UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO ANTONIO CARVALHO MONTEIRO DE OLIVEIRA

AVALIAÇÃO DE CÓDIGO-FONTE ORIENTADO A OBJETOS USANDO REQUISITOS NÃO-FUNCIONAIS, MÉTRICAS E LÓGICA *FUZZY*

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2015

JOÃO ANTONIO CARVALHO MONTEIRO DE OLIVEIRA

AVALIAÇÃO DE CÓDIGO-FONTE ORIENTADO A OBJETOS USANDO REQUISITOS NÃO-FUNCIONAIS, MÉTRICAS E LÓGICA *FUZZY*

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof.ª Simone Nasser Matos

PONTA GROSSA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

Avaliação de Código Fonte Orientado a Objetos Usando Requisitos Não-Funcionais, Métricas e Lógica *Fuzzy*

por

JOÃO ANTONIO CARVALHO MONTEIRO DE OLIVEIRA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 24 de novembro de 2015 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Profª Drª Simone Nasser Matos
Orientadora

_____
Prof. Dr. Ariangelo Hauer Dias
Membro Externo

_____
Profª. Msc. Sarah Gomes Sakamoto
Membro titular

_____
Prof. Dr. Tarcizio Alexandre Bini
Membro titular

_____
Prof. Dr. Ionildo José Sanches
Responsável pelos Trabalhos
de Conclusão de Curso

_____
Prof. Dr Erikson Freitas de Morais
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Agradeço a minha professora orientadora Simone Nasser Matos pelo apoio e dedicação, essenciais para o desenvolvimento deste trabalho, além da compreensão e de todo o tempo aplicado para garantir a finalização deste trabalho.

Agradeço aos meus familiares em especial meus pais Valdemar e Maria Elinete aos quais sempre me forneceram conforto e me auxiliando em cada decisão tomada.

Agradeço aos meus amigos e a todos aqueles que estiveram comigo durante a realização do curso com os quais contribuíram direta e indiretamente para a realização deste trabalho

RESUMO

OLIVEIRA, JOÃO ANTONIO CARVALHO MONTEIRO. **Avaliação de Código Fonte Orientado a Objetos Usando Requisitos Não-Funcionais, Métricas e Lógica *Fuzzy***. 2015. 78 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

A avaliação de programas garante que o produto final tenha um nível de qualidade e pode ser realizada por meio de ferramentas automatizadas que permitem gerar valores quantitativos para as métricas de software tais como: quantidade de parâmetros, número de linhas e complexidade ciclomática. Na literatura foi proposto o uso destas métricas com a lógica *fuzzy* para avaliar o nível de qualidade do código-fonte, porém, o processo não é realizado de forma automatizada e não se estabeleceu a correlação entre quais métricas são necessárias para medir um determinado requisito de qualidade (por exemplo, a métrica acoplamento entre classes de objetos para medir o requisito de qualidade acoplamento). Este trabalho desenvolveu uma avaliação de código-fonte usando requisitos de qualidade associando-os às suas respectivas métricas com o objetivo de verificar o nível em que o software atende aos requisitos. Para medir o nível de atendimento de cada requisito de qualidade foi aplicada a lógica *fuzzy* para o tratamento da incerteza. A avaliação automatizada permite ao desenvolvedor criar um projeto, selecionar os requisitos que serão avaliados, correlacionar as métricas aos requisitos de qualidade, obter os valores numéricos das métricas por meio da ferramenta *CKJM Extended* e aplicar as etapas da lógica *fuzzy* usando *jFuzzyLogic*. Os resultados são exibidos para o desenvolvedor na forma gráfica. As vantagens do uso da avaliação automatizada é que se conseguiu integrar o uso de métricas com os requisitos de qualidade e lógica *fuzzy*.

Palavras-chave: Lógica *Luzzy*. Requisito de Qualidade. Métricas. Avaliação de código-fonte.

