VISUALIZAÇÃO DE SOFTWARE COMO SUPORTE AO DESENVOLVIMENTO CENTRADO EM MÉTRICAS ORIENTADAS A OBJETOS

Elidiane P. dos Santos e Sandro S. Andrade
Instituto Federal de Educação, Ciência e Tecnologia da Bahia- IFBa
Rua Emídio dos Santos, S/N
Barbalho, Salvador Bahia
{elidiane, sandroandrade}@ifba.edu.br

RESUMO

Este trabalho apresenta o projeto e implementação de um *plugin* para visualização de métricas em Sistemas Orientado a Objetos (OO), bem como a sua avaliação experimemental em atividades de refatoração e evolução. As métricas são calculadas pelo *plugin* através dos dados extraídos de códigos C++ utilizados no *Qt Creator* (IDE para o desenvolvimento de aplicativos *Qt*). A análise quantitativa é necessária para a avaliação do *software*, que poderá ser qualificado como bom ou ruim de acordo com os resultados. O estudo inclui alguns exemplos de sistemas que seguem o paradigma OO.

1. INTRODUÇÃO

As organizações adotam a Programação Orientada a Objetos (POO) como uma opção para a solução de alguns dos seus problemas, como o melhor gerenciamento da complexidade em projetos de *software* que tendem a ter uma grande complexidade. Essas organizações querem o máximo possível de qualidade em suas aplicações, produzindo artefatos que possam ser facilmente interpretados e com uma boa arquitetura.

A abordagem OO suporta muitas técnicas para a implementação de sistemas mais flexíveis e de fácil análise tais como: reusabilidade de código, herança e outros. Através dos benefícios disponibilizados pela orientação a objetos é possível tornar o sistema mais robusto e de fácil manutenção.

O desenvolvimento de *software* utilizando o paradigma OO surge como uma possibilidade para a melhoria da qualidade e produtividade. Este paradigma permite modelar o problema em termos de objetos capazes de diminuir a distância entre o problema do mundo real e sua abstração [12].

A realidade da informática atual aponta para o sério problema em relação à qualidade dos produtos desenvolvidos, onde a única preocupação das equipes de desenvolvimento é o cumprimento dos prazos. É com esta preocupação que os programadores entregam ao cliente apenas as funções básicas do sistema, incluindo vários defeitos. Estas aplicações são novamente avaliadas pelos desenvolvedores, muitas vezes um produto aparentemente já concluído passa a ser cobrado adicionalmente do cliente levando-o ao prejuízo [4].

As Métricas de *Software* desempenham um papel fundamental para a programação OO, possibilitando um maior controle do desenvolvimento dos sistemas e auxiliando o processo de implementação. Esta metodologia é importante para auxiliar o arquiteto de *software* a tomar decisões corretas e estabelecer uma visão geral da qualidade planejada para o sistema.

Este estudo pretende verificar a utilização das Métricas de *Software* orientado a objetos, trazendo uma visão geral das principais utilizadas, suas características e suas vantagens na programação. O objetivo é comprovar a hipótese definida: *"a utilizacao de tecnicas para visualizacao de metricas OO conduz a operacoes de refatoracoes mais e cientes e rapidas"*.

O *Qt Creator*, IDE para o desenvolvimento de aplicativos *Qt* em C++ [7]. possui uma carência na visualização da qualidade de um código OO. Este recurso é fundamental para que haja um bom gerenciamento da produtividade dos projetos de *software*, possibilitando a criação de sistemas confiáveis. À medida que se faz uma extração de dados do programa há um domínio maior sobre eles, eliminando os riscos que poderão ocorrer e permitindo ao desenvolvedor ter uma visão geral do sistema.

Com a intenção de suprir essa necessidade do *Qt Creator*, foram implementadas métricas OO que são utilizadas para a avaliação da qualidade do *software*. Essas métricas são necessárias e indispensáveis para examinar cuidadosamente a qualidade e o rendimento de um *software*. Através delas há uma compreensão maior sobre a complexidade do sistema, suas principais viabilidades e conhecimentos dos caminhos necessários para a resolução de problemas que poderão surgir.

O restante deste artigo está organizado como segue. A seção 2 descreve os conceitos gerais das métricas orientadas a ob-

jetos apresentando algumas delas, suas vantagens, desvantagens e utilidades e algumas ferramentas de métricas OO já existentes. A seção 3 contém uma breve explicação das ferramentas utilizadas: o *Qt* e o *Qt Creator*; e uma explicação detalhada sobre a ferramenta desenvolvida, utilizando um estudo de caso e análise dos resultados de suas métricas. A seção 4 apresenta a avaliação do experimento de validação do *software*. A seção 5 apresenta conclusão do trabalho.

2. VISUALIZAÇÃO DE SOFTWARE E MÉTRICAS OO

De acordo com [10] a Visualização de Software auxilia o programador a analisar e compreender a estrutura e o funcionamento de um programa, em um nível maior de abstração do que quando comparada a uma simples leitura do código-fonte. Um ambiente de visualização de software é uma ferramenta responsável pela apresentação visual do software de forma clara e atrativa.

A combinação de métricas e visualização oferece facilidades à compreensão e avaliação de softwares que ajudam o projetista a desenvolver estratégias mais eficientes para a atividade de manutenção, por exemplo. A visualização de Software é importante para a economia de tempo e dinheiro, melhor compreensão de software, aumento da produtividade e qualidade, gestão de complexidade e utilizada para encontrar erros.

Segundo [1], algumas métricas que podem ser utilizadas em sistemas orientados a objetos são: Contagem de Métodos, Número de Atributos de Instância, Profundidade da Árvore de Herança, Número de Filhos, Utilização Global, Quantidade de Atributos de Classe, Tamanho do Método, Resposta do Método e Comentários por Método.

Segundo [14], as métricas OO estão divididas em três categorias: Métricas de Análise, Métricas de Projeto e Métricas de Construção. Tais métricas podem ajudar no esforço de desenvolvimento, desta forma detectando os problemas que poderão ocorrer. De acordo com [1] as métricas de projeto e de construção, além de medir os aspectos importantes do *software* são bem fáceis de automatizar e coletar. As categorias das métricas OO estão representadas em forma de diagrama na Figura 1 para uma melhor compreensão.

Segundo [3], as métricas OO podem ser divididas segundo *Lorenz e Kidd* e segundo *Chidamber e Kemerer*, conforme segue na Figura 2. *Chidamber e Kemerer* definiram seis métricas para o cálculo de complexidade de sistemas OO e são conhecidas como as métricas de *CK*. Elas são referências para análise quantitativa e têm o objetivo de concentrar nos testes de classes, que possuem uma grande possibilidade de apresentar um maior número de defeitos. As métricas de *Lorenz e Kidd* têm como base o cálculo quantitativo de alguns aspectos fundamentais da OO, como os atributos e serviços, herança, coesão e acoplamento.

Este estudo focaliza nas métricas de construção, implementando aquelas que além de avaliar a qualidade do projeto, podem melhorar a qualidade do código. Conceitos gerais sobre as outras métricas são também apresentados.

2.0.1 Métricas de Análise

De acordo com [14] *apud* [1] as Métricas de Análise são usadas para a medição da qualidade dos esforços da análise e podem medir a porcentagem das classes mais importantes do sistema que devem ficar entre 30 e 50 %. O valor determina se as classes-chaves são suficientes. Caso a porcentagem seja baixa haverá a necessidade de criar outras.

As métricas de análise podem medir também os números de cenários de utilização (descrições dos requisitos), estes números de cenários identificados podem ser usados para verificar se a análise está completa. Aplicações com poucos meses de desenvolvimento possuem entre cinco e dez cenários de utilização, as de tamanho médio com menos de um ano de desenvolvimento possuem entre vinte e trinta cenários e as aplicações de tamanho grande têm quarenta ou mais cenários.

2.0.2 Métricas de Projeto

Segundo [4], as métricas de projeto ajudam a estabelecer comparações entre vários sistemas e criar a base de futuras recomendações para um novo projeto que poderá eventualmente evoluir para normas em nível organizacional.

WMC, DIT, NOC, CBO, LCOM e RFC são as métricas que segundo [3] foram propostas por *Chidamber* e *Kemerer(1994)* e são conhecidas como CK.

a) Contagem de Metodos: a técnica de contagem de métodos é uma métrica muito fácil de coletar [1]. As classes com um número alto de métodos tendem a ser específicas para os seus objetivos e geralmente as com uma quantidade menor tendem a ser mais reutilizáveis. Para [13], o número de métodos de uma classe não deveria passar de 20, contudo aceita-se que esta quantidade chegue a 40.

b) WMC - Metodos Ponderados por Classes: para [13], a WMC *(Weigthed Methods Per Class)* é a medida da complexidade individual de uma classe. Esta métrica é responsável pela contagem dos métodos implementados em uma classe e a soma de suas complexidades é dada pela complexidade ciclomática (número de caminhos criados pelos fluxos de controle em um módulo de software). [5] afirma que quanto maior for a quantidade de métodos em uma classe maior será o potencial de impacto nos filhos.

c) RFC - Respostas de uma Classe: segundo [5], a RFC *(Response For a Class)* definida pelo número de métodos diferentes que podem ser chamados em resposta a uma mensagem para um objeto ou algum método da classe, incluindo os métodos que são acessados dentro da hierarquia da classe. Conclui-se então por [13] que quanto maior for o número de chamadas de métodos feitas por uma classe maior será a sua complexidade. O número aceitável de métodos para esta métrica seria 100, entretanto na maioria das classes este número é menor ou igual a 50.

d) DIT - Profundidade da Arvore de Heranca: segundo [3], DIT *(Depth Of Inheritance Tree)* é determinada pelo número máximo de superclasses posicionadas hierarquicamente acima da classe em foco. Segundo [14] *apud* [13], a profundidade da hierarquia da herança é o número máximo de passos da classe nó até a raiz da árvore e é medida

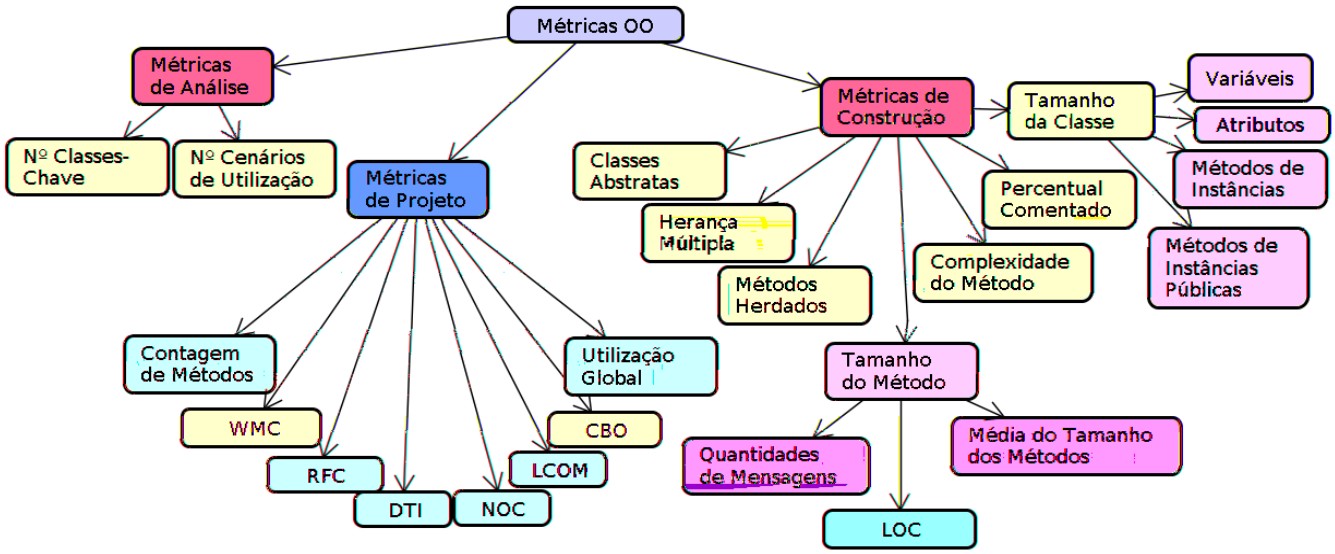

Figura 1: Tipos de Metricas OO (segundo Seibt)

