Exemplos Práticos para Análise de Code Smells - Aula 12/09/2025

📋 Preparação do Ambiente

Instalação das Ferramentas

```
# Instalar as ferramentas principais
pip install pylint flake8 radon vulture

# Verificar instalação
pylint --version
flake8 --version
radon --version
vulture --version

# Clonar o repositório do curso
git clone https://github.com/fmarquesfilho/bpp-2025-2
cd bpp-2025-2
```

🎯 Exercício 1: Análise com pylint

Arquivo: `problema1.py`

```
# problema1.py - Exemplo com múltiplos code smells
import os
import json
import requests
from datetime import datetime

class UserManager:
    def __init__(self):
        self.users = []
        self.temp_data = {}

    def process_user_registration(self, name, email, password, age,
address, phone, country, city, zipcode, company, department, salary,
manager_email):
        # Validação (método muito longo - Long Method)
        if name == None or name == "" or len(name) < 2:
            print("Nome inválido")
            return False
        if email == None or email == "" or "@" not in email or "." not in
email:
            print("Email inválido")
            return False
        if password == None or password == "" or len(password) < 8:
```

```
            print("Senha muito fraca")
            return False
        if age == None or age < 18 or age > 120:
            print("Idade inválida")
            return False
        if address == None or address == "":
            print("Endereço obrigatório")
            return False
        if phone == None or phone == "" or len(phone) < 10:
            print("Telefone inválido")
            return False

        # Formatação
        formatted_name = name.strip().title()
        formatted_email = email.lower().strip()
        formatted_phone = phone.replace("-", "").replace(" ",
"").replace("(", "").replace(")", "")

        # Cálculo de score (lógica complexa - High Complexity)
        score = 0
        if age > 25:
            score += 10
        if age > 35:
            score += 15
        if age > 45:
            score += 20
        if len(password) > 12:
            score += 5
        if any(c.isupper() for c in password):
            score += 5
        if any(c.islower() for c in password):
            score += 5
        if any(c.isdigit() for c in password):
            score += 5
        if any(c in "!@#$%^&*" for c in password):
            score += 10
        if country.lower() == "brazil":
            score += 15
        if salary > 5000:
            score += 20
        if salary > 10000:
            score += 30

        # Criação do objeto usuário
        user_data = {
            'id': len(self.users) + 1,
            'name': formatted_name,
            'email': formatted_email,
            'password': self.hash_password(password),
            'age': age,
            'address': address,
            'phone': formatted_phone,
            'country': country,
            'city': city,
```

```
            'zipcode': zipcode,
            'company': company,
            'department': department,
            'salary': salary,
            'manager_email': manager_email,
            'score': score,
            'created_at': datetime.now().isoformat(),
            'status': 'active',
            'verified': False
        }

        # Persistência
        self.users.append(user_data)
        self.save_to_file(user_data)
        self.send_welcome_email(formatted_email, formatted_name)
        self.send_manager_notification(manager_email, formatted_name)
        self.log_registration(user_data)
        self.update_statistics()

        return True

    def hash_password(self, password):
        # Implementação simplificada (não use em produção!)
        return f"hash_{password}_{len(password)}"

    def save_to_file(self, user_data):
        # Código duplicado - Duplicate Code
        filename = "users.json"
        if os.path.exists(filename):
            with open(filename, 'r') as f:
                existing_users = json.load(f)
        else:
            existing_users = []
        existing_users.append(user_data)
        with open(filename, 'w') as f:
            json.dump(existing_users, f, indent=2)

    def send_welcome_email(self, email, name):
        # Simulação de envio de email
        print(f"Enviando email de boas-vindas para {email}")

    def send_manager_notification(self, manager_email, name):
        # Simulação de notificação
        print(f"Notificando manager {manager_email} sobre novo usuário
{name}")

    def log_registration(self, user_data):
        # Log simples
        print(f"LOG: Usuário {user_data['name']} registrado em
{user_data['created_at']}")

    def update_statistics(self):
        # Atualização de estatísticas
        unused_var = "esta variável não é usada"  # Dead Code
```

```
        print(f"Total de usuários: {len(self.users)}")

    def generate_user_report(self, user_id):
        # Código duplicado - mesmo padrão de carregamento de arquivo
        filename = "users.json"
        if os.path.exists(filename):
            with open(filename, 'r') as f:
                existing_users = json.load(f)
        else:
            existing_users = []

        user = None
        for u in existing_users:
            if u['id'] == user_id:
                user = u
                break

        if not user:
            return None

        # Geração do relatório
        report = f"""
        === RELATÓRIO DO USUÁRIO ===
        Nome: {user['name']}
        Email: {user['email']}
        Idade: {user['age']}
        Score: {user['score']}
        Status: {user['status']}
        Criado em: {user['created_at']}
        """

        # Salvar relatório
        report_filename = f"report_{user_id}.txt"
        with open(report_filename, 'w') as f:
            f.write(report)

        # Enviar por email
        print(f"Relatório salvo em {report_filename}")
        return report

# Classe com poucos métodos - Too Few Public Methods
class EmailValidator:
    def is_valid(self, email):
        return "@" in email and "." in email

# Função com nome inadequado - Poor Naming
def doStuff(data):
    result = []
    for item in data:
        if item > 0:
            result.append(item * 2)
    return result

# Variável global não utilizada - Dead Code
```

```
UNUSED_CONSTANT = "Esta constante nunca é usada"

if __name__ == "__main__":
    manager = UserManager()

    # Teste com muitos parâmetros - Long Parameter List
    success = manager.process_user_registration(
        "João Silva",
        "joao@email.com",
        "senhaSegura123!",
        30,
        "Rua das Flores, 123",
        "(84) 99999-9999",
        "Brazil",
        "Natal",
        "59000-000",
        "Tech Corp",
        "Desenvolvimento",
        8000,
        "manager@techcorp.com"
    )

    print(f"Registro {'bem-sucedido' if success else 'falhou'}")
```

