

Universidade Federal de Sergipe  
Engenharia de Software II

**Implementação de Integração Contínua com GitHub Actions**  
Atividade 3 – Etapa 2 e 3

**Membros:**

José Fernando Bispo dos Santos – 202200014210  
Valter Fabricio dos Santos – 202000066991

**Professor:** Dr. Glauco de Figueiredo Carneiro  
**Fevereiro de 2026**

## SUMÁRIO

<b>1. Implementação Prática (Hands-on).....</b>	<b>3</b>
<b>1.1 Estratégia Adotada.....</b>	<b>3</b>
<b>1.2 Descrição do Workflow Implementado.....</b>	<b>3</b>
<b>1.3 Trecho do Workflow (.yml) .....</b>	<b>4</b>
<b>1.4 Diagrama do Pipeline Implementado.....</b>	<b>5</b>
<b>1.5 Critério de Sucesso.....</b>	<b>5</b>
<b>2. Justificativa do Não Uso das Outras Estratégias.....</b>	<b>6</b>
<b>2.1 Estratégia B – Quality Gate (Análise Estática).....</b>	<b>6</b>
<b>2.2 Estratégia C – Pipeline de Contingência.....</b>	<b>6</b>
<b>3. Etapa 3 – Análise de Impacto na Evolução do Software.....</b>	<b>7</b>
<b>4. Conclusão.....</b>	<b>7</b>

**Projeto:** *langextract*

**Repositório base:** <https://github.com/google/langextract>

**Fork:** <https://github.com/Fernand0-jf/langextract>

---

## 1. Implementação Prática (Hands-on)

### 1.1 Estratégia Adotada

Para a implementação do pipeline de Integração Contínua (CI), a equipe optou pela **Estratégia A – Pipeline de Construção (Build & Test)**.

Essa escolha foi motivada pelo fato de o projeto *langextract* ser desenvolvido em Python e possuir dependências que podem ser instaladas e executadas normalmente no ambiente gratuito do GitHub Actions, sem exigir recursos especiais de hardware ou tempos de execução elevados.

Dessa forma, foi possível configurar um workflow que realiza a instalação do projeto, a verificação de sintaxe (compilação) e a execução de testes básicos, garantindo a validação funcional mínima do código a cada alteração.

---

### 1.2 Descrição do Workflow Implementado

O workflow foi configurado utilizando **GitHub Actions** e é acionado automaticamente nos seguintes eventos:

- `push` para a branch `main`
- `pull_request` direcionado para a branch `main`

O pipeline executa as seguintes etapas:

#### 1. Checkout do repositório

O código-fonte do projeto é obtido a partir do fork do repositório original.

#### 2. Configuração do ambiente Python

O ambiente de execução é configurado com a versão **Python 3.10**, garantindo compatibilidade com o projeto.

### **3. Atualização do gerenciador de pacotes (pip)**

O `pip` é atualizado para evitar problemas de compatibilidade durante a instalação das dependências.

### **4. Instalação do projeto (Build)**

O projeto é instalado em modo editável (`pip install -e .`), simulando o processo de build típico de projetos Python e validando a configuração do pacote.

### **5. Verificação de sintaxe (Compilação)**

É executado o comando `python -m compileall langextract`, que percorre todos os arquivos do módulo principal e verifica se o código pode ser compilado para bytecode, detectando erros de sintaxe ou indentação antes da execução.

### **6. Execução de teste básico (Smoke Test)**

Por fim, é realizado um teste simples que importa o módulo principal do projeto (`langextract`). Esse teste valida se o pacote foi corretamente instalado e se suas dependências principais estão resolvidas. Esse tipo de verificação é conhecido como *smoke test*, garantindo que o sistema está funcional em um ambiente limpo.

---

## **1.3 Trecho do Workflow (.yml) -**

<https://github.com/Fernand0-jf/langextract/blob/main/.github/workflows/etapa2-build-test.yml>

name: Etapa 2 - Build and Test

on:

push:

  branches: [ "main" ]

pull\_request:

  branches: [ "main" ]

jobs:

build-test:

  name: Build & Smoke Test

  runs-on: ubuntu-latest

steps:

- name: Checkout repository  
uses: actions/checkout@v4
  - name: Set up Python  
uses: actions/setup-python@v5  
with:  
  python-version: '3.10'
  - name: Upgrade pip  
run: python -m pip install --upgrade pip
  - name: Install project (editable)  
run: |  
  pip install -e .
  - name: Compile code (syntax check)  
run: |  
  python -m compileall langextract
  - name: Smoke test - import langextract  
run: |  
  python - <<EOF  
  import langextract  
  print("LangExtract import OK")  
  EOF
- 

## 1.4 Diagrama do Pipeline Implementado

Push / Pull Request (main)



