Ferramenta de Análise Código Fonte em Delphi

Aluno - Roque César Possamai

Orientador - Everaldo Artur Grahl

Roteiro

- Introdução
- Fundamentação Teórica
  - Conceitos Básicos
  - Métricas de Software
  - Métricas de Software para Código Fonte
- Desenvolvimento do Trabalho
  - Especificação
  - Implementação
  - Operacionalidade
- Considerações Finais

Introdução

- Objetivos das organizações: Qualidade do produto de software
- Utilização de métricas de software para garantia da qualidade
- Ferramentas que auxiliem a obtenção/uso das métricas
- Vantagens da utilização de métricas de software para código-fonte

Fundamentação Teórica

Conceitos Básicos

Medição: processo de codificação, objetiva e empírica, de algumas propriedades de uma selecionada classe de entidades em um sistema de símbolos formais, tão bem quanto a descrição das mesmas (McDermic, John)**.**

Métrica: medição que caracteriza alguns aspectos de uma entidade, seja ela um produto, processo ou companhia (McDermic, John)**.**

Métrica de Software: método de determinar quantitativamente a extensão em que o processo e o produto têm certos atributos (Fernandes, Agnaldo)**.**

Fundamentação Teórica

Métricas de Software

- Surgiram em meados dos anos 70;
- Aperfeiçoamento do gerenciamento de projetos de software em meados de 80;
- Aperfeiçoamento da qualidade do software;
- Produtividade da equipe de desenvolvimento;
- Obtenção dos objetivos organizacionais;
- Conscientização dos benefícios da utilização de métricas.

Fundamentação Teórica

Métricas de Software para Código Fonte

Pontos Positivos

- Obtenção da complexidade do Código;
- Estimativas de construção do Software;
- Produtividade/eficiência da equipe de desenvolvimento

Pontos Negativos

- Dependência de linguagem
- Dependência de outras métricas para análise

Desenvolvimento do Trabalho

Objetivo do Protótipo: fornecer métricas de código-fonte para programas em Delphi;

Método de Desenvolvimento: Orientação a Objeto.

Abordagem da Especificação: UML (Rational Rose)

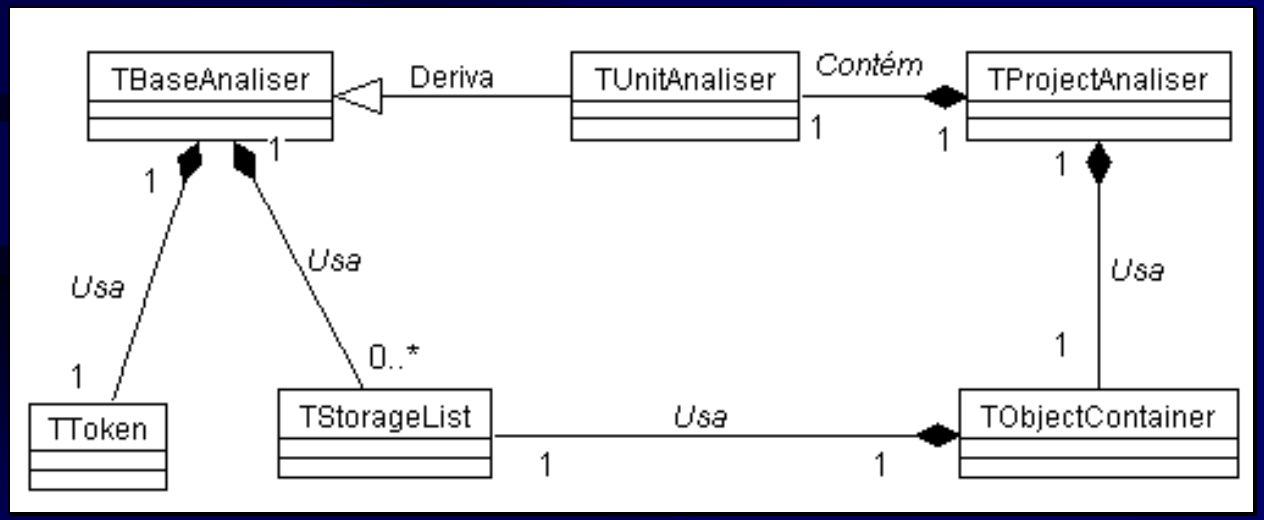
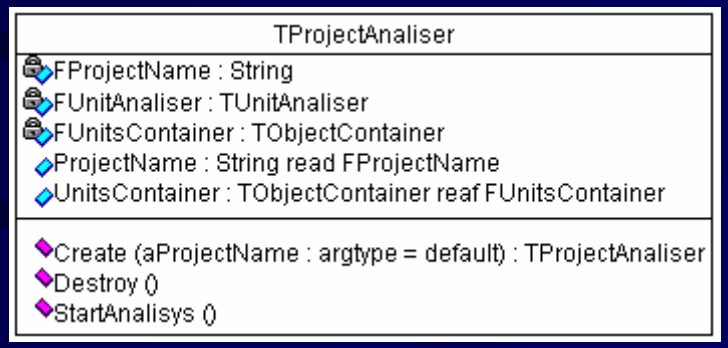


