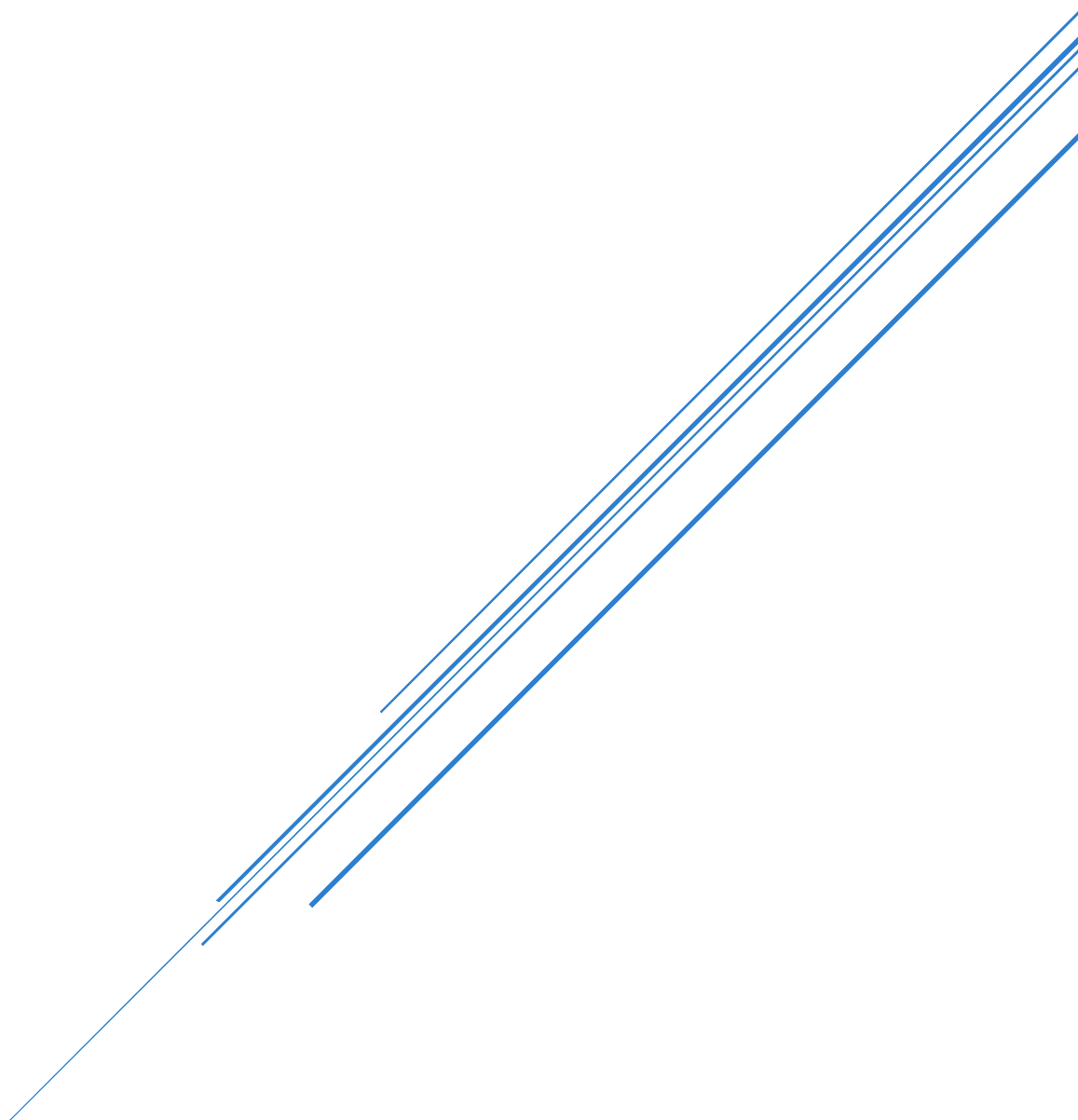


# RELATÓRIO FINAL

GESTÃO E QUALIDADE DE SOFTWARE



UNIFACS  
Junho de 2024

# RELATÓRIO FINAL

## GESTÃO E QUALIDADE DE SOFTWARE

Bruno Sales Fiaes Carneiro

Marcelo Miranda Cavalcanti

Zanoni Campos Fernandes

Luis Eduardo da Silva Belinelli

Diego Amaro Ferreira

George Gebers Brizolla

Levi de Oliveira Dourado

UNIFACS  
Junho de 2024

# INTRODUÇÃO

A gestão de qualidade no desenvolvimento de software é crucial para garantir que as entregas atendam às expectativas dos clientes e dos usuários finais. Ao implementar práticas e processos de qualidade desde as fases iniciais do desenvolvimento, as equipes podem identificar e resolver problemas de forma eficaz, contribuindo para a satisfação do cliente e para a manutenção de uma boa reputação da empresa no mercado.

O emprego de sistemas de controle de versão é essencial para gerenciar o código-fonte de maneira organizada e eficaz durante o ciclo de vida do desenvolvimento de software. Ferramentas como Git possibilitam o rastreamento de alterações, a colaboração entre desenvolvedores e a reversão de modificações indesejadas, promovendo a coesão e a integridade do código. Isso contribui para a produção de software de qualidade, com histórico claro de evolução e manutenção.

Em conjunto com o Git, o uso de Conventional Commits ajuda a promover uma padronização na forma como as alterações são registradas no controle de versão. Ao seguir um conjunto de convenções pré-definidas para mensagens de commit, as equipes podem categorizar e rastrear facilmente as mudanças, facilitando a compreensão e revisão do histórico de desenvolvimento.

Uma das práticas da entrega contínua é a automatização da entrega do software ao ambiente de produção sempre que uma nova funcionalidade é implementada e passa nos testes automatizados. Essa abordagem acelera o ciclo de entrega, permitindo que as equipes entreguem valor de forma mais rápida e consistente. Além disso, ao automatizar todo o processo de implantação, desde a integração até a entrega, a entrega contínua reduz o risco de erros humanos e garante que o software entregue ao usuário final seja confiável e de alta qualidade.

