Atividade 1 - Teste de Software

Jorge Lucas Mota Sousa

1. Problema escolhido **3**
1.2 Oque foi reportado 3
1.3 Resposta escolhida pela comunidade 3
1.4 Outras respostas 4
2 GitHub **5**
3 Testes **6**
3.1 - Unittest discover 6

1. Problema escolhido

Os testes unitários são fundamentais no desenvolvimento de software, pois permitem que os desenvolvedores verifiquem a funcionalidade de partes específicas do código. Devido a essa grande importância, a pergunta cadastrada no Stack Overflow "How do I run all Python unit tests in a directory?" chamou minha atenção e foi a escolhida para meu trabalho. Atualmente, essa questão conta com mais de 400 votos e muitas interações, refletindo seu valor e relevância para a comunidade de desenvolvimento.

1.2 Oque foi reportado

O desenvolvedor enfrentou dificuldades ao tentar implementar uma solução para executar todos os testes unitários em um diretório específico de forma automatizada através de um arquivo centralizado chamado `all_test.py`. O objetivo era criar um mecanismo onde todos os arquivos que seguem o padrão `test_*.py` pudessem ser executados simultaneamente , fornecendo uma saída única dos resultados.

Na primeira abordagem, o desenvolvedor tentou importar todos os módulos de teste utilizando padrões de nome de arquivo e, em seguida, chamar `unittest.main()` para executar os testes. Essa estratégia, no entanto, resultou em uma execução que não identificou nenhum teste, como evidenciado pela saída que indicava "Ran 0 tests". Isso ocorreu porque `unittest.main()` é projetado para lidar com a execução de testes em um contexto onde o script é chamado diretamente como o ponto de entrada principal, o que não é o caso aqui.

Já na segunda tentativa, o desenvolvedor adotou uma abordagem mais manual. Utilizou-se da biblioteca `glob` para localizar os arquivos e depois carregou cada módulo de teste. Esses testes foram adicionados a uma instância de `unittest.TestSuite`, e o conjunto de testes foi executado usando `unittest.TestResult`. Embora essa abordagem tenha conseguido executar os testes, a formatação e apresentação dos resultados não estavam alinhadas com as saídas padrão de `unittest`, e havia incertezas sobre a correta contagem e relato dos testes executados.

1.3 Resposta escolhida pela comunidade

A resposta escolhida pela comunidade do Stack Overflow mostra como executar todos os testes unitários em um diretório usando recursos integrados do Python a partir da versão 2.7. A solução consiste em utilizar o módulo `unittest` com o comando `discover`. Os comandos fornecidos são:

1. `python -m unittest discover <diretorio_teste>`: Executar testes recursivamente no diretório especificado.
2. `python -m unittest discover -s <diretorio> -p 'teste.py'`: Permite especificar o diretório e o padrão de nome de arquivo para os testes a serem descobertos.

A resposta também aborda problemas comuns e verificações, como a necessidade de todos os diretórios de teste terem um arquivo `__init__.py` para que sejam reconhecidos pelo comando `discover`. Comentários adicionais na discussão esclarecem outras dúvidas específicas e confirmam a funcionalidade dos comandos em diferentes ambientes e versões do Python.

Essa alternativa possui maior flexibilidade nos testes fazendo sua popularidade na comunidade, pois se adaptam melhor às variadas necessidades de projetos de software. A preferência por soluções mais adaptáveis reflete uma tendência da comunidade em valorizar ferramentas que oferecem menos restrições e mais personalização. Além disso, oferece uma solução direta e incorporada para executar todos os testes unitários em um diretório, utilizando apenas recursos padrões do Python. A explicação é clara e sucinta, e fornece opções de comando para diferentes necessidades, como a execução recursiva de testes.

1.4 Outras respostas

Proposta 1:

111

You could use a test runner that would do this for you. nose is very good for example. When run, it will find tests in the current tree and run them.

Updated:

Here's some code from my pre-nose days. You probably don't want the explicit list of module names, but maybe the rest will be useful to you.

```
testmodules = [
    'cogapp.test_makefiles',
    'cogapp.test_whiteutils',
    'cogapp.test_cogapp',
    ]

suite = unittest.TestSuite()

for t in testmodules:
    try:
        # If the module defines a suite() function, call it to get the suite.
        mod = __import__(t, globals(), locals(), ['suite'])
        suitefn = getattr(mod, 'suite')
        suite.addTest(suitefn())
    except (ImportError, AttributeError):
        # else, just load all the test cases from the module.
        suite.addTest(unittest.defaultTestLoader.loadTestsFromName(t))

unittest.TextTestRunner().run(suite)
```

Este código utiliza dinâmica para carregar módulos de teste, o que pode introduzir complexidade desnecessária e riscos de erro. A necessidade de manipular exceções como ImportError e AttributeError para lidar com falhas de importação ou a ausência de uma função específica aumenta a complexidade do script e pode ocultar problemas que seriam mais facilmente diagnosticados em uma abordagem mais declarativa.