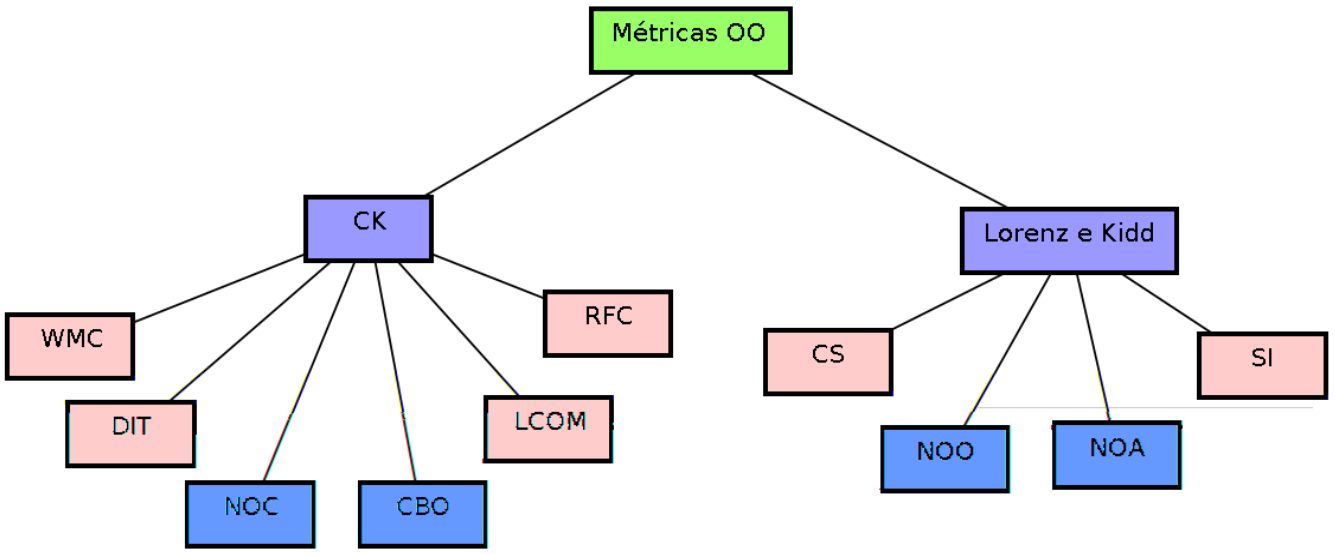

Figura 2: Tipos de Metricas OO (segundo Borges)

pelo número de classes ancestrais. Um DIT elevado significa dizer que muitas características das classes foram herdadas e existem muitas classes que fazem parte desta hierarquia e possuem características comum entre elas. As superclasses possuem um alto nível de abstração, sendo capazes de serem reutilizáveis.

Segundo [13], quando o DIT for zero indica uma classe raiz, um alto percentual de DIT entre dois e três indica um alto grau de reutilização, se a maioria dos ramos da árvore forem poucos profundos (DIT < 2) pode indicar que poucas vantagens do desenvolvimento OO foram exploradas. Quando uma árvore for muito profunda (DIT> 5) a complexidade do projeto irá aumentar e precisará de uma reavaliação.

e) NOC - Numero de Filhos: de acordo com [13], NOC *(Number Of Children)* consiste do número de subclasses imediatamente abaixo de uma classe na hierarquia, esta métrica é contrária a DIT. Um NOC alto indica um nível baixo de abstração, pois uma superclasse com uma quantidade alta de filhos tem a tendência de possuir poucas características em comum com todas as subclasses.

f) LCOM - Falta de Coesao: segundo [14] *apud* [13], LCOM *(Lack Of Cohesion)* é definida pelo número dos diferentes métodos em uma classe que referenciam uma determinada variável de instância. Segundo [14], uma alta coesão representa uma boa subdivisão das classes, tornando-a mais simples e com uma alta capacidade de reutilização, o que é uma das principais características de um *software* OO e garante que os métodos exerçam suas funções adequadamente. Uma coesão baixa torna a classe mais complexa, viabilizando a possibilidade de erros em sua utilização, pois os serviços ficam ligados pelos atributos indicando que foram mal projetados.

Uma das maneiras de medir uma coesão descrita por [13] é, para cada atributo de uma classe calcular o percentual dos métodos que o utilizam, obter a média destes percentuais e subtraí-lo de 100%. Percentuais com valores baixos representam uma maior coesão entre atributos e métodos na classe.

g) CBO - Acoplamento entre Objetos: para [13], CBO *(Coupling Between Object Classes)* é definido pelo número de classes acopladas a uma determinada classe. Duas classes são acopladas quando métodos declarados em uma delas utilizam métodos ou variáveis de instância de outra. As classes objetos podem ser acopladas de três maneiras diferentes: a) quando uma mensagem é passada entre os objetos. b) quando os métodos declarados em uma classe utilizam atributos ou métodos de outras classes. c) através da herança que introduz um significativo acoplamento entre as superclasses e suas subclasses.

Quanto maior a ligação entre as classes, menor a possibilidade de reutilização, pois a classe torna-se dependente de outras classes para cumprir suas obrigações. Um alto acoplamento indica uma baixa independência de classes, o que aumenta a complexidade do *software*.

2.0.3 Métricas de Construção

Para [1], as métricas além de medir a qualidade do projeto podem também ser aplicadas para melhorar a qualidade do código. Esta é a finalidade das métricas de construção.

a) Tamanho do Metodo: de acordo com [14] *apud* [13], a métrica do tamanho de um método pode ser obtida de várias maneiras, tais como a contagem de todas as linhas físicas de código, o número de comandos, o número de linhas em branco e o número de linha comentadas, dentre outras formas existentes. Abaixo segue uma dessas métricas:

Linhas de Codigo (LOC)- esta métrica conta o número de linhas físicas de código ativo que estão em um método. Para criar métodos fáceis de manter eles devem ser pequenos. Deve-se esperar 30 linhas para códigos C++ [1] .

b) Percentual Comentado: segundo [1], a quantidade de linhas comentadas por métodos é uma métrica muito útil para estimar a qualidade do código. Poucas linhas comentadas no código podem ajudar em sua compreensão, porém quando há um exagero indica tempo perdido para documentar tais métodos. O percentual de linhas de código é obtido com a divisão do total de linhas comentadas pelo total de linhas de código, o que para [13] este resultado deveria ficar entre 20 e 30% para facilitar a manutenção e reusabilidade do código.

c) Complexidade do Metodo: a métrica de complexidade por método é obtida com o número de complexidades dividido pelo número dos métodos. [9] utilizou alguns pesos para computar a complexidade do método, o valor esperado para esta métrica é 65.

1- Chamadas de API: 5,0

2- Atribuicoes: 0,5

3- Operadores Matematicos (C++): 2,0

4- Mensagens com Parâmetros (C++): 3,0

5- Expressoes Aninhadas: 0,5

6- Parâmetros: 0,3

7- Chamadas Primitivas: 7,0

8- Variaveis Temporarias: 0,5

9- Mensagens sem Parâmetros: 1,0

d) Tamanho da Classe: existem diversas formas de medir uma classe, abaixo segue algumas métricas:

Quantidade de Metodos em uma classe: esta métrica conta todos os métodos de uma classe. Segundo [9], o valor recomendado para esta métrica é de 40 para classes de interface com o usuário e 20 para as demais.

Quantidade de Atributos por Classe: ao contar a quantidade de atributos das classes é possível ter um indicador da qualidade do projeto [9]. Uma classe com um número alto de atributos indica que tem muitos

relacionamentos com outros objetos do sistema. As classes com mais de três ou quatro atributos mascaram o problema de acoplamento da aplicação [1]. Classes de interface com o usuário os números de atributos podem chegar a nove, pois estas classes necessitam de mais atributos para lidar com componentes de telas [9].

Media de Atributos por Metodos em uma Classe: é obtida pelo total de atributos dividido pelo total de métodos [9]. Muitos atributos indicam a possibilidade das classes estarem fazendo além do que deveriam.

Quantidade de Metodos de Classe em uma Classe: as próprias classes são objetos que podem fornecer serviços que são globais para as suas instâncias [9]. O número de métodos que são disponibilizados para a classe e não para suas instâncias afetam o seu tamanho, onde este número poderia ser menor em relação ao número de métodos de instância. Para [9] as classes raramente necessitam mais do que quatro métodos de classe.

Quantidade de Metodos Publicos em uma Classe: segundo [9] esta é uma boa medida da responsabilidade total da classe. Os métodos públicos são aqueles serviços que estão disponíveis como serviços para outras classes. Métodos públicos são indicadores do trabalho total feito por uma classe, uma vez que eles são serviços utilizados por outras classes.

2.0.4 Trabalhos Correlatos

Esta seção exibirá algumas ferramentas relacionadas a métricas OO como resultados de outros trabalhos relativos a este tema.

As ferramentas apresentadas são: *Visual Metrica [3], Ferramenta para calculo de metricas OO [14] e Metricas para Acoplamento e Coesao em Sistemas Orientados a Objetos [10].*

1. **Visual Metrica:** esta ferramenta desenvolvida por [3] é responsável por calcular algumas métricas de construção e de projeto, possibilitando a análise do código fonte OO em C# e Java. As informações para cálculo das métricas são coletadas através da extração das classes, métodos e de seus atributos através das análises dos arquivos de um projeto. Com a visual métrica o usuário poderá: selecionar arquivos de projeto que será analisado; selecionar arquivos individuais para uma análise específica; escolher as métricas que serão calculadas; gravar o resultado do cálculo; abrir um projeto gravado anteriormente; definir limites mínimo e máximo que uma determinada métrica poderá atingir; e por fim calcular métricas. A Figura 3 apresenta a utilização desta ferramenta.

2. **Ferramenta para Calculos das Metricas em *Software* Orientados a Objetos:** a ferramenta desenvolvida por [14] é um protótipo que fornece métricas pré-definidas a partir da análise de código fonte de programas codificados em *Delphi*. As informações para cálculo das métricas são obtidas através da extração das classes de seus métodos e de seus atributos através da análise das *units* de um projeto em *Delphi*. Após a

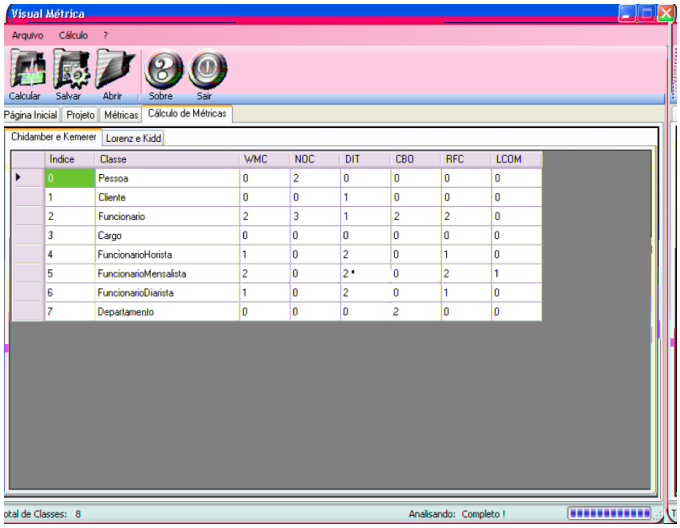

Figura 3: Visual Metrica [3]

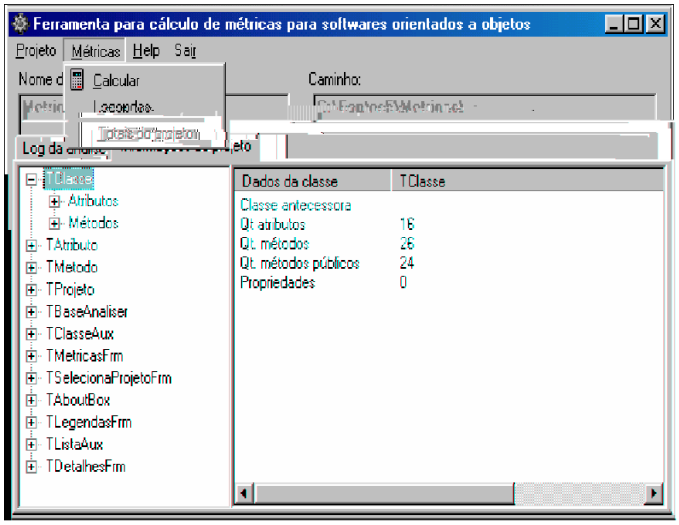

Figura 4: Ferramenta para Calculos das Metricas [14]

extração destes dados são calculadas as métricas. As métricas selecionadas por [14] foram as de projetos e construção totalizando dezenove métricas que a ferramenta permite calcular. A Figura 4 apresenta a ferramenta.

