Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação

CE-230
Qualidade, Confiabilidade e Segurança de Software

Prof. Dr. Adilson M. Cunha

Prova 2
Aplicação de Métricas e Análise de Sensitividade e Traçabilidade de Requisitos

Aluno: **Walter Abrahão dos Santos**
walter@dss.inpe.br

Novembro-2004

1. Introdução

Para a aplicação de métricas foi escolhido o software gerado para o Protótipo do Veículo Experimental TRIPHIBIUS da disciplina de Sistemas Embarcados e de Tempo Real (CE235/2003).

O módulo de interesse a ser analisado é o protótipo da Central de Alarmes (CEAL) do veículo que foi posteriormente integrado com os subsistemas Elétrico (ELET) e Barramento de Dados (BADA). Estes três subsistemas juntos formam Supervisão (SUP).

Os diagramas e código fonte gerados para o subsistema CEAL foram agora submetidos à ferramenta Together da Borland para medições quanto à sua qualidade e confiabilidade. Para detalhes de projeto, com os diagramas e o código fonte reporte-se ao URL: http://cdt.br/sites/daniel.pegas/sup-t.htm.

O diagrama de classes visto via Together é apresentado na Fig.1.

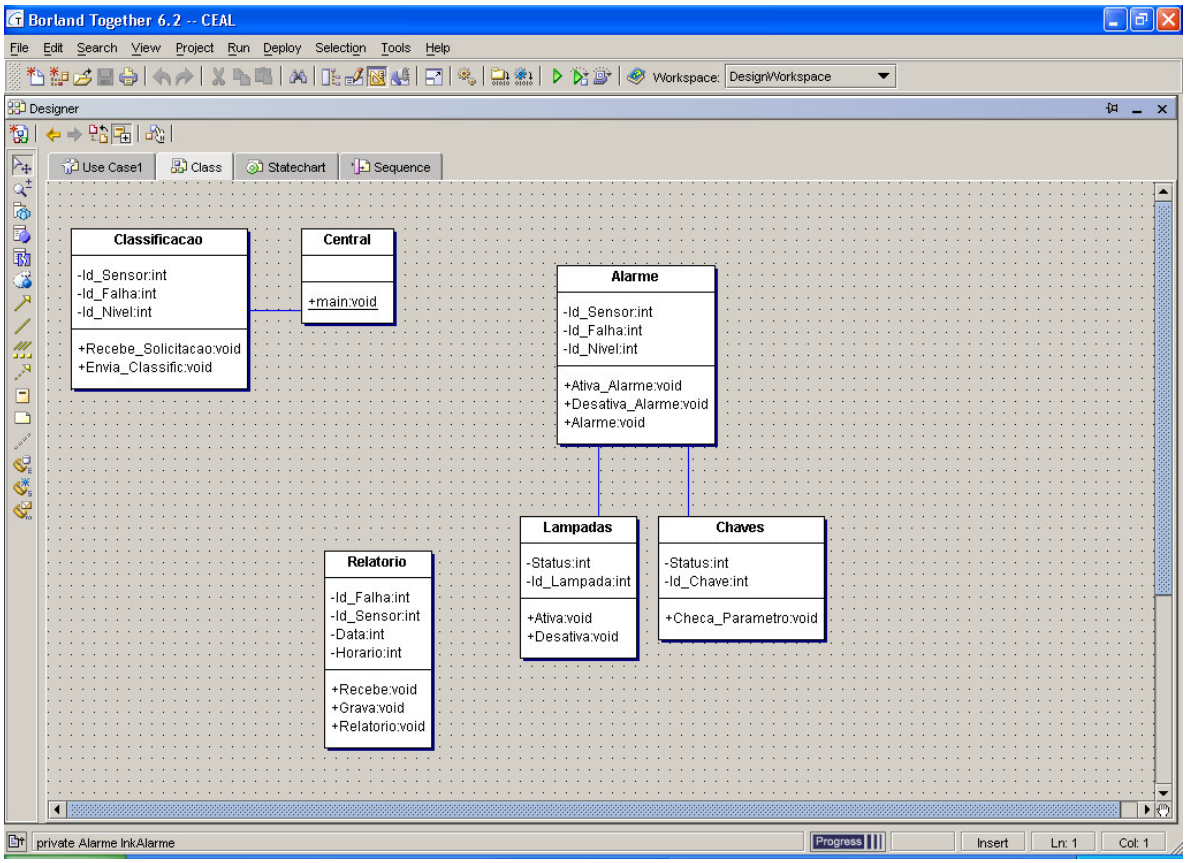

Fig. 1 - O diagrama de classes do CEAL

2. Aplicação das Métricas

Para análise do código do CEAL foram adotadas as seguintes métricas:

- **CR - Comment Ratio**
- **LOCOM2 - Lack Of Cohesion Of Methods 2**
- **NOO - Number Of Operations**

