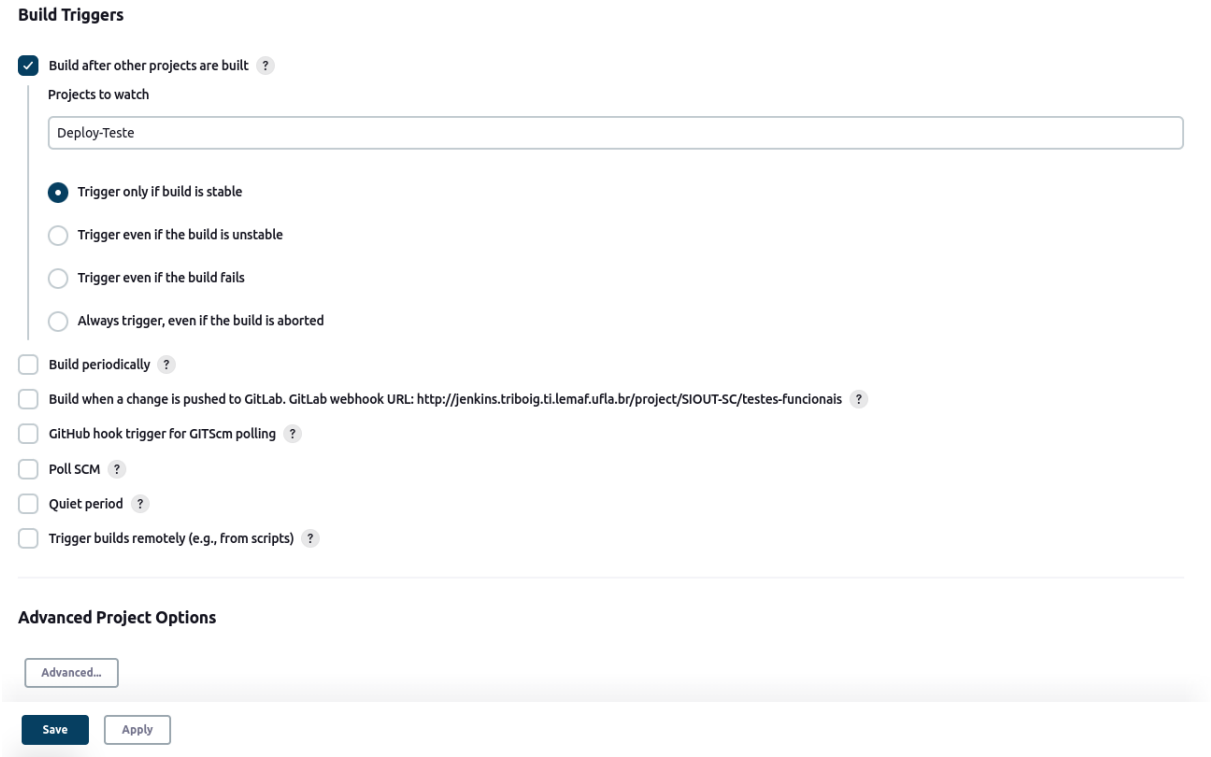
Fonte: Do autor (2022)

Com todos os recursos disponíveis no SonarQube, a manutenção do código se torna algo bem mais simples de ser realizado. Essa refatoração somada aos testes de regressão, e testes End to End, são essenciais para a escalabilidade do projeto, e para a realização destes testes de forma automatizada, inicialmente foi escolhida a ferramenta conhecida como Selenium, que apesar de gratuita, com o tempo foi possível verificar o alto custo de sua implantação, já que possui uma curva de aprendizado alta, e obteve uma baixa aceitação pela equipe. Como alternativa ao Selenium, foi analisada e escolhida a ferramenta Cypress para a efetivação da automação dos testes.

3.2.4 Adicionando testes com Cypress

Outra ferramenta implementada na esteira CI/CD foi o Cypress com o intuito de realizar testes de interface, verificando se os elementos da interface gráfica têm as ações esperadas, testes de *API-endpoints*, verificando se as requisições enviadas à API recebem as respostas esperadas e testes *End to End (e2e),* que verifica um fluxo ou funcionalidade do sistema do início ao fim. Para a realização dos testes, é necessário o *deploy* do sistema em algum servidor, pois os testes são executados com o sistema em funcionamento. Por isso, o projeto de testes automatizados possui sua própria configuração, mostrada na Figura 35, e seu próprio Jenkinsfile, que é apresentado inteiramente no Apêndice B.

Figura 35 - Configuração Jenkins testes funcionais

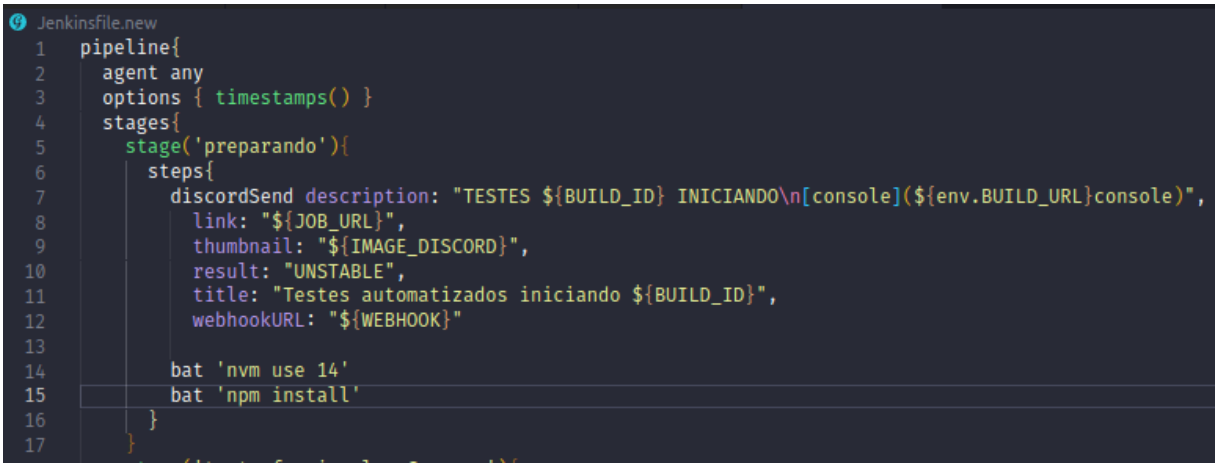


Fonte: Do autor (2022)

A única alteração em relação à outra página de configuração exibida anteriormente, está no gatilho de acionamento, pois nesse projeto é marcada a opção de executar após a execução estável do projeto Deploy-Teste.

Já no script Jenkinsfile, começamos com o *stage* de preparação, em que é enviada uma notificação para o Discord, e define a versão do Node que será usada, conforme destacado na Figura 36.

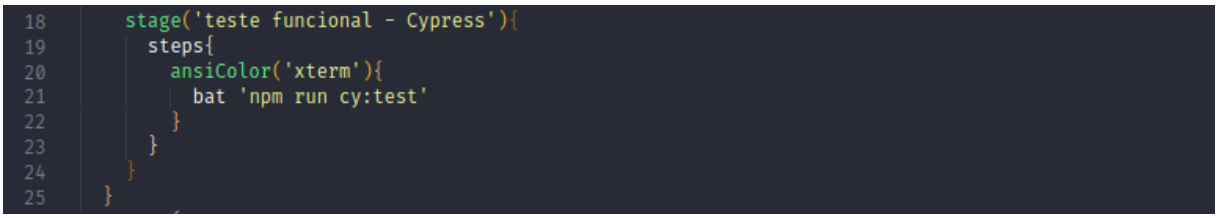
Figura 36 - Primeiro estágio Jenkinsfile testes funcionais



Fonte: Do autor (2022)

Em seguida, é executado o comando para execução dos testes, conforme Figura 37.

