RELATÓRIO FERRAMENTA LIZARD
Calculadora de Imposto de Renda Retido na Fonte (IRRF)

Alunos: Gabriel Edmundo Rocha - 2020054412
Selene Melo Andrade - 2019054986

1 - Introdução

O objetivo deste relatório é identificar e analisar a função mais complexa obtida ao executar a ferramenta Lizard no sistema desenvolvido para o trabalho prático da disciplina de Engenharia de Software II.

A ferramenta realiza análise de complexidade de cada função definida no programa. Ao rodá-lo no diretório onde o sistema está definido, ela busca por métodos ou funções que fujam dos valores limites estabelecidos por métricas definidas, como a quantidade de caminhos de execução independentes (CCN). Quanto maior for esse valor, maior a dificuldade de se entender, modificar e, consequentemente, testar o código fonte. Com essa análise, conseguimos identificar as funções que precisavam ser refatoradas, a fim de promover maior eficiência e testabilidade do código.

2- Relatório Lizard

Ao executar a ferramenta Lizard dentro do diretório do projeto, obtivemos o seguinte relatório:

```
Selene-MacBook-Pro:testetp selene$ lizard ClienteMultiUnitario.py InterfaceVisual.py LogicaCalculadora.py RelatorioPDF.py Grafico.py
================================================
  NLOC    CCN   token  PARAM  length  location
------------------------------------------------
      43      2    605      1      70 __init__@9-78@ClienteMultiUnitario.py
      40      6    861      1      55 criar_graficos_e_relatorios@81-135@ClienteMultiUnitario.py
     120     18   1117      0     155 interface_visual@122-276@InterfaceVisual.py
      19      1    109     13      23 __init__@15-37@LogicaCalculadora.py
      32     11    452      1      52 formulas_entrada@40-91@LogicaCalculadora.py
       2      1     43      1       3 enviardadosparagrafico@96-98@LogicaCalculadora.py
       2      1     33      1       3 enviardadospararelatorio@100-102@LogicaCalculadora.py
      13      1     68     10      17 __init__@7-23@RelatorioPDF.py
      34      1    373      1      42 gerarrelatorio@25-66@RelatorioPDF.py
      13      1     70     10      17 __init__@8-24@Grafico.py
       3      2     70      2       3 adddin@26-28@Grafico.py
       3      2     78      3       3 adddin2@30-32@Grafico.py
      20      1    248      1      31 criargrafico1@34-64@Grafico.py
      20      1    248      1      31 criargrafico2@66-96@Grafico.py
      20      1    248      1      31 criargrafico3@98-128@Grafico.py
5 file analyzed.
==============================================================
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt    file
--------------------------------------------------------------
     88      41.5     4.0      733.0         2     ClienteMultiUnitario.py
    232     120.0    18.0     1117.0         1     InterfaceVisual.py
     61      13.8     3.5      159.2         4     LogicaCalculadora.py
     51      23.5     1.0      220.5         2     RelatorioPDF.py
     84      13.2     1.3      160.3         6     Grafico.py

!!!! Warnings (cyclomatic_complexity > 15 or length > 1000 or nloc > 1000000 or parameter_count > 100) !!!!
================================================
  NLOC    CCN   token  PARAM  length  location
------------------------------------------------
     120     18   1117      0     155 interface_visual@122-276@InterfaceVisual.py
================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
--------------------------------------------------------------------------------
       516      25.6     3.3      308.2       15            1      0.07    0.31
```

Imagem 1 - Print do relatório Lizard após ser executado no diretório raiz

Dele, é possível extrair algumas informações importantes para a análise de complexidade do sistema:

NLOC = Número de linhas não comentadas da função
CCN (cyclomatic complexity) = Mede a complexidade de cada função analisando o número de caminhos independentes
PARAM = Número de parâmetros utilizados
length = Comprimento da função (NLOC + linhas comentadas)
location = Indica a função analisada e a localização dela no arquivo
Total nloc = Número total de linhas (desconsiderando comentadas) do programa
Fun Cnt = Número de funções no arquivo
Warning cnt = Número de funções que excedem o limite de complexidade

As métricas limítrofes estabelecidas pela ferramenta são mostradas abaixo:

```
(cyclomatic_complexity > 15 or length > 1000 or nloc > 1000000 or parameter_count > 100)
```

Imagem 2 - Métricas estabelecidas no relatório Lizard

A mensagem de aviso (Warnings), indica que existe uma função no arquivo que excede o limite de complexidade estabelecido pela ferramenta Lizard. Avaliando os valores de CCN gerados pelo relatório, notamos que uma função possui valor 18, ultrapassando o valor limite (CCN > 15). Tal função está indicada no parâmetro “location” do relatório, sendo ela interface_visual(), localizada no arquivo InterfaceVisual.py.

Também é possível notar que existe um função - formulas_entrada() - que, embora não exceda o valor limite para CCN, possui um valor relativamente alto (CCN = 11) e deve ser avaliada com atenção.

