Teste de Software II

Laboratórios de Informática I

Nuno Macedo

MIEI 15/16 — Universidade do Minho

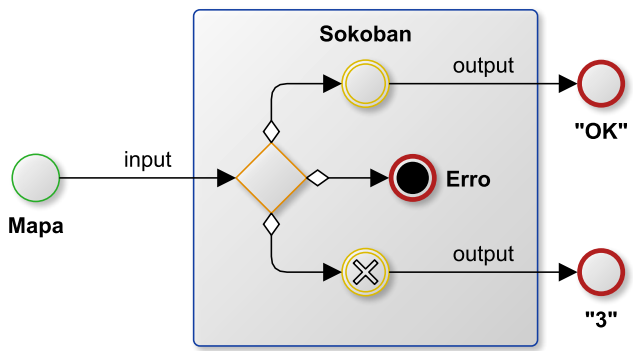
30 de Novembro de 2015

Métodos de Teste de Software

Diferentes níveis e estratégias de teste, por exemplo:

- *Black-box testing* testa sem conhecimento do código.

- *White-box testing* testa com conhecimento total do código.



White-box Testing

Na primeira fase do projeto abordamos o *black-box testing*, o método utilizado pelo Mooshak.

Nesta fase vamos explorar o *white-box testing*, que permite uma análise mais refinada do código através da exploração da estrutura interna do programa.

White-box Testing

Permite definir testes que efetivamente exercitem as estruturas internas do código, proporcionando uma maior **cobertura** do código.

A ideia é testar as estruturas de controlo, o fluxo de dados, cobertura de instruções e decisões, ...

Obriga por isso a **raciocinar** sobre o código desenvolvido.

Por outro lado, torna-se mais **complexo** definir e manter testes *white-box*.

Teste Unitário

Os testes *white-box* podem ser aplicados a diferentes níveis.

O teste unitário tem como objetivo testar as unidades mais simples do programa (e.g., funções individuais).

Facilita a identificação de problemas (em contraste com testar todo o programa como um todo).

Promove a deteção de erros nas fases iniciais do desenvolvimento, evitando também a introdução de erros quando o código evolui.

Acabam por servir também como documentação do comportamento esperado das unidades.

HUnit

Definir testes *white-box* é mais custoso do que testes puramente *black-box*.

Para que o esforço não seja desperdiçado, devem ser re-utilizáveis e a sua execução automatizada.

Várias *frameworks* de teste unitário foram desenvolvidas para este efeito (*xUnit*).

HUnit

HUnit é a *framework* para teste unitário em Haskell (`cabal install hunit`).

Testes são definidos como asserções de propriedades e combinados em *suites* de testes.

Estas são depois executadas automaticamente pelo HUnit, que reporta os resultados ao utilizador.

HUnit: Asserções

Dado um valor booleano, reporta uma falha se falso.

```
assertBool :: String -> Bool -> Assertion
```

Dados dois valores comparáveis, reporta uma falha se diferentes.

```
assertEqual :: (Eq a, Show a) => String -> a -> a ->
    Assertion
```

Exemplo:

```
assertEqual ``soma'' (minha_soma 2 3) 5
```

HUnit: Testes

Essencialmente, compõem testes individuais em *suites*.

Exemplo:

```
tE1 = TestCase $ assertEqual ...
tE2 = TestCase $ assertEqual ...
testesTE = TestLabel "Tarefa E" $ TestList [tE1,tE2]
```

HUnit: Execução

Testes são executados com a função `runTestTT`, que reporta as falha ao utilizador.

Exemplo:

```
*Main> runTestTT testesTE
### Failure:
/Users/user/hunit.hs:8
Teste E1,
expected: 10
but got: 11
Counts {cases = 2, tried = 1, errors = 0, failures = 1}
```

Exemplo

Podemos também usar o HUnit para simular o comportamento black-box do Mooshak, aproveitando os ficheiros de input/output.

```
testesMooshak :: IO Test
testesMooshak = do
 is <- find (depth ==? 0) (extension ==? ".in") "../tests/C"
 let tE = TestLabel "Tarefa C" $ TestList $ map (
    testesTarefa tarefa3) is
 return $ TestLabel "mooshak" $ tE

testesTarefa :: (String -> String) -> String -> Test
...
```

http://lpaste.net/4556833393362337792

Cobertura dos Testes

Quando consideramos testes em *white-box* devemos ter em consideração a cobertura do código desses testes.

Critérios básicos incluem a quantidade das funções definidas foram executadas, de instruções do código, ou as estruturas de controlo de fluxo exercitadas.

Critérios avançados incluem testar todas as combinações possíveis das estruturas de controlo, o que rapidamente se pode tornar impraticável.

HPC

A ferramenta HPC (*Haskell program coverage*) permite testar alguns dos critérios básicos de cobertura.

O código é 'instrumentalizado' ao ser compilado, gerando anotações durante a execução.

Exemplo:

```
$ ghc -fhpc HUnit.hs
$ ./HUnit
...
$ hpc report HUnit
23% expressions used (564/2378)
42% boolean coverage (3/7)
    100% guards (0/0)
     42% 'if' conditions (3/7), 4 unevaluated
    100% qualifiers (0/0)
17% alternatives used (38/219)
40% local declarations used (20/50)
39% top-level declarations used (43/109)
```

Limitações

O teste unitário tem limitações (e.g., erros que emergem da interação de diferentes unidades de código).

De facto, em geral é impossível testar todas as possíveis execuções de um programa.

O teste de *software* deve ser sempre acompanhado de outras técnicas que promovam a qualidade.