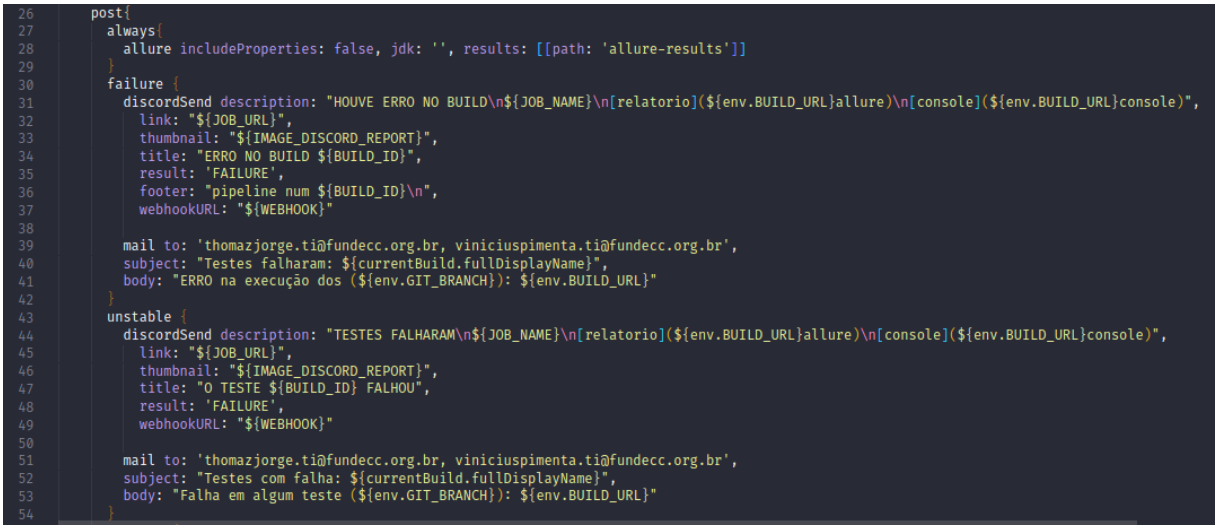
Figura 37 - Segundo estágio Jenkinsfile testes funcionais



Fonte: Do autor (2022)

Finalizando o *script*, temos as notificações que são enviadas ao Discord, de acordo com o resultado dos testes, e um e-mail é enviado para a pessoa responsável pelos testes e para o estagiário, de acordo com a Figura 38.

Figura 38 - Post scripts Jenkinsfile testes funcionais

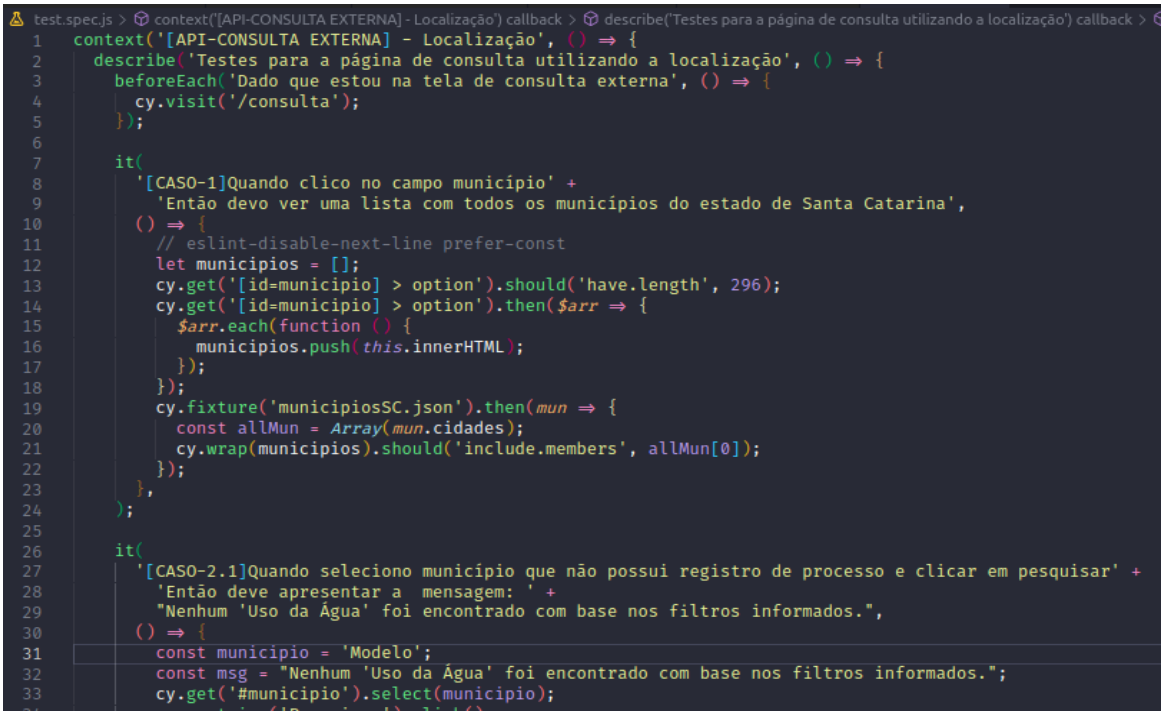


```
post{
  always{
    allure includeProperties: false, jdk: '', results: [[path: 'allure-results']]
  }
  failure {
    discordSend description: "HOUVE ERRO NO BUILD\n${JOB_NAME}\n[relatorio](${env.BUILD_URL}allure)\n[console](${env.BUILD_URL}console)",
      link: "${JOB_URL}",
      thumbnail: "${IMAGE_DISCORD_REPORT}",
      title: "ERRO NO BUILD ${BUILD_ID}",
      result: 'FAILURE',
      footer: "pipeline num ${BUILD_ID}\n",
      webhookURL: "${WEBHOOK}"

    mail to: 'thomazjorge.ti@fundecc.org.br, viniciuspimenta.ti@fundecc.org.br',
    subject: "Testes falharam: ${currentBuild.fullDisplayName}",
    body: "ERRO na execução dos (${env.GIT_BRANCH}): ${env.BUILD_URL}"
  }
  unstable {
    discordSend description: "TESTES FALHARAM\n${JOB_NAME}\n[relatorio](${env.BUILD_URL}allure)\n[console](${env.BUILD_URL}console)",
      link: "${JOB_URL}",
      thumbnail: "${IMAGE_DISCORD_REPORT}",
      title: "O TESTE ${BUILD_ID} FALHOU",
      result: 'FAILURE',
      webhookURL: "${WEBHOOK}"

    mail to: 'thomazjorge.ti@fundecc.org.br, viniciuspimenta.ti@fundecc.org.br',
    subject: "Testes com falha: ${currentBuild.fullDisplayName}",
    body: "Falha em algum teste (${env.GIT_BRANCH}): ${env.BUILD_URL}"
  }
```

Fonte: Do autor (2022)

Os *scripts* do Cypress são escritos utilizando a linguagem javascript, um exemplo de um *script* desenvolvido para a aplicação encontra-se no Apêndice C. Os *scripts* devem começar definindo o contexto em que será realizado o teste, em seguida é apresentada uma breve descrição. O trecho presente no bloco "*beforeEach*" é executado antes de todos os casos de testes definidos nos blocos "*it*" (FIGURA 39).