ABSTRACT

OLIVEIRA, JOÃO ANTONIO CARVALHO MONTEIRO. **Object Oriented Code Evaluation Using Non-Functional Requirements, Metrics and Fuzzy Logic**. 2015. 78 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

The software evaluation to ensure that the product has a quality level and it can be done by automatizated tools that are able to generate quantitatives values for the metrics of software like: number of parameters, number of line of code, cyclomatic complexity. In the literature was proposed the use of these metrics in conjunction with fuzzy logic in order to evaluate the level of code's quality. The evaluation process is not automated and is stablished a relashinship between the metrics that are necessary to mesure a quality requirement (i.e., the metric coupling between object classes can be used to mesure the quality of the coulpling external requeriment). This work developted a evaluation of source-code using external requirements and creating a relation between metrics and requirements. The quality's level of source-code is mesured using fuzzy logic for treatment of uncertainty. The automated evaluation allow the developer the create a Project, select the requirements that will be evaluated, create the relation between metrics and external requirements, obtain the numeric values of each metrics in using CKJM extended tools and apply the fuzzy logics phases are done using jFuzzyLogic. The results are show to developer in a grahical form. The advantages of using the automated evaluation is the possibility to integrate the use of metrics with external requiremets and fuzzy logic.

Keywords: Fuzzy Logic. External Requeriment. Metrics. Source-code Evaluation.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AHF	Attribute Hiding Factor (Fator de Ocultação de Atributo)
AIF	Attribute Inheritance Factor (Fator de Herança dos Atributos)
CBO	Coupling Between Object (Acoplamento Entre Classes de Objetos)
CC	Number of Calls (Número de Chamadas)
CK	Chidamber e Kemerer
COF	Coupling Factor (Fator de Acoplamento)
CYCLO	Cyclomatic Complexity (Complexidade Ciclomática)
DIT	Depth of Inheritance Tree (Profundidade da Árvore de Herança)
ISO	International Organization for Stardardization
LOC	Lines of Code (Linhas de Código)
LCOM	Lack of Cohesion in Methods (Ausência de Coesão em Métodos)
MHF	Method Hiding Factor (Fator de Ocultação de Método)
MIF	Method Inheritance Factor (Fator de Herança de Método)
MOOD	Metrics for Object Oriented Design
NOC	Number of Children (Número de Filhos)
POF	Polymorphism Factor (Fator de Polimorfismo)
SEI	Software Engineering Institute
RFC	Response For a Class (Resposta de Classe)
WMC	Weighted Methods per Class (Métodos Ponderados por Classe)

SUMÁRIO

1 INTRODUÇÃO

Um software com qualidade é aquele que além de atender as necessidades do usuário, contempla características como: ser manutível, flexivel, reusável, entre outros. Empresas de desenvolvimento de software têm como objetivo criar produtos com qualidade, por isso é necessário medir se o produto gerado está de acordo com a qualidade esperada apresentado requisitos tais como: manutenibilidade, flexibilidade, baixo acoplamento e alta coesão.

Medir é o processo pelo qual números ou símbolos são atribuídos aos membros de uma entidade, possibilitando que abstrações sejam facilmente entendidas e controladas . Dessa forma, medir e avaliar a qualidade de um software permite o desenvolvimento de um produto melhor elaborado e de qualidade elevada.

Métricas de software são úteis para avaliar qualidade do software. Algumas métricas podem medir: complexidade, usabilidade, testabilidade e manutenibilidade do software (FENTON; PFLEEGER, 1997), porém esta avaliação é mais complexa (PRESSMAN, 2001). As métricas de software são importantes pois podem identificar problemas no software, tal como um código mau elaborado que acarreta em grande manutenibilidade.

Avidagic, Boskovic e Causevic (2008) propõem o uso de métricas para a avaliação do código-fonte utilizando a lógica *fuzzy* com o intuito de manipular problemas de incerteza e imprecisão, uma vez que a definição de requisitos de qualidade pode ser subjetiva. Sua avaliação foi no requisito de qualidade manutenibilidade do código-fonte, porém o processo é realizado de forma manual e não apresenta a correlação entre os requisitos de qualidade e as métricas.