Checkout do Repositório

↓  
Setup do Python 3.10  
↓  
Upgrade do pip  
↓  
Instalação do Projeto (Build)  
↓  
Compilação / Verificação de Sintaxe (compileall)  
↓  
Smoke Test (import do módulo principal)  
↓  
Pipeline finalizado com sucesso

---

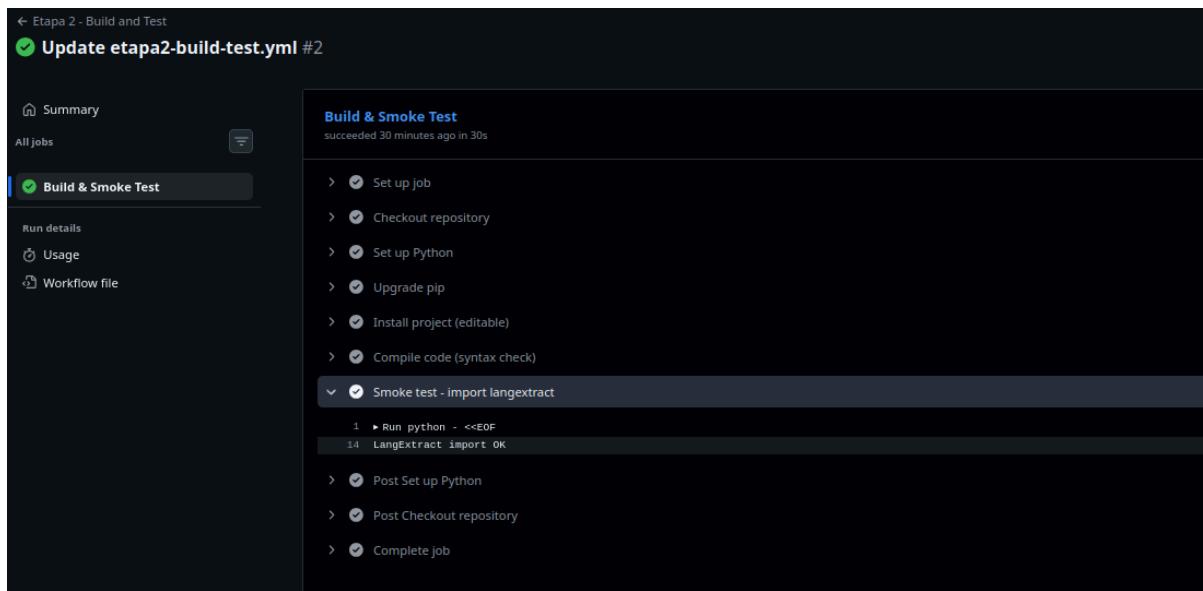
## 1.5 Critério de Sucesso

Actions

<https://github.com/Fernand0-jf/langextract/actions/runs/21712596731/job/62619814748>

O pipeline foi executado com sucesso no fork do repositório da equipe, apresentando o **sinal verde (Success)** na aba **Actions** do GitHub, atendendo integralmente ao critério de sucesso definido para a Etapa 2.





---

## 2. Justificativa do Não Uso das Outras Estratégias

### 2.1 Estratégia B – Quality Gate (Análise Estática)

A Estratégia B é indicada para projetos muito grandes ou legados, nos quais a execução do build é inviável dentro das limitações do ambiente de CI. No entanto, o projeto *langextract* permite a instalação, compilação e execução de testes básicos no GitHub Actions sem restrições técnicas.

Dessa forma, limitar o pipeline apenas a análises estáticas resultaria em uma validação menos completa do software.

---

### 2.2 Estratégia C – Pipeline de Contingência

A Estratégia C é destinada a cenários em que o projeto possui dependências impossíveis de serem satisfeitas no ambiente de Integração Contínua, exigindo a criação de scripts artificiais apenas para demonstrar o funcionamento do pipeline. Como o projeto *langextract* pôde ser validado diretamente no GitHub Actions, não houve necessidade de adotar essa abordagem.

---

### **3. Etapa 3 – Análise de Impacto na Evolução do Software**

A automação implementada por meio do pipeline de Integração Contínua contribui diretamente para a evolução saudável do software. A validação automática do build, da compilação e da execução mínima do projeto permite identificar erros de integração de forma precoce, reduzindo o risco de degradação gradual do código (*software decay*).

Além disso, o pipeline facilita o onboarding de novos desenvolvedores, pois estabelece um processo padronizado e automatizado de verificação. Assim, qualquer novo integrante da equipe pode confiar que, se o pipeline estiver verde, o projeto encontra-se em um estado funcional e consistente.

---

### **4. Conclusão**

A implementação do pipeline de Build & Test utilizando GitHub Actions atendeu plenamente aos objetivos propostos, demonstrando na prática a aplicação de Integração Contínua e evidenciando seus benefícios para a qualidade, manutenção e evolução do software.

---