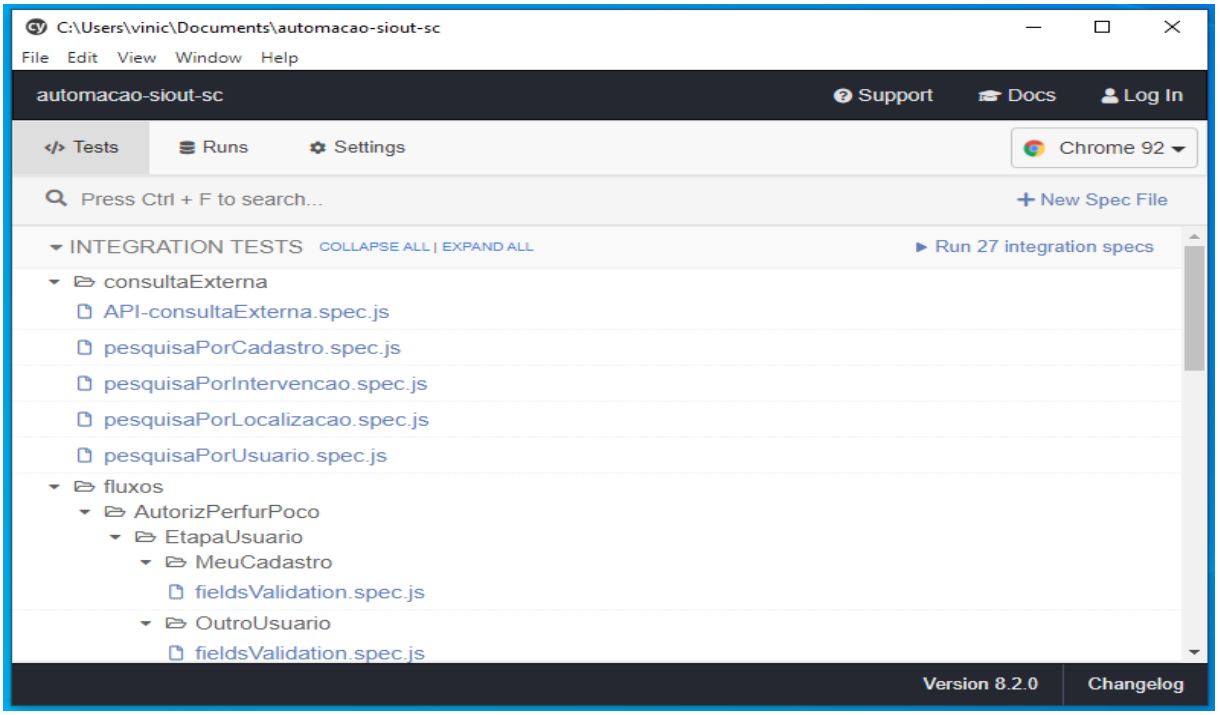
Figura 39 - Exemplo script Cypress



Fonte: Do autor (2022)

Para a criação e desenvolvimento dos testes automatizados, o Cypress possui uma aplicação *Web* , onde é possível executar e acompanhar de forma visual a execução dos *scripts*. A Figura 40 mostra essa interface gráfica para a execução dos testes.

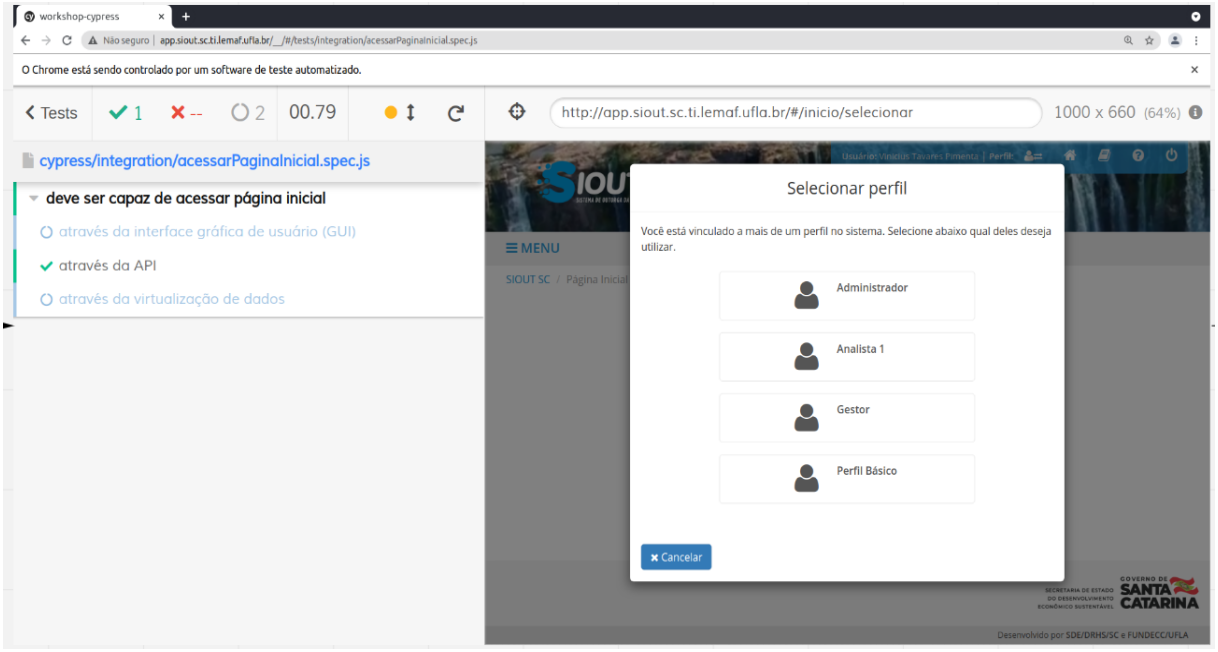
Figura 40 - Interface de testes Cypress



Fonte: Do autor (2022)

Já a Figura 41, mostra um *printscreen* da execução de um teste, onde está verificando se é possível acessar a página inicial do sistema testado:

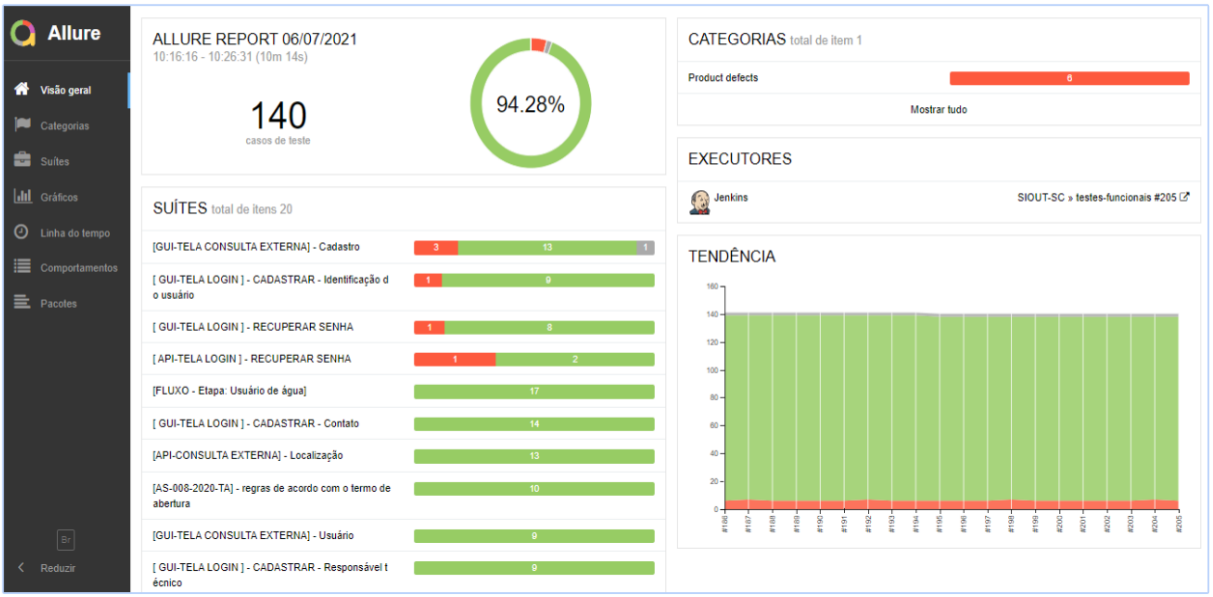
Figura 41 - Execução de um teste com Cypress



Fonte: Do autor (2022)

Após a execução dos testes, é utilizado um framework conhecido como *Allure Report*, responsável por criar uma página web com um relatório da execução dos testes, e enviado para o servidor publicando em uma url, para que a equipe possa acompanhar. A Figura 42 mostra um *printscreen* do dashboard deste relatório.