3- Explicação da função mais complexa

Com base na análise realizada, pode-se concluir que a função mais complexa do programa é interface_visual(), localizada no arquivo InterfaceVisual.py. Devido ao seu alto valor de complexidade ciclomática e os demais valores obtidos, pode ser que essa função seja difícil de manter ou modificar, sendo uma boa candidata a refatoração.

A função é responsável pela criação da interface gráfica com o usuário, permitindo que ele insira os valores que serão utilizados para realizar os cálculos do sistema e visualize os resultados em forma de gráficos ou de um relatório. O código está organizado em um loop “while true” que mantém a janela da interface aberta e reage a diversos eventos gerados pela interação do usuário.

A lógica da função está mostrada abaixo em pseudo-código:

```
Função interface_visual:

Enquanto Verdadeiro:

          Ler evento e valores da interface gráfica

          Se o evento for fechamento da janela:
          Encerrar o laço

          Se o evento for 'Gráfico':
          Tornar a coluna do gráfico visível
          Tornar a coluna do relatório invisível

          Se o evento for 'Relatório':
          Tornar a coluna do relatório visível
          Tornar a coluna do gráfico invisível

          Se o evento for 'Adicionar nome':
          Armazenar o nome do contribuinte

          Se o evento for 'Adicionar CPF':
          Armazenar o CPF do contribuinte

          Se o evento for 'Upar dados do salário':
          Capturar e armazenar os dados do salário e bônus

          Se o evento for 'Atualizar dados do salário':
          Atualizar os dados do salário e bônus

          Se o evento for 'Upar período de cálculo':
          Definir o período de cálculo dos dados

          Se o evento for 'Atualizar período de cálculo':
          Atualizar o período de cálculo dos dados

          Se o evento for 'Upar descontos cabíveis no IRRF':
          Definir os descontos e outras variáveis relacionadas ao IRRF
          Calcular o IRRF e preparar dados para o gráfico
          Criar o gráfico com os dados calculados

          Se o evento for 'Atualizar descontos cabíveis no IRRF':
          Atualizar os descontos e outras variáveis relacionadas ao IRRF
          Calcular o IRRF e preparar dados para o relatório
          Gerar o relatório com os dados calculados

Fechar a interface gráfica

Laço Principal:

Enquanto Verdadeiro:

          Ler evento e valores da janela principal

          Se o evento for fechamento da janela:
          Encerrar o laço

          Se o evento for 'Entrar com dados de múltiplas pessoas através de uma tabela':
          Executar funções para processar dados de múltiplas pessoas
          Criar gráficos e relatórios baseados nesses dados
          Fechar a janela principal

          Se o evento for 'Entrar com seus dados via interface visual':
          Chamar a função interface_visual
```

Imagem 3 - Pseudocódigo da função interface_visual()

Aspectos importantes do funcionamento da função:

- **Alternância de visibilidade:** Eventos 'Gráfico' e 'Relatório' alternam entre duas seções da interface, tornando uma visível e a outra invisível. Isso muda o foco da interface entre visualização de gráficos e relatórios.

- **Entrada de dados:** Eventos como 'Adicionar nome' e 'Adicionar CPF' capturam dados de entrada do usuário e os armazenam em variáveis como nome_contribuinte e cpf_contribuinte.

- **Atualização de dados salariais:** Eventos como 'Upar dados do salário' e 'Atualizar dados do salário' capturam e atualizam as informações sobre o salário e bônus do contribuinte.

- **Definição do período de cálculo:** Eventos como 'Upar período de cálculo' e 'Atualizar período de cálculo' são usados para definir ou atualizar o período de tempo para o qual os cálculos serão realizados.

- **Cálculo de IRRF:** Eventos 'Upar descontos cabíveis no IRRF' e 'Atualizar descontos cabíveis no IRRF' iniciam o processo de cálculo do IRRF com os dados fornecidos. Uma instância da classe LogicaCalculadora é criada, para processar as informações inseridas e calcular o salário líquido, o IRRF recolhido e outros valores.

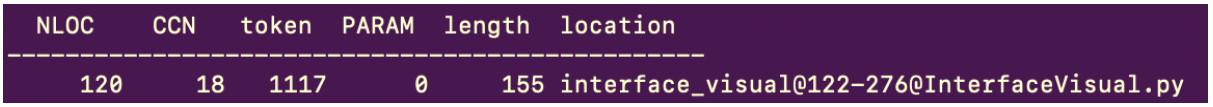

Imagem 4 - Relatório Lizard específico para a função interface_visual

A função interface_visual() apresentou alta complexidade ciclomática (CCN) por possuir muitos caminhos independentes que podem ser seguidos durante sua execução. Isso é devido a grande quantidade de declarações condicionais (if, elif, else). Quanto mais caminhos, mais cenários de teste são necessários para cobrir completamente a função em testes unitários. A grande contagem de tokens e o número elevado de linhas de código (NLOC), reforçam a noção de que a função realiza uma ampla gama de tarefas, potencialmente indo além de uma responsabilidade única.