Uma descrição de cada métrica é apresentada a seguir tal como fornecida pela ferramenta.

CR - Comment Ratio

Counts the ratio of documentation and/or implementation comments to total lines of code (comments are included in the code count). Blank lines can be optionally interpreted as code.
For Java, C++, and C#, you can also specify which type of comments to use for the ratio.

LOCOM2 - Lack Of Cohesion Of Methods 2

Counts the percentage of methods that do not access a specific attribute averaged over all attributes in the class. A high value of cohesion (a low lack of cohesion) implies that the class is well designed. A cohesive class will tend to provide a high degree of encapsulation, whereas a lack of cohesion decreases encapsulation and increases complexity.

Supported languages: Java, C++

NOO - Number Of Operations

Counts the number of operations. Inherited members are counted if the Ancestors box is checked. If a class has a high number of operations, it might be wise to consider whether it would be appropriate to divide it into subclasses.

Supported languages: Java, C++, Visual Basic, VisualBasic.Net, C#

Para fins de ilustração, a Fig. 2 mostra o gráfico de Kiviat do código CEAL para todas as métricas disponíveis no Together.

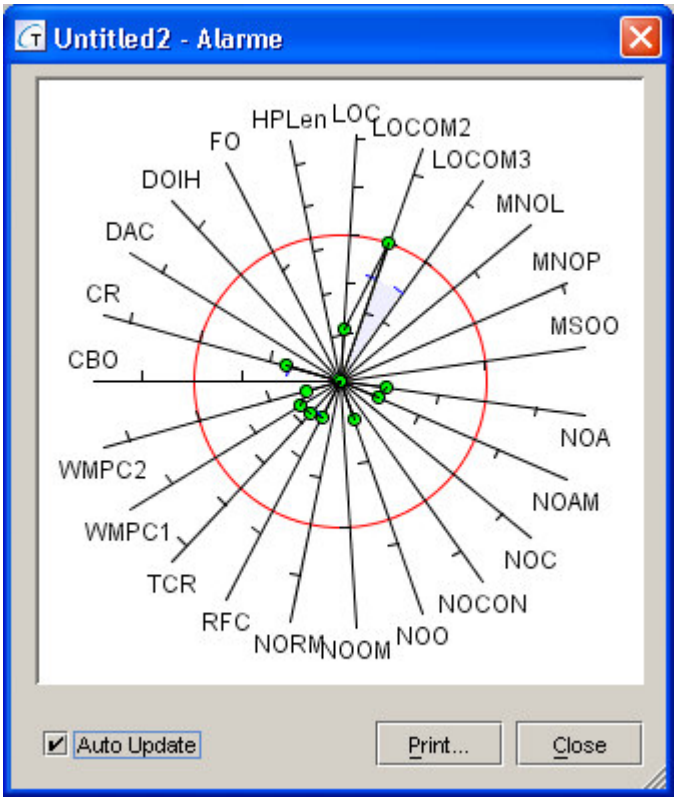

Fig. 2 - Gráfico de Kiviat para todas as métricas aplicadas ao código do CEAL.