Tarefas do Exercício 1:

1. **Execute pylint no arquivo:**

   ```
   pylint problema1.py
   ```

2. **Analise os resultados e identifique:**
   - Quantos warnings/errors foram encontrados?
   - Qual a pontuação final do código?
   - Quais os principais code smells detectados?

3. **Categorize os problemas encontrados:**
   - Long Method (process_user_registration)
   - Duplicate Code (save_to_file vs generate_user_report)
   - Long Parameter List (13 parâmetros!)
   - Dead Code (variáveis não utilizadas)
   - Poor Naming (função doStuff)
   - Too Few Public Methods (EmailValidator)

🎯 Exercício 2: Análise de Complexidade com Radon

Comando:

```
radon cc problema1.py -s
```

Tarefas do Exercício 2:

1. **Execute radon e analise:**
    - Qual a complexidade ciclomática do método `process_user_registration`?
    - Quais funções/métodos têm maior complexidade?
    - O que isso indica sobre a manutenibilidade?
2. **Compare com métricas recomendadas:**
    - A (1-5): Baixa complexidade
    - B (6-10): Moderada
    - C (11-20): Alta
    - D (21-50): Muito alta
    - F (50+): Extremamente alta

🎯 Exercício 3: Detecção de Código Morto com Vulture

Arquivo: `problema2.py`

```python
# problema2.py - Exemplo com Dead Code
import sys
import os
from datetime import datetime, timedelta

# Constante nunca usada
UNUSED_CONFIG = {
    'timeout': 30,
    'retries': 3,
    'debug': True
}

class DataProcessor:
    def __init__(self):
        self.data = []
        self.backup_data = []  # nunca usado
        self.temp_storage = {}

    def process_data(self, items):
        """Processa lista de itens"""
        result = []
        for item in items:
            processed = self._transform_item(item)
            result.append(processed)
        return result

    def _transform_item(self, item):
        """Transforma um item individual"""
```

```
            return item.upper() if isinstance(item, str) else str(item)

        def _backup_data(self, data):
            """Método nunca chamado"""
            self.backup_data = data.copy()
            print("Backup realizado")

        def _legacy_method(self):
            """Método antigo que não é mais usado"""
            old_format = "formato antigo"
            return old_format

        def get_stats(self):
            """Retorna estatísticas"""
            unused_var = "esta variável não é usada"
            return {
                'total': len(self.data),
                'processed_at': datetime.now()
            }

    def old_utility_function(x, y):
        """Função que não é mais chamada"""
        return x + y * 2

    def another_unused_function():
        """Outra função nunca usada"""
        temp = []
        for i in range(10):
            temp.append(i * 2)
        return temp

    if __name__ == "__main__":
        processor = DataProcessor()
        data = ["hello", "world", 123, "python"]
        result = processor.process_data(data)
        print(f"Resultado: {result}")
        print(f"Stats: {processor.get_stats()}")
```