Figura 42 - Relatório Allure dos testes com Cypress



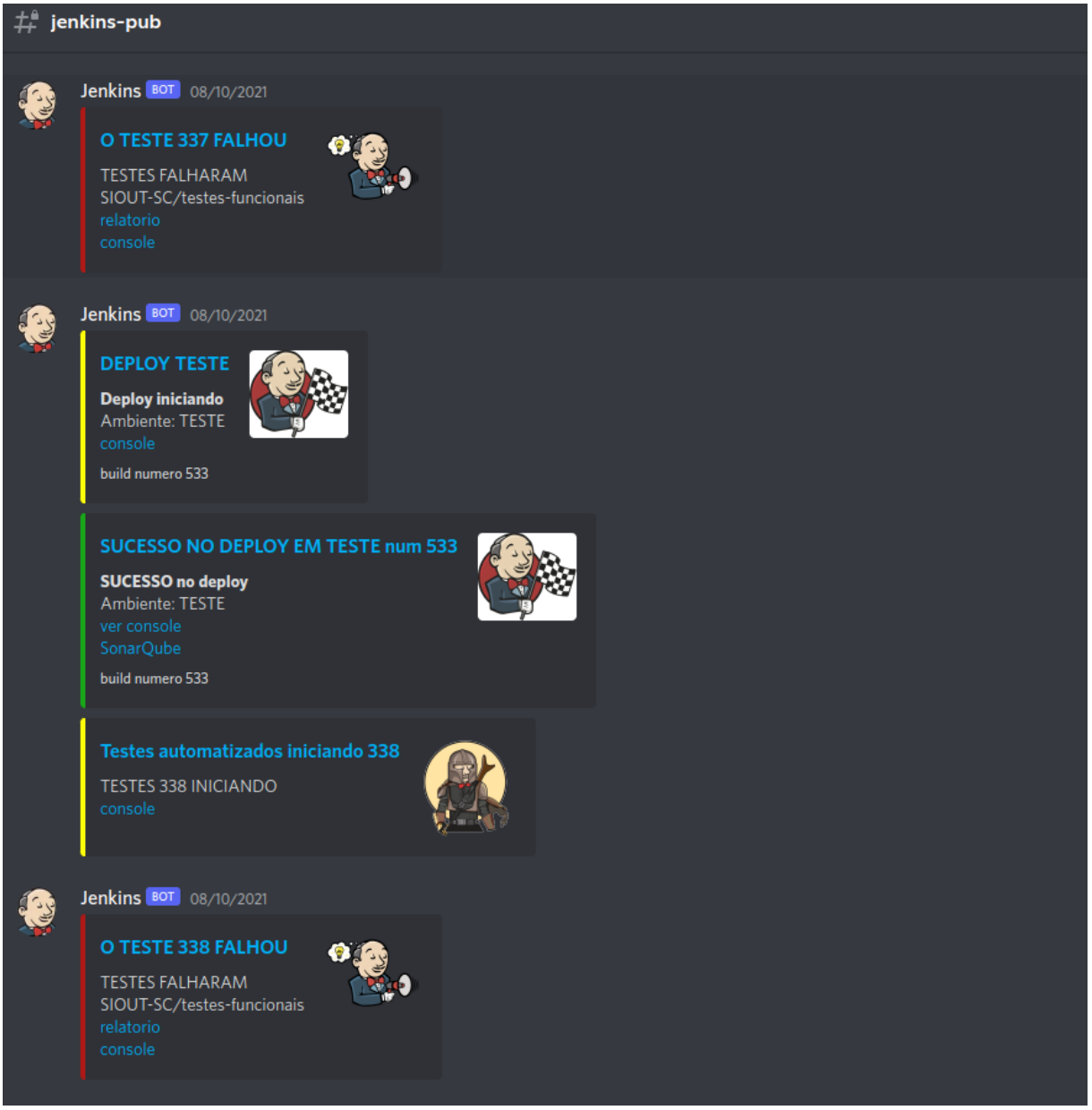
Fonte: Do autor (2022)

3.2.5 Notificando a equipe

Como todo esse processo ocorre sem que alguém impere, é necessário manter toda a equipe informada das ocorrências de implantação e testes do sistema que está sendo desenvolvido, e para isso foi de suma importância utilizar o canal de comunicação já usado pela equipe. O canal é conhecido como Discord e possui salas de bate-papo em texto e canais de voz e vídeo. Foi criada uma sala com o nome de Jenkins-Pub, e adicionada a ela uma integração com um plugin do Jenkins.

O texto e o momento da notificação é definido no Jenkinsfile, e o resultado dessas notificações está apresentado na Figura 43.

Figura 43 - Notificações Discord



Fonte: Do autor (2022)

Como toda a equipe tem acesso às notificações, caso ocorra falha no deploy, os desenvolvedores verificam o motivo, acessando os logs e/ou o relatório do Sonarqube, corrigindo as possíveis causas do erro no processo. Caso seja notificado falha nos testes, cabe à equipe de qualidade realizar a verificação do teste que foi quebrado, reproduzindo e

reportando para os desenvolvedores. Caso todas as notificações sejam positivas, a equipe de qualidade continua com a criação de novos casos de teste que cubram a nova funcionalidade.

3.3 Resultados

Após a realização deste estágio, foi possível vivenciar um aumento da qualidade de código, devido a preocupação em manter as métricas utilizadas pelo SonarQube sempre em um padrão aceitável e criando estratégias de refatoração junto à equipe, focando na diminuição da dívida técnica.

Também houve um aumento na qualidade do produto e consequentemente o cliente mais satisfeito, pois com a implantação de testes automatizados, que executam sempre que há alguma alteração, diminui consideravelmente a repetição de um erro ou falha, criando o que é chamado de testes de regressão.

Ocorreu também um benefício geral na equipe, aumentando sua produtividade, além de desonerar os desenvolvedores com mais experiência, permitindo que foquem realmente no desenvolvimento. A redução de falhas humanas para zero durante o processo de implantação também foi um benefício relevante.

As tabelas a seguir, foram criadas juntamente com a equipe do projeto, descrevendo os resultados obtidos:

a) Tabela 3 representa os ganhos na área de segurança;
b) Tabela 4 representa os benefícios no processo de desenvolvimento;
c) Tabela 5 representa os resultados obtidos nos testes de sistema e end to end;
d) Tabela 6 representa os testes dos endpoints da API do projeto;
e) Tabela 7 representa os testes de unidade e integração.