Este trabalho usou como base a técnica aplicada por Avidagic, Boskovic e Causevic (2008), no qual criou-se a relação entre requisitos de qualidade e métricas. Esta relação foi armazenada em um banco de dados para que posteriormente seja usada como um modelo para qualquer projeto que se queira realizar a avaliação do código-fonte orientado a objeto.

A avaliação proposta por este trabalho é baseada na ideia de projeto, no qual o desenvolvedor deve selecionar um conjunto de classes compiladas (*.class*) referente a um projeto, escolher ou acrescentar os requisitos que deseja avaliar com suas respectivas métricas e criar as regras de produção da lógica *fuzzy*. A avaliação carrega o modelo armazenado no banco de dados, copia os arquivos .class para a pasta do projeto e usa a ferramenta *CKJM Extended* nesses arquivos para se obter os valores númericos associados às métricas, os quais são gravados em um arquivo que é a entrada para a ferramenta *jFuzzyLogic*, que executa os processo de *Fuzzyficação* (modelar o problema em questão para que possa ser aplicada a lógica *fuzzy*) e *Deffuzificação* (converter um termo *fuzzy* para um valor numérico).

Os resultados da avaliação do código-fonte são exibidos de forma gráfica para permitir uma melhor análise dos dados.

1.1 OBJETIVOS

O objetivo geral deste trabalho é avaliar a qualidade de um projeto orientado a objetos, desenvolvido em Java, usando métricas, requisitos de qualidade e lógica *fuzzy*. Para isto são identificados os seguintes objetivos específicos:

- Identificar métricas de software aplicadas em código-fonte.
- Correlacionar métricas e requisitos de qualidade.
- Identificar ferramentas automatizadas para métricas e lógica *fuzzy*.
- Construção de uma ferramenta automatizada para avaliação de código-fonte.
- Analisar os resultados obtidos.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos. O Capítulo 2 apresenta a importância e relaciona as métricas orientadas a objetos. O Capítulo 3 relata os conceitos sobre a lógica *fuzzy* bem como descreve o processo de *Fuzzyficação* e *Desfuzzyficação*. O Capítulo 4 aborda o processo proposto para a avaliação do código-fonte. O Capítulo 5 descreve os resultados obtidos por este trabalho de pesquisa. Por fim, Capítulo 6 apresenta as considerações finais e os trabalhos futuros que podem ser realizados a partir desta pesquisa.

2 MÉTRICAS DE SOFTWARE

Este Capítulo apresenta uma breve descrição do que são métricas de software. A seção 2.1 relata a importância do uso de métricas de software. A seção 2.2 apresenta como são aplicadas métricas no desenvolvimento de software para que se garanta ou aprimore o controle do processo de desenvolvimento de software. A seção 2.3 descreve as métricas de software orientado a objetos, bem como cita quais são as mais utilizadas e seus respectivos valores de referência.

2.1 IMPORTÂNCIA DAS MÉTRICAS DE SOFTWARE

Uma das finalidades da engenharia de software é controlar os processos de desenvolvimento do software em suas diferentes etapas, a fim de identificar problemas comuns ao desenvolvimento. Fenton e Pfleeger (1997) citam quatro problemas frequentemente encontrados em projetos:

- Dificuldade em estabelecer metas mensuráveis para produtos de software.
- Dificuldade de entender e quantificar o custo dos componentes do projeto.
- Ausência da quantificação ou predição da qualidade dos produtos.
- Não se permite a introdução de novas tecnologias para dentro de projetos sem a confirmação de que a nova tecnologia é eficiente e efetiva.

Com o intuito de sanar os problemas identificados anteriormente, vários autores sugerem o emprego de métricas (MILLS, 1988; CHIDAMBER; KEMERER, 1994).