Tarefas do Exercício 3:

1. **Execute vulture:**

   ```
   vulture problema2.py
   ```

2. **Identifique o código morto:**

   - Quais funções nunca são chamadas?
   - Quais variáveis nunca são usadas?
   - Quais imports são desnecessários?

3. **Calcule o desperdício:**
    - Quantas linhas de código podem ser removidas?
    - Qual o percentual de código não utilizado?

🎯 Exercício 4: Análise de Estilo com flake8

Comando:

```
flake8 problema1.py problema2.py --max-complexity=10 --statistics
```

Tarefas do Exercício 4:

1. **Analise violações de estilo:**
    - Quantas violações PEP 8 foram encontradas?
    - Quais os tipos mais comuns de violação?
2. **Foque na complexidade:**
    - Quais funções excedem a complexidade máxima (10)?
    - Como isso se relaciona com os code smells?

📊 Exercício 5: Análise Comparativa

Arquivo Refatorado: `solucao1.py`

```python
# solucao1.py - Versão refatorada
from abc import ABC, abstractmethod
from dataclasses import dataclass
from datetime import datetime
from typing import List, Optional
import json
import os

@dataclass
class UserData:
    name: str
    email: str
    age: int
    address: str
    phone: str
    country: str
    city: str
    zipcode: str
    company: str
    department: str
    salary: float
    manager_email: str
```

```
@dataclass
class User:
    id: int
    name: str
    email: str
    password_hash: str
    age: int
    score: int
    created_at: str
    status: str = 'active'
    verified: bool = False

class UserValidator:
    """Responsável pela validação de dados de usuário"""

    def validate(self, user_data: UserData) -> bool:
        """Valida todos os dados do usuário"""
        return (
            self._validate_name(user_data.name) and
            self._validate_email(user_data.email) and
            self._validate_age(user_data.age) and
            self._validate_phone(user_data.phone)
        )

    def _validate_name(self, name: str) -> bool:
        return name and len(name.strip()) >= 2

    def _validate_email(self, email: str) -> bool:
        return email and "@" in email and "." in email

    def _validate_age(self, age: int) -> bool:
        return 18 <= age <= 120

    def _validate_phone(self, phone: str) -> bool:
        return phone and len(phone.replace("-", "").replace(" ", "")) >=
10

class ScoreCalculator:
    """Calcula score do usuário baseado em critérios"""

    def calculate(self, user_data: UserData, password: str) -> int:
        score = 0
        score += self._age_score(user_data.age)
        score += self._password_score(password)
        score += self._location_score(user_data.country)
        score += self._salary_score(user_data.salary)
        return score

    def _age_score(self, age: int) -> int:
        if age > 45:
            return 45
        elif age > 35:
            return 25
        elif age > 25:
```

```
            return 10
        return 0

    def _password_score(self, password: str) -> int:
        score = 0
        if len(password) > 12:
            score += 5
        if any(c.isupper() for c in password):
            score += 5
        if any(c.islower() for c in password):
            score += 5
        if any(c.isdigit() for c in password):
            score += 5
        if any(c in "!@#$%^&*" for c in password):
            score += 10
        return score

    def _location_score(self, country: str) -> int:
        return 15 if country.lower() == "brazil" else 0

    def _salary_score(self, salary: float) -> int:
        if salary > 10000:
            return 50
        elif salary > 5000:
            return 20
        return 0

class UserRepository:
    """Gerencia persistência de usuários"""

    def __init__(self, filename: str = "users.json"):
        self.filename = filename

    def save(self, user: User) -> None:
        users = self._load_users()
        users.append(user.__dict__)
        self._save_users(users)