Tabela 3 - Resultados na área de segurança

Como era	Como está	Benefícios gerados	Como pode melhorar
Não havia inspeção	• Inspeção via Sonar • Testes de API através do Cypress	• Identificação das vulnerabilidades via sonar • Correção de vulnerabilidades identificadas pelo Cypress • Melhoria no relacionamento com o cliente • Redução de issues/bugs	• As vulnerabilidades devem ser atacadas uma a uma • Rotinas de segurança devem ser incorporadas na linha de produção

Fonte: Do autor (2022)

Tabela 4 - Resultados no processo de desenvolvimento

Como era	Como está	Benefícios gerados	Como pode melhorar
• Manual - Tomando 30 minutos por build de um desenvolvedor com mais experiência • Poucos builds durante uma sprint • Vinculado à AS	• Build automático em pipeline do Jenkins • Builds a cada modificação do código (integração contínua) • Comunicação automática via email e discord para a squad	• Redução do tempo de build e deploy • Economia de 10 horas em média por mês de um desenvolvedor experiente • Aumento na confiabilidade do deploy realizado, não permitindo deploy com falhas • Melhoria no relacionamento com o cliente	• Implantar deploy automático em homologação e produção (economia de mais 2 horas ao mês)

Fonte: Do autor (2022)

Tabela 5 - Resultados nos testes de sistema / E2E

Como era	Como está	Benefícios gerados	Como pode melhorar
• Manuais	• Automação dos testes • Criação de 149 casos de testes utilizando a ferramenta Cypress • Relatório dos testes realizados, (allure)	• Testes executados a cada modificação (commit) • Testes de Regressão • Cobertura de funcionalidades • identificação de melhorias • identificação e correção de bugs • Redução do tempo de execução dos testes • Aumento na confiabilidade dos testes • Redução de issues/bugs	• Aumento da quantidade de scripts de testes

Fonte: Do autor (2022)

Tabela 6 - Resultados nos testes de endpoints da API

Como era	Como está	Benefícios gerados	Como pode melhorar
não havia	• Implantação de ferramenta para gerenciar endpoints (Swagger) • Automação das requisições com Cypress	• Testes executados a cada modificação (commit) • Testes de Regressão • Cobertura de funcionalidades • identificação de melhorias • identificação e correção de bugs • Redução do tempo de execução dos testes • Aumento na confiabilidade dos testes • Redução de issues/bugs	• Aumento da quantidade de scripts de testes • Incorporar versionamento na rota • Padronizar rotas

Fonte: Do autor (2022)

Tabela 7 - Resultados nos testes de unidade e integração

Como era	Como está	Benefícios gerados	Como pode melhorar
Não havia	• Capacidade de inserir nova fase no pipeline • Capacidade de automatizar a execução • Capacidade de planejare treinar equipe • Aguardando interesse e disponibilidade do squad/tribo.	• Aguarda implantação	• Implantar testes de unidade • Implantar testes de integração • Incorporar fase no pipeline

Fonte: Do autor (2022)

4 CONSIDERAÇÕES FINAIS

O aprendizado do estagiário durante as atividades desenvolvidas e relatadas aqui, deve ser levado em conta, tendo em vista que foi a primeira experiência profissional, com o apoio e incentivo de toda a equipe e do orientador do estágio.

Todo o conhecimento construído ao longo da graduação é útil para a execução das atividades aqui relatadas e para o desenvolvimento profissional. Todavia, algumas disciplinas foram fundamentais para o desenvolvimento do estágio. A disciplina de Sistemas de Informação, realizada no segundo módulo do curso, foi indispensável com sua abordagem sobre Infraestrutura de TI, hardware e software, auxiliando ainda no entendimento de alguns conceitos sobre ética em sistemas de informação. A disciplina de Engenharia de Software, realizada no quarto módulo da graduação, foi o suporte para todas as atividades desenvolvidas, definindo e destacando os objetivos da Engenharia de Software, planejando o projeto, levantando estimativas e definindo metas e, além de toda a teoria, mostrou que a Engenharia de Software deve ser usada também em benefício de si própria. Gestão de Tecnologia da Informação, outra disciplina do quarto módulo, essencial para acompanhar o desenvolvimento das práticas desempenhadas, alicerçou a análise do estudo de caso, coadjuvando para tomadas de decisões relacionadas ao desempenho dos processos.

Porém, seria de grande valia para este estágio, se houvesse no curso, disciplinas focadas em práticas de gerência de configuração e modificação, uma vez que há muita dificuldade no uso de ferramentas como git e gitlab por exemplo, que foi aprendido de forma eficaz durante as práticas do estágio. Alguma disciplina voltada para a cultura DevOps e seu universo, como automação de processos ou esteira CI/CD. Outro quesito que houve dificuldade, principalmente no início do estágio, foi saber quais testes implementar e o que testar, dificuldade essa que seria diminuída se houvesse alguma disciplina focada em testes, apresentando estratégias e, sobretudo, mostrando o que deve ser testado em um sistema e quais testes podem ser automatizados.

O apoio e união da equipe responsável pelo desenvolvimento do projeto onde o estagiário atuou, conduzida por uma gestão que prioriza pessoas antes de metas, assegurou o

sucesso do resultado obtido, fruto de um investimento em uma cultura que está se propagando por toda a empresa.

Os próximos passos para o estagiário são continuar a manutenção da esteira CI/CD e implantar outras ferramentas DevOps, e ainda utilizar a tecnologia de containers Docker para a aplicação e as ferramentas envolvidas. Após os resultados obtidos com essa experiência, a empresa deve expandir as práticas envolvidas para todos os outros projetos e equipes.

REFERÊNCIAS

BOAGLIO, Fernando. **Jenkins:** Automatize tudo sem complicações. Editora Casa do Código, 2016.

CAMPBELL, G. Ann; SONARSOURCE SA. **COGNITIVE COMPLEXITY:** A new way of measuring understandability. Treinaweb, Switzerland, v. 1.5, p. 01-21, 5 abr. 2021. Disponível em: https://assets-eu-01.kc-usercontent.com/edec468c-78ec-013b-f802-04894160c9cc/e89714ed-2e5b-4119-9ff7-3ccdef359ee9/CognitiveComplexity.pdf. Acesso em: 8 ago. 2022.

CAVALCANTI, Vicenzo. **Framework Scrum:** tudo o que você deveria saber. In: Blog.geekhunter. WEB: Time de Redação da GeekHunter, 24 set. 2020. Disponível em: https://blog.geekhunter.com.br/framework-scrum-voce-esta-usando-corretamente/. Acesso em: 6 ago. 2022.

CYPRESS. **Why Cypress?**. WEB, 2022. Disponível em: https://docs.cypress.io/guides/overview/why-cypress. Acesso em: 7 ago. 2022.

DEFRANCO , Joanna F.; VOAS , Jeffrey. Revisiting Software Metrology: Some topics never fade away but also never advance. Software metrology seems to be one of those. Let's revisit it here.. Computer, Magazine Article, ano 2022, v. 55, ed. 6, 3 jun. 2022. DOI 10.1109/MC.2022.3146648. Disponível em: https://ieeexplore.ieee.org/document/9789303. Acesso em: 11 ago. 2022.

DEV BLOG, QUALITY ASSURANCE. **Cypress:** o novo conceito em testes automatizados. In: ATECH GRUPO EMBRAER. Atech. WEB, 8 fev. 2021. Disponível em: https://atech.com.br/cypress-o-novo-conceito-em-testes-automatizados/. Acesso em: 8 ago. 2022.

DUARTE, Luiz. **Scrum e Métodos Ágeis:** Um Guia Prático. 2. ed. atual. [S. l.: s. n.], outubro de 2016.

FUNDECC, Sobre a Fundecc. 2022. Disponível em: http://www.fundecc.org.br/sobre-a-fundecc . Acesso em: 06 de Agosto de 2022.

_____. Política de Integridade, 2020. Disponível em: http://www.fundecc.org.br/politica-de-integridade/ . Acesso em: 6 ago. 2022.

