Protótipo de Métricas para Programação Orientada a Objetos

Aluno: Eduardo José Cardoso

Orientador: Everaldo Artur Grahl

Roteiro da Apresentação

- Introdução
- Conceitos Básicos de Métricas
- Conceitos Básicos de OO
- Métricas Orientada a Objetos
- Especificação
- Funcionamento do Protótipo
- Conclusão
- Sugestões

Introdução

- Origem

As métricas originaram-se da aplicação de cálculos para quantificar indicadores sobre o processo de desenvolvimento do software. A partir de 1970 os projetos de software tornaram-se grandes e mais complexos, sendo necessário o controle do processo de forma emergente.

Introdução

- Objetivo

O objetivo principal do trabalho é a especificação e implementação de um software que irá analisar o código fonte de programas orientado a objetos em Delphi e fornecer resultados de cálculos de algumas métricas estudadas.

Conceitos Básicos de Métricas

- Objetivo com a Utilização de Métricas
- Importância da Medição do Software
- Problemas na Implantação de Métricas
- Como são as Métricas Atuais

Conceitos Básicos de Orientação a Objetos

- **Objetos -** Qualquer indivíduo, lugar, coisa, evento, tela, relatório ou conceito que seja aplicável ao projeto do sistema. Os objetos possuem dados e funcionalidade que definem seus comportamentos.

- **Classe** - Uma descrição de um conjunto de objetos semelhantes.

Conceitos Básicos de Orientação a Objetos

- **Herança** - Permite tirar vantagem das similaridades entre as classes.
- **Atributos** - Algo que um objeto ou uma classe sabem. Um atributo é basicamente um único dado ou informação.
- **Métodos** - Algo que um objeto ou uma classe faz. Um método é semelhante a uma função ou procedimento em uma programação estruturada.

Métricas Orientada a Objetos

DC=Classes Descendentes

- DC(TCustomTextSupplier) = 2

```
function TMainClass.GetDescendentes(aClassDef: TDefinitionClass): integer;
var
  xClassDef: TDefinitionClass;
  i: integer;
begin
  Result := 0;
  for i := 0 to FListClass.Count -1 do
  begin
    xClassDef := TDefinitionClass(FListClass.Objects[i]);
    if CompareText(xClassDef.BaseClass, aClassDef.ClassName) = 0 then
    begin
      inc(Result);
      Result := Result + xClassDef.ClassesDescendentes;
    end;
  end;
end;
```

Métricas Orientada a Objetos

MN=Métodos Novos

- MN(TTextSupplier) = 1

```
function TMainClass.GetMetodosNews(aMetodos: TStrings): integer;
var
  ipub: Integer;
begin
  Result := 0;
  for ipub := 0 to aMetodos.Count -1 do
    if TTypeMetodos(aMetodos.Objects[ipub]) in [tmHerdavel, tmNone] then
      inc(Result)
end;

function TMainClass.GetNovos(aClassDef: TDefinitionClass): integer;
begin
  //Todos os metodos que estão na classe e não estão como virtual(Herdavél)
  Result := 0;
  Result := Result + GetMetodosNews(aClassDef.MetodosPublic) +
                     GetMetodosNews(aClassDef.MetodosProtected) +
                     GetMetodosNews(aClassDef.MetodosPrivate);
end;
```

Métricas Orientada a Objetos

MO=Métodos Redefinidos

- MO(TFileSupplier) = 1

```
//Todos os métodos do typo public e protected que sao herdados(override)
function TMainClass.GetReDefinidos(aClassDef: TDefinitionClass): integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to aClassDef.MetodosPublic.Count -1 do
  begin
    if TTypeMetodos(aClassDef.MetodosPublic.Objects[i]) = tmHerdado then
      Inc(Result);
  end;

  for i := 0 to aClassDef.MetodosProtected.Count -1 do
  begin
    if TTypeMetodos(aClassDef.MetodosProtected.Objects[i]) = tmHerdado then
      Inc(Result);
  end;

end;
```