Diagrama de Classes do protótipo relativo a Análise de código

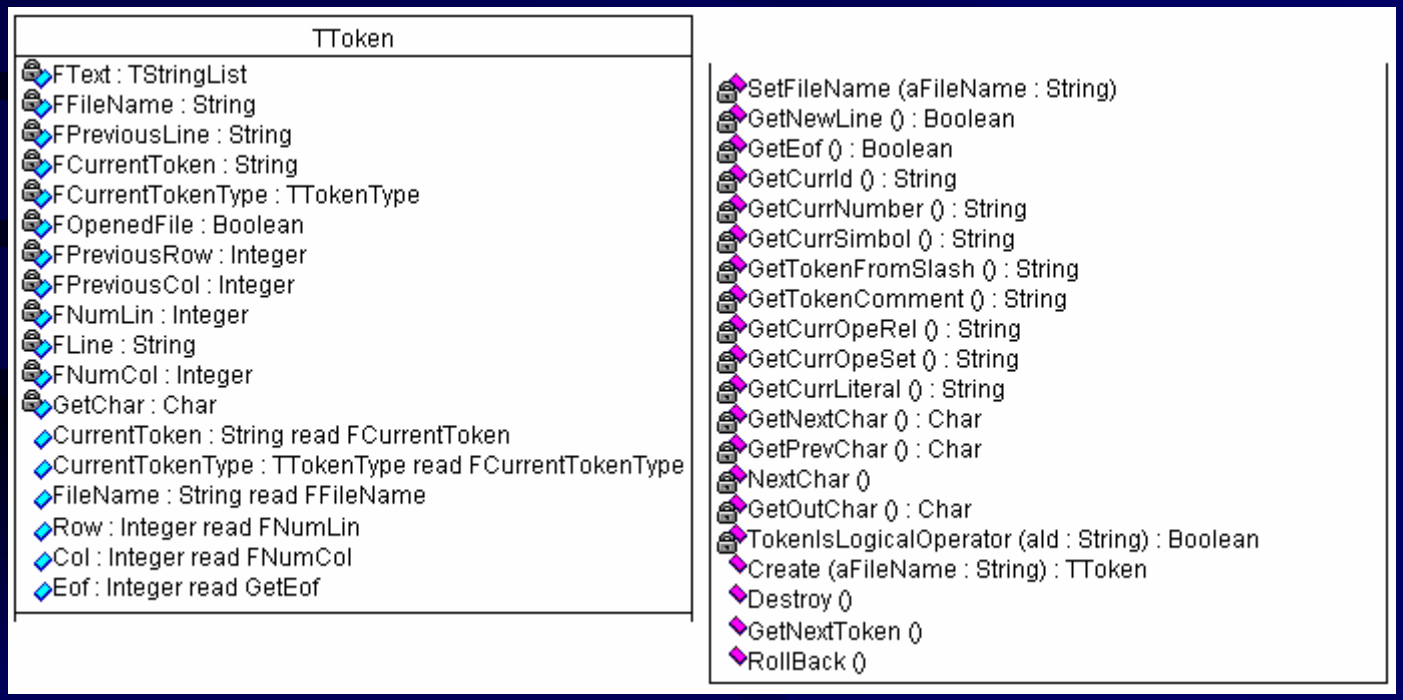
Desenvolvimento do Trabalho

Classe - TProjectAnaliser



Desenvolvimento do Trabalho

Classe - TToken

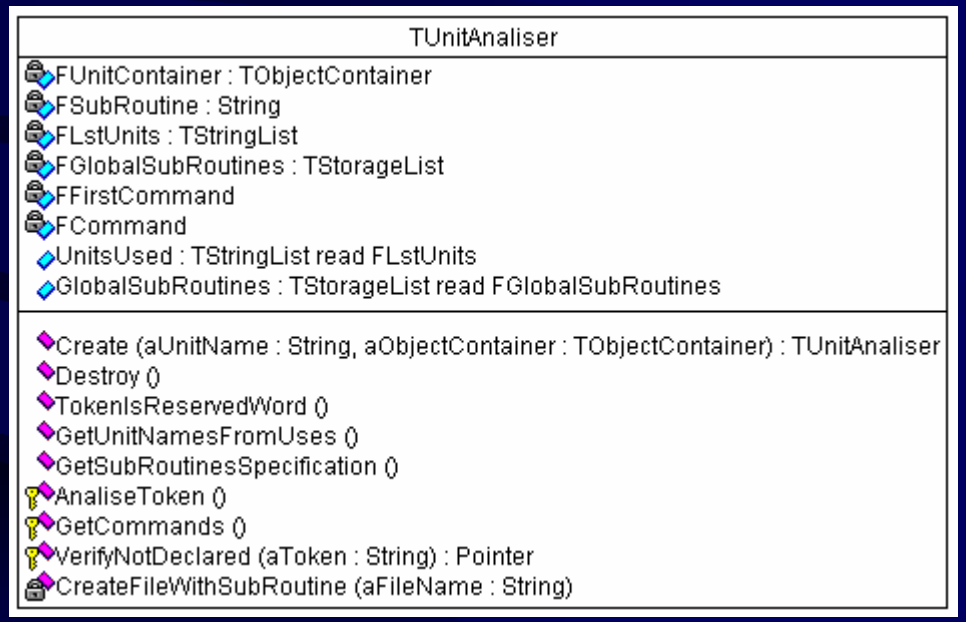


Desenvolvimento do Trabalho

Classe - TBaseAnaliser

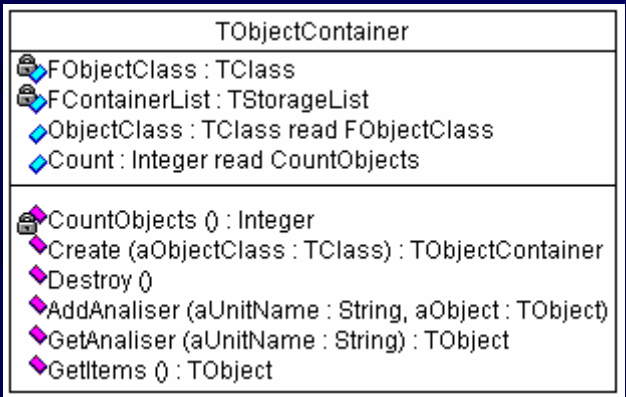
Desenvolvimento do Trabalho

Classe - TUnitAnaliser



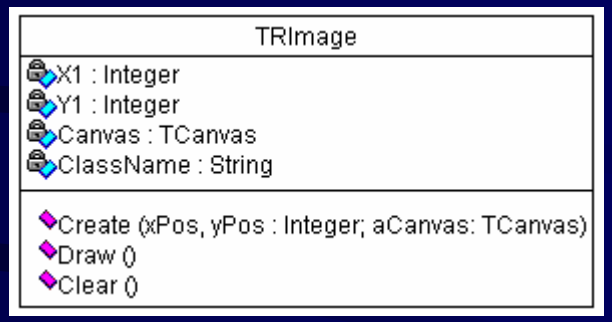
Desenvolvimento do Trabalho

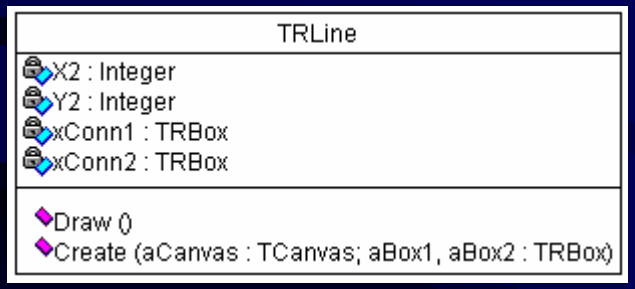
Classe - TObjectContainer

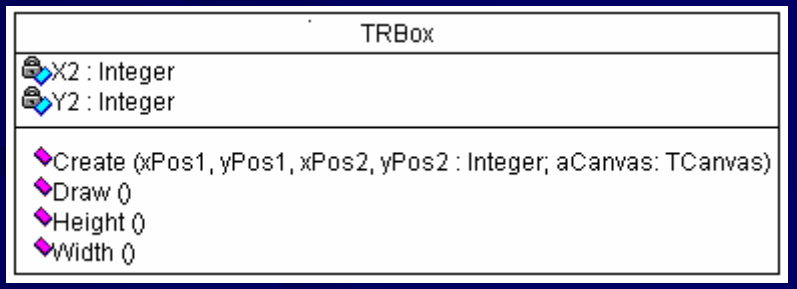


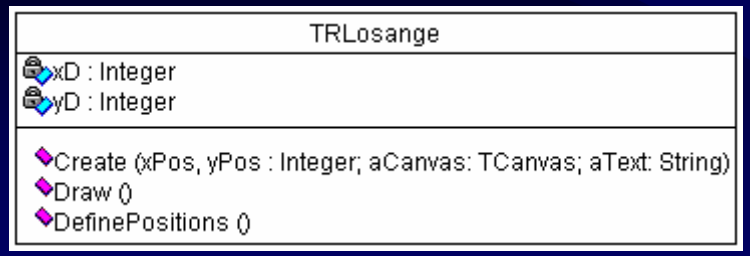
Desenvolvimento do Trabalho

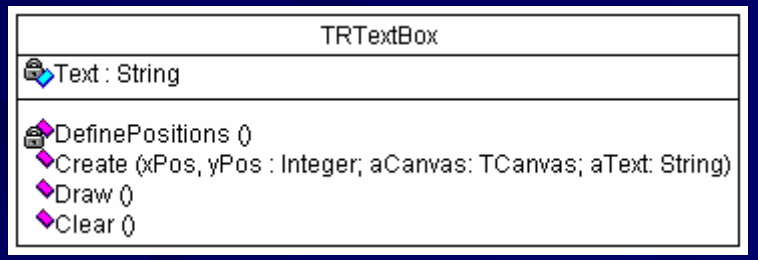
Classes para Desenho











Desenvolvimento do Trabalho

Forma de Trabalho do Analisador