Na Fig. 3 mostra o gráfico de Kiviat do código CEAL somente para as **métricas escolhidas** para análise no Together. Para a análise foram adotados os seguintes limites:

NOO – de 0 à 50 CR – de 5 à 101 LOCOM2 – de 30 à 101

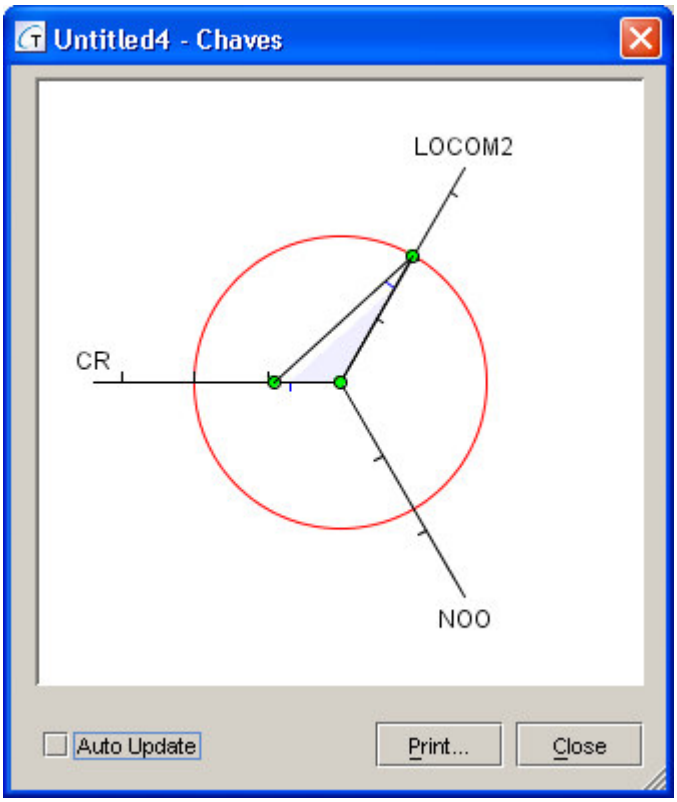

Fig. 3 - Gráfico de Kiviat do código CEAL para métricas CR, NOO e LOCOM2.

3. Análise Detalhada e Utilização das Métricas Escolhidas

A seguir, cada uma das métricas escolhidas são analisadas individualmente e apresentadas na forma tabular e em gráfico de barras para melhor visualização da distribuição da ocorrência das métricas dentro de cada classe do projeto CEAL.

Adicionalmente, comentários de uso e recomendações gerados dinamicamente pela ferramenta Together são apresentados.

3.1 CR - Comment Ratio

How to use CR - Comment Ratio

The appropriate number of comments depends on the corporate policy in this area. In any case, comments improve readability of the program and simplify changes.

It is recommended to provide at least 5% comments relative to the total amount of the code and comment lines. The upper value is unlimited.

Current Highs

Class	CR	Full name
Interface1	50	Interface1
Chaves	8	Chaves
Lampadas	7	Lampadas
Classificacao	6	Classificacao
Alarme	6	Alarme

Current Lows

Class	CR	Full name
Central	3	Central
Relatorio	4	Relatorio
Alarme	6	Alarme
Classificacao	6	Classificacao
Lampadas	7	Lampadas

