Métricas e Smells para C– e MSP

Análise e Teste de Software

Carlos Sá[1] Filipe Oliveira[1] Sérgio Caldas[1]

[1]Departamento de Informática
Universidade do Minho
Grupo 11

Apresentação ATS
21 de Janeiro de 2016

Sumário

- Catálogo Métricas Criadas;
- Programas em C– Utilizados;
- Estratégias de *Refactoring*;
- *Bad Smells*;
- Sistema de Classificação dos Programas/Funções;
- Overview Pyramid;
- Overview Gráfica das Métricas;
- Refactoring Automático;
- Overview Gráfica do Refactor;
- Demonstração.

Catálogo Métricas Criadas :: Parte 1/3

- Cálculo do número de funções;
- Contagem do número de argumentos por função;
- **Complexidade Ciclomática** (**CYCLO**);
- Número complexidade operacional intrinsica (**CYCLO/LOC**)
- Linhas de código por operação (**LOC/NOM**);
- Cálculo do número de **operadores por função**;
- Número total de **operadores únicos por função**;
- Número total de **operandos por função**;
- Número total de **operandos únicos por função**;
- Cálculo da **métrica de Halstead** (Volume, Dificuldade, Esforço, e Bugs);
- **Número de comentários** por função;

Catálogo Métricas Criadas :: Parte 2/3

- Número total de linhas de código (**LOC**);
- Cálculo do *Path length* de instruções (**IPL**);
- Contagem do Número **Total de Operações de Comparação**;
- Cálculo do número **total de operações de incremento/decremento**;
- Número total de **operações de atribuição**;
- Número de **argumentos não utilizados** por função;
- Total de declarações não usadas por função;
- Número de packages (**NOP**). (1 ficheiro código C– corresponde a 1 package);
- Número total de Classes (**NOC**). No nosso caso 1;
- Número total de classes por package (**NOC/Package**). No nosso caso 1;
- Número total de operações (**NOM**);

Catálogo Métricas Criadas :: Parte 3/3

- Número total de operações por classe (**NOM/Class**);
- Número de chamadas (**CALLS**) a operações;
- **Total de classes** invocadas;
- **Coupling intensity**: número de chamadas por operação (CALLS/Operações);
- **Coupling dispersion**: FANOUT/Operation_Call (número de classes envolvidas por operação;
- Número médio de classes derivadas (**ANDC**): Número de subclasses directas de uma classe. No nosso caso, 0;
- **AHH** Altura média da hierarquia (tamanho médio do path length desde a raíz até à subclasse mais profunda). No nosso caso é assumido por omissão 1 subclasse e o respectivo path length máximo da função em estudo;
- Indice Maintainability;
- Número de operações por classe (NOM/NOC).

Exemplo Estratégia para Cálculo das Métricas

```
/** Metric to compute total number of functions **/
%strategy CollectNumberFuncs(func:HashMap, mapArgs:↩
    HashMap) extends Identity() {
visit Instrucao {
    Funcao(_,tipo,_,nome,_,_,argumentos,_,_,inst,_) -> ↩
        {
         func.put( nome,  argumentos);
         mapArgs.put( nome, 0);
          TopDown(countArgsFunction(mapArgs,nome)).visit↩
             ( argumentos);
         }
    }
}
```

Exemplos de Programas em C– Utilizados

- max (Cálculo do valor máximo entre dois números);
- max3numbers (Cálculo do máximo de três números);
- factorialI (Cálculo do factorial *Imperativo*);
- factorialR (Cálculo do factorial *Recursivo*);
- descN (que calcula a sequência de descendentes inteiros de N);
- oddOrEven (que verifica se um valor é par ou impar);
- swap_ab (troca o valor de duas variáveis);
- mult2num (multiplicação de dois valores inteiros);
- printNnaturals (imprime os naturais de 1 até N);
- armstrongnumber (que calcula valores de Armstrong)

Estratégias de Refactoring

- Refactoring da negação das condições dos if's;
- Dupla negação de condições;
- Remoção de argumentos não utilizados por uma função;
- Remoção de declarações não utilizadas;

