Quality stages of Python development

Priscila Oliveira

$ whoami

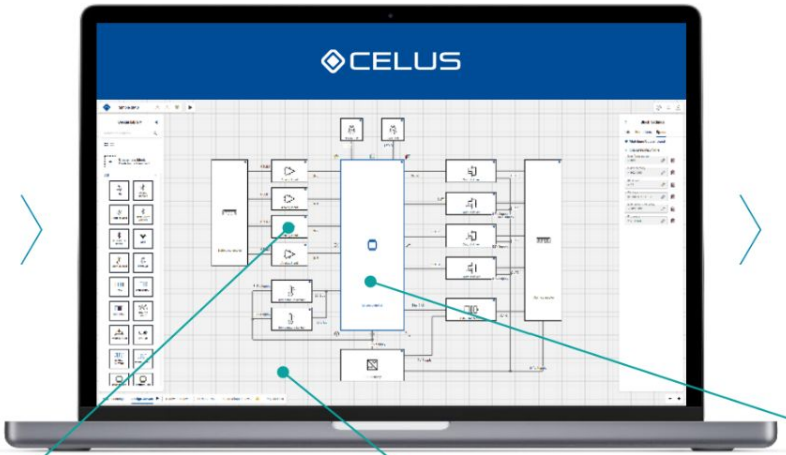
```python
from typing import Literal


class Me:
    occupation: str = 'Computer Scientist'
    expertise_area: str = 'Web Development'
    python_experience: int = 6
    experience_unit: Literal['years', 'months'] = 'years'

    current_role: str = 'Tech Lead'
    company: str = 'CELUS'
```

Hardware Design

in Minutes!

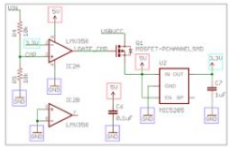
Input

- Functional requirements definition
- Electrical, environmental & mechanical constraints



Output

- ECAD native schematics
- Bill of materials
- PCB floorplan proposal
- Early feasibility analysis
- Early cost estimation
- Geometric fit
- Instant estimation of development effort



Functional blocks with technical parameters to find the best matching module

Very large library of components, circuits and reference designs

CELUS Engineering Platform selects the right components through smart algorithms and the use of AI for the entire design

Code Quality

Code Quality

- Clean code
- Performance
- Review
- **Static analysis**
- **Tests**

Static Analysis

Analyze the source code without executing it.

Checks for:

- **Quality**
- Security
- Correctness
- Performance

Linter

Helps identify

- Syntax errors
  - Leading to runtime errors
  - E.g. Missing parenthesis
- Potential issues
  - E.g. Too complex code
- Dangerous logic
  - Redefining built-in code can lead to unintended side-effects
  - E.g. redefining `id` (built-in var)

Linter

- Code smell
  - Poorly designed code, even if it is technically correct
  - E.g Duplicated code, long methods
- Code style violations
  - E.g. Naming conventions, formatting
- Common security vulnerabilities
  - Known injections, vulnerabilities
  - E.g. Passwords/credentials in the code

Linter - Code Standards

- Readability
- Maintainability
- Collaboration/Code Review
  - Common set of expectations
- PEP 8
  - Style guide for Python
- Example: different IDEs

Static Type checking

- Dynamically typed language
  - Type is identified in runtime
  - Type can change during execution
- Typings, introduced in Python 3.5, supports type hints
  - Does not enforce types

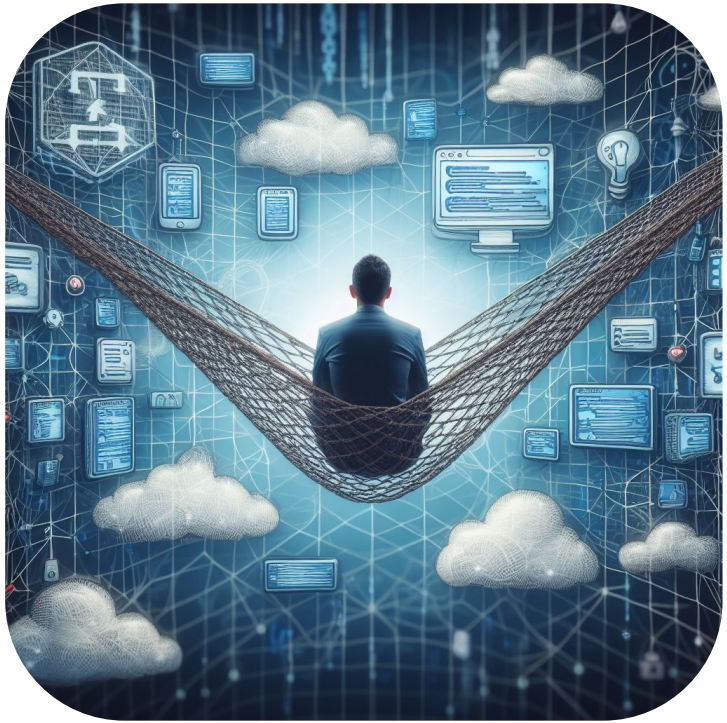
Linter tools

- Logical
  - pylint
  - flake8
  - ruff
- Code standard
  - pycodestyle: checks PEP 8
  - pydocstyle: checks compliance with docstring conventions (PEP 257)
  - black: code formatter
  - isort: organize imports
- Security
  - bandit
- Type checking
  - mypy

Tests

- Helps identifying bugs early
- Regression checking
- Supports refactoring
- Documentation
  - Provides examples of usage
- Can reduce maintenance time

- *Tools*: pytest, selenium



Code example

Check the in the repository

Q&A

Thank you!