    def find_by_id(self, user_id: int) -> Optional[User]:
        users = self._load_users()
        for user_data in users:
            if user_data['id'] == user_id:
                return User(**user_data)
        return None

    def _load_users(self) -> List[dict]:
        if not os.path.exists(self.filename):
            return []
        with open(self.filename, 'r') as f:
            return json.load(f)

    def _save_users(self, users: List[dict]) -> None:
        with open(self.filename, 'w') as f:
            json.dump(users, f, indent=2)
```

```python
class NotificationService:
    """Serviço de notificações por email"""

    def send_welcome_email(self, email: str, name: str) -> None:
        print(f"Enviando email de boas-vindas para {email}")

    def send_manager_notification(self, manager_email: str, name: str) ->
None:
        print(f"Notificando manager {manager_email} sobre novo usuário
{name}")

class UserManager:
    """Orquestra o processo de registro de usuários"""

    def __init__(self):
        self.validator = UserValidator()
        self.score_calculator = ScoreCalculator()
        self.repository = UserRepository()
        self.notification_service = NotificationService()
        self._user_counter = 0

    def register_user(self, user_data: UserData, password: str) -> bool:
        """Registra um novo usuário no sistema"""
        if not self.validator.validate(user_data):
            return False

        score = self.score_calculator.calculate(user_data, password)

        user = User(
            id=self._get_next_id(),
            name=user_data.name.strip().title(),
            email=user_data.email.lower().strip(),
            password_hash=self._hash_password(password),
            age=user_data.age,
            score=score,
            created_at=datetime.now().isoformat()
        )

        self.repository.save(user)
        self.notification_service.send_welcome_email(user.email,
user.name)
        self.notification_service.send_manager_notification(
            user_data.manager_email,
            user.name
        )

        return True

    def _get_next_id(self) -> int:
        self._user_counter += 1
        return self._user_counter

    def _hash_password(self, password: str) -> str:
```

```
        return f"hash_{password}_{len(password)}"

if __name__ == "__main__":
    user_data = UserData(
        name="João Silva",
        email="joao@email.com",
        age=30,
        address="Rua das Flores, 123",
        phone="(84) 99999-9999",
        country="Brazil",
        city="Natal",
        zipcode="59000-000",
        company="Tech Corp",
        department="Desenvolvimento",
        salary=8000,
        manager_email="manager@techcorp.com"
    )

    manager = UserManager()
    success = manager.register_user(user_data, "senhaSegura123!")
    print(f"Registro {'bem-sucedido' if success else 'falhou'}")
```

Tarefas do Exercício 5:

1. **Compare as métricas:**

```
# Análise do código problemático
pylint problema1.py --reports=y
radon cc problema1.py -s

# Análise do código refatorado
pylint solucao1.py --reports=y
radon cc solucao1.py -s
```

2. **Documente as melhorias:**
    - Redução na complexidade ciclomática
    - Aumento da pontuação pylint
    - Eliminação de code smells
    - Aplicação dos princípios SOLID

🚀 Desafio Extra

Para quem terminar cedo, analise um projeto real:

```
# Clone um projeto Python popular
git clone https://github.com/psf/requests
cd requests
```

```
# Execute análise completa
pylint requests/ --reports=y > analise_requests.txt
radon cc requests/ -a > complexidade_requests.txt
vulture requests/ > deadcode_requests.txt

# Compare com suas próprias métricas!
```

📚 Recursos Adicionais

- **Catálogo de Code Smells**: https://luzkan.github.io/smells/
- **Documentação pylint**: https://pylint.pycqa.org/
- **Guia radon**: https://radon.readthedocs.io/
- **PEP 8 Style Guide**: https://pep8.org/