Exemplo de Estratégias para Refactoring

```
public static Instrucao removeDeclaracoesNaoUtilizadas ↩
    ( Instrucao inst, TreeSet<String> idsNaoUtilizados ↩
    ) {
      %match(inst) {
        Declaracao(_,_,_,declaracoes,_,_) -> {
          %match(declaracoes) {
            ListaDecl( dec1,taliDec*) -> {
              %match(dec1) {
                Decl(id,_,_,_,_) -> {
                  if ( idsNaoUtilizados.contains( id))
                    return  Exp(Empty());
                }
              }
            }
          }
        }
      }
      return inst;
```

Bad Smells

- Para detectar *Bad Smells* num programa em *C–*, é necessário comparar os valores obtidos das Métricas obtidas com os valores de referência. Esses valores podem ser consultados na tabela em baixo:

	Low	Average	High	Fonte
Cyclo/Lines of Code	0.20	0.25	0.30	OOMP[3]
LOC/Operation	5	10	16	OOMP [3]
NOM/Class	4	9	15	OOMP [3]
NOC/Package	3	19	35	OOMP [3]
CALLS/Operation	1.17	1.58	2	OOMP [3]
Fanout/CALL	0.20	0.34	0.48	OOMP [3]
ANDC	0.19	0.28	0.37	OOMP [3]
AHH	0.05	0.13	0.21	OOMP [3]
Indice Maintainability	0.05	0.10	0.95	MICROSOFT [1]
LOC	9	19	39	NDEPEND CODE ANALYSER [2]
Number of Arguments	3	5	8	NDEPEND CODE ANALYSER [2]
Unused Arguments	0	0	1	NDEPEND CODE ANALYSER [2]

Figura: Valores de Referência para as Métricas

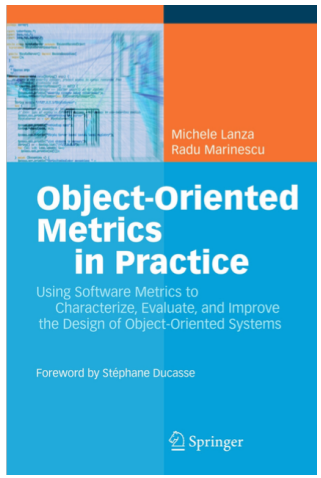
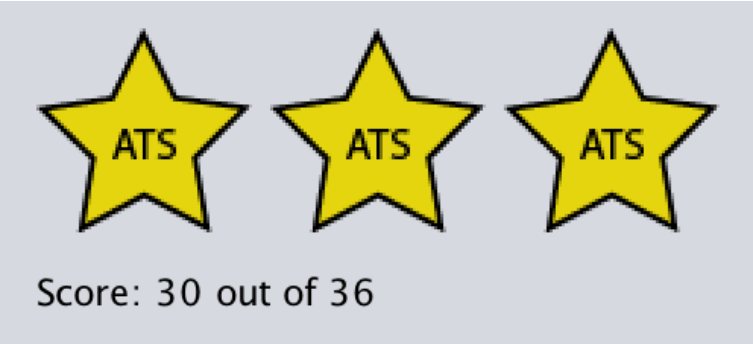


Figura: Livro OOMP

Bad Smells

- Se um dado valor obtido para um métrica através do nosso programa está acima do valor tabelado *High* então um bad smell é detetado;
- A partir do momento em que um *Bad Smell* é detetado pelo nosso programa é necessário classificar negativamente o valor da métrica obtida através de um sistema de classificação.
- Se o valor da métrica estiver dentro do intervalo de valores estabelecidos, então é atribuída uma classificação positiva.