Métricas Orientada a Objetos

MI=Métodos Herdados

- MI(TFileSupplier) = 0

```
function TMainClass.GetHerdados(aClassDef: TDefinitionClass): integer;
var
  xAllMetHerdados: TStringList;
  xclassBase: TDefinitionClass;
  i: integer;
begin
  Result := 0;
  if aClassDef.BaseClass <> '' then
  begin
    xclassBase := GetClassDef(aClassDef.BaseClass);
    if xClassBase <> nil then
    begin
      xAllMetHerdados := TStringList.Create;
      try
        xAllMetHerdados.Duplicates := dupIgnore;
        xAllMetHerdados.Sorted := true;
        GetAllMetHerdados(xAllMetHerdados, xClassBase);
        Result := xAllMetHerdados.Count;
        for i := 0 to aClassDef.MetodosPublic.Count -1 do
          if xAllMetHerdados.IndexOf(aClassDef.MetodosPublic.Strings[i]) <> -1 then
            Dec(Result);
        for i := 0 to aClassDef.MetodosProtected.Count -1 do
          if xAllMetHerdados.IndexOf(aClassDef.MetodosPublic.Strings[i]) <> -1 then
            Dec(Result);
      finally
        xAllMetHerdados.Free;
```

Ambiente Delphi 4

Desenvolvimento do Protótipo

Diagrama de Contexto

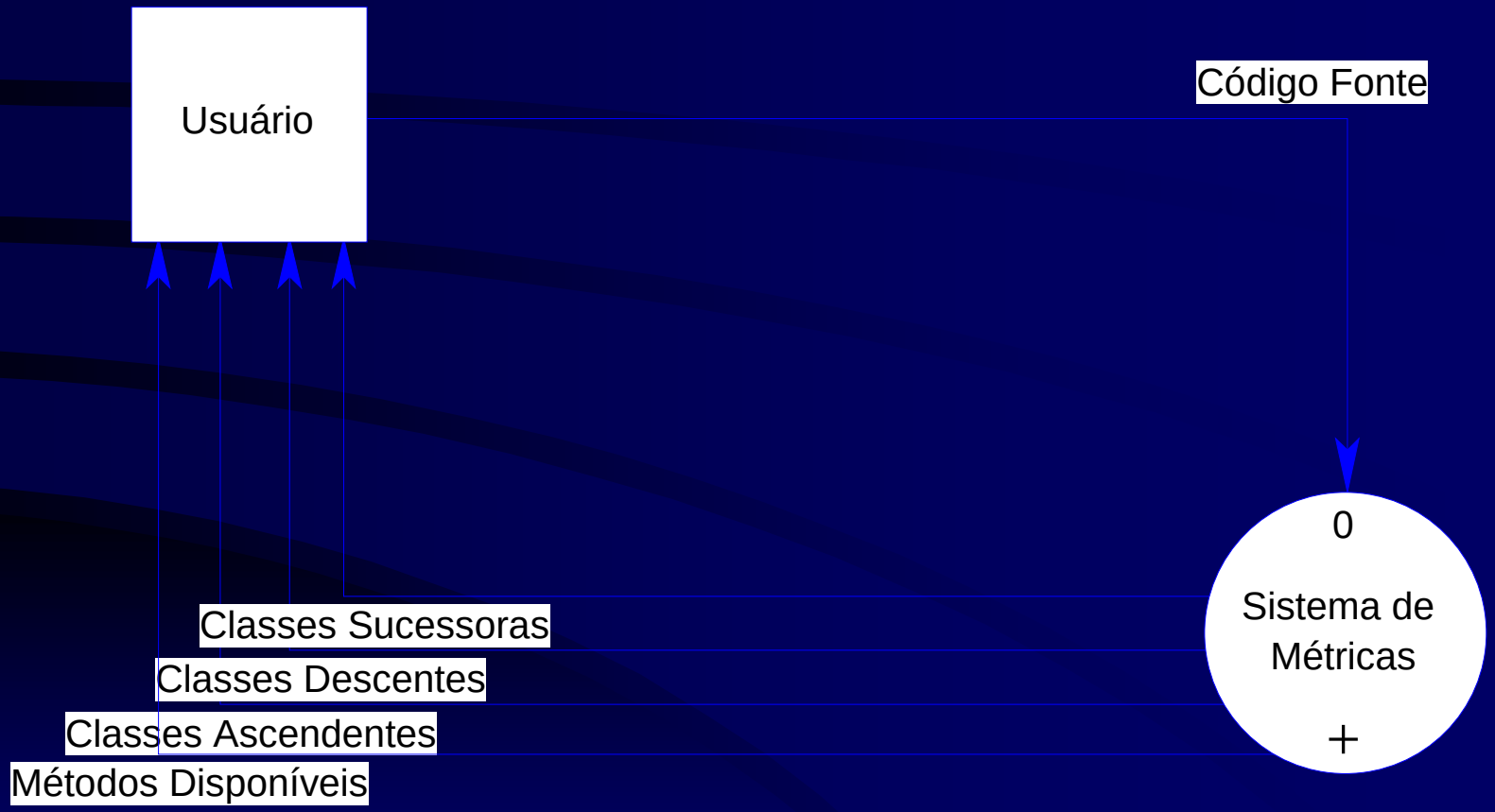
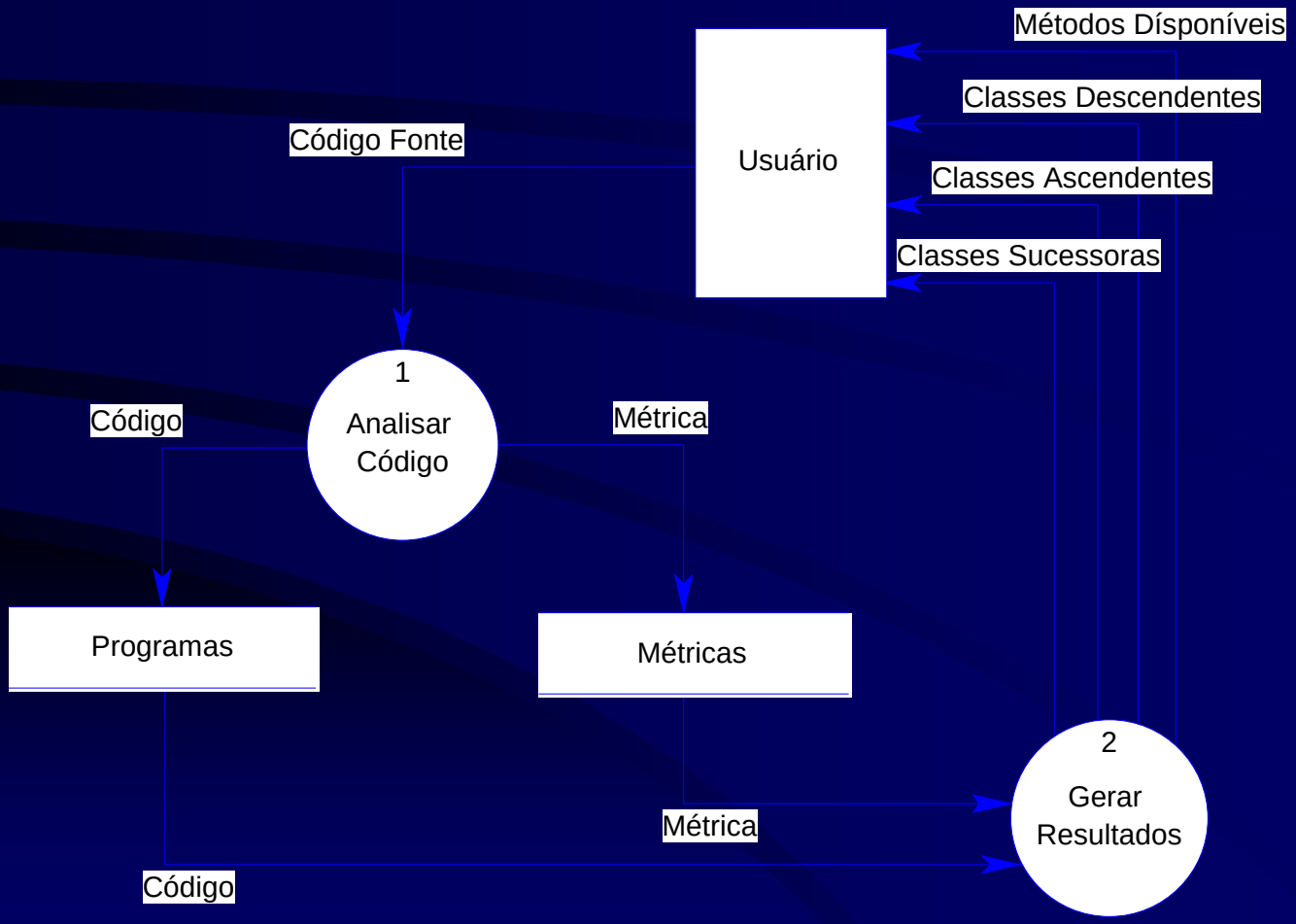
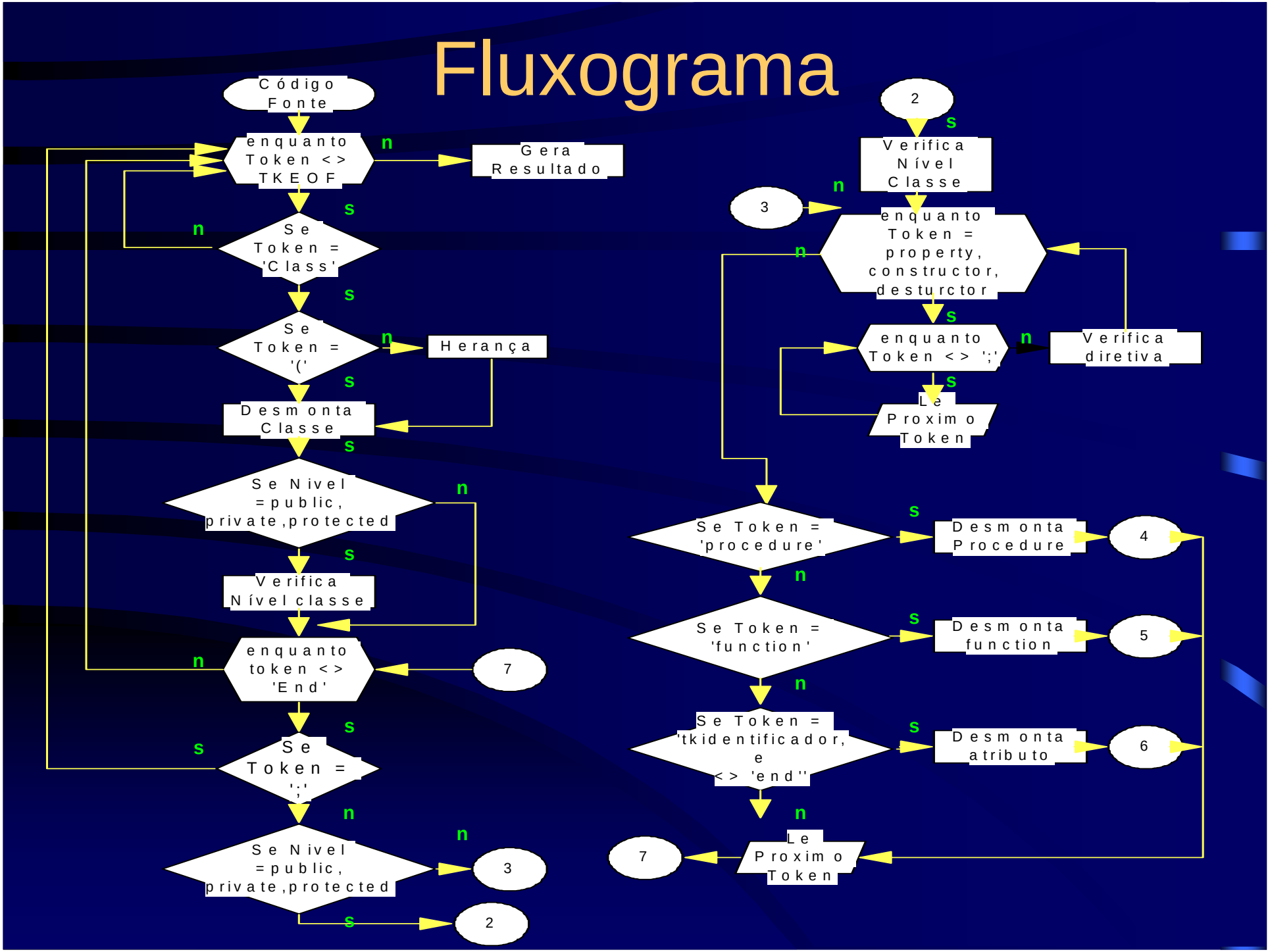