O objetivo deste trabalho é utilizar em conjunto ferramentas de testes de software, de controle de versão e de automatização para criar um ambiente de entrega contínua. As ferramentas em questão são: Git, Junit, Mockito, JaCoCo, Github Actions e SonarCloud. No final deste processo, é possível visualizar os resultados dos testes de software e da análise de qualidade do código-fonte do projeto.

## **ESTRATÉGIA USADA PARA ELABORAR OS TESTES**

O projeto utilizado para a realização deste trabalho é um programa de gerenciamento de um mercado de pequeno porte. Os testes propostos para este trabalho são testes unitários e testes de integração.

### **CLIENTE**

Neste teste é verificado o comportamento da classe Cliente do sistema. Ele inclui testes para os métodos de configuração e obtenção de atributos do cliente, como ID, nome, CPF e data de nascimento. Além disso, há testes de validação para o nome, CPF e data de nascimento do cliente, garantindo que as entradas sejam válidas. Esses testes ajudam a assegurar que a classe Cliente funcione corretamente e valide corretamente as informações dos clientes no sistema.

### **PRODUTO**

Neste teste é verificado o comportamento da classe Produto. Ele inclui testes que verificam se os getters e setters estão funcionando corretamente. Também testa os métodos de verificação de Strings vazias, de datas inválidas, e de valores numéricos nulos.

### **PEDIDO**

Neste teste é verificado o comportamento da classe Pedido. Ele testa se a lista de pedidos do objeto criado está funcionando corretamente, armazenando a quantidade correta de pedidos.

### **DBTEST**

Neste teste é verificado o comportamento da classe que interage com classes de acesso ao banco de dados. Para realizar este teste sem utilizar o banco de dados, a classe de acesso ao banco de dados é “mockada” utilizando o Mockito. Seu comportamento é simulado, e a classe que interagiria com o BD tem as suas respostas validadas interagindo com a classe “mockada”.

### **ITCASE**

Neste teste é verificado o comportamento das classes que realizam o acesso ao banco de dados. São testes de integração, para verificar o comportamento lógico do programa, e a

conexão entre o programa e o serviço externo. O teste realiza inserções no banco de dados e verifica o resultado obtido após leituras, para diferentes campos da tabela no BD. Os dados inseridos neste teste são deletados no final, para evitar problemas futuros (por exemplo, a reexecução deste mesmo teste).