Métrica é um conjunto de procedimentos para a definição de escalas e métodos para medidas. O *Software Engineering Institute* (SEI) define métricas de software como a composição de medidas de uma atividade vinculada ao desenvolvimento de software (ISO/IEC9126-1, 2001).

Métricas preveem prazos e custos de projetos de software, aumentam a qualidade do produto e produtividade dos funcionários. Estes objetivos podem ser alcançados mediante maior controle no processo de desenvolvimento do produto, provendo informações que auxiliam o gerência do projeto. O gerente tem um papel fundamental para o uso eficiente das métricas, pois estas lhe oferecem indicadores de quais são os requisitos de qualidade que estão sendo atendidos, além daqueles que não estão. Considerando-se que dificilmente é possível se alcançar altos níveis de qualidade em todos os requisitos externos de qualidade, cabe ao gerente optar por quais requisitos serão preferencialmente atendidos. O processo de gerenciamento de software depende da melhoria na identificação, medida e controle dos parâmetros envolvidos no processo de desenvolvimento de software (MILLS, 1988).

A adoção de métricas de software vem da constatação de que a gerência dos projetos de software necessita de formas para avaliar o produto e os processos. Quando aplicam-se métricas no processo de desenvolvimento de software é possível obter um produto final com um nível de qualidade mais elevado, uma vez que as métricas são em última instância uma forma de controle de qualidade do produto. Assim, o gerente do projeto tem um grande benefício com o uso de métricas, pois estas lhe fornecem informações precisas sobre a arquitetura do produto (MILLS, 1988).

O emprego de métricas de software proporciona uma base quantitativa para o desenvolvimento e validação dos modelos dos processos de desenvolvimento de software. Segundo Mills (1988), uma boa métrica possibilita a medição dos parâmetros de qualidade definidos de forma simples e objetiva. A métrica ideal deve ser:

- Simples: deve ser claro o que a métrica se propõe a avaliar.
- Objetiva: um único aspecto deve ser avaliado.
- De fácil obtenção: a métrica deve ser obtida sob um custo razoável.
- Válida: a métrica deve realizar aquilo que se propõe.
- Robusta: deve permanecer imutável perante mudanças insignificantes no processo ou produto.

As métricas podem ser classificadas quanto aos critérios utilizados para determiná-las. Sob esta perspectiva, as métricas são classificadas como: *objetivas* e *subjetivas* (SOMMERVILLE, 2010). As métricas objetivas são aquelas obtidas mediante regras bem definidas, tais como número de linhas de código (LOC) produzidas, consumo de memória, velocidade de execução, etc. As métricas subjetivas avaliam aspectos qualitativos do produto.

A ISO/IEC9126-1 (2001) definiu um padrão que estabelece um *framework* que define um modelo de qualidade do software, e de acordo com cada domínio, diferentes formatações desse modelo devem ser aplicadas de acordo com o domínio em específico. Além de elencar seis características de qualidade mostradas abaixo (ISO/IEC9126-1, 2001):

- Funcionalidade: capacidade do software realizar satisfatoriamente as funções que atendam às necessidades explicitas e implícitas do projeto de software.
- Confiabilidade: capacidade do software ter um nível de desempenho especificado.
- Usabilidade: capacidade do produto ser compreendido, aprendido, operado e atraente ao usuário.
- Eficiência: capacidade do software apresentar um desempenho apropriado, relativo à quantidade de recursos usados.

- Manutenibilidade: capacidade do software ser modificado. As modificações podem incluir correções, melhorias ou adaptações devido à mudanças no ambiente, requisitos ou especificações funcionais.

- Portabilidade: capacidade do software ser transferido de um ambiente para outro.

Para medir as características de qualidade existem as métricas subjetivas, que são formadas por um conjunto de métricas objetivas. Por exemplo, manutenibilidade é um fator de qualidade que possui subfatores, tais como: simplicidade, modularidade, boa documentação do código.