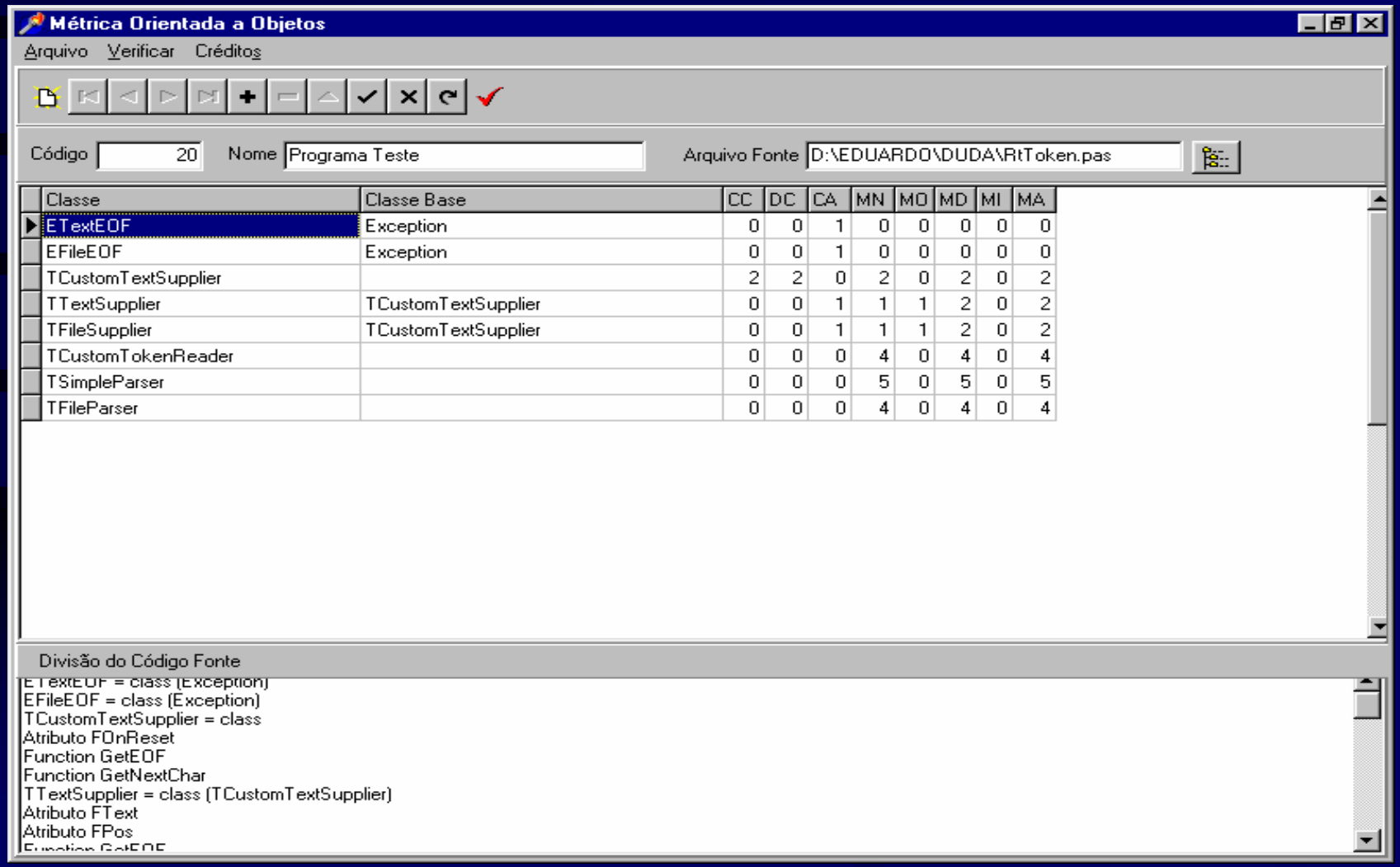
Diagrama de Fluxo de Dados



Fluxograma


Código Fonte
enquanto Token <> TKEOF
n
Gera Resultado
s
Se Token = 'Class'
n
s
Se Token = '('
n
Herança
s
Desmonta Classe
s
Se Nivel =public, private,protected
n
s
Verifica Nível classe
enquanto token <> 'End'
n
7
s
Se Token = ';'
s
n
Se Nivel =public, private,protected
n
3
s
2
2
s
Verifica Nível Classe
3
n
enquanto Token = property, constructor, desturctor
n
s
enquanto Token <> ';'
n
Verifica diretiva
s
Le Proximo Token
Se Token = 'procedure'
s
Desmonta Procedure
4
n
Se Token = 'function'
s
Desmonta function
5
n
Se Token = 'tkidentificador, e <> 'end''
s
Desmonta atributo
6
n
7
Le Proximo Token

Visualização do Protótipo



Conclusão

- A utilização de métricas orientada a objetos é uma necessidade cada vez mais evidente;

- As métricas contribuem para maior qualidade no processo;

- A ferramenta desenvolvida poderá ser utilizada para auditar sistemas, verificando a relação dos métodos, atributos da classe com outras classes do sistema analisado.

Sugestões

- Disponibilizar um número maior de métricas ao protótipo implementado, fazer busca de herança em outras units do sistema avaliado e incluir relatórios.

FIM