GIT. --distributed-even-if-your-workflow-isnt. Documentation. WEB, 2022. Disponível em: https://git-scm.com/. Acesso em: 7 ago. 2022.

GITLAB. **The One DevOps Platform**. About WEB, 2022. Disponível em: https://about.gitlab.com/. Acesso em: 8 ago. 2022.

GOCACHE. **O que é Jenkins? Para iniciantes**. In: GOCACHE. WEB, 14 abr. 2021. Disponível em: https://www.gocache.com.br/dicas/o-que-e-jenkins-para-iniciantes/. Acesso em: 7 ago. 2022.

JÂNIO, Fábio. Métricas e sua importância para a engenharia de software. WEB, 8 maio 2019. Disponível em: https://fabiojanio.medium.com/m%C3%A9tricas-e-sua-import%C3%A2ncia-para-a-engenharia-de-software-2c48905f1eb9. Acesso em: 6 set. 2022.

JENKINS CD. **Jenkins User Documentation**. WEB, 2021. Disponível em: https://www.jenkins.io/doc/. Acesso em: 7 ago. 2022.

KERZNER, H. **Gestão de Projetos**: As Melhores Práticas. 4 ed. Porto Alegre: Bookman, 2016. 9788582605301. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582605301/. Acesso em: 07 Aug 2022

LOUZADA, Paula. **O que é DevOps:** Quebre a barreira entre desenvolvimento e operações. In: FM2S. FM2S: Educação e Consultoria. WEB, 4 ago. 2019. Disponível em: https://www.fm2s.com.br/o-que-e-devops-quebre-a-barreira-entre-desenvolvimento-e-operacoes/. Acesso em: 8 ago. 2022.

LUCANIA, Israel. **O que é o SonarQube?**. In: Konia. [S. l.], 2020. Disponível em: https://konia.com.br/o-que-e-o-sonarqube/. Acesso em: 9 ago. 2022.

MAIA, C. L. B. et al. **A Multi-Objective Approach for the Regression Test Case Selection Problem**, XLI Simpósio Brasileiro de Pesquisa Operacional, 2009.

OLIVEIRA, Eloiza da Silva Gomes de; CUNHA, Vera Lúcia. **O estágio Supervisionado na formação continuada docente a distância: desafios a vencer e Construção de novas subjetividades**. Publicación en línea. Murcia (España). Año V. Número 14.- 31 de Marzo de 2006. Disponível em https://www.um.es/ead/red/14/oliveira.pdf. Acesso em: 7 ago. 2022.

OLIVEIRA, Welliton. O que é scrum? Conceito, definições e etapas. In: EVOLVE. **Evolve:** CONSULTORIA PARA STARTUPS. WEB, 16 set. 2019. Disponível em: https://evolvemvp.com/o-que-e-scrum-conceito-definicoes-e-etapas/. Acesso em: 7 ago. 2022.

PASSERINI, Gislaine Alexandre. **O estágio supervisionado na formação inicial de**

PMBOK. **Um Guia de Conjunto de Conhecimentos em Gerenciamento de Projetos** (Guia PMBOK®). Project Management Institute, Four Campus Boulevard, Newton Square, Pennsylvania, USA, Sixth Edition, 2017.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software.** 8 ed. Porto Alegre: Bookman, 2016. 9788580555349. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580555349/. Acesso em: 07 Aug 2022

RED HAT. CI/CD: integração e entrega contínuas. In: RED HAT. **CI/CD:** integração e entrega contínuas. WEB, 12 abr. 2018. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd. Acesso em: 7 ago. 2022.

_____. **Introdução ao DevOps.** WEB, 19 abr. 2018. Disponível em: https://www.redhat.com/pt-br/topics/devops. Acesso em: 8 ago. 2022.

SCHWABER, Ken; SUTHERLAND, Jeff. The 2020 Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. In: SUTHERLAND, Jeff; SCHWABER, Ken. **Scrum Guides**. Oficial. WEB, novembro de 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf. Acesso em: 6 ago. 2022.

SIOUT-SC. **Sistema de Outorga de SC**, 2022. Disponível em: http://siout.aguas.sc.gov.br/. Acesso em: 8 ago. 2022.

SONARQUBE. Docs. WEB, 2022. Disponível em: https://docs.sonarqube.org/latest/. Acesso em: 7 ago. 2022.

TEDESCO, Kennedy. **Complexidade ciclomática, análise estática e refatoração**. In: TREINAWEB. Treinaweb: Escola online para desenvolvedores. WEB, 2016. Disponível em: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao/. Acesso em: 8 ago. 2022.

UNYSCAPE.COM. Scrum vs Waterfall: Which one is More Suitable for Your Project. In: UNYSCAPE.COM. **Unyscape:** Crafting Digital Success Stories for over 11 Years. WEB, 12 abr. 2019. Disponível em: https://unyscape.com/scrum-vs-waterfall-which-one-is-more-suitable-for-your-project/. Acesso em: 7 ago. 2022.

VALENTE, Marco Túlio. **Engenharia de Software Moderna:** Princípios e Práticas para Desenvolvimento de Software com Produtividade. 1. ed. rev. [S. l.]: Independente, 2020. *E-book* v. 1. ISBN 978-65-00-00077-1.

Version One. The 11th annual State of Agile™ Report. [S.l.], 2017

ZETTLER, Kev. O que é desenvolvimento baseado em tronco?. In: ATLASSIAN. **Desenvolvimento de software**. WEB, 2020. Disponível em: https://www.atlassian.com/br/continuous-delivery/continuous-integration/trunk-based-development#:~:text=O%20desenvolvimento%20baseado%20em%20tronco,%22tronco%22%20ou%20ramifica%C3%A7%C3%A3o%20principais. Acesso em: 8 ago. 2022.