Proposta 2:

This is now possible directly from unittest: unittest.TestLoader.discover.

```
import unittest
loader = unittest.TestLoader()
start_dir = 'path/to/your/test/files'
suite = loader.discover(start_dir)

runner = unittest.TextTestRunner()
runner.run(suite)
```

Neste script, há uma dependência explícita da estrutura de diretórios. Isso limita a flexibilidade e pode complicar a configuração em ambientes de desenvolvimento diversificados, onde os caminhos podem variar ou ser configurados dinamicamente.

Proposta 3:

Well by studying the code above a bit (specifically using `TextTestRunner` and `defaultTestLoader`), I was able to get pretty close. Eventually I fixed my code by also just passing all test suites to a single suites constructor, rather than adding them "manually", which fixed my other problems. So here is my solution.

```
import glob
import unittest

test_files = glob.glob('test_*.py')
module_strings = [test_file[0:len(test_file)-3] for test_file in test_files]
suites = [unittest.defaultTestLoader.loadTestsFromName(test_file) for test_file in module_
test_suite = unittest.TestSuite(suites)
test_runner = unittest.TextTestRunner().run(test_suite)
```

A utilização do padrão de nomeação test_*.py e da biblioteca glob para identificar arquivos de teste pode ser restritiva e propensa a falhas, especialmente em sistemas de arquivos complexos. Este método pode não capturar todos os testes desejados se os nomes dos arquivos não seguirem o padrão esperado, limitando a abrangência dos testes executados.

Proposta 4:

If you want to run all the tests from various test case classes and you're happy to specify them explicitly then you can do it like this:

```
from unittest import TestLoader, TextTestRunner, TestSuite
from uclid.test.test_symbols import TestSymbols
from uclid.test.test_patterns import TestPatterns

if __name__ == "__main__":

    loader = TestLoader()
    tests = [
        loader.loadTestsFromTestCase(test)
        for test in (TestSymbols, TestPatterns)
    ]
    suite = TestSuite(tests)

    runner = TextTestRunner(verbosity=2)
    runner.run(suite)
```

Este código requer que cada classe de teste seja explicitamente carregada, aumentando a verbosidade e a necessidade de manutenção manual do script à medida que novos testes são adicionados. Este tipo de configuração manual pode ser propenso a erros e dificulta a escalabilidade do processo de teste em projetos que evoluem rapidamente.

Proposta 5:

I have used the `discover` method and an overloading of `load_tests` to achieve this result in a (minimal, I think) number lines of code:

```
def load_tests(loader, tests, pattern):
''' Discover and load all unit tests in all files named ``*_test.py`` in ``./src/``
'''
    suite = TestSuite()
    for all_test_suite in unittest.defaultTestLoader.discover('src', pattern='*_tests.py')
        for test_suite in all_test_suite:
            suite.addTests(test_suite)
    return suite

if __name__ == '__main__':
    unittest.main()
```

Execution on fives something like

```
Ran 27 tests in 0.187s
OK
```

Embora utilize um método de descoberta automática para encontrar testes, este código ainda envolve uma complexidade adicional ao iterar sobre todos os conjuntos de testes descobertos. A exigência de um padrão de nomenclatura específico para os arquivos (*_tests.py) também pode ser um limitador, não oferecendo flexibilidade suficiente para diferentes convenções de nomenclatura adotadas por diferentes equipes ou projetos.

2 GitHub

Todos os códigos utilizados neste trabalho estão disponíveis no repositório Teste_Software_2024_Sousa_Jorge, que está organizado nas seguintes pastas:

- Codigos_nao_aceitos: Esta pasta contém os códigos que não foram aceitos pela comunidade.
- Testes_erro: Esta pasta foi criada para analisar o comportamento do unittest quando há um erro lógico no código.
- Testes_stackoverflow: Esta pasta inclui soluções que foram aceitas pela comunidade, após discussões no Stack Overflow.
- Testes_txt: Esta pasta é destinada a testar a funcionalidade de catalogar todos os testes em um arquivo .txt

3 Testes

Todos os códigos abaixo estão disponíveis no Github.

3.1 - Unittest discover

A solução aceita pela comunidade e bem útil e versátil como já foi dito anteriormente, para testar eu criei o seguintes códigos de teste:

```python
import unittest

class TesteExemplo1(unittest.TestCase):
    def teste_adicao(self):
        self.assertEqual(1 + 1, 2)

    def teste_subtracao(self):
        self.assertEqual(2 - 1, 1)

if __name__ == "__main__":
    unittest.main()
```

```python
import unittest

class TesteExemplo2(unittest.TestCase):
    def teste_multiplicacao(self):
        self.assertEqual(2 * 2, 4)

    def teste_divisao(self):
        self.assertEqual(4 / 2, 2)

if __name__ == "__main__":
    unittest.main()
```

