MAC0340 - Laboratório de Engenharia de Software - 2014/S1

António Augusto Tavares Martins Miranda (7644342)
amartmiranda@gmail.com

Rodrigo Duarte Louro (7240216)
digao.louro@gmail.com

Exercício 6: Avaliação da Adequação de testes

- **Ferramentas auxiliares para geração ou análise dos testes**

  Para a geração dos testes nós não usamos usamos nenhuma ferramenta, ou seja, criamos todos os nossos testes manualmente. Para executar os testes usamos a ferramenta JUnit.

- **Geração dos test cases para a análise da cobertura do MC/DC**

  Tal como foi dito no tópico anterior, as malhas de testes, usadas no processo de determinação do grau de cobertura do MC/DC, foram elaboradas manualmente, tendo em conta o código fonte alvo e o grau de MC/DC que queríamos cobrir. O diretório Atividade6/src/test/java/malhaDeTestesASerAnalisada possui todas as malhas de teste elaboramos para este protótipo. As malhas de testes que se encontram neste diretório são:

  - ExemploClasseUmTeste: esta classe de teste é usada para exercitar a classe ExemploClasseUm em relação aos requisitos do MC/DC da mesma. Dentro dele elaboramos dois testes com 100% de cobertura para o MC/DC.
  - ExemploClasseDoisTeste: esta classe de teste é usada para exercitar a classe ExemploClasseDois em relação aos requisitos do MC/DC da mesma. Dentro dele elaboramos dois testes com 100% de cobertura para o MC/DC.
  - ExemploClasseTresTeste: esta classe de teste é usada para exercitar a classe ExemploClasseTres em relação aos requisitos do MC/DC da mesma. Dentro dele elaboramos dois testes. Um com 100% de cobertura para o MC/DC e outro que nunca vai cobrir 100% do MC/DC.

- **Análise da Adequação da malha de testes**

  Para fazer a avaliação das malhas de teste, estamos aproveitando o xml que gerado pelo protótipo do exercício 4, que contém os requisitos de teste para o cobrimento do MC/DC, e compará-los aos valores observados nas variáveis das decisões/condições do código fonte, resultantes da execução da malha de testes.

  Primeiramente extraímos os requisitos de teste para o MC/DC do código fonte, presentes no xml gerado pelo programa do exercício 4, para uma hashtable de decisões/condiçõe. Em seguida implantamos diretamente no código fonte (do programa a ser testado), operações de captura e armazenamento dos valores das variáveis essenciais, particulares a cada decisão/condição do programa e geramos (manualmente) as malhas de teste para exercitar o código fonte. Após a execução das malhas de teste é criada uma hashtable com as valorações de cada uma das variáveis essenciais às decisões/condições do programa e que foram exercitadas pela malha de teste. Para computar o grau de cobrimentodo mcdc comparamos as valorações da hashtable com os requisitos para o cobrimento do mcdc com a hashtable dos valores exercitados pela malha de testes.

- **Manual de usuário**

  Antes de executarmos o protótipo temos que ter os seguintes ficheiros:

  - O código fonte que queremos testar;
  - A malha de testes para exercitar o código fonte;
  - o arquivo xml do código fonte com os requisitos para o cobrimento do mcdc.

  Cabe ao usuário criar os dois primeiros ficheiros. Atenção a malha de testes pode ser gerada automaticamente por meio de ferramentas apropriadas. O arquivo xml com do código fonte com os requisitos do mcdc deve ser pelo protótipo desenvolvido na atividade 4. Dado um código fonte, o protótipo da atividade 4 cria 3 arquivos XML

cada um com os requisitos para o cobrimento dos critérios de todas as condições, todas as decisões e mcdc. Para mais informações sobre o protótipo da atividade 4 e sobre como usá-lo recomendamos a leitura do relatório do mesmo. Agora que já temos os xml com os requisitos do mcdc basta colocá-lo no diretório Atividade6/resources. Atenção no diretório Atividade6/resources só pode existir um ficheiro xml com os requisitos do mcdc. Agora o código fonte a ser testado deve ser instrumentado (manualmente) e só depois adicionado ao diretório Atividade6/src/main/java/analisar. Seja ClasseExemplo.java,

```
public class ExemploDeClasseDois {
  public static void metodoExemplo (int a, int b) {
    // Codigo executado pelo metodo
    if (a > 0 && b == 0) a = 0;
  }
}
```

, o código instrumentado seria,

```
public class ExemploDeClasseDois {
  public static void metodoExemplo (int a, int b) {
    // Instrumentacao
    if (a > 0) {
      HashTable.getInstance()
               .setHashExecutados(``ClasseExemplo.'', ``metodoExemplo.'',
                                  ``a>0 && b==0'', ``a>0'', true);
    } else {
      HashTable.getInstance()
               .setHashExecutados(``ClasseExemplo.'', ``metodoExemplo.'',
                                  ``a>0 && b==0'', ``a>0'', false);
    }
    if (b == 0) {
      HashTable.getInstance()
               .setHashExecutados(``ClasseExemplo.'', ``metodoExemplo.'',
                                  ``a>0 && b==0'', ``b==0'', true);
    } else {
      HashTable.getInstance()
               .setHashExecutados(``ClasseExemplo.'', ``metodoExemplo.'',
                                  ``a>0 && b==0'', ``b==0'', false);
    }
    // Codigo executado pelo metodo
    if (a > 0 && b == 0) a = 0;
  }
}
```

, ou seja, dada uma decisão, queremos pegar todas as valorações possíveis de cada condição da decisão.