3. **Metricas para Acoplamento e Coesao em Sistemas Orientados a Objetos:** desenvolvida por [10] esta ferramenta utiliza as métricas para indicar o grau de coesão e acoplamento das classes, exibindo pontos do projeto que pareçam estar mal estruturados ou mal implementados. A ferramenta analisa o código-fonte de um projeto Java, indicando intensidade de coesão e acoplamento em sistemas orientado a objetos. O grau de coesão pode ser alto, médio e baixo e o de acoplamento forte, médio e fraco. As métricas que a ferramenta permite calcular relacionadas com coesão e acoplamento em sistemas orientados a objetos são

CBO, DIT, NOC e LCOM3.

A ferramenta *Visualization Of Metrics*, cujo desenvolvimento foi apresentado neste trabalho, conta com algumas vantagens relacionadas as outras, tais como:

Possui a representação de gráficos em sua implementação com vantagens em relação ao Protótipo de Seibt [14];

É integrada a uma IDE, vantagem que nenhuma outra abordagem apresenta. Dedicada para o programador que não terá a necessidade de utilizar outra ferramenta externa para avaliar a qualidade do seu código, pois ela já está acoplada ao seu ambiente de desenvolvimento, o que torna uma programação mais interativa e dinâmica.

Implementa algumas métricas de construção, fundamentais e importantes, que a visual Métrica [3] e a ferramenta de Marques[10] não utiliza.

Possui um gráfico para cada Métrica e detalhes de cada uma, o que nas outras ferramentas apresenta somente um para todas as métricas;

FERRAMENTAS	MÉTRICAS					LINGUAGEM			GRÁFICO	IDE
	MAM	QMC	TMC	QMP	QAC	Java	C++	Delphi		
Visualization Of Metrics	+	+	+	+	+	-	+	-	+	+
Visual Métrica	-	-	-	-	-	+	-	-	+	-
Protótipo de Seibt(2001)	+	+	+	+	+	-	-	+	-	-
Ferramenta de Marques(2008)	-	-	-	-	-	+	-	-	+	-

Tabela 1: Comparacao entre as ferramentas de metricas OO

MAM = Média de Atributo por Métodos; QMC = Quantidade de Métodos por Classe; TMC = Tamanho dos Métodos por Classe; QMP = Quantidade de Métodos Publico; QAC = Quantidade de Atributos por Classe;

na *IDE Qt Creator*.

A ferramenta foi desenvolvida em C++ no *Qt Creator*, é um sistema *Desktop* e necessita somente da instalação do ambiente de desenvolvimento. Qualquer programador que já utiliza este ambiente poderá instalar esta ferramenta facilmente.

3.3 IMPLEMENTAÇÃO DAS MÉTRICAS

3.3.1 Métricas Definidas

As métricas implementadas foram algumas de construção, definidas como as mais importantes para avaliar a qualidade do código e do projeto.

1.

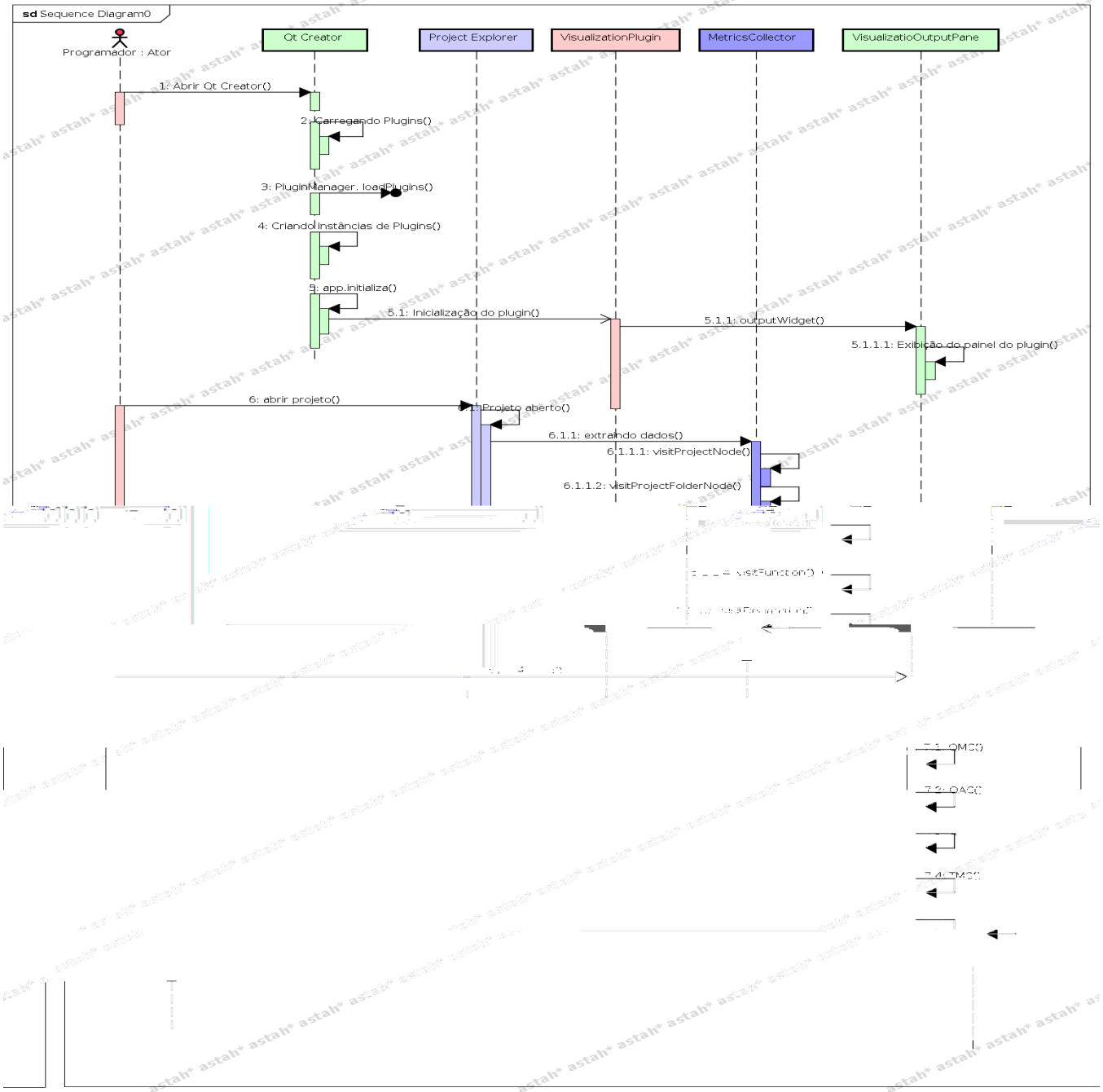

Figura 6: Diagrama de Sequência

Tabela 2: Descricao dos Metodos e Atributos da classe *VisualizationMetricsPlugin*

MÉTODOS	DESCRIÇÃO
VisualizationMetricsPlugin()	construtor da classe
initialize()	método principal que inicializa o *plugin*
ATRIBUTOS	DESCRIÇÃO
m_ visualizationMetricsPane	guarda um objeto da classe VisualizationMetricsPane

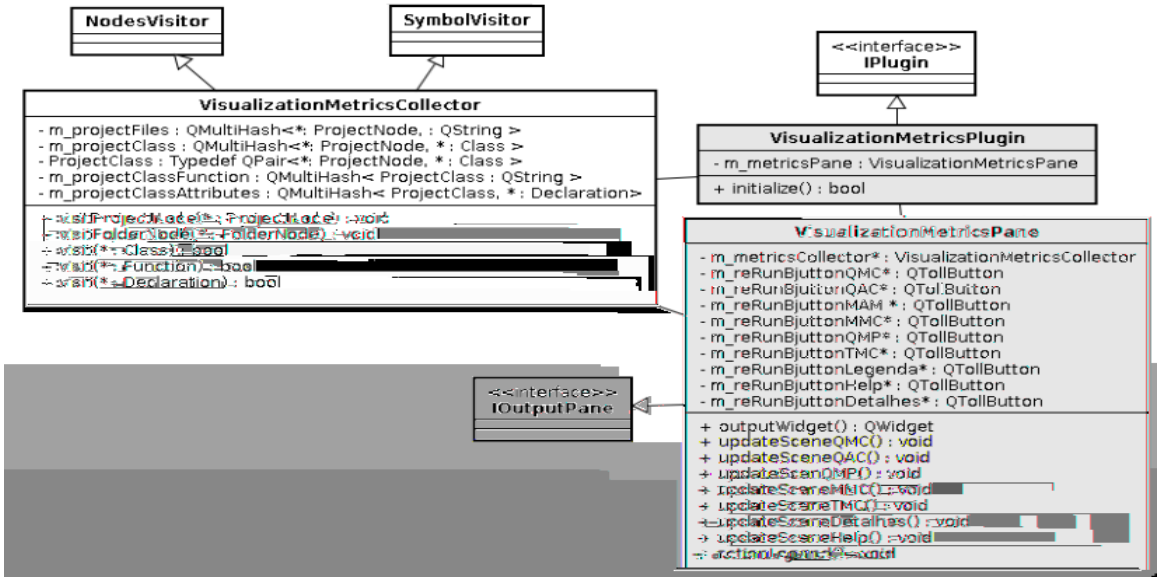

Figura 7: Diagrama de Classes

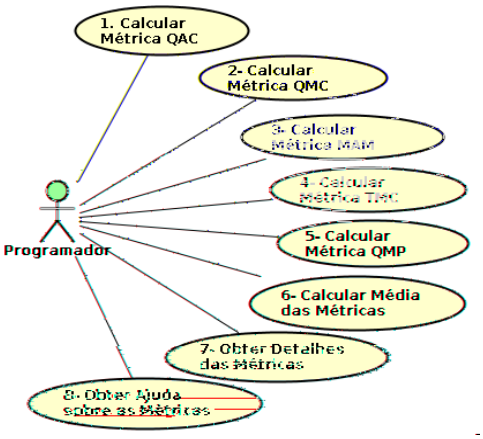

Figura 5: Diagrama de Casos de Uso

A tabela 2 apresenta a classe *VisualizationMetricsPlugin*, é a principal que implementa a interface *IPlugin* que é responsável pela criação de um *plugin* no *Qt Creator*.

A tabela 3 apresenta a classe *VisualizationMetricsPane*, é a responsável pelos cálculos das métricas e pela parte gráfica do sistema, exibe o painel gráfico das métricas no *Qt Creator*.

A tabela 4 apresenta a classe *VisualizationMetricsCollector*, é a que extrai todos os dados do sistema para o cálculo das métricas.

A ilustração de alguns códigos do projeto está representada nas Figuras 8, 9 e 10, apresentando alguns para a extração dos dados e para os cálculos das métricas.

3.4 Entendendo os Resultados das Métricas

O gráfico de barras utilizado é útil pois exibe os valores de uma forma que é possível compará-los relativamente uns com os outros. O gráfico é formado por dois eixos o horizontal que representa as classes do projeto e o vertical os valores das métricas para cada classe.

Foram selecionadas 5 métricas. Cada uma representa a qualidade do código relativo a cada classe. O gráfico ilustra a variação de valores para estas métricas, onde cada uma possui um limite estático e quando este limite é ultrapassado a cor da barra muda para vermelho que representa um nível ruim de qualidade, quando o valor da métrica está dentro do limite definido, esta cor pode variar entre verde e laranja. A cor verde representa uma boa qualidade do código e a cor laranja significa um valor aceitável que pode ser definido como razoável. O gráfico possui uma informação extra que é o nome do projeto que está representado por um circulo pequeno de cor azul que exibe o nome do projeto para cada classe.

3.4.1 Funcionamento da Ferramenta

A ferramenta implementada tem um funcionamento prático e auto-explicativo, quando um programador inicia o *Qt Creator* e cria ou abre um projeto, o *plugin* estará ativo na parte inferior da IDE disponibilizando uma barra de ferramentas contendo todo o suporte necessário para a utilização das métricas. Abaixo seguem as funcionalidades do sistema as respectivas telas de exibições estão incluídas no apêndice D, com a intenção de facilitar a compreensão do sistema, a análise dos cálculos das métricas será feito com o seu próprio código.