⇒ Instancia-se a classe *TProjectAnaliser*, passando como parâmetro o nome do projeto selecionado. Ao instanciar-se a classe *TProjectAnaliser*, a mesma cria uma instância das classes *TUnitAnaliser* e *TObjectContainer*. Durante a criação do objeto *TUnitAnaliser*, o mesmo instancia a classe *TToken* para a extração dos tokens do projeto selecionado;

⇒ Inicia-se a análise do projeto através do método *StartAnalisys*, que efetua a análise do programa-fonte associado ao projeto;

Desenvolvimento do Trabalho

⇒ *TUnitAnaliser.StartAnalisys* inicia o processamento do programa-fonte, executando o método *AnaliseToken* para cada token extraído;

⇒ O método *AnaliseToken*, verifica se este token é uma palavra reservada ou um símbolo não definido. Se for uma palavra reservada, é chamado o método *TokenIsReservedWord*, caso contrário chama-se o método *TokenIsNotDefined*.

Desenvolvimento do Trabalho

⇒ O método *TokenIsReservedWord* verifica se a palavra reservada é uma das palavras que contenham instruções inerentes à análise como: *VAR, USES, CONST, TYPE*, etc.

⇒ Caso esta palavra reservada seja *VAR*, é chamado o método *GetUnitVars* que extrairá as variáveis adicionando-as na lista de variáveis da unit.

Desenvolvimento do Trabalho

⇒ Caso a palavra reservada seja *USES*, é chamado o método *GetUnitNamesFromUses*, que armazenará as units utilizadas em uma lista, efetuando automaticamente a sua análise. Deve-se dar especial atenção à este método, pois para cada unit encontrada, é iniciado uma análise, visto que as informações utilizadas na unit atual poderão estar presentes na unit indicada no *USES*. Deste modo, a análise desta unit somente prosseguirá ao fim da análise das units encontradas na cláusula *USES*.