Sistema de Classificação dos Programas/Funções

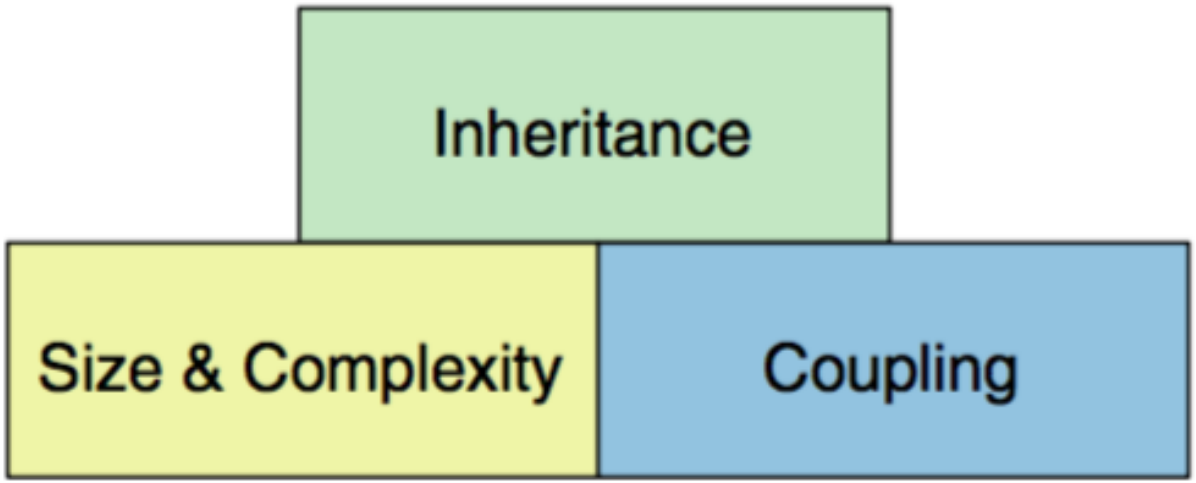


- A classificação de cada Programa/Função é calculada da seguinte forma:
  - Cada métrica possui os seus valores de referência (como foi visto em cima);
  - Cada valor da métrica é comparado com o respectivo valor de referência;
  - Por fim é atribuída a classificação (1 até 3 estrelas) ao Programa/Função;
    - 1 estrela – valores entre 0 e 12 pontos;
    - 2 estrelas – valores entre 13 e 24 pontos;
    - 3 estrelas – valores entre 25 e 36 pontos;

Overview Pyramid

- Primeiras 8 métricas representam a chamada "Overview Pyramid";
- Calcular essa mesma pirâmide para o código em análise.
- Organizar primeiras 8 métricas por 3 super grupos iguais aos apresentados no livro:
  - Métricas que relacionam Herança;
  - Métricas que relacionam Tamanho e Complexidade;
  - Métricas que relacionam Coupling (relação entre classes);

Atente no seguinte exemplo do livro:



Overview Pyramid (slide 2)

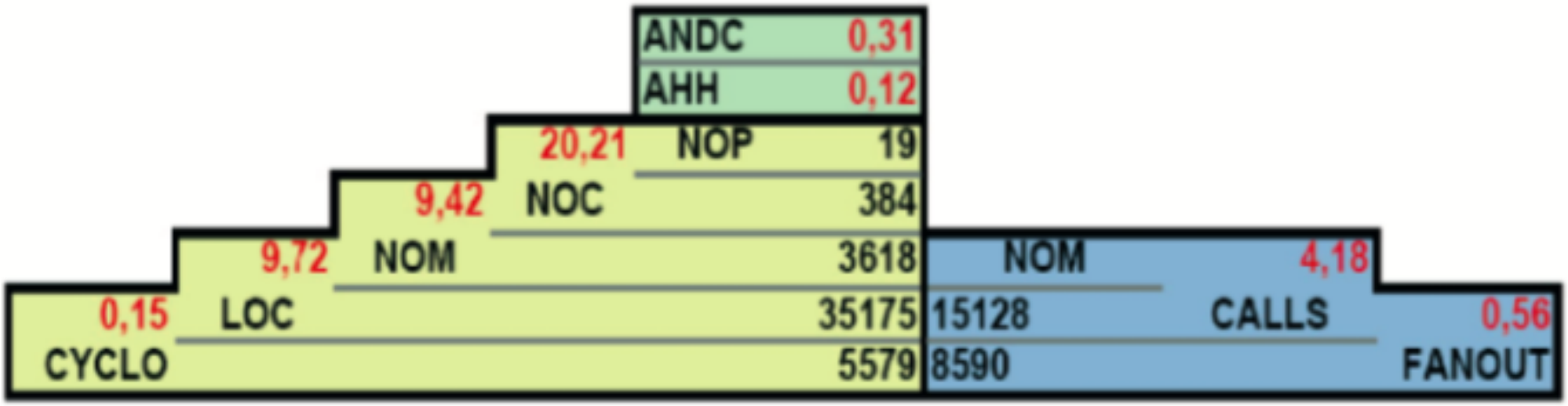


Figura: Overview completa da Pirâmide

Overview Gráfica das Métricas

- Número elevado de métricas calculadas;
- Essencial visualizar e interpretar métricas de uma forma simples.