## **EXECUÇÃO DE TESTES E GERAÇÃO DE RELATÓRIO**

Os testes unitários foram executados através do JUnit e Mockito, em conjunto com a ferramenta de automatização de compilação e gerenciamento de dependências, o Maven. Além das dependências necessárias para a realização dos testes unitários e de integração, também foi especificado no Maven o uso da ferramenta JaCoCo.

O JaCoCo verifica os testes que são executados em um projeto Java, analisando quais instruções do código são de fato utilizadas em cada teste. No fim, é gerado um relatório que mostra a cobertura de teste de código-fonte. O relatório é armazenado localmente em formatos diversos, como HTML. Com este relatório, é possível visualizar informações superficiais, como a porcentagem de cobertura de teste, e também informações mais específicas, como quais instruções do código foram verificados nos testes.

## GITHUB ACTIONS

O GitHub é uma plataforma online de repositórios Git. Uma das ferramentas que o GitHub oferece é o GitHub Actions, uma ferramenta de integração contínua e entrega contínua, totalmente integrada ao repositório Git hospedado na plataforma.

Utilizando arquivos localizados em uma pasta específica do repositório, o GitHub provisiona máquinas virtuais para executar “trabalhos”, cujas ações são especificadas em arquivos YAML. Esses trabalhos ocorrem baseados em “eventos”, que servem como gatilhos para dar início a um ou mais trabalhos. Alguns exemplos de eventos são: realizar um push, criar um Pull Request, criar um Issue, realizar um merge entre branches, etc.

Com essa ferramenta, é possível executar testes unitários e de integração, além da geração de relatório do JaCoCo, de forma automatizada sempre que for feito um push no repositório.

Também é possível verificar as mensagens de commit do repositório, utilizando expressões regulares (Regex), para verificar que as mensagens seguem as regras estabelecidas pelo Conventional Commits.

Além de estabelecer padrões para prefixos de mensagens de commits, o Conventional Commits também estabelece regras para incrementos de número de versões, baseado no Semantic Versioning. Com o GitHub Actions, também é possível automatizar essa tarefa de geração de versões.

## **GIT FLOW**

O Git Flow é um modelo de fluxo de trabalho que organiza o desenvolvimento de software em branches distintos, promovendo uma abordagem estruturada para colaboração de desenvolvimento. Ele estabelece uma hierarquia de branches, como o "main", "dev" e "release" (entre outros). Este modelo facilita o gerenciamento de versões do software, permitindo uma transição entre o desenvolvimento de novos recursos, a preparação de lançamentos e a correção de problemas em produção.

Neste projeto, o branch principal de desenvolvimento foi o "dev". O branch main foi configurado para impedir a realização de push, possível apenas através de um PR no GitHub. Assim, o branch main fica protegido contra commits sem antes passarem pelos testes automatizados, e sem antes ser revisado e receber aprovações dos desenvolvedores designados.

No GitHub Actions, todo push é um evento para realizar a verificação das mensagens de commit e executar os testes unitários. Ao alcançar um estado de desenvolvimento considerado adequado para o ambiente de produção, é realizado um PR para a branch release. Os testes unitários são executados neste branch.

Caso todos os testes sejam bem-sucedidos, pode-se abrir um PR para realizar o merge com o branch main. Se aprovado, o GitHub Actions verifica as mensagens de commit para gerar ou incrementar a versão do projeto.




# RESULTADOS

Triggered via push last week

Status

Total duration

Artifacts

 mmcav pushed · 74a9a0d · dev


Success

55s


—

ci\_cd.yml


on: push

 preflight

7s

 build


7s


 test


13s

preflight

succeeded last week in 7s

>  Set up job

>  Run actions/checkout@v4


>  Validação da mensagem de commit


1 ▶ Run echo "Validando mensagens de commit..."