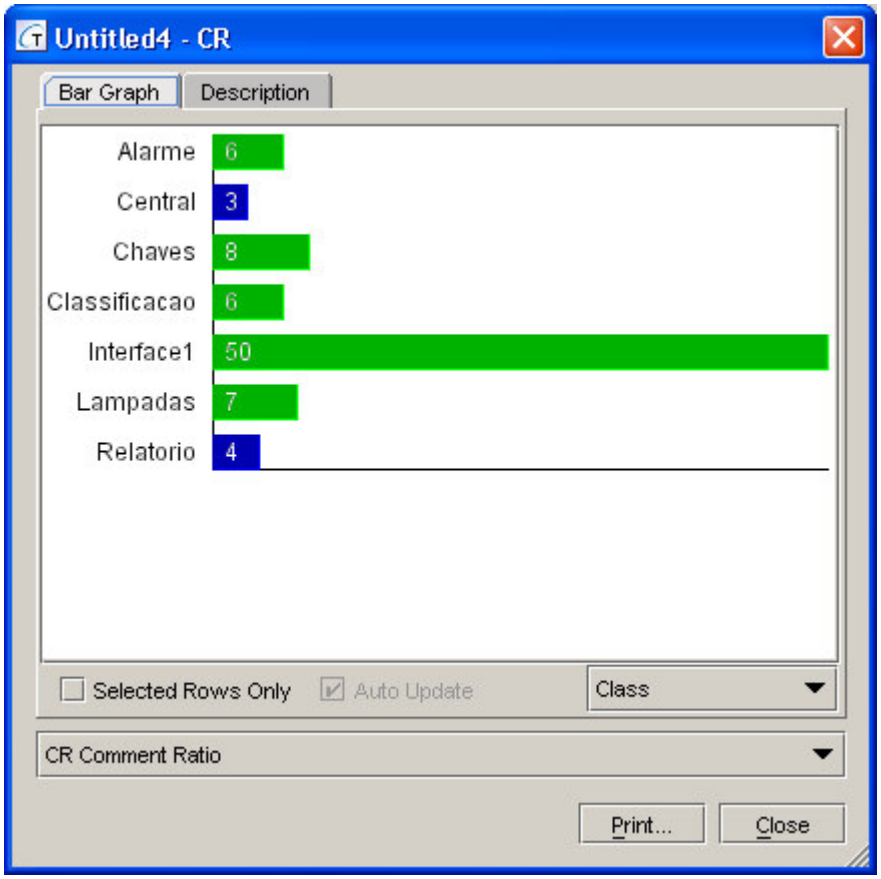

Fig. 4 - Gráfico de barras para métrica CR em classes do CEAL

3.2 LOCOM2 - Lack Of Cohesion Of Methods 2

How to use LOCOM2 - Lack Of Cohesion Of Methods 2

Analyze only those classes for which this metric exists. Avoid classes that have a value less than 30%. A low value suggests a poor class design, which requires increased effort for testing or modifying the class. To increase the value, it is recommended to redesign the class.
This metric, along with LOCOM1, has a few drawbacks (Brian Henderson-Sellers "Object-Oriented Metrics. Measure of Complexity"). It's possible to use LOCOM3 as an alternative.

Current Highs

Class	LOCOM2	Full name
Relatorio	100	Relatorio
Lampadas	100	Lampadas
Classificacao	100	Classificacao
Chaves	100	Chaves
Alarme	100	Alarme

Current Lows

Class	LOCOM2	Full name
Alarme	100	Alarme
Chaves	100	Chaves
Classificacao	100	Classificacao
Lampadas	100	Lampadas
Relatorio	100	Relatorio