APÊNDICE A - Jenkinsfile Siout-SC ambiente de teste

```
pipeline{
 agent any
 options { timestamps() }
 tools {nodejs "nodejs12.18.2"}
 stages{
   stage('Preparando o ambiente'){
     steps{
       discordSend description: "Deploy iniciando\n
                                 Ambiente:
${SIOUT_SC_DEFINE_CONSTANTS}\n
                                 [console](${env.BUILD_URL}console)",
         link: "${JOB_URL}",
         thumbnail: "${IMAGE_DISCORD}",
         result: "UNSTABLE",
         title: "DEPLOY ${SIOUT_SC_DEFINE_CONSTANTS}",
         webhookURL: "${WEBHOOK}",
         footer: "build numero ${BUILD_ID}"

       dir('Interface'){
         nodejs(nodeJSInstallationName: 'nodejs12.18.2'){
           bat 'copy %SIOUT_SC_DEPENDENCIAS%\\token'
           bat 'robocopy %SIOUT_SC_DEPENDENCIAS%\\node_modules
./node_modules /mt /z /s /e & EXIT /B 0 '
           bat 'robocopy %SIOUT_SC_DEPENDENCIAS%\\bower_components
./bower_components /mt /z /s /e & EXIT /B 0 '
           bat 'nvm use 12.18.2'
           bat 'node --version'
         }
       }
     }
   }
   stage('Construindo frontend'){
     steps{
       dir('Interface'){
         bat 'gulp'
```

```
        }
      }
    }
    stage('Construindo backend'){
      steps{
        bat "nuget restore SioutSC.sln"
        bat "MSBuild.exe ./Siout/Siout.csproj /t:Rebuild
/p:DeployOnBuild=true
            /p:DefineConstants=%SIOUT_SC_DEFINE_CONSTANTS%
/p:PublishProfile=%SIOUT_SC_PUBLISH_PROFILE%"
      }
    }
    stage('Construindo o Zip e Preparando o BKP'){
      steps{
        bat 'copy %SIOUT_SC_DEPENDENCIAS%\\versao.html .\\Interface'
        dir('Interface'){
          bat 'gulp gerarZip'
        }
        dir('Interface/dist'){
          bat 'dir'

          //Agrupar os zips
          bat 'WinRAR.exe a -r -ag+YYYY-MM-DD-
"%SIOUT_SC_NAME_ZIP_BKP%.zip" logica.zip interface.zip'

          //Extraindo os arquivos gerados
          bat 'WinRAR.exe x -y logica.zip logica/ & EXIT /B 0'
          bat 'WinRAR.exe x -y interface.zip interface/ & EXIT /B 0'

          //Remover os zips desnecessarios
          bat 'del interface.zip'
          bat 'del logica.zip'

          //criar a pasta bkp
          bat 'mkdir backup'
          bat 'move *.zip backup'
        }
```

```
    }
  }
  stage('Analise do sonarqube'){
    steps{
      bat "SonarScanner.MSBuild.exe begin
/k:${'"%SIOUT_SC_SONAR_KEY%"'}
          /d:sonar.host.url=${'"%SIOUT_SC_SONAR_URL%"'}
/d:sonar.login=${'"%SIOUT_SC_SONAR_LOGIN%"'}"
      bat "MSBuild.exe /t:Rebuild"
      bat "SonarScanner.MSBuild.exe end
/d:sonar.login=${'"%SIOUT_SC_SONAR_LOGIN%"'}"
    }
  }
  stage('Deploy para ambiente de teste'){
    steps{
      dir('Interface/dist'){
        //Entrar na pasta do servidor
        bat 'net use
\\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout
          "%SIOUT_SC_PASSWORD_SERVER%" /USER:%SIOUT_SC_USER_SERVER%'

        //Copiar o codigo gerado para as pastas
        bat 'robocopy logica/
\\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\Logica
\\
              /mt /z /s /e & EXIT /B 0'
        bat 'robocopy interface/
\\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\Interf
ace\\
              /mt /z /s /e & EXIT /B 0'
        bat 'robocopy backup/
\\\\%SIOUT_SC_SERVER_PUBLISH%\\%SIOUT_SC_FOLDER_PUBLISH%\\Siout\\backup
\\
              /mt /z /s /e & EXIT /B 0'
      }
    }
  }
```

```
 }
 post{
   failure {
     discordSend description: "Houve falha no deploy\n
                               Ambiente: ${SIOUT_SC_DEFINE_CONSTANTS}\n
                               [ver console](${env.BUILD_URL}console)\n

[SonarQube](${env.SONAR_UR}${env.SIOUT_SC_SONAR_KEY})",
         link: "${JOB_URL}",
         thumbnail: "${IMAGE_DISCORD}",
         result: "FAILURE",
         title: "FALHA NO DEPLOY ${SIOUT_SC_DEFINE_CONSTANTS} num
${BUILD_ID}",
         webhookURL: "${WEBHOOK}",
         footer: "build numero ${BUILD_ID}"
   }
   success {
     discordSend description: "SUCESSO no deploy\n
                               Ambiente: ${SIOUT_SC_DEFINE_CONSTANTS}\n
                               [ver console](${env.BUILD_URL}console)\n

[SonarQube](${env.SONAR_UR}${env.SIOUT_SC_SONAR_KEY})",
         link: "${JOB_URL}",
         thumbnail: "${IMAGE_DISCORD}",
         result: "SUCCESS",
         title: "SUCESSO NO DEPLOY EM ${SIOUT_SC_DEFINE_CONSTANTS} num
${BUILD_ID}",
         webhookURL: "${WEBHOOK}",
         footer: "build numero ${BUILD_ID}"
   }
 }
}
```

APÊNDICE B - Jenkinsfile testes automatizados

```
pipeline{
 agent any
 options { timestamps() }
 stages{
   stage('preparando'){
     steps{
       discordSend description: "TESTES ${BUILD_ID}
INICIANDO\n[console](${env.BUILD_URL}console)",
         link: "${JOB_URL}",
         thumbnail: "${IMAGE_DISCORD}",
         result: "UNSTABLE",
         title: "Testes automatizados iniciando ${BUILD_ID}",
         webhookURL: "${WEBHOOK}"

       bat 'nvm use 14'
       bat 'npm install'
     }
   }
   stage('teste funcional - Cypress'){
     steps{
       ansiColor('xterm'){
         bat 'npm run cy:test'
       }
     }
   }
 }
   post{
     always{
       allure includeProperties: false, jdk: '', results: [[path:
'allure-results']]
     }
     failure {
       discordSend description: "HOUVE ERRO NO
BUILD\n${JOB_NAME}\n[relatorio](${env.BUILD_URL}allure)\n[console](${en
v.BUILD_URL}console)",
```

```
        link: "${JOB_URL}",
        thumbnail: "${IMAGE_DISCORD_REPORT}",
        title: "ERRO NO BUILD ${BUILD_ID}",
        result: 'FAILURE',
        footer: "pipeline num ${BUILD_ID}\n",
        webhookURL: "${WEBHOOK}"

      mail to: 'thomazjorge.ti@fundecc.org.br,
viniciuspimenta.ti@fundecc.org.br',
      subject: "Testes falharam: ${currentBuild.fullDisplayName}",
      body: "ERRO na execução dos (${env.GIT_BRANCH}):
${env.BUILD_URL}"
    }
    unstable {
      discordSend description: "TESTES
FALHARAM\n${JOB_NAME}\n[relatorio](${env.BUILD_URL}allure)\n[console](${
env.BUILD_URL}console)",
        link: "${JOB_URL}",
        thumbnail: "${IMAGE_DISCORD_REPORT}",
        title: "O TESTE ${BUILD_ID} FALHOU",
        result: 'FAILURE',
        webhookURL: "${WEBHOOK}"

      mail to: 'thomazjorge.ti@fundecc.org.br,
viniciuspimenta.ti@fundecc.org.br',
      subject: "Testes com falha: ${currentBuild.fullDisplayName}",
      body: "Falha em algum teste (${env.GIT_BRANCH}):
${env.BUILD_URL}"
    }
    success {
      discordSend description: "TODOS OS TESTES FORAM EXECUTADOS COM
SUCESSO\n${JOB_NAME}\n[relatorio](${env.BUILD_URL}allure)\n[console](${
env.BUILD_URL}console)",
        link: "${JOB_URL}",
        thumbnail: "${IMAGE_DISCORD_REPORT}",
        title: "SUCESSO NO TESTE ${BUILD_ID}",
        result: 'SUCCESS',
```

```
          webhookURL: "${WEBHOOK}"

        mail to: 'thomazjorge.ti@fundecc.org.br,
viniciuspimenta.ti@fundecc.org.br',
        subject: "Testes com sucesso: ${currentBuild.fullDisplayName}",
        body: "Sucesso nos testes (${env.GIT_BRANCH}): ${env.BUILD_URL}"
      }
  }
}
```