Os botões QMC, QAC, QMP, MAM, TMC quando pressionados exibem as informações gráficas das respectivas métricas, representando cada classe através de barras coloridas, com uma numeração que representa o valor atualizado desta métrica. Cada cor das barras representa os níveis de qualidades, ao lado de cada gráfico tem uma legenda que identifica as suas colorações. Para TMC é exibida uma tabela com as quantidades de linhas dos métodos de cada classe.

Tabela 3: Descricao dos Metodos e Atributos da classe VisualizationMetricsPane

MÉTODOS	DESCRIÇÃO
visualizationMetricsPane()	Construtor da classe

```
void VisualizationOutputPane::updateSceneQMC()
{
    int i = 10;
    m_scene->clear();
    updateSceneLegend();

    m_scene->addSimpleText(trUtf8("QUANTIDADE DE MÉTODOS POR CLASSE (QMC)"))->setPos(-300,-120);

     foreach(ProjectExplorer::ProjectNode *project, m_metricCollector->m_projectClasses.uniqueKeys())
     {
         if (!project) continue;
         foreach(CPlusPlus::Class *clazz, m_metricCollector->m_projectClasses.values(project))
         {
             if (!clazz) continue;
             if (clazz->isUnavailable()) continue;
             if (clazz->name() && clazz->name()->identifier() && m_metricCollector)
             {

                 int numberOfMethods = m_metricCollector->m_projectClassFunctions.count(MetricCollector::ProjectClass(project, clazz));

                 QVariant v;
                 v.setValue(numberOfMethods);
                 QString s = v.toString();
                 if(numberOfMethods <=20){
                     QGraphicsRectItem *rect = m_scene->addRect(0, -numberOfMethods*10, 20, numberOfMethods*10,
                                                                QPen(Qt::darkGreen), QBrush(Qt::green, Qt::SolidPattern));

                     rect->setPos(i, 0);
                     rect->setToolTip(clazz->name()->identifier()->chars());
                     m_scene->addSimpleText(s)->setPos(i+2,-numberOfMethods*9.5);

                 }
                 else
                     if(numberOfMethods > 20 && numberOfMethods<=40 ){
                         QGraphicsRectItem *rect = m_scene->addRect(0, -numberOfMethods*10, 20, numberOfMethods*10, QPen(Qt::darkGreen),
                                                                    QBrush(QColor(254,150,0), Qt::SolidPattern));

                         rect->setPos(i, 0);
                         rect->setToolTip(clazz->name()->identifier()->chars());
                         m_scene->addSimpleText(s)->setPos(i+2,-numberOfMethods*9.5);

                     }
```

Figura 10: Metodo que calcula a metrica QMC

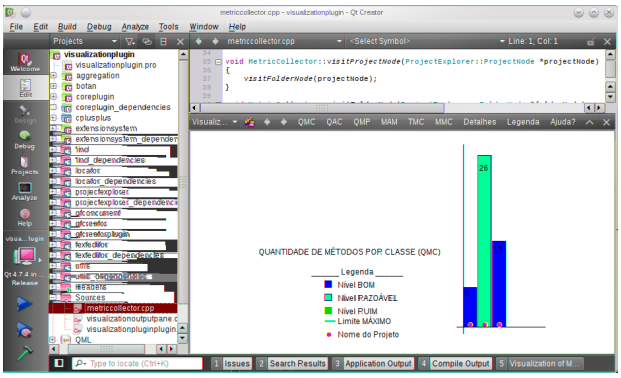

Figura 11: Tela de Exibicao do Calculo da Metrica QMC

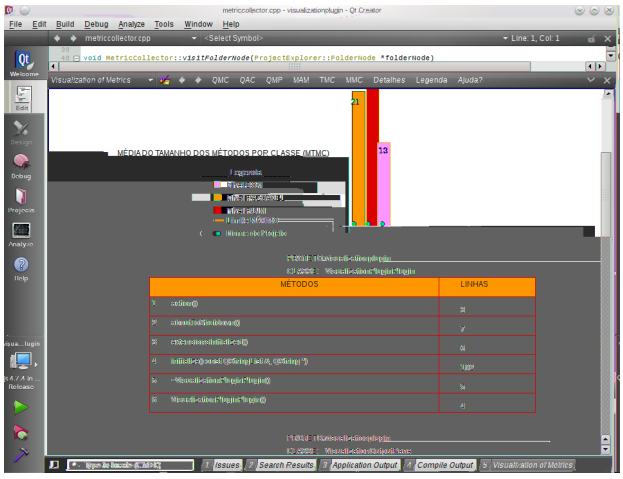

Figura 12: Tela de exibicao do calculo da metrica TMC

As Figuras 11 e 12 apresentam as funcionalidades de QMC e TMC.

O botão MMC pressionado exibe as informações gráficas da média aritmética das métricas, em um único gráfico, representando cada métrica através de barras coloridas, com uma numeração que representa o valor atualizado da média da métrica e o nome da respectiva métrica. Cada cor das barras representa os seu níveis de qualidade. Além de apresentar o gráfico, também exibe uma tabela com todas as médias das métricas por projeto.

Detalhes: esta funcionalidade exibe todos os detalhes das métricas através de uma tabela de dados, representando cada valor das métricas de cada classe. Também é exibido ao lado de cada valor das métricas um quadrado pequeno que colore de acordo como o nível da métrica. Quando verde a métrica está com um nível bom, ao ficar laranja a métrica está com um nível razoável e quando a cor representada é vermelha indica um nível ruim da métrica.

Legenda: a legenda tem a função de informar todas as siglas das métricas, ao clicar neste botão é exibida uma janela

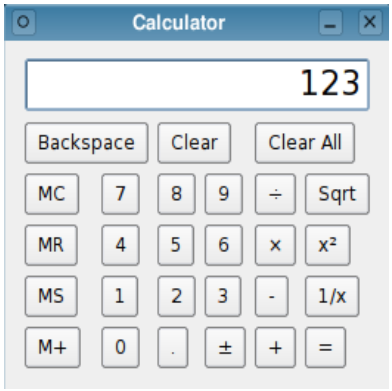

Figura 13: Estudo de Caso(Fonte: *Qt Creator*)

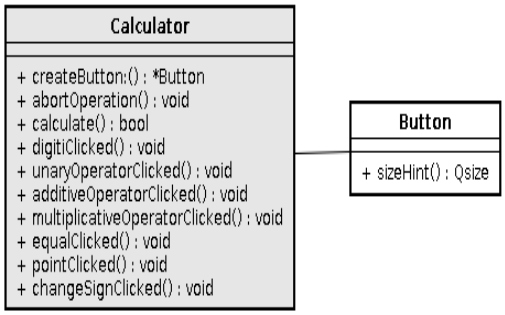

Figura 14: Diagrama de classes do projeto *Calculator*

informativa contendo todas as siglas das métricas e seus respectivos significados.

Ajuda: esta opção traz os conceitos gerais de cada métrica, suas vantagens e desvantagens e os níveis mínimos e máximos que podem alcançar.

3.4.2 Estudo de Caso

Neste trabalho já foram analisadas as métricas relativas ao próprio código do *visualization of Metrics*. Para algumas classes os resultados foram bons porém para outras razoáveis. Além de avaliar a qualidade do próprio *software*, o *Visualization of Metrics* avaliará um outro projeto básico para demonstração, disponibilizado nos exemplos do *Qt Creator*.

O projeto utilizado é uma calculadora com interface gráfica que utiliza os *widgets* do *Qt*, contém várias funcionalidades além das operações matemáticas básicas. A Figura 13 ilustra esta calculadora em execução. A Figura 14 ilustra o diagrama de classes desse projeto que possui duas classes, uma chamada *calculator* que é responsável por todas as funcionalidades da calculadora e outra é a classe *button* responsável pelo *layout*. As Figuras 15, 16 e 17 ilustram os cálculos de algumas métricas para o projeto *calculator*.

3.4.3 Análise dos Resultados

A Figura 15 apresenta a tela de exibição dos cálculos das médias das métricas de todo o projeto. Analisando os resultados obtidos, tem-se que o projeto possui um bom nível de qualidade pois a maioria das médias das métricas estão classificadas com nível bom e somente uma que é a média de atributos por classe ultrapassou o limite definido e ficou com um nível ruim.

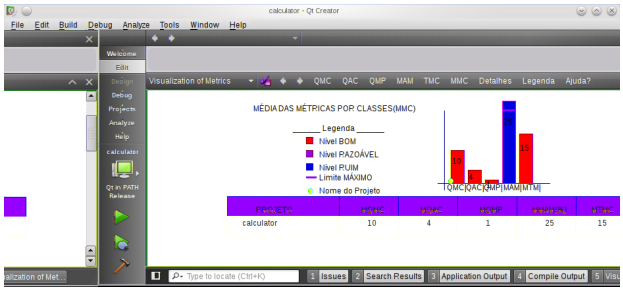

Figura 15: Media geral das metricas do projeto *Calculator*

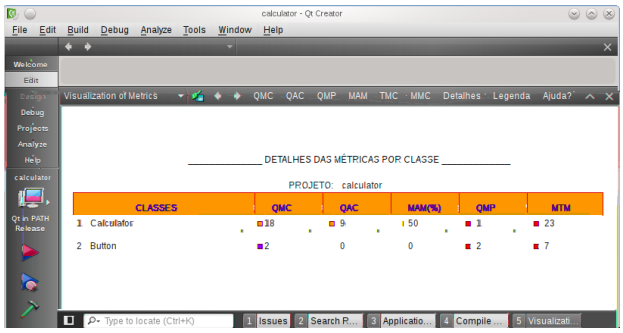

Figura 16: Detalhe de todas as metricas do projeto *Calculator*

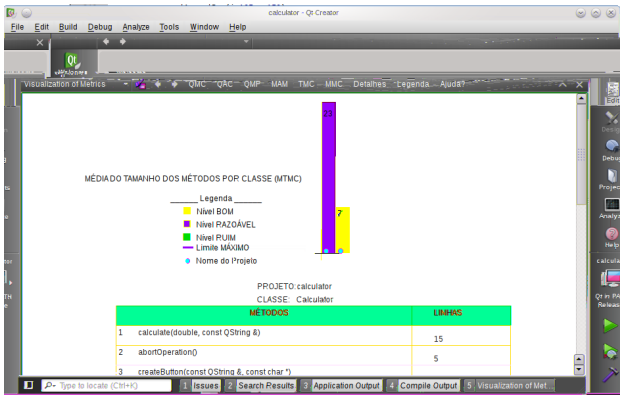

Figura 17: Tela de calculo da metrica TMC do projeto *Calculator*

A Figura 16 apresenta a tela com a tabela de todos os valores das métricas para cada classe individual, onde para cada resultado obtido o nível é apresentado pelos quadrados posicionados ao lado. Para a classe *Calculator*, a métrica QMC e QAC estão com níveis razoáveis, as QMP e MTM estão com níveis bons e somente a MAM(%) está com um nível ruim. Para a classe *Button* existem duas métricas com nível ruim, duas com nível bom e uma com nível razoável.

A Figura 17 demonstra a tela de cálculo da métrica TMC exibindo o gráfico com a média do TMC e nas tabelas a quantidade de linhas de cada método de cada classe. Através do gráfico o nível para a classe *Calculator* é razoável e a da classe *Button* nível bom.

4. AVALIAÇÃO DO EXPERIMENTO DE VALIDAÇÃO DO SOFTWARE

4.0.4 Descrição do Experimento

Hipotese Nula (H0): a utilização do *plugin* de visualização de software NÃO implica em uma diferença estatisticamente considerável na diferença média das métricas antes e depois da refatoração.

Hipotese Alternativa (H1): a utilização do *plugin* de visualização de software IMPLICA em uma diferença estatisticamente considerável na diferença média das métricas antes e depois da refatoração.

Espera-se que, se rejeitada a hipótese nula, o valor das métricas após a refatoração utilizando o *plugin* indique uma melhoria na qualidade interna do sistema. Ou seja, se uma métrica menor indicar um sistema de melhor qualidade espera-se que Média (com *plugin*) < Média (sem *plugin*).