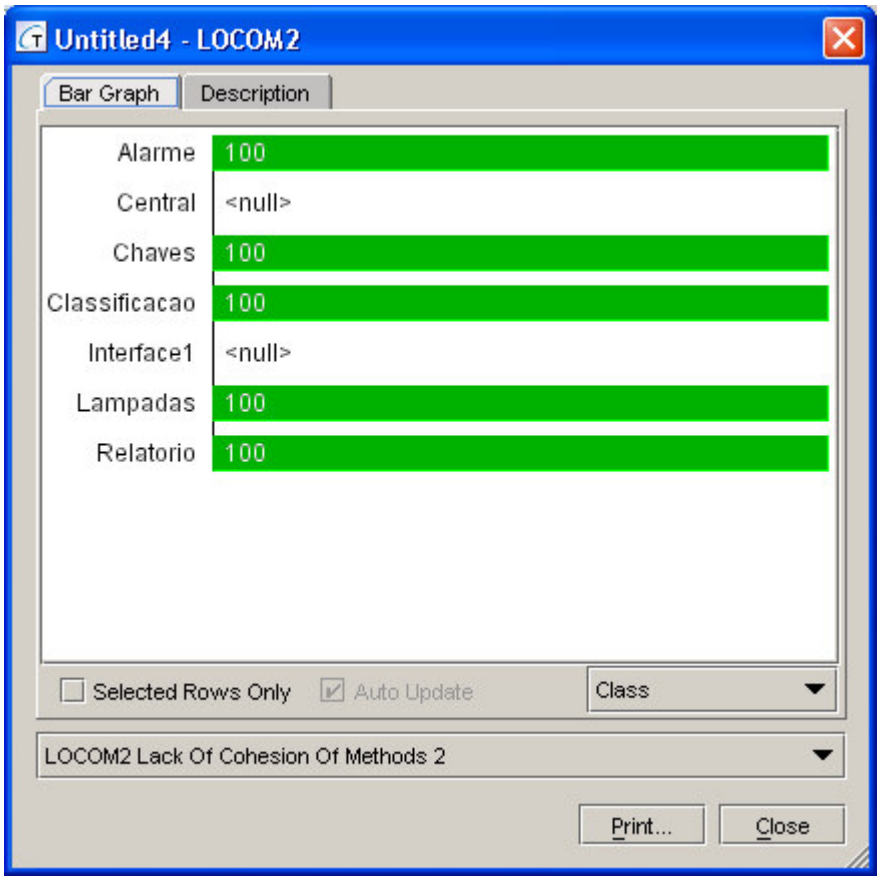

Fig. 5 - Gráfico de barras para métrica LOCOM2 em classes do CEAL.

3.2 NOO - Number Of Operations

How to use NOO - Number Of Operations

Avoid classes where the number of methods exceeds 50 (all visibility modifiers except for base). To reduce the number of methods, split classes into subclasses or spread out parts of methods over the hierarchy of classes.

Current Highs

Class	NOO	Full name
Relatorio	3	Relatorio
Alarme	3	Alarme
Lampadas	2	Lampadas
Classificacao	2	Classificacao
Chaves	1	Chaves

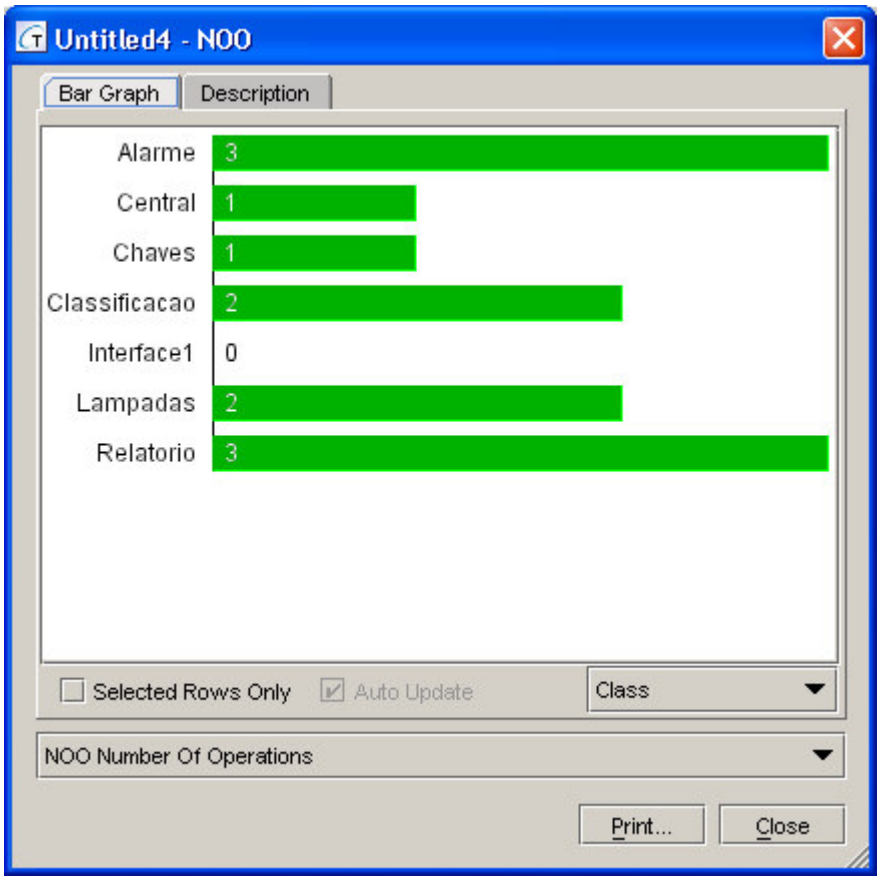

Fig. 6 - Gráfico de barras para métrica NOO em classes do CEAL.