Ambos possuem êxito ao testar:

```
PS C:\Users\PC\Desktop\Teste_Software_2024_Sousa_Mota\Testes_stackoverflow> python -m unittest discover .
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

3.2 - Unittest discover ao encontrar erro lógico

Afim de extrair 100% do Unittest eu decidir fazer testes com erro lógico:

```python
import unittest

class TesteExemplo1(unittest.TestCase):
    def teste_adicao(self):
        self.assertEqual(1 + 1, 2)

    def teste_subtracao(self):
        self.assertEqual(2 - 1, 3) #Erro

if __name__ == "__main__":
    unittest.main()
```

```
PS C:\Users\PC\Desktop\Teste_Software_2024_Sousa_Mota\Testes_erro> python -m unittest discover .
.F
======================================================================
FAIL: test_subtraction (test_erro1.TestExample1.test_subtraction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\PC\Desktop\Teste_Software_2024_Sousa_Mota\Testes_erro\test_erro1.py", line 8, in test_subtraction
    self.assertEqual(2 - 1, 3) #Erro
    ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 1 != 3

----------------------------------------------------------------------
Ran 2 tests in 0.002s

FAILED (failures=1)
```

3.3 - Unittest discover com txt

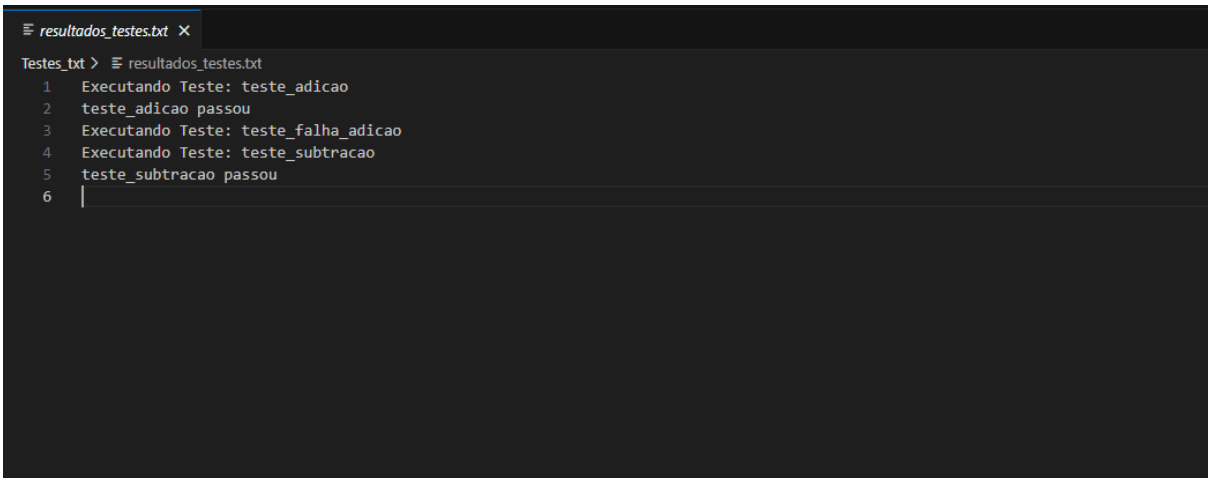
Para fazer o unittest discover gerar um txt é necessário o seguinte comando:

```
python -m unittest discover > resultados_testes.txt
```

```python
import unittest

class TesteExemplo1(unittest.TestCase):
    def teste_adicao(self):
        print("Executando Teste: teste_adicao")
        self.assertEqual(1 + 1, 2)
        print("teste_adicao passou")

    def teste_subtracao(self):
        print("Executando Teste: teste_subtracao")
        self.assertEqual(2 - 1, 1)
        print("teste_subtracao passou")

    def teste_falha_adicao(self):
        print("Executando Teste: teste_falha_adicao")
        self.assertEqual(1 + 1, 3)  # Este teste irá falhar
        print("teste_falha_adicao passou")

if __name__ == "__main__":
    unittest.main()
```

```
PS C:\Users\PC\Desktop\Teste_Software_2024_Sousa_Mota\Testes_txt> python -m unittest discover > resultados_testes.txt
.F.
======================================================================
FAIL: teste_falha_adicao (test_example1.TesteExemplo1.teste_falha_adicao)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\PC\Desktop\Teste_Software_2024_Sousa_Mota\Testes_txt\test_example1.py", line 16, in teste_falha_adicao
    self.assertEqual(1 + 1, 3)  # Este teste irá falhar
    ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 2 != 3

----------------------------------------------------------------------
Ran 3 tests in 0.002s

FAILED (failures=1)
```

Txt gerado:


resultados_testes.txt
Testes_txt > resultados_testes.txt
1 Executando Teste: teste_adicao
2 teste_adicao passou
3 Executando Teste: teste_falha_adicao
4 Executando Teste: teste_subtracao
5 teste_subtracao passou
6