Para a comprovação de tais hipóteses foi feito um experimento onde o sistema utilizado foi o jogo de tetrix. Nesse experimento o objetivo era fazer uma refatoração e inclusão de nova funcionalidade no sistema. Foi definido um grupo de controle que não utilizou o *plugin* e o outro grupo de uso do *plugin*. As variáveis definidas a serem medidas foram, tempo e métricas após o experimento. O experimento possui um fator (uso do *plugin*) com dois níveis (sim ou não) sendo executado de forma emparelhada (os dois grupos foram sujeitos ao mesmo conjunto de atividades realizadas).

4.0.5 Gráficos dos Resultados

O experimento foi feito com dois grupos de 4 pessoas, sendo o primeiro chamado Grupo de Controle (pessoas que não usaram o *plugin*) e o segundo chamado Grupo de uso do *plugin* (pessoas que utilizaram o *plugin*). Os gráficos de todo o projeto representam os valores de cada Métrica x Indivíduo, onde as barras azuis representam a Média das Métricas Obtidas pelo Indivíduo em cada Classe do Projeto (MCls) e as barras vermelhas as Médias das Métricas Originais do Projeto. Nos Gráficos do Projeto consolidado são representado os valores das Médias das Métricas x Todas as Métricas, onde as barras azuis representam as MCls Obtidas em cada Indivíduo x as Métricas Originais do Projeto. Abaixo seguem as Figuras com os gráficos do projeto e suas descrições.

Figura 18: ilustra o gráfico da Métrica QMC de todo o projeto, o gráfico do grupo de controle, indivíduos que não usaram o *plugin*. Contém, para cada indivíduo, as médias das métricas obtidas de todas as classes do projeto em comparação com as métricas originais. O indivíduo 1 e 2 tiveram um bom desempenho pois conseguiram diminuir a quantidade de métodos em excesso das classes, deixando com uma quantidade aceitável. O indivíduo 3 se manteve instável não acrescentou nem retirou nenhum método das classes. O indivíduo 4 acrescentou mais métodos nas classes, porém continuou na faixa aceitável. No gráfico do Grupo de Uso do *plugin*, os indivíduos 1, 2, 3 acrescentaram métodos as classes e contudo não ultrapassaram o limite. O indivíduo 4 se manteve instável.

Figura 19: ilustra o gráfico da Métrica QAC de todo o projeto, onde a média do projeto original está razoável, não permitindo que seja acrescentato mais que um atributo, pois o limite é 9 por classe e a média chegou a 8,6. Os indivíduos 1, 2 e 4 respeitaram esse limite mesmo sem a utilização do *plugin* e se mantiveram instáveis, não acrescentando nem retirando nenhum atributo das classes. O indivíduo 3 acrescentou somente um atributo e permaneceu no limite desejável. O grupo de uso do *plugin* se manteve totalmente instável não acrescentando nem retirando nenhum atributo das classes.

Figura 20: ilustra o gráfico da Métrica QMP de todo o projeto, onde a média do projeto original se encontra em um nível bom. Para o grupo de controle o indivíduo 1 não acrecentou nenhum método, os indivíduos 2 e 3 diminuiram a média em relação ao projeto original e se mantiveram com as mesmas médias, o indíviduo 4 obteve uma média maior que o projeto original, mais contudo todos permaneceram em um nível bom. No grupo de uso do *plugin* os indivíduos 1, 3 e 4 obtiveram uma média menor que a original. O indivíduo 2 teve uma média um pouco maior que a original, mas contudo, todos ficaram com um nível bom.

Figura 21: ilustra o gráfico da Métrica MAM% que tem o limite de 22% ou seja em uma classe é permitido ter no máximo 9 atributos e 40 métodos. A média do projeto original ultrapassou o limite recomendado com 167%. Para o grupo de controle os indivíduos 1 e 3 tiveram uma média alta ultrapassando a média original e o limite da métrica. Os indivíduos 2 e 4 conseguiram minimizar a média mas ainda não foi o suficiente e permaneceram acima do limite. Para o grupo de uso do *plugin* os indivíduos 1, 2 e 3 minimizaram as médias em relação ao projeto original mas não conseguiram chegar ao limite permitido. O indivíduo 4 ultrapassou a média original e todos continuaram sem atingir o limite permitido.

Figura 22: ilustra o gráfico da Métrica MTM (linhas) onde é permitido até 30 linhas de códigos por métodos. O projeto original ultrapassou esse limite com a média de 32 linhas por métodos. Para o grupo de controle, os indivíduos 1 e 5 também ultrapassaram esse valor, porém os indivíduos 2 e 3 conseguiram chegar ao limite permanecendo com um nível razoável. Para o grupo de uso do *plugin* todos obteram uma média menor que a média original, porém só o indivíduo 2 conseguiu ficar com a média de nível bom.

Figura 23: ilustra os gráficos do projeto consolidado com os valores de todas as métricas do projeto por grupo e não mais por indivíduos. Comparando os dois grupos temos:

a) Métrica QMC: o grupo de controle obteve um valor instável igual a métrica do projeto original de 10,33 e

o grupo de uso do *plugin* obteve um valor de 11,08 e também com nível bom e não permaneceu instável.

b) Métrica QAC: o grupo de controle obteve um valor de 8,75, mais que a métrica original e permanecendo com um nível bom, o grupo de uso do *plugin* obteve um valor 8,66 com um nível bom e melhor que o outro grupo.

c) Métrica QMP: o grupo de controle obteve um valor 5, igual ao da métrica original nível bom e instável, o grupo de uso do *plugin* obteve um valor de 4,91 nível bom e não permaneceu instável e com um valor melhor que o outro grupo.

d) Métrica MAM(%): o grupo de controle obteve um valor de 147,25 valor menor que a métrica original porém com um nível ruim. O grupo de uso do *plugin* obteve o valor de 141,91 apesar de continuar com nível ruim, seu valor foi melhor que o da métrica original e do que o do outro grupo.

e) Métrica MTM (linhas): o grupo de controle obteve um valor de 29,75 menor que o da métrica original e com um nível razoável. O grupo de uso do *plugin* obteve valor de 26,58 também com um nível razoável porém com um valor melhor que o outro grupo.

4.0.6 Conclusões dos Resultados

Com as indicações representadas nos gráficos percebe-se que fora rejeitada a hipótese nula e para que se confirme a hipótese alternativa, o valor das métricas após a refatoração utilizando o *plugin* indica uma melhoria na qualidade interna do sistema.

A questao-chave e: de quanto seria esta melhoria e com qual confiança pode-se afirmar que esta melhoria irá ocorrer ao utilizar o *plugin*.

Será utilizado um teste de hipótese, conhecido como *t-test*, para avaliar se a diferença entre as médias das duas amostras é estatisticamente significante.

Será realizado um *t-test* para cada métrica sendo avaliada e assume-se que as duas amostras (sem o uso do *plugin* e com o uso do *plugin*) são amostras obtidas de duas populações com distribuição normal, médias desconhecidas e variâncias desconhecidas porém iguais.

Deseja-se obter o maior nível de confiança para o qual o *plugin* produz uma diferença estatisticamente significante na média das métricas.

a) T-TEST para a metrica QMC

p-value = 0,3618456438;

N vel maximo de con anca (1 - p-value) = 0,6381543562;

b) T-TEST para a metrica QAC

p-value = 0,391002219

N vel maximo de con anca (1 - p-value) = 0,608997781

c) T-TEST para a metrica QMP

p-value = 0,8542703292

N vel maximo de con anca (1 - p-value) = 0,1457296708

d) T-TEST para a metrica MAM (%)

p-value = 0,8354105574

N vel maximo de con anca (1 - p-value) = 0,1645894426

e)-T-TEST para a metrica MTM (linhas)

p-value = 0,2622190236

N vel maximo de con anca (1 - p-value) = 0,7377809764

Conclusao: pode-se afirmar que a métrica mais influenciada pela presença do *plugin* é a MTM (linhas) já que pode-se afirmar com 73% de certeza que o *plugin* faz diferença na diferença desta métrica antes e depois da refatoração.

Outras metricas potencialmente afetadas: QMC (63%) e QAC (60%). As demais possuem nível de confiança extremamente baixo.

Ameacas a validade do experimento:

Baixo número de amostras;

Amostragem selecionada de forma não-randômica (diferentes habilidades e experiências dos alunos envolvidos);

Sistema utilizado pode não ser representativo do domínio de aplicação onde o *plugin* será utilizado;

5. CONCLUSÃO E TRABALHOS FUTUROS

A objetivo principal deste trabalho foi o desenvolvimento de um *plugin* para o *Qt Creator* para calcular métricas de sistemas orientados a objeto desenvolvido em C++. A ferramenta calcula cinco métricas fundamentais para avaliar a qualidade de um *software*.

Com o o estudo de caso e a análise do código-fonte do próprio sistema, foi possível perceber o quanto a utilização das métricas é importante para garantir a qualidade do *software*, mesmo em sistemas simples com uma ou duas classes.

Este trabalho apesar de ter alcançado o seu principal objetivo propõe possibilidades de desenvolvimentos futuros com o intúito de aumentar as suas funcionalidades e contribuições para áreas de qualidades de *software*. Um número maior de métricas poderá ser desenvolvido e outras funcionalidades na parte visual, como ao clicar em uma barra que representa uma determinada classe, ser aberto o editor na classe indicada. A opção de arquivar os cálculos das métricas extraídas do sistemas é uma necessidade futura que poderá ser implementada, para casos de comparações em cada refatoramento de projetos.