27 Validando mensagens de commit...


28 Commit Message: ci: removing the extra step in favor of using the Job Summary feature

29 Valid commit message.

>  Instalação do JDK for x64


>  Post Instalação do JDK for x64


>  Post Run actions/checkout@v4


>  Complete job


test


succeeded last week in 13s

>  Set up job

>  Run actions/checkout@v4

>  Rodar testes unitários

>  Post Run actions/checkout@v4

>  Complete job

integration-tests

succeeded last week in 1m 21s

>

✓

Set up job

>

✓

Run actions/checkout@v4

>

✓

Rodar testes de integração

>

✓

Envio de relatório do Jacoco via FTP

>

✓

Post Run actions/checkout@v4

>

✓



Complete job

Triggered via workflow run last week

Status

Total duration

Artifacts

 mmcav completed  74a9a0d

Success

1m 32s

—

release.yml

on: workflow\_run

✓

integration-tests

1m 21s

✓

versioning

0s

integration-tests summary

Link para o Relatório do Jacoco

[https://joaobsjunior.com.br/trabalho-a3/facs-qualidade-software-projetoA3/relatorio\\_jacoco\\_2024-06-06\\_15-13-21/](https://joaobsjunior.com.br/trabalho-a3/facs-qualidade-software-projetoA3/relatorio_jacoco_2024-06-06_15-13-21/)

Job summary generated at run-time

## mercadinho.app

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Pedido.java	<div><div></div></div>	62%	<div><div></div></div>	50%	6	11	8	19	3	7	0	1
Produto.java	<div><div></div></div>	75%	<div><div></div></div>	71%	5	20	10	33	1	13	0	1
Cliente.java	<div><div></div></div>	89%	<div><div></div></div>	61%	13	33	7	64	0	15	0	1
DbConexao.java	<div><div></div></div>	50%	<div><div></div></div>	n/a	1	2	6	11	1	2	0	1
App.java	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	7	7	2	2	1	1
DbSelecao.java	<div><div></div></div>	84%	<div><div></div></div>	75%	2	8	4	23	1	6	0	2
DbAtualizacao.java	<div><div></div></div>	77%	<div><div></div></div>	50%	2	6	4	19	1	5	0	1
DbInsercao.java	<div><div></div></div>	72%	<div><div></div></div>	50%	2	3	4	14	1	2	0	1
DbDelecao.java	<div><div></div></div>	58%	<div><div></div></div>	50%	2	3	4	11	1	2	0	1
ControladorEntidades.java	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	2	2	2	2	1	1
ProdutoUnidade.java	<div><div></div></div>	93%	<div><div></div></div>	75%	1	7	1	17	0	5	0	1
ProdutoPeso.java	<div><div></div></div>	91%	<div><div></div></div>	50%	1	5	1	14	0	4	0	1
ClienteReaderDb.java	<div><div></div></div>	100%	<div><div></div></div>	100%	0	4	0	7	0	3	0	1
Total	179 of 814	78%	28 of 76	63%	39	106	58	241	13	68	2	14

## Main Branch Status

Quality Gate ?  
Passed



Enjoy your sparkling clean code!

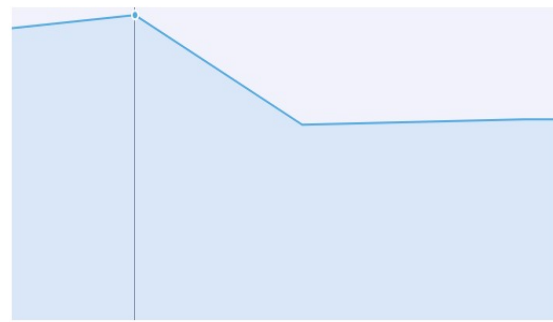
[See Full Analysis](#)

## Main Branch Evolution since 27 days ago

77 Issues ↘ -35

Issues Coverage Duplications

117 Issues



Issues New Code

[See full history](#)

## versioning

succeeded 3 weeks ago in 3s

- > Set up job
- > Run actions/checkout@v4
- > Configuração do Git
- > Tag release
  - 1 ▶ Run GENERATE\_VERSION=true
  - 101 Generate version: true
  - 102 Last tag: #1.1.5#
  - 103 New version: 1.1.6
  - 104 Generating new version...
  - 105 To <https://github.com/facs-qualidade-software/projeto-A3>
  - 106 \* [new tag] 1.1.6 -> 1.1.6
- > Post Run actions/checkout@v4
- > Complete job