A falta de parâmetros sugere que a função pode depender de variáveis globais ou de estado, o que pode acarretar em um acoplamento mais forte e menor reusabilidade do código. Isso porque a função interface_visual() é altamente dependente de outras partes do código, como a classe “LogicaCalculadora”, “Grafico” e “RelatorioPDF”. Além disso, a grande quantidade de código em um único bloco dificulta a manutenção, uma vez que os desenvolvedores precisam compreender uma grande quantidade de lógica para fazer modificações ou correções.

Para melhorar o design e manutenção do código, poderia ser considerada a refatoração da função interface_visual(), talvez dividindo-a em funções menores, cada uma com uma única responsabilidade. Isso ajudaria a reduzir a complexidade, facilitaria o teste e melhoraria a legibilidade.

4 - Rodando o Lizard após refatoração

Após a refatoração do código, obtivemos o seguinte relatório:

```
Selene-MacBook-Pro:teste2 selene$ lizard ClienteMultiUnitario.py InterfaceVisual.py LogicaCalculadora.py RelatorioPDF.py Grafico.py
================================================
  NLOC    CCN   token  PARAM  length  location
------------------------------------------------
      33     13    493      1      43 __init__@9-51@ClienteMultiUnitario.py
      17      2    279      1      22 prencher_variaveis_multiplas@53-74@ClienteMultiUnitario.py
      35      4    606      1      48 criar_graficos_e_relatorios@77-124@ClienteMultiUnitario.py
      87     13    681      0     117 interface_visual@122-238@InterfaceVisual.py
      24      1    142     13      28 __init__@4-31@LogicaCalculadora.py
       3      1     31      1       5 calculo_bonus_e_salario_bruto@33-37@LogicaCalculadora.py
      14      5    203      1      22 calculo_deducoes_e_salario_base@39-60@LogicaCalculadora.py
      13      6    199      1      20 calculo_irrf_recolhido_salario_liquido_aliquota@62-81@LogicaCalculadora.py
       5      2     54      1       8 definir_anos_vigencia@83-90@LogicaCalculadora.py
      35      4    467      7      44 preparar_graficos@92-135@LogicaCalculadora.py
      16      3    295      6      21 preparar_relatorio@137-157@LogicaCalculadora.py
       2      1     43      1       3 enviardadosparagrafico@159-161@LogicaCalculadora.py
       2      1     33      1       3 enviardadosspararelatorio@163-165@LogicaCalculadora.py
      13      1     68     10      17 __init__@7-23@RelatorioPDF.py
      34      1    373      1      42 gerarrelatorio@25-66@RelatorioPDF.py
      13      1     70     10      17 __init__@8-24@Grafico.py
       3      2     70      2       3 formatar_texto_da_barra_de_baixo_do_grafico@26-28@Grafico.py
       3      2     78      3       3 formatar_texto_da_barra_de_cima_do_grafico@30-32@Grafico.py
      20      1    248      1      31 criargrafico1@34-64@Grafico.py
      20      1    248      1      31 criargrafico2@66-96@Grafico.py
      20      1    248      1      31 criargrafico3@98-128@Grafico.py
5 file analyzed.
==============================================================
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt    file
--------------------------------------------------------------
     90      28.3     6.3      459.3         3     ClienteMultiUnitario.py
    199      87.0    13.0      681.0         1     InterfaceVisual.py
    115      12.7     2.7      163.0         9     LogicaCalculadora.py
     51      23.5     1.0      220.5         2     RelatorioPDF.py
     84      13.2     1.3      160.3         6     Grafico.py

===============================================================================================================
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or nloc > 1000000 or parameter_count > 100)
==========================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
------------------------------------------------------------------------------------------
       539      19.6     3.1      234.7       21            0      0.00    0.00
```

Imagem 5 - Print do terminal após executar o Lizard no código refatorado

É possível notar que a complexidade ciclomática da função interface_visual() foi reduzida de CCN = 18 para CCN = 13. Como o atual valor não excede o limite de complexidade, não foram geradas mensagens de aviso. A mudança no valor de CCN é devida a uma das refatorações realizadas, que diminuiu as responsabilidades de interface_visual() ao aplicar um Move Method em dois métodos que, por realizarem operações lógicas-matemáticas, cabiam melhor na classe LogicaCalculadora.

Além disso, outra refatoração realizada foi a extração do método formulas_entrada(), que estava muito genérico, em outros métodos, mais focados e menores. Os novos métodos são de mais fácil compreensão e com nomes muito mais comunicativos. São eles:

- calculo_bonus_e_salario_bruto()
- calculo_deducoes_e_salario_base()
- calculo_irrf_recolhido_salario_liquido_aliquota()
- definir_anos_vigencia()

Após tais modificações, o código ficou mais compreensível, reutilizável e testável, melhorando sua eficiência como um todo.