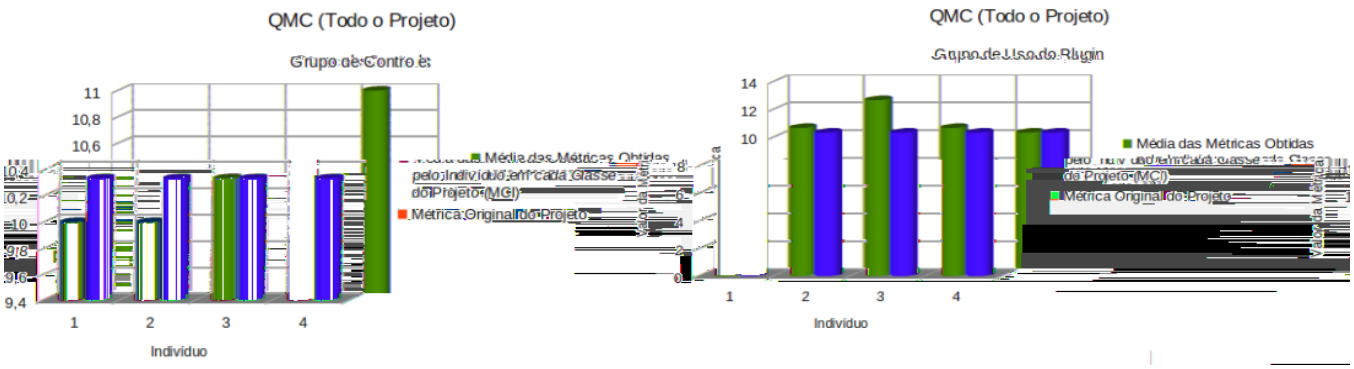

Figura 18: Gra cos da Metrica QMC de todo o Projeto para cada Grupo

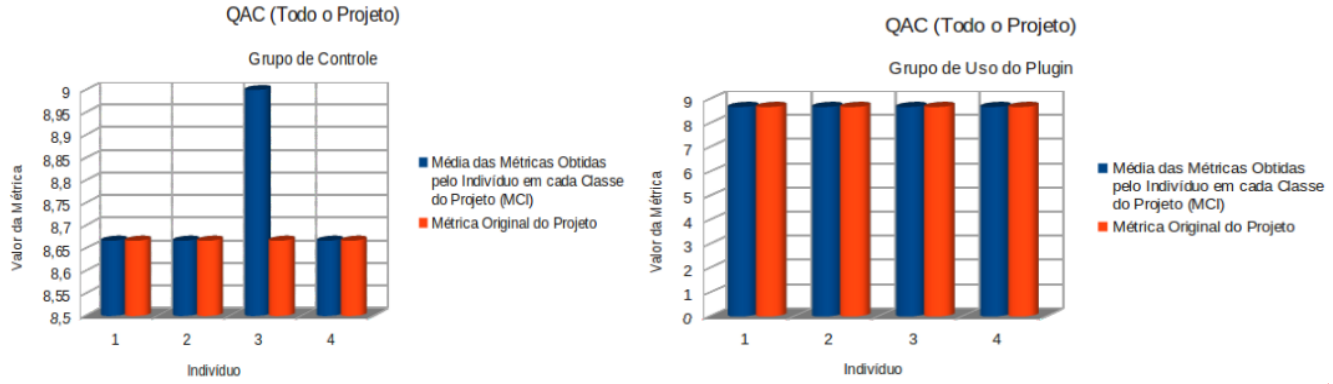

Figura 19: Gra cos da Metrica QAC de todo o Projeto para cada Grupo

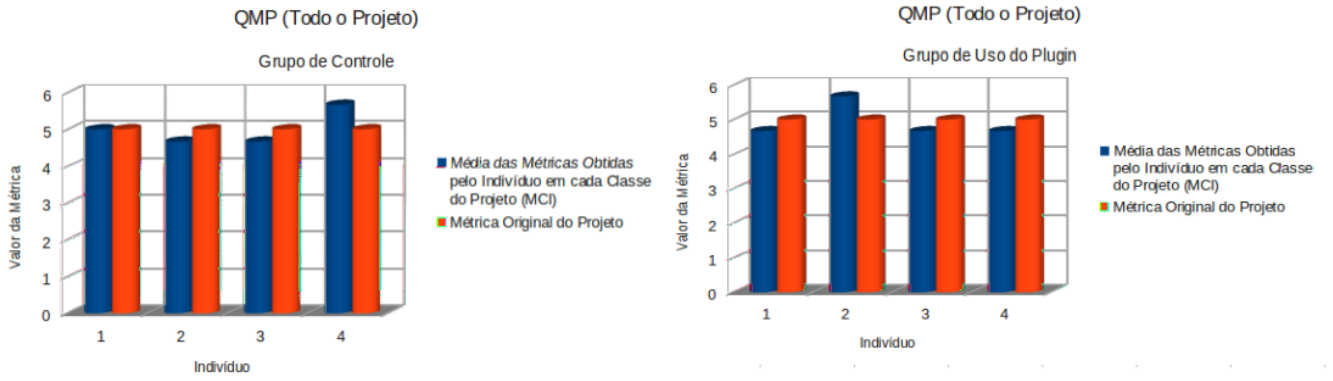

Figura 20: Gra cos da Metrica QMP de todo o Projeto para cada Grupo

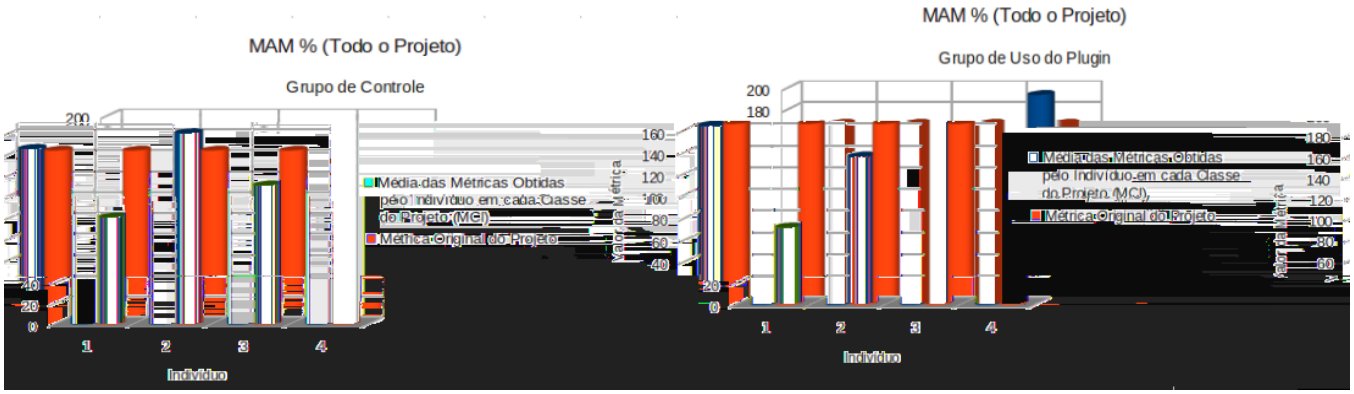

Figura 21: Gra cos da Metrica MAM% de todo o Projeto para cada Grupo

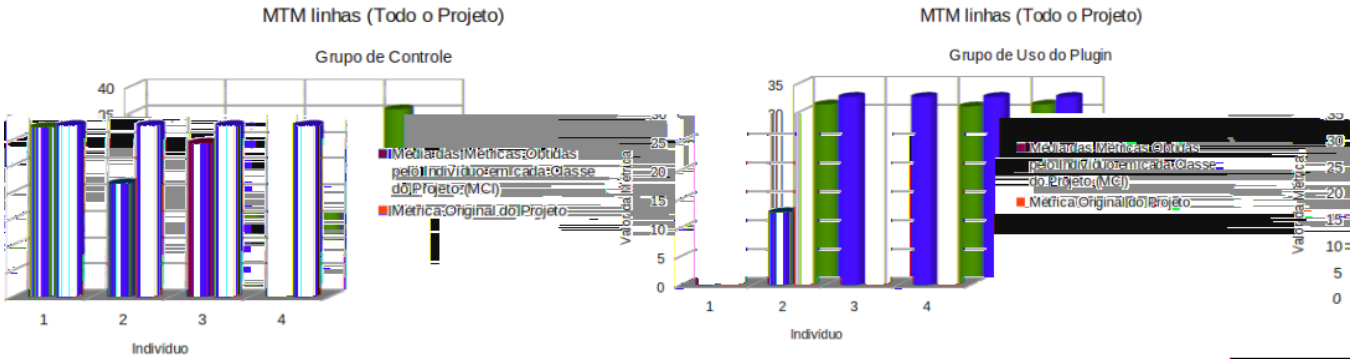

Figura 22: Gra cos da Metrica MTM(Linhas) de todo o Projeto para cada Grupo

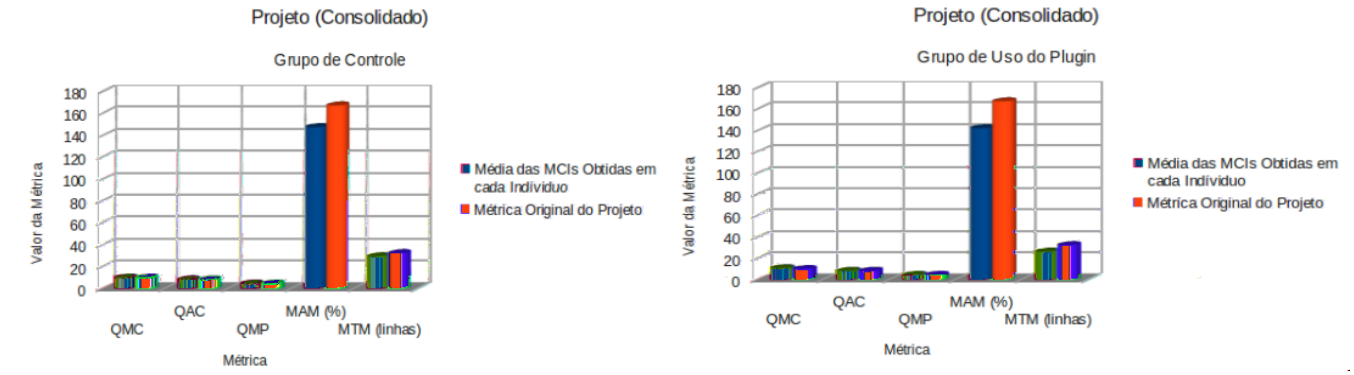

Figura 23: Gra cos do Projeto Consolidado com todas as Metricas para cada Grupo

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] S. W. AMBLER. Análise e projeto orientado a objetos: seu guia para desenvolver sistemas robustos com tecnologia de objetos. 1998.

[2] A. ARIGOFLU. A methodology for cost estimation. 1993.

[3] G. B. Borges. Applying and interpreting object oriented metrics. 1998.

[4] E. J. CARDOSO. Métricas para programação orientada a objetos, trabalho de conclusão de curso. 1999.

[5] C. F. Chidamber, Shyam R; Kemerer. A metrics suite for object oriented desing. 1994.

[6] J. M. M. FERNANDES. Métricas para análise de complexidade de pogramas orientado a objetos. 2008.

[7] Q. C. IDE and TOOLS. http://qt.nokia.com/products/developer-tools/.

[8] I. e. a. JACOBSON. Object oriented software engineering: a use case driven appriach. 1992.

[9] J. LORENZ, Mark; KIDD. Object-oriented software metrics. 1994.

[10] M. M. D. MARQUES. Métricas para acoplamento e coesão em sistemas orientado a objetos em ambiente de visualização de software, trabalho de conclusão de curso. 2008.

[11] P. D. J. MOLLER, Kurt. H. Software metrics: a practitioneris guide to improved product development. 1999.

[12] M. J. C. W. K. C. ROCHA, Ana R. Qualidade de software : Teoria e prática. 2001.

[13] L. ROSENBERG. Applying and interpreting object oriented metrics. 1998.

[14] P. R. R. d. S. SEIBT. Ferramenta para cálculo de métricas em software orientados a objetos, trabalho de conclusão de curso. 1998.

[15] M. SHEPPERD. Foundations of software measurement. 1995.

7. APÊNDICES

7.1 APÊNDICE A - VISÃO GERAL DO TRABALHO

1- O que e uma metrica OO?

É a metodologia responsável pela medição da qualidade do *software* orientado a objetos, auxiliando no desenvolvimento.

2- Qual a vantagem em sua utilizacao?

A obtenção de dados importantes sobre o *software* em uso, ter uma visão geral do projeto planejado e acompanhar o seu desenvolvimento.

3- Qual o papel fundamental que as metricas OO desempenham?

Controle no desenvolvimento do *software* e auxílio na tomada de decisões.

4- Quais as categorias que as metricas OO estao divididas?

Métricas de Análise - são usadas para a medição da qualidade dos esforços da análise e podem medir as classe importantes do sistema. Métricas de Projeto - ajudam a estabelecer comparações entre vários sistemas e criar uma base de futuras recomendações para um novo projeto. Métricas de

13- O que acontece quando ha um numero grande de atributos em uma classe?

Haverá muitos relacionamentos com outras classes, o que gerará problemas com a reusabilidade.

7.2 APÊNDICE B - CÓDIGO FONTE DO EXEMPLO ANALISADO

Da Figura 24 à 32 segue partes dos códigos do projeto Calculator que foi o exemplo utilizado para o caso de uso.

```
class Calculator : public QDialog
{
    Q_OBJECT

public:
    Calculator(QWidget *parent = 0);

private slots:
    void digitClicked();
    void unaryOperatorClicked();
    void additiveOperatorClicked();
    void multiplicativeOperatorClicked();
    void equalClicked();
    void pointClicked();
    void changeSignClicked();
    void backspaceClicked();
    void clear();
    void clearAll();
    void clearMemory();
    void readMemory();
    void setMemory();
    void addToMemory();
```

Figura 24: De nicao da classe Calculator - slots privados

```
private:
    Button *createButton(const QString &text, const char *member);
    void abortOperation();
    bool calculate(double rightOperand, const QString &pendingOperator);
```

Figura 25: De nicao da classe Calculate - metodos privados

```
void Calculator::digitClicked()
{
    Button *clickedButton = qobject_cast<Button *>(sender());
    int digitValue = clickedButton->text().toInt();
    if (display->text() == "0" && digitValue == 0.0)
        return;

    if (waitingForOperand) {
        display->setText(display->text() + QString::number(digitValue));
    }
```

Figura 26: Codigo fonte da classe Calculator - slots digitClicked

```
void Calculator::unaryOperatorClicked()
{
    Button *clickedButton = qobject_cast<Button *>(sender());
    QString clickedOperator = clickedButton->text();
    double operand = display->text().toDouble();
    double result = 0.0;

    if (clickedOperator == tr("Sqrt")) {
        if (operand < 0.0) {
            abortOperation();
            return;
        }
        result = sqrt(operand);
    } else if (clickedOperator == tr("x\262")) {
        result = pow(operand, 2.0);
    } else if (clickedOperator == tr("1/x")) {
        if (operand == 0.0) {
            abortOperation();
            return;
        }
        result = 1.0 / operand;
    }
    display->setText(QString::number(result));
    waitingForOperand = true;
}
```

Figura 27: Codigo fonte da classe Calculator - slots unaryOperatorClicked

```
void Calculator::equalClicked()
{
    double operand = display->text().toDouble();

    if (!pendingMultiplicativeOperator.isEmpty()) {
        if (!calculate(operand, pendingMultiplicativeOperator)) {
            abortOperation();
            return;
        }
        operand = factorSoFar;
        factorSoFar = 0.0;
        pendingMultiplicativeOperator.clear();
    }
    if (!pendingAdditiveOperator.isEmpty()) {
        if (!calculate(operand, pendingAdditiveOperator)) {
            abortOperation();
            return;
        }
        pendingAdditiveOperator.clear();
    } else {
        sumSoFar = operand;
    }

    display->setText(QString::number(sumSoFar));
    sumSoFar = 0.0;
    waitingForOperand = true;
}
```

Figura 28: Codigo fonte da classe Calculator - slots equalClicked

7.3 APÊNDICE C - GRÁFICOS DO EXPERIMENTO

Da Figura 33 à 41 segue os gráficos da análise do projeto Tetrix para cada grupo participante do experimento é exibido o gráfico de cada métrica para cada classe do projeto.