Todos os ficheiros a serem instrumentados têm que incluir hash.Hastable e pertencer a package analisar, visto que isso é necessário para a criação do hastable dos valores exercitados pela malha de testes.

O método setHashExecutados recebe como parâmetros a string com o nome da classe a ser instrumentada, string com o nome do método onde se encontra a decisão, string da decisão, string da condição e por fim o valor assumido pela condição. Atenção que as strings que representam classe e o método devem terminar em ponto, exemplo “ClasseExemplo.” e “metodoExemplo.”.

A malha de testes deve ser colocada no diretório Atividade6/src/test/java/malhaDeTestesASerAnalisada com package malhaDeTestesASerAnalisada. Neste mesmo diretório existe o ficheiro TodosOsTestes.java, onde devemos acrescentar o nome da malha de testes a ser executada, mais concretamente, no @SuiteClasses. Exemplo: @SuiteClasses(ClasseExemplo.class)

Agora, finalmente, estamos aptos para rodar o programa. Executando o protótipo como junit test, deve ser imprimido no output o grau de cobertura da malha de teste para cada classe, método e decisão do código fonte.

- **Testes para a validação do protótipo**

Para a validação do nosso protótipo criamos duas classes de testes. A HashTest, visto todas as funcionalidades mais importantes do mesmo se encontram implementadas na classe Hash. Esta classe contém as implementações da

montagem das hashs como as valores da execução das malhas de teste e os valores necessários para a cobertura do mcdc, comparação das hashs com os valores resultantes da execução das malhas de teste e os valores necessários para o cobrimento do mcdc e por fim a implementação da determinação de cobertura do mcdc de cada uma da classes e a sua respectiva impressão. A LeituraXMLTest é simplesmente para garantir que as condições do mcdc estão sendo lidas conrretamente do arquivo xml.

Na HashTest temos os seguintes testes:

- setAndGetHashExecutadosTest: este método testa os geters e seters da Hashtable que contem as valorações da execução das malhas de teste;
- comparaHashTablesTest: este método testa a comparação da hashtable dos valores exercitados e a hashtable dos valores para o cobrimento do mcdc para a determinação da porcentagem de cobertura do mcdc.

Na LeituraXMLTest temos o seguinte teste:

- getRequisitosMCDCTeste: este método testa se a extração dos requisitos do mcdc está sendo feita corretamente.

- **Conclusões**

A implementação da atividade 6 foi muito tranquila. Acreditamos que isso deve-se grande parte pela atividade 5, onde simplesmente pensamos em como implementar a adequação da malha de testes para o mcdc a partir do protótipo da atividade 4.

As facilidades que tivemos na elaboração do projeto além do conceito da avaliação da adequação da malha de testes ser simples e da professora não ter exigido uma instrumentação de código automática(muito provavelmente no bytecode) foram:

- reúso dos requisitos do mcdc gerados pelo protótipo da atividade 4;
- leitura e interpretação das informações contidas no xml trivial graças à biblioteca xstream.jar que usamos desde a atividade 4 para gerar o xml com os requisitos do mcdc; ividade 4;
- instrumentação do código manual e diretamente no código fonte;
- a linguagem usada para implementar a atividade 6 possui um leque variado de estruturas e recursos (usamos principalmente hastables e json object) que nos ajudaram no armazenamento, determinação e impressão do grau de cobertura da malha de testes.

A única verdadeira dificuldade que tivemos durante a atividade 6 foi o cálculo da porcentagem de cobertura do mcdc. Esse cálculo tinha que ser feito em relação á classe, método e decisão. Isso gerou alguns problemas de duplicação das porcentagens de cobertura que posteriormente foram resolvidas.

Um ponto negativo no nosso projeto é que ele não foi integrado no protótipo da atividade 4, logo, como dito no manual do usuário é necessário usar os dois protótipos para determiarmos o grau de cobertura da malha de testes, ou seja, reduz drasticamente o usabilidade do mesmo. Além disso como o protótipo é dependente dos requisitos do mcdc do protótipo da atividade 4, que não trata todos os casos do mcdc, exemplo tratamento das mácaras e a negação, então a portcentagem de cobertura do mcdc apresentada por este protótipo não é 100% coerente com a realidade. Apesar de tudo estamos satisfeitos com os resultados obtidos.