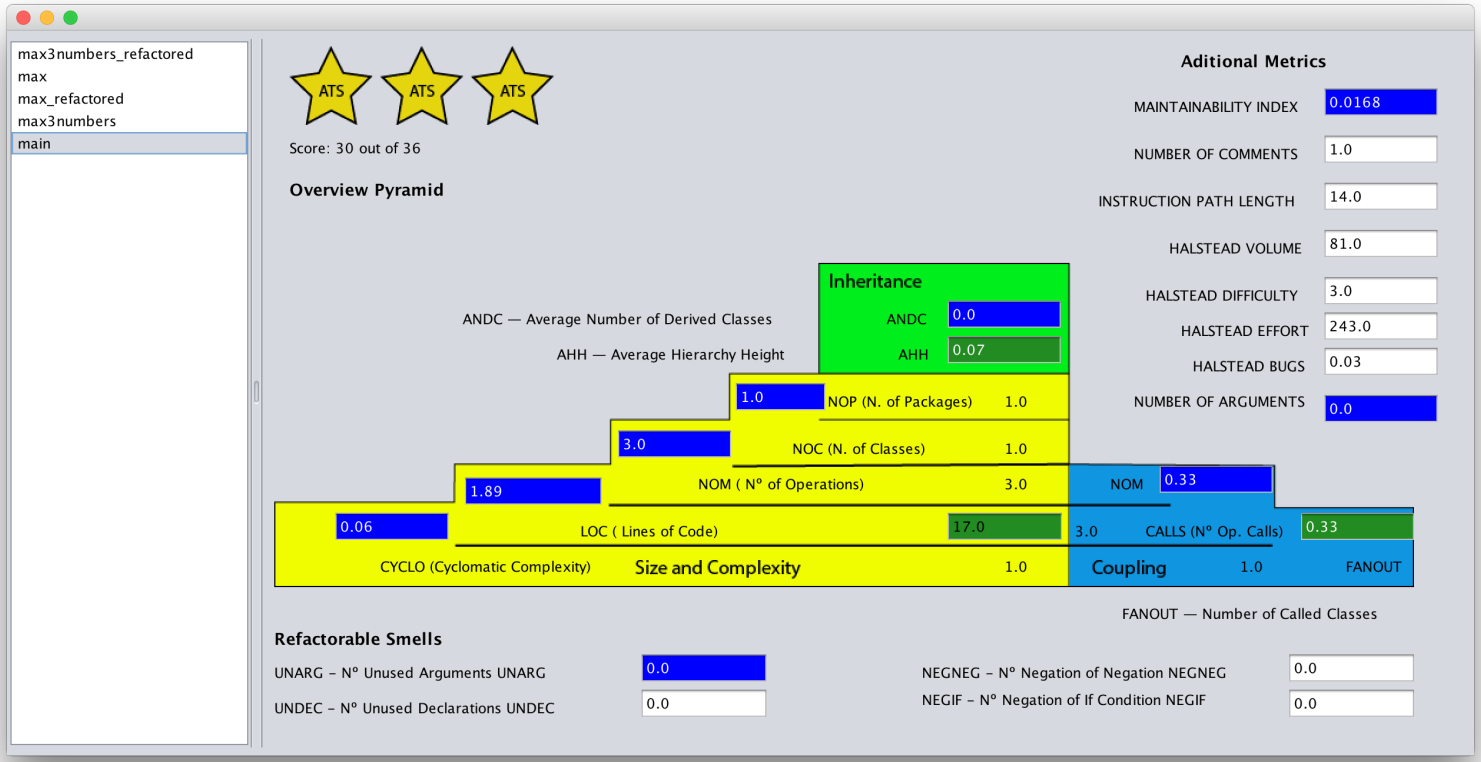


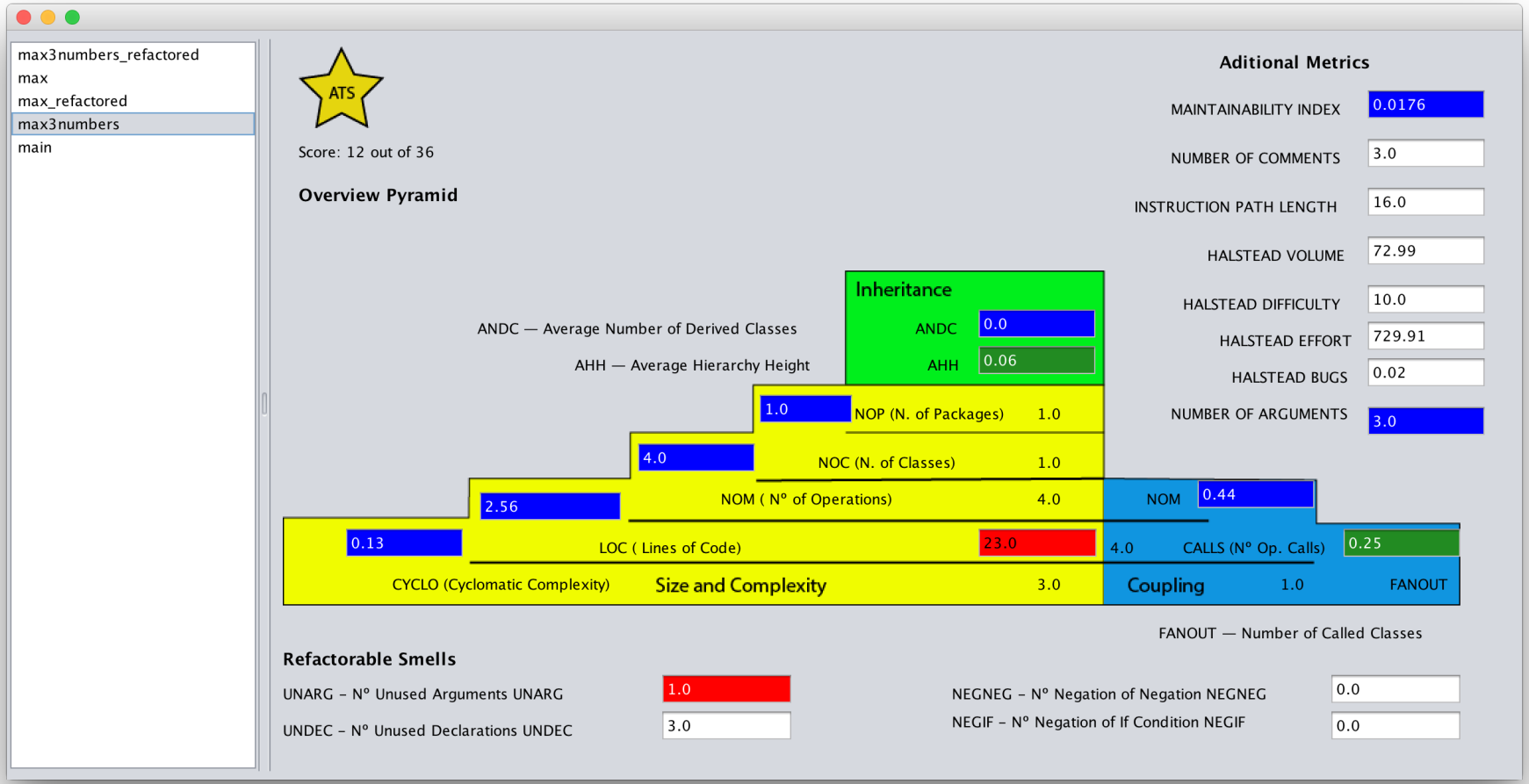
Figura: Interface Gráfico do Sistema de Classificaçãdo dos Programas/Funções

Refactoring Automático

- Sempre que é identificado um *Bad Smell* é aplicado uma estratégia de *refactoring* ao código fonte;
- Como não existem refabricações para todo o tipo de *Smells* indentificados, são aplicadas transformações de código apenas nos casos apresentados em cima;
- A transformação do código fonte é feita de forma automática pelo programa.