7.4 APÊNDICE D - TELAS DO FUNCIONAMENTO DA FERRAMENTA

Da Figura 42 à 49 segue as telas de exibições do funcionamento da ferramenta.

```
bool Calculator::calculate(double rightOperand, const QString &pendingOperator)
{
    if (pendingOperator == tr("+")) {
        sumSoFar += rightOperand;
    } else if (pendingOperator == tr("-")) {
        sumSoFar -= rightOperand;
    } else if (pendingOperator == tr("\327")) {
        factorSoFar *= rightOperand;
    } else if (pendingOperator == tr("\367")) {
        if (rightOperand == 0.0)
            return false;
        factorSoFar /= rightOperand;
    }
    return true;
}
```

Figura 29: Codigo fonte da classe Calculator - metodo Calculate

```
class Button : public QToolButton
{
    Q_OBJECT

public:
    Button(const QString &text, QWidget *parent = 0);

    QSize sizeHint() const;
};
```

Figura 30: De nicao da classe Button

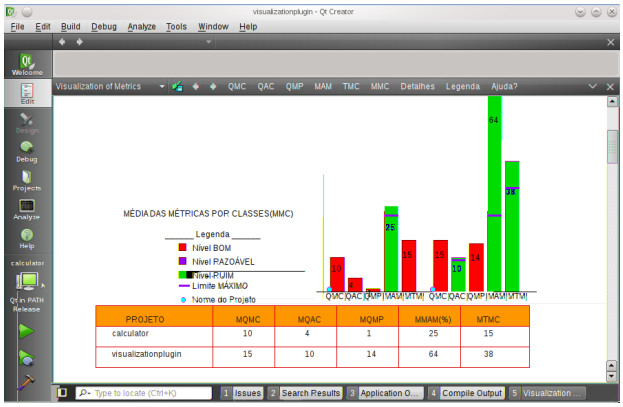

Figura 55: Tela de exibicao com dois projetos abertos, exibindo a media de suas metricas.

```
QSize Button::sizeHint() const
{
    QSize size = QToolButton::sizeHint();
```