APÊNDICE C - Script de testes com Cypress

```
context('[API-CONSULTA EXTERNA] - Localização', () => {
 describe('Testes para a página de consulta utilizando a localização',
() => {
   beforeEach('Dado que estou na tela de consulta externa', () => {
     cy.visit('/consulta');
   });

   it(
     '[CASO-1]Quando clico no campo município' +
       'Então devo ver uma lista com todos os municípios do estado de
Santa Catarina',
     () => {
       // eslint-disable-next-line prefer-const
       let municipios = [];
       cy.get('[id=municipio] > option').should('have.length', 296);
       cy.get('[id=municipio] > option').then($arr => {
         $arr.each(function () {
           municipios.push(this.innerHTML);
         });
       });
       cy.fixture('municipiosSC.json').then(mun => {
         const allMun = Array(mun.cidades);
         cy.wrap(municipios).should('include.members', allMun[0]);
       });
     },
   );

   it(
     '[CASO-2.1]Quando seleciono município que não possui registro de
processo e clicar em pesquisar' +
       'Então deve apresentar a  mensagem: ' +
       "Nenhum 'Uso da Água' foi encontrado com base nos filtros
informados.",
     () => {
       const municipio = 'Modelo';
```

```
      const msg = "Nenhum 'Uso da Água' foi encontrado com base nos
filtros informados.";
      cy.get('#municipio').select(municipio);
      cy.contains('Pesquisar').click();
      cy.get('[ng-show="pesquisado"] > :nth-child(2) > .text-center')
        .should('be.visible')
        .should('contain.text', msg);
    },
  );

  it(
    '[CASO-2.2]Quando seleciono qualquer município que possua registro
de processo e clicar em pesquisar' +
      'Então deve carregar todos os cadastros relacionados ao
município selecionado',
    () => {
      const municipio = 'Abelardo Luz';
      cy.get('#municipio').select(municipio);
      cy.contains('Pesquisar').click();
      cy.get('[ng-repeat="item in $data"]')
        .should('be.visible')
        .then(() => {
          cy.get('tr').then(a => {
            expect(a).contain(municipio);
          });
        });
    },
  );

  it(
    '[CASO-3]Quando clico em Bacia hidrográfica sem ter selecionado
algum município' +
      'Então deve exibir uma lista com todas bacias hidrográficas do
estado de Santa Catarina',
    () => {
      // eslint-disable-next-line prefer-const
      cy.reload();
```

```
      const bacias = [];
      cy.get('#baciaHidrografica > option').should('have.length', 26);
      cy.get('#baciaHidrografica > option').then($arr => {
        $arr.each(function () {
          bacias.push(this.innerHTML);
        });
      });
      cy.fixture('baciasSC.json').then(bac => {
        const allBacias = Array(bac.bacias);
        cy.wrap(bacias).should('include.members', allBacias[0]);
      });
    },
  );

  it(
    '[CASO-4]Quando seleciono qualquer município e clico em bacia
hidrográfica' +
      'Então deve exibir uma lista com as bacias hidrográficas
localizadas no município',
    () => {
      const baciasFloripa = [
        'Bacias Hidrográficas da Ilha de Santa Catarina',
        'Rio Biguaçú e bacias contíguas',
        'Rio Cubatão e bacias contíguas',
      ];

cy.intercept('/siout/BaciaHidrografica/ListarPorMunicipio?idMunicipio=4
205407').as(
        'baciaMunicipio',
      );
      cy.get('#municipio').select('Florianópolis');
      cy.wait('@baciaMunicipio').then(() => {
        cy.get('#baciaHidrografica > option').then($res => {
          cy.wrap($res).should('have.length', 4);
          const baciasListadas = [];
          $res.each(function () {
            baciasListadas.push(this.innerHTML);
```

```
            });
            cy.wrap(baciasListadas).should('include.members',
baciasFloripa);
          });
        });
      },
    );

    it(
      '[CASO-5]Quando clico em  Corpo hídrico sem ter selecionado uma
bacia hidrográfica e um município' +
        'Então o campo deve estar bloqueado',
      () => {
        cy.reload();
        cy.get(':nth-child(6) > .row > :nth-child(1) > .col-md-8 >
.form-control').should(
          'be.disabled',
        );
      },
    );

    it(
      '[CASO-6]Quando clico em corpo hídrico após ter selecionado alguma
bacia hidrográfica' +
        'Então deve exibir uma lista com todos os corpos hídricos
pertencentes à bacia',
      () => {
        cy.reload();
        cy.get('#baciaHidrografica').select('Rio Canoas');
        cy.get('#corpoHidrico > option').should('have.length', 603);
      },
    );

    it(
      '[CASO-7]Quando clico em sistema aquífero' +
        'Então deve exibir uma lista com todos os tipos de sistema
aquíferos',
```

```
    () => {
      cy.fixture('sistemaAquiferos.json').then($resp => {
        Array.from($resp.Entity).forEach;
      });

      cy.get('#sistemaAquifero');
    },
  );

  it(
    '[CASO-8]Quando seleciono alguma bacia hidrográfica qualquer e
clicar em pesquisar' +
      'Então deve exibir o resultado com todos os cadastros realizados
na bacia em questão',
    () => {
      cy.get('#baciaHidrografica').select('Rio Canoas');
      cy.contains('Pesquisar').click();
      cy.get('[ng-show="pesquisado"]').should('be.visible');
    },
  );

  it(
    '[CASO-9]Quando seleciono alguma bacia hidrográfica qualquer sem
selecionar município e clicar em ' +
      'pesquisar, E verificar algum cadastro do resultado' +
      'Então o cadastro deve pertencer à algum município da bacia
selecionada',
    () => {
      cy.reload();
      const municipios = [];
      const bacia = 'Rio Chapecó e bacias contíguas';
      cy.get('#baciaHidrografica').select(bacia);
      cy.contains('Pesquisar').click();
      cy.get('[data-title-text="Município"]')
        .should('be.visible')
        .then(l => {
          const linha = Array.from(l);
```

```
            linha.forEach(el => {
              if (municipios.includes(el.innerHTML) === false) {
                municipios.push(el.innerHTML);
              }
            });
          });
        cy.wrap(municipios).each(mun => {
          if (mun !== ' <!-- ngRepeat: (name, filter) in
$column.filter(this) --> ') {
            cy.get('#municipio')
              .select(mun)
              .then(() => {
                cy.get('.ajax-loader').should('be.not.visible');
                cy.get('#baciaHidrografica').should('contain', bacia);
              });
          }
        });
      },
    );
    it(
      '[CASO-10]Quando selecionar algum sistema aquífero e clicar em
pesquisar' +
        'Então deve exibir o resultado com todos os cadastros
realizados',
      () => {
        cy.get('#sistemaAquifero').select('Aquíferos fraturados de menor
potencialidade (af3)');
        cy.contains('Pesquisar').click();
        cy.get('[ng-show="pesquisado"]').should('be.visible');
      },
    );

    it(
      '[CASO-11]Quando seleciono todos os campos e clico em cancelar' +
        'Então todos os campos devem estar sem valor selecionado',
      () => {
        cy.get('#municipio').select('Abdon Batista');
```

```
        cy.get('#baciaHidrografica').select('Rio Canoas');
        cy.get('#corpoHidrico').select('Arroio Boa Vista');
        cy.get('#sistemaAquifero').select('Aquíferos fraturados de menor
potencialidade (af3)');
        cy.contains('Limpar').click();
        cy.get('#municipio option:selected').should('have.text',
'Selecione...');
        cy.get('#baciaHidrografica option:selected').should('have.text',
'Selecione...');
        cy.get(':nth-child(6) > .row > :nth-child(1) > .col-md-8 >
.form-control').should(
          'be.disabled',
        );
        cy.get('#sistemaAquifero option:selected').should('have.text',
'Selecione...');
      },
    );

    it(
      '[CASO-12]Quando seleciono qualquer bacia e qualquer corpo hídrico
e clico em pesquisar' +
        'Então deve exibir o resultado com os cadastros enquadrados na
pesquisa, ou a mensagem: ' +
        "Nenhum 'Uso da Água' foi encontrado com base nos filtros
informados.",
      () => {
        cy.get('#baciaHidrografica').select('Rio Canoas');
        cy.get('#corpoHidrico').select('Arroio Boa Vista');
        cy.contains('Pesquisar').click();
        cy.get('[ng-show="pesquisado"]').should('be.visible');
      },
    );
  });
});
```