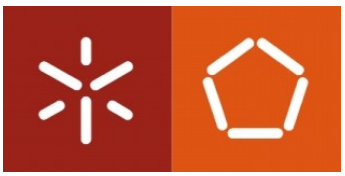

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Métodos Formais em Engenharia de Software

Análise e Teste de Software

Métricas e Smells para C– e MSP
(2ª Fase)

Grupo 11
Carlos Sá - A59905
Filipe Oliveira - A57816
Sérgio Caldas - A57779

Resumo

O presente documento constitui o relatório da primeira fase do projeto desenvolvido no âmbito da UCE de Análise e Teste de Software.

Este projeto consiste no desenvolvimento de um catálogo de métricas para a linguagem C– e MSP. Assim, pretende-se definir um catálogo de métricas que permita avaliar código fonte escrito na linguagem **C–** e MSP. Tal será feito com o recurso ao **Tom** - uma linguagem para manipulação de àrvores e **Gom** que permite a representação da árvore numa especificação algébrica. Através destas métricas poderemos extrair indicadores que permitam localizar pontos de *correção*.

Entre as várias métricas temos, por exemplo, as que vão contabilizar o número de linhas de código, o número de funções, ciclos, chamadas a funções recursivas entre outras.

Com o presente catálogo pretende-se localizar *bad smells* em C– em função de valores dessas métricas. Baseando-nos num conjunto de bons programas em C– definem-se o valor de referência para algumas dessas métricas. Caso seja analisado um programa que não siga esse valor de referência então um "bad smell"é localizado (por exemplo, uma função com muitas linhas). Ao longo deste documento explicaremos como foram criadas essas métricas e como foram utilizadas para um conjunto de programas. O projeto pode ser acedido através do seu respectivo repositório **GitHub** aqui.

1 Métricas de Software

1.1 Primeira Fase

Abaixo segue um conjunto de métricas/contadores implementadas pelo grupo até ao final da primeira fase do projeto:

```
Integer numberFunctions;
HashMap <String,Integer> functionsNumberParameters;
```

- Contar o número total de funções do código;
- Contar o número de parâmetros das funções;

1.2 Segunda Fase

Nesta segunda fase o grupo implementou as métricas em baixo listadas.

- Complexidade Ciclomática;

A função responsável por esse cálculo pode ser consultada de seguida:

```
%strategy startCollectCyclomaticInside( complexidade:HashMap, funcao:String , maximo_medido:int ) extends Identity() {
  visit Instrucao {
    For(_,_,_,_,_,_,_,_,_,_,inst,_) -> {
      maximo_medido++;
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa <= maximo_medido ){
        complexidade.put(funcao,maximo_medido);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao,maximo_medido)).visit(`inst);
    }
    If(_,_,_,_,_,_,inst,_) -> {
      maximo_medido++;
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa <= maximo_medido ){
```

```
        complexidade.put(funcao,maximo_medido);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao,↩
          maximo_medido)).visit(`inst);
    }
    While(_,_,_,_,_,_,inst,_) -> {
      maximo_medido++;
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa <= maximo_medido ){
        complexidade.put(funcao,maximo_medido);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao,↩
          maximo_medido)).visit(`inst);
    }
  }
}



%strategy startCollectCyclomatic(HashMap complexidade, String funcao) ↩
    extends Identity() {
  visit Instrucao {
    For(_,_,_,_,_,_,_,_,_,_,_,inst,_) -> {
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa < 1 ){
        complexidade.put(funcao,1);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao, 1)).visit(`↩
          inst);

    }
    If(_,_,_,_,_,_,inst,_) -> {
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa < 1 ){
        complexidade.put(funcao,1);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao, 1)).visit(`↩
          inst);
    }
    While(_,_,_,_,_,_,inst,_) -> {
      int valor_mapa = (int) complexidade.get(funcao);
      if ( valor_mapa < 1 ){
        complexidade.put(funcao,1);
      }
      `TopDown(startCollectCyclomaticInside(complexidade,funcao, 1)).visit(`↩
          inst);
    }
  }
}


/** Tentativa definir uma métrica para contar o número de funć es  ***/
%strategy CollectNumberFuncs(func:HashMap,complex_c:HashMap) extends ↩
    Identity() {
  visit Instrucao {
    Funcao(_,tipo,_,nome,_,_,argumentos,_,_,inst,_) -> {
      func.put(`nome, `argumentos);
      complex_c.put(`nome,0);
      `TopDown(startCollectCyclomatic(complex_c,nome)).visit(`inst);
    }
  }
}
```

1.2.1 Ficheiro de métricas gerado

Para além desta métrica que implementamos nesta fase, o grupo corrigiu alguns aspetos, no código da primeira fase, que consideramos não estarem implementados da melhor maneira.

Assim sendo, o ficheiro com as métricas apresenta agora o seguinte aspecto:

```
Number of fuctions: 3
Calculated Cyclomatic Complexity: 4
Number of Arguments per function:
 max : 2
 max3numbers : 3
 main : 0
Cyclomatic Complexity per function:
 max : 4
 max3numbers : 2
 main : 1
```

2 Trabalho Futuro

Como trabalho futuro na próxima fase, o grupo tem como objetivo implementar os seguintes métodos/contadores:

```
HashMap <String,Integer> LongMap;
Integer totalLinesOfCode;
```

- Calcular o comprimento das funções (**LongMap**);
- Contabilizar o número total de linhas de código;

Denote que, caso consigamos implementar os métodos/métricas anteriormente definidas, pretendemos calcular mais métricas que permitam avaliar padrões de código de maior complexidade;

3 Dificuldades e limitações

Apesar desta fase ser ainda uma fase muito preliminar do projeto, para obter os resultado incluídos neste relatório o grupo enfrentou um conjunto de várias dificuldades que limitou a progressão do trabalho.

A primeira dificuldade está relacionada com o código fornecido pela equipa docente. Precisamos de despender de muito tempo para conseguir perceber o programa e perceber como é definiríamos as métricas utilizando o respetivo código. O facto de nunca termos utilizado Tom e Gom antes desta unidade curricular poderá estar na origem dessas dificuldades.