Figura 32: Codigo fonte da classe Button - metodo sizeHint

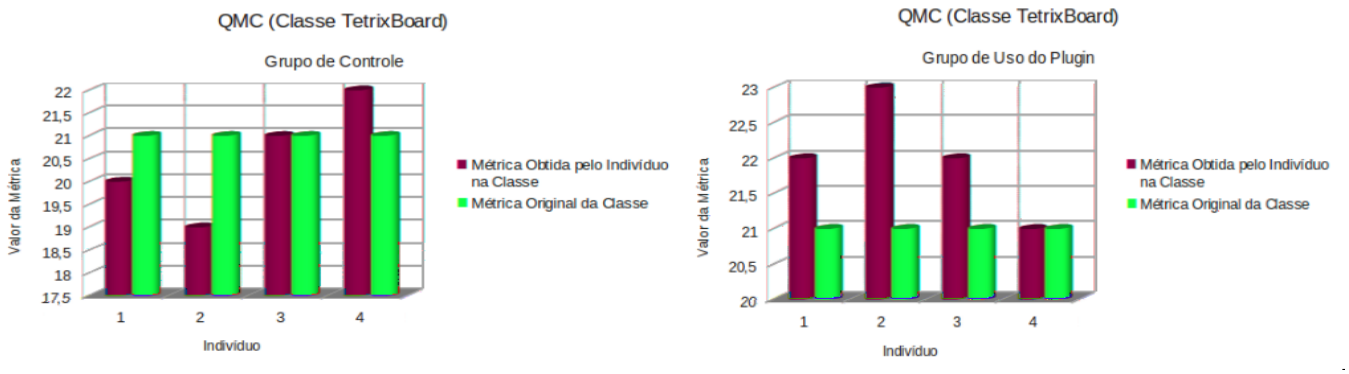

Figura 33: Gra co da Metrica QMC da Classe TetrixBoard para cada Grupo

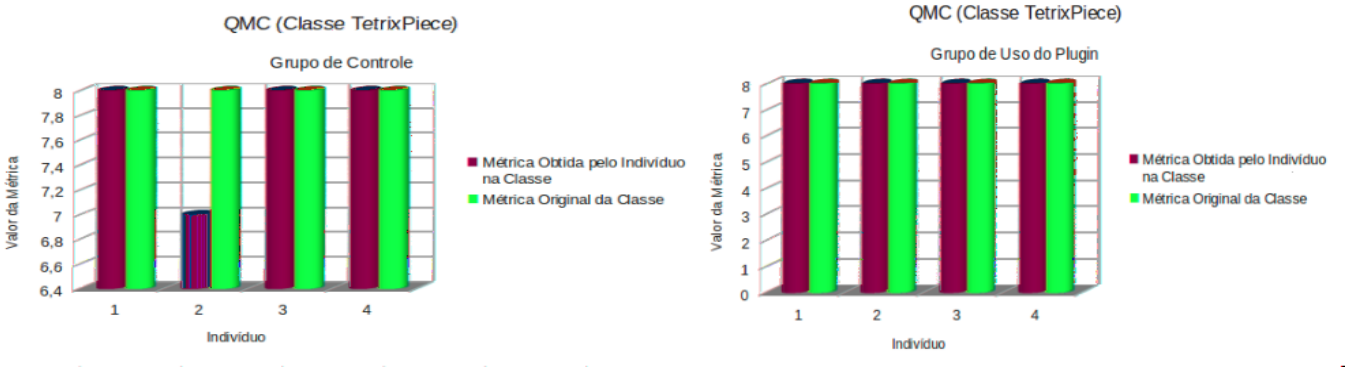

Figura 34: Gra co da Metrica QMC da Classe TetrixPiece para cada Grupo

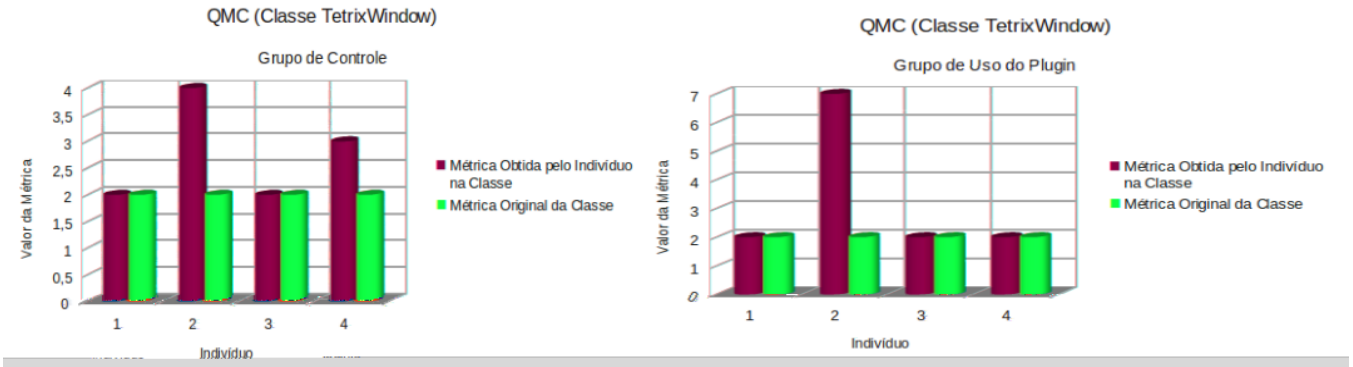

Figura 35: Gra co da Metrica QMC da Classe TetrixWindow de cada Grupo

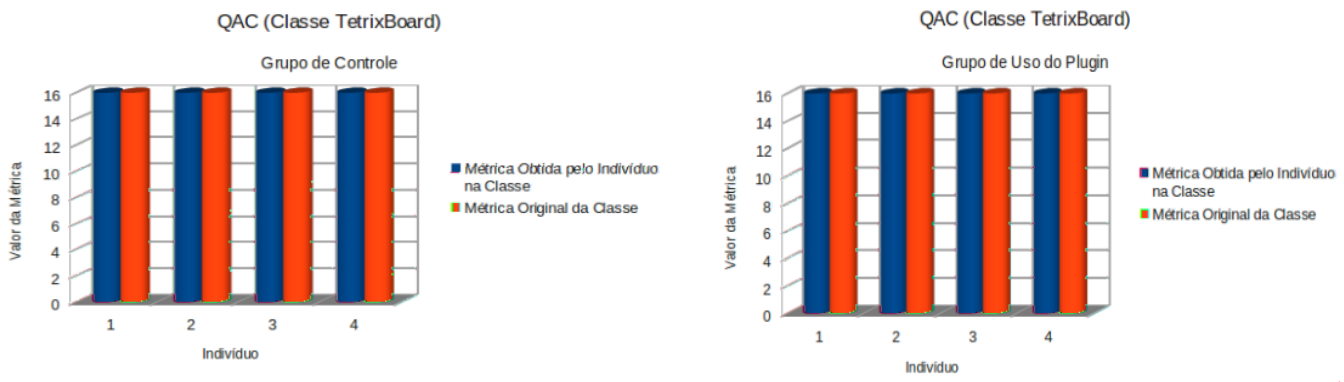

Figura 36: Gra co da Metrica QAC da Classe TetrixBoard para cada Grupo

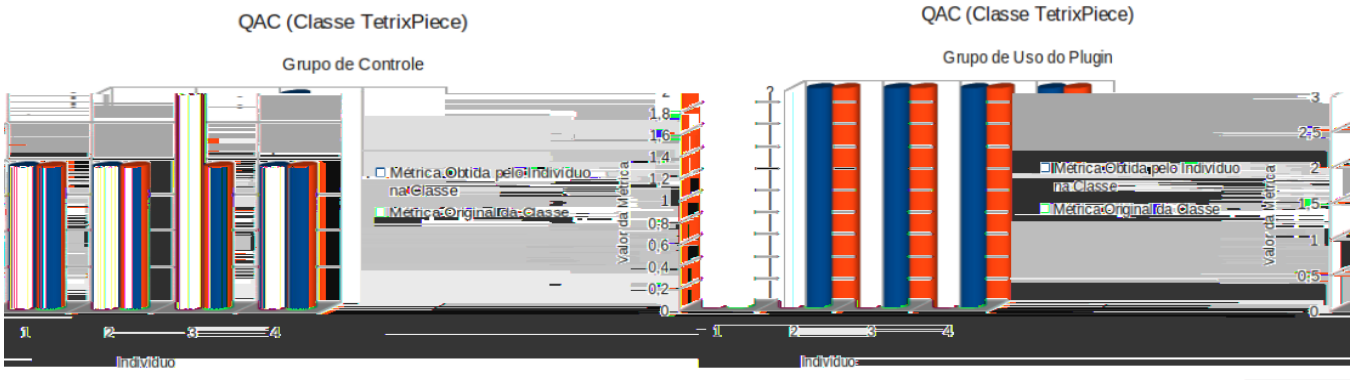

Figura 37: Gra co da Metrica QAC da Classe TetrixPiece para cada Grupo

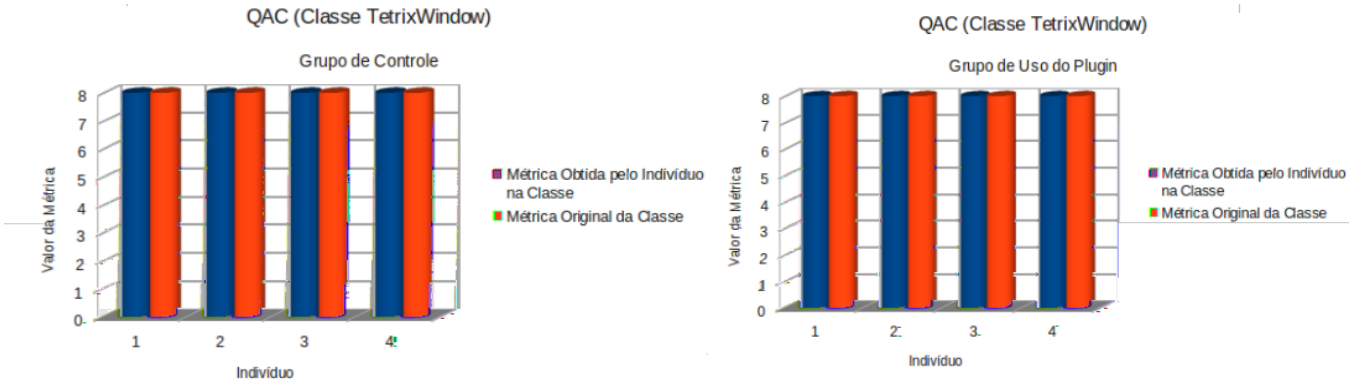

Figura 38: Gra co da Metrica QAC da Classe TetrixWindow de cada Grupo

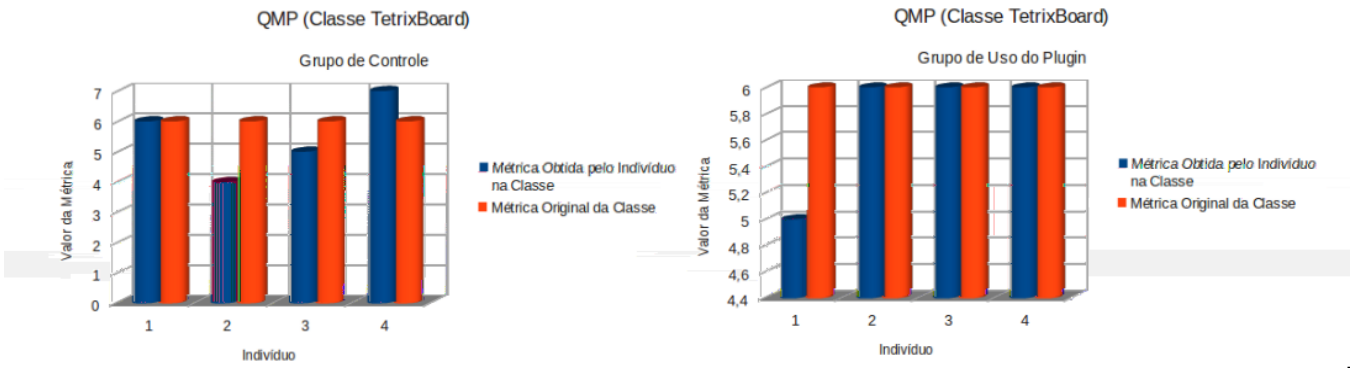

Figura 39: Gra co da Metrica QMP da Classe TetrixBoard para cada Grupo

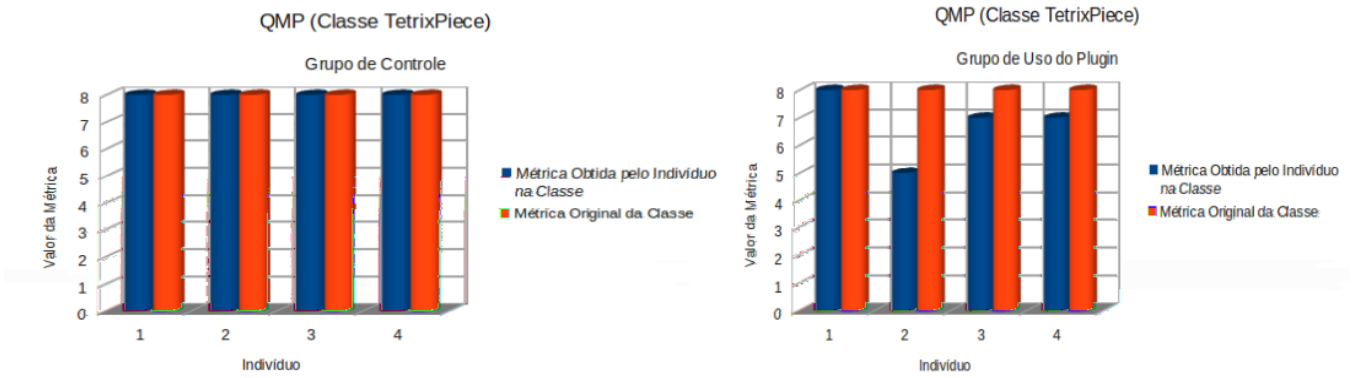

Figura 40: Gra co da Metrica QMP da Classe TetrixPiece para cada Grupo

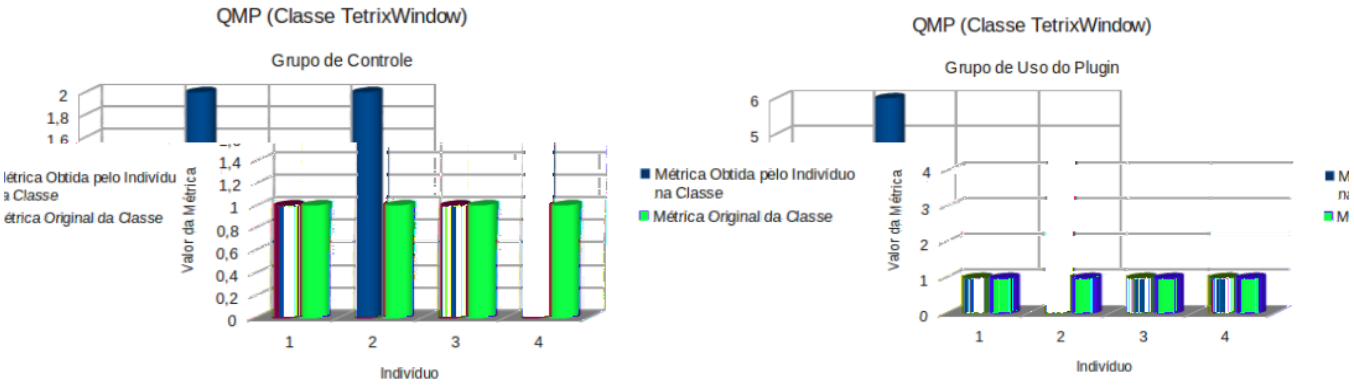

Figura 41: Gra co da Metrica QMP da Classe TetrixWindow de cada Grupo

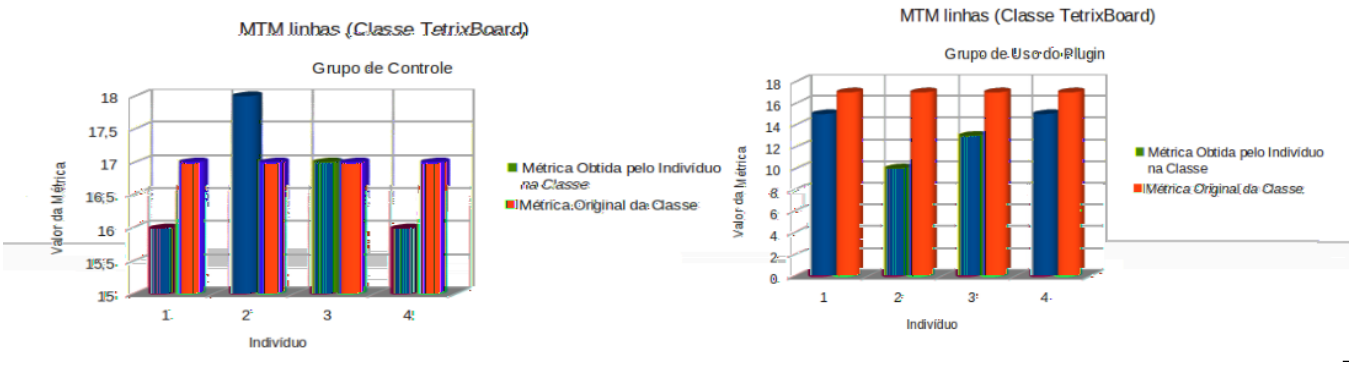

Figura 42: Gra co da Metrica MTM (Linhas) da Classe TetrixBoard para cada Grupo

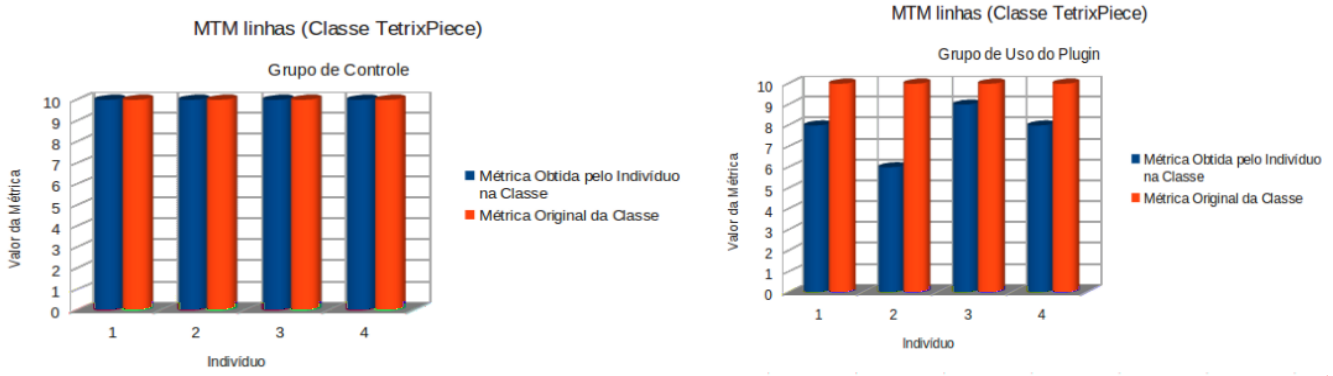

Figura 43: Gra co da Metrica MTM (Linhas)da Classe TetrixPiece para cada Grupo

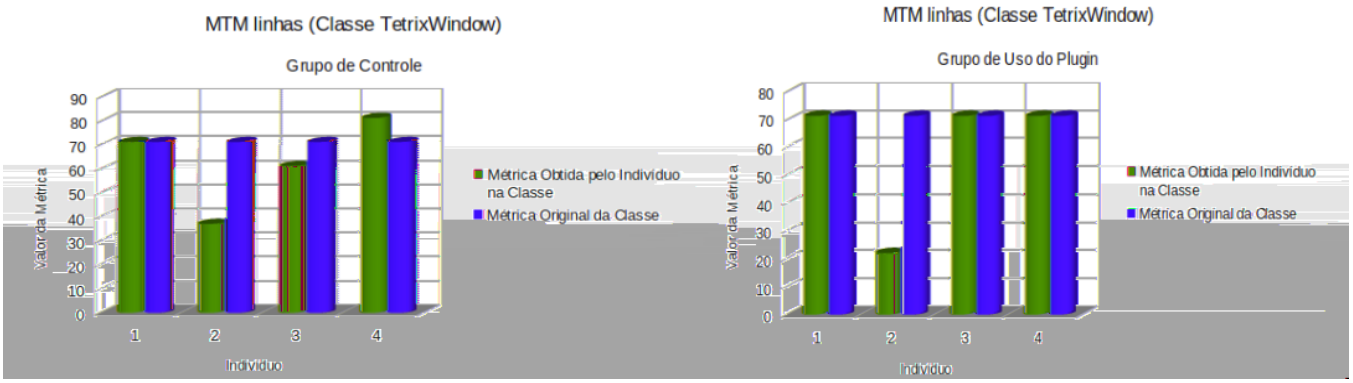

Figura 44: Gra co da Metrica MTM (Linhas) da Classe TetrixWindow de cada Grupo