⇒ Caso a palavra reservada seja *CONST*, é chamado o método *GetUnitConstants*, que extrairá todas as constantes declaradas por esta cláusula. O mesmo procedimento é adotado para a palavra *TYPE*.

Desenvolvimento do Trabalho

⇒Quando palavra reservada é *FUNCTION* ou *PROCEDURE*, o processamento é diferente: primeiro, certifica-se de que esta não é somente a declaração da subrotina. Caso não seja somente e declaração, todo o corpo da rotina é extraído e gravado em um arquivo temporário. Após isto, é instanciada uma classe *TUnitAnaliser* para verificar este arquivo temporário. Isto permite que a subrotina tenha uma análise adequada, da mesma forma como é feito com o código-fonte da unit.

⇒O método *TokenIsNotDefined*, que é chamado quando o identificador não é uma palavra reservada, verifica se este identificador é uma variável, constante, tipo, ou subrotina que já tenha sido analisada nesta unit ou nas units as quais tenha feito uso (*USES*). Caso o identificador não seja reconhecido o mesmo é armazenado na lista de palavras não reconhecidas do analisador.

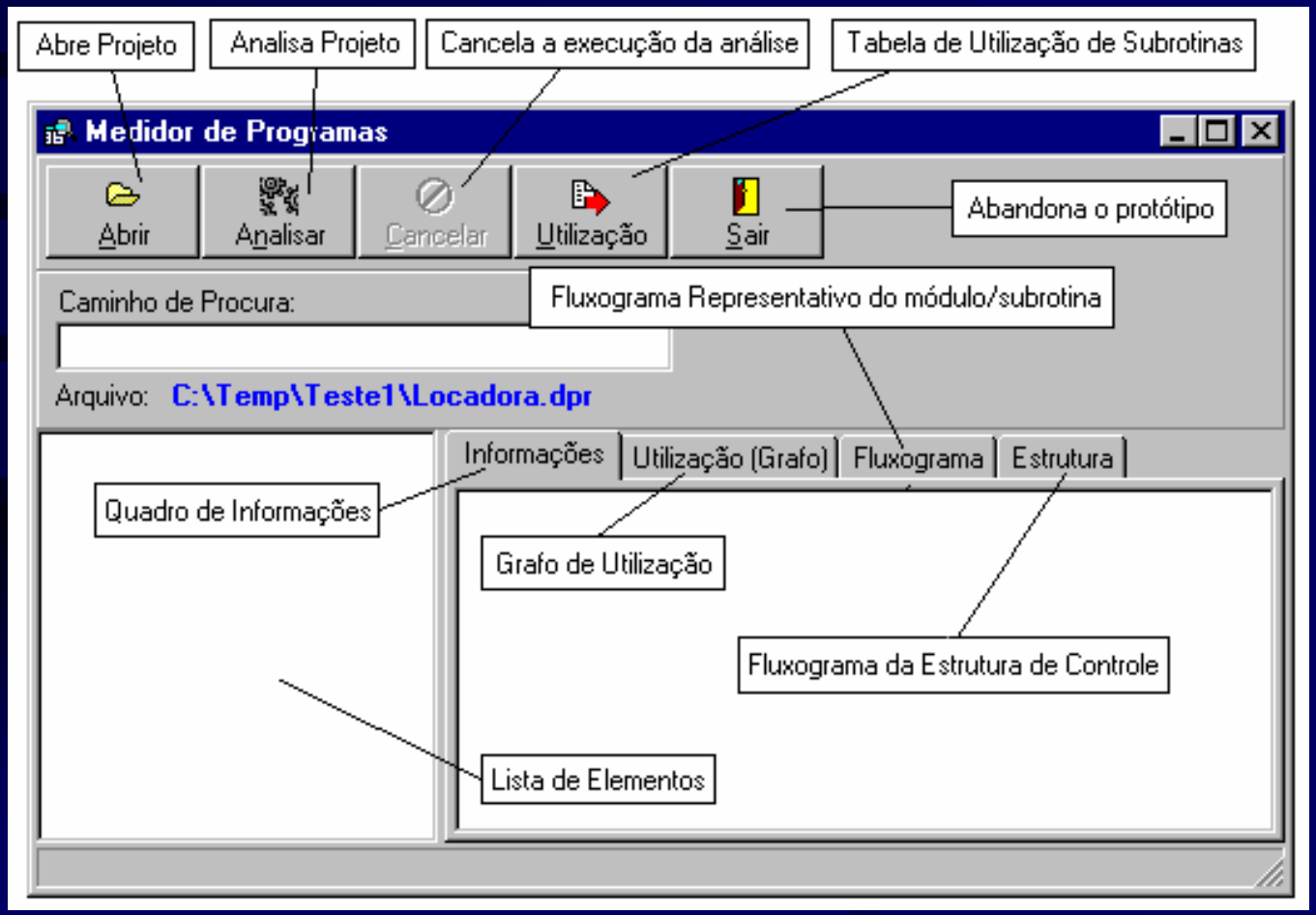
Desenvolvimento do Trabalho

⇒ O método *TokenIsNotDefined*, procura em todas as listas de identificadores para encontrar o tipo do mesmo. Quando encontrado, incrementa o contador de utilização deste identificador. Caso este identificador seja uma subrotina, adiciona o local ao qual a mesma está sendo referenciada na sua *UseList* (lista de usos).

⇒ Esta *UseList* é utilizada para a formação do grafo de utilização da subrotina.