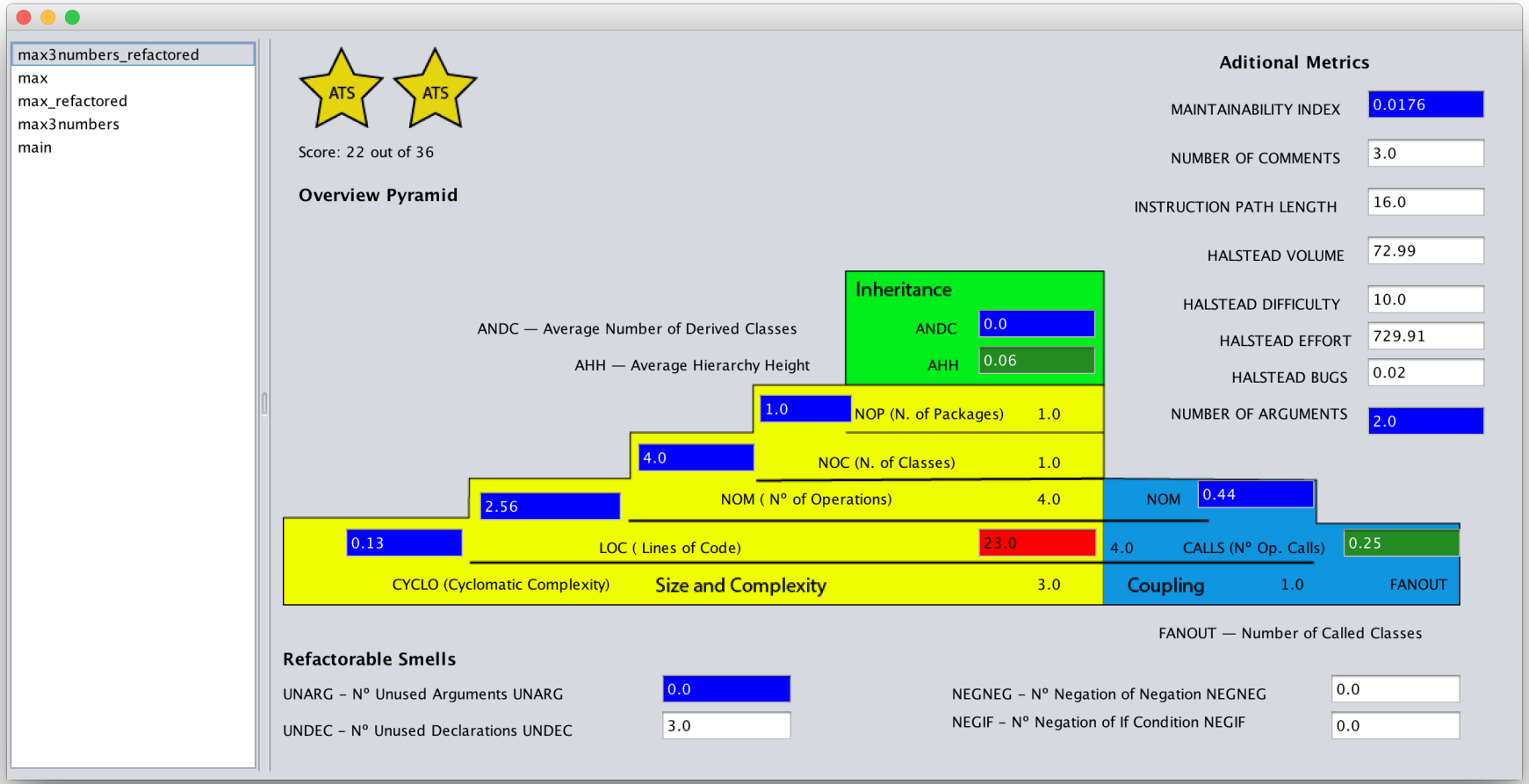
Overview Gráfica do Refactoring - parte I

- Overview antes do Refactoring



Overview Gráfica do Refactoring - parte II

- Overview antes do Refactoring



Demo

- DEMO

Métricas e Smells para C– e MSP

Análise e Teste de Software

Carlos Sá[1] Filipe Oliveira[1] Sérgio Caldas[1]

[1]Departamento de Informática
Universidade do Minho
Grupo 11

Apresentação ATS
21 de Janeiro de 2016