4. Análise de Sensitividade e Traçabilidade de Requisitos

Para a análise de sensitividade foi feita a introdução de um novo método `Mess_up( )` no código da classe Alarme conforme mostrado na Fig. 7. Este novo método interfere no código original fazendo acesso à atributos privados da classe Alarme o que prejudica a métrica LOCOM2 conforme a definição apresentada anteriormente.

De acordo com mostrado na Fig. 8, nota-se que a métrica LOCOM2 passa de 100% originalmente para 75% o que empobrece o código.

```
/* Generated by Together */
import java.*;

public class Alarme {
    public void Ativa_Alarme() {
    }

    public void Desativa_Alarme() {
    }

    public void Alarme() {
    }

       // Código intruso

    private void Mess_up(){
        Id_Sensor = 1;
        Id_Falha = 1;
        Id_Nivel = 1;
    }

    private int Id_Sensor;
    private int Id_Falha;
    private int Id_Nivel;
}
```

Fig. 7 – Introdução de código intruso na definição da classe Alarme.

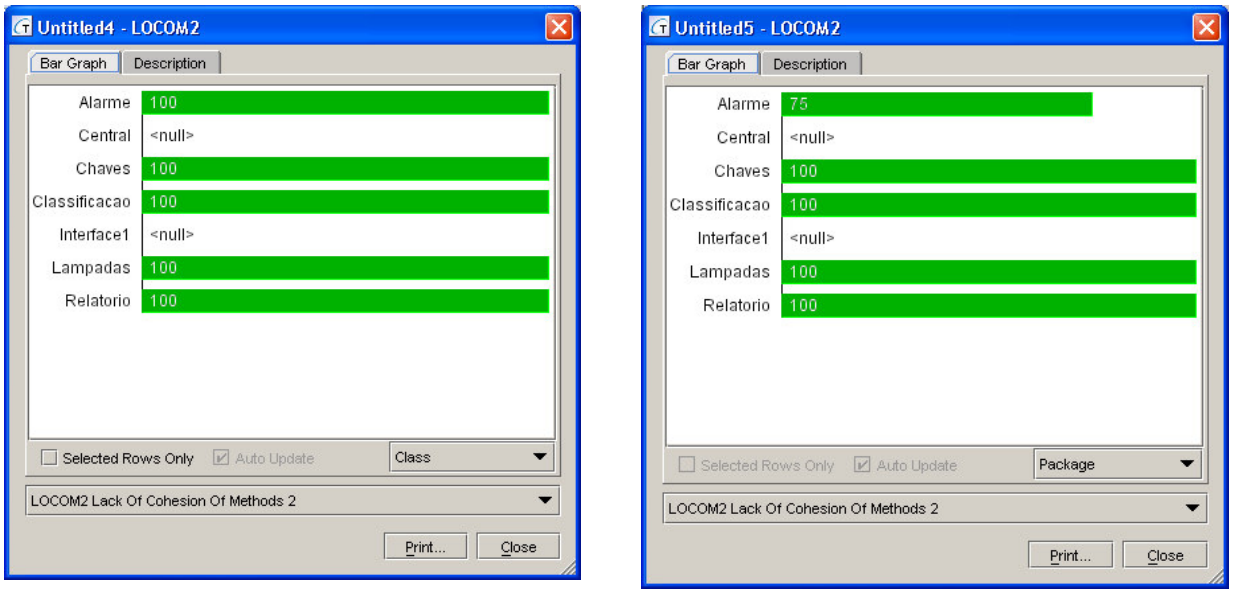

Fig. 8 – Efeito na métrica LOCOM2 antes e depois da introdução de método Mess_up( ), código intruso na definição da classe Alarme

A análise de traçabilidade de requisitos não foi realizada por desconhecimento de detalhes do projeto CEAL.