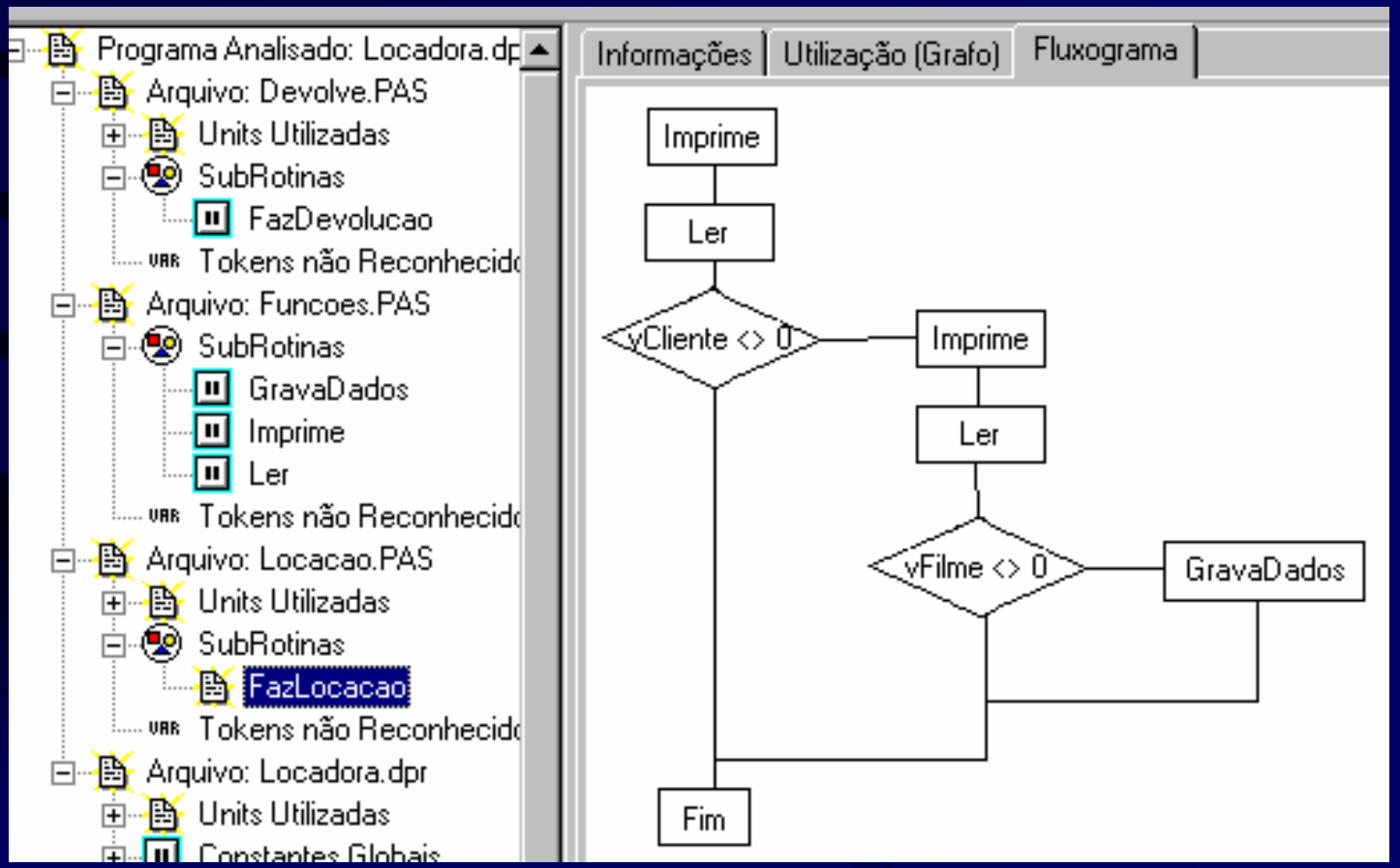
Desenvolvimento do Trabalho

Operacionalidade do Protótipo: Tela Principal



Desenvolvimento do Trabalho

Operacionalidade do Protótipo: Fluxograma



Desenvolvimento do Trabalho

Operacionalidade do Protótipo: Grafo de Utilização

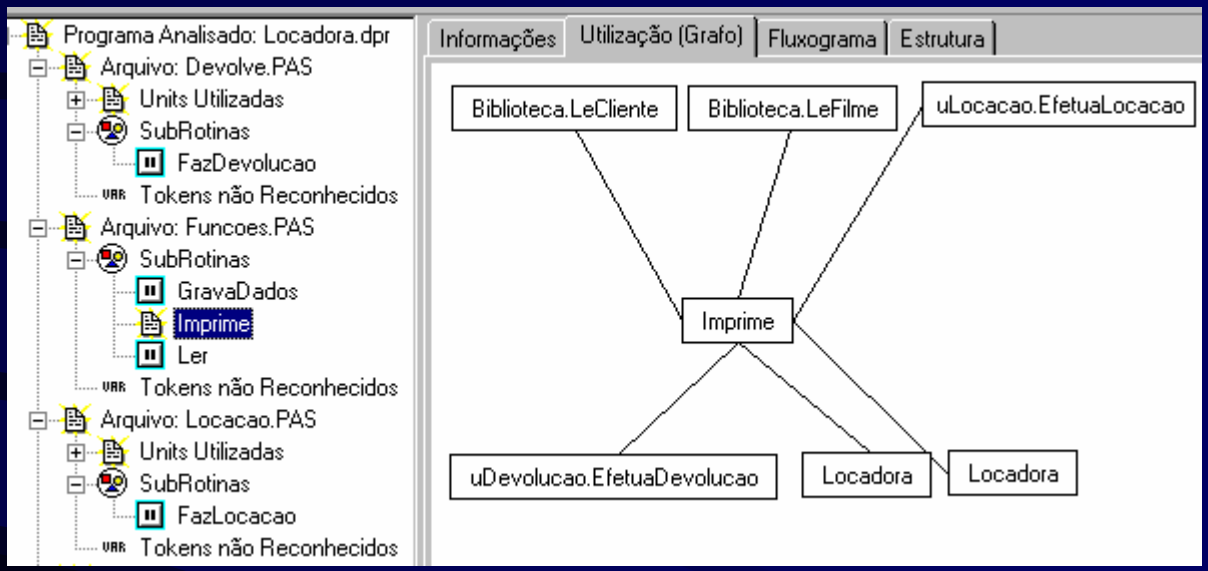
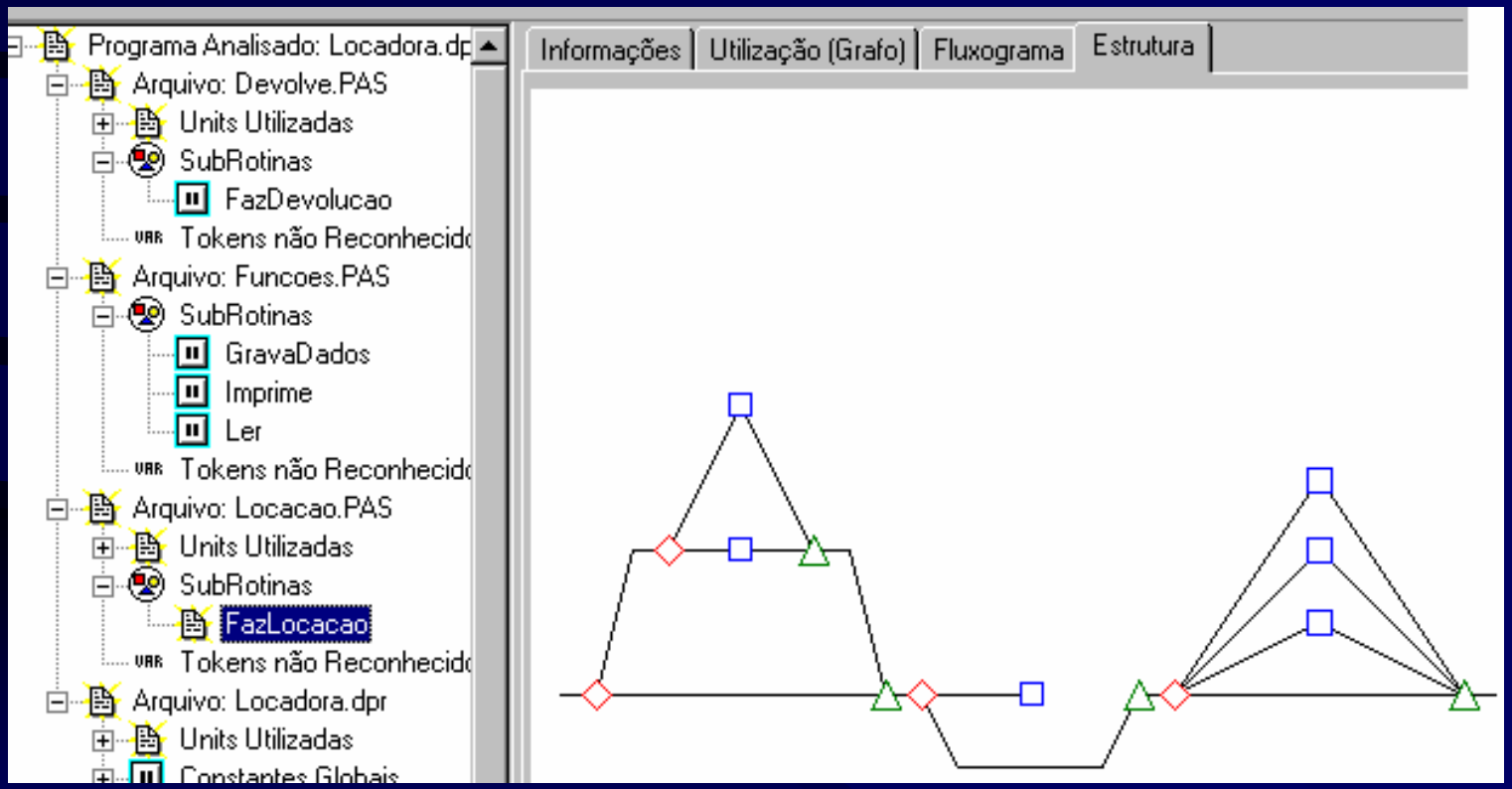


Tabela de Utilização

Rotinas/Utilização	FazDevolucao	GravaDados	Imprime	Ler	FazLocacao
FazDevolucao		X	X	X	
GravaDados					
Imprime					
Ler					
FazLocacao		X	X	X	

Desenvolvimento do Trabalho

Operacionalidade do Protótipo: Estruturas de Controle



Considerações Finais

Vantagens da Utilização da Ferramenta

Resultados auxiliam no entendimento do sistema, podendo ser utilizada no período de aprendizagem do mesmo.

Aplicação automática das métricas de código fonte para programas em Delphi.

Empresas precisam utilizar Métricas de Software para melhoramentos na qualidade de seu Software.

Considerações Finais

Extensões/Sugestões

- **Utilização de um Analisador Sintático para efetuar o reconhecimento do código fonte;**
- **Definição de listas de subrotinas próprias do Delphi;**
- **Inclusão de outras métricas de sistema, como FPA, COCOMO;**
- **Inclusão de mais resultados como relatórios, gráficos;**
- **Manutenção de uma base de dados auxiliando na verificação do andamento do sistema em um determinado período.**