Diferentes projetos podem exigir diferentes métricas e resultados, entretanto, as métricas subjetivas devem ser sempre escolhidas pelos gerentes dos projetos pois são, na maioria dos casos, as mais importantes para o cliente. Uma vez que este dificilmente se interessa por fatores internos ao programa, geralmente o cliente têm sua atenção voltada a fatores como: usabilidade, confiabilidade e integridade (GALIN, 2004), isto é, a fácil utilização e confiança nos resultados são fatores primordiais para um usuário final.

Medir o custo e o esforço necessários à construção de um software, o número de linhas de código produzido, e outras medidas objetivas são relativamente fáceis de se obter. No entanto, a qualidade, funcionalidade, eficiência e manutenibilidade são mais complexas de serem medidas, além de envolverem certo grau de subjetividade (PRESSMAN, 2001).

2.2 MÉTRICAS APLICADAS AO DESENVOLVIMENTO DE SOFTWARE

Para a aplicação de medidas no desenvolvimento de software se faz necessário identificar as entidades e os atributos que se deseja medir. No desenvolvimento de software existem três partes básicas que se relacionam entre si, sendo elas (SOMMERVILLE, 2010):

- **Processos**: coleção de atividades ligadas ao software.

- **Produtos**: todo artefato ou documento resultante de um processo.

- **Recursos**: são as entidades necessárias por um processo.

Processos estão intimamente ligados ao tempo, uma vez que a execução de um projeto baseia-se em uma sequência ordenada de processos, em que cada nova etapa se inicia após o término da anterior. O desenvolvimento de um software é baseado em uma cadeia de processos que devem ser executados.

Recursos e produtos se relacionam com o processo, pois cada processo utiliza de recursos disponíveis e produtos para a geração de novos produtos, por exemplo, pode-se gerar um Diagrama de Classe que será usado como entrada para a geração de um Diagrama de Sequência (FENTON; NEIL, 2000).

Internamente, cada uma das partes citadas apresentam dois atributos distintos: características internas e externas (PRESSMAN, 2001):

- **Atributos internos** de um produto, processo ou recurso são aqueles que podem ser facilmente medidos.
- **Requisitos de qualidade (Atributos externos)** de um produto, processo ou recurso são aqueles que se dedicam a medir a relação entre o produto, processo ou recurso com o ambiente, isto é, o comportamento do produto, processo ou recurso é tão importante quanto a própria entidade.

No caso de um software é possível dizer que grande parte da avaliação dos atributos internos pode ser feita avaliando-se o código-fonte. Assim, é possível se identificar, por exemplo, atributos internos, tais como: tamanho, número de linhas de código (LOC), complexidade ciclomática (CYCLO) e profundidade da árvore de herança (DIT) (MILLS, 1988). Dessa forma, os atributos internos contemplam uma visão quantitativa do produto, processo ou recurso. Essa *medidas diretas* são mais fáceis de serem obtidas, contanto que os objetivos e os atributos internos sejam estabelecidos previamente. No entanto, a qualidade e funcionalidade de um software somente podem ser medidas indiretamente (PRESSMAN, 2001).

É difícil medir precisamente requisitos de qualidade do software, tais como manutenibilidade, compreensibilidade e usabilidade são pois estão relacionados a experiência do usuário com o software. Esses fatores por serem subjetivos estão sujeitos a variações de acordo com a interação entre o software e quem o manipula (SOMMERVILLE, 2010).

O Quadro 1 apresenta de maneira geral como qualquer métrica de software mede um atributo seja ele interno ou externo de um produto, processo ou recurso. Além disso, é possível observar que existe uma relação entre os atributos internos e externos. Os requisitos de qualidade são aqueles em que há maior interesse em se prever, uma vez que representam a interação entre o usuário e o software. Para que atributos internos indiquem a qualidade do projeto faz-se necessário que o framework de Kitchenham (1990) seja seguido:

1. O atributo interno deve ser medido com precisão.
2. Deve haver uma relação entre o valor de um atributo que pode ser medido e o atributo externo relacionado.
3. A relação entre atributos internos e externos deve ser entendida e formalizada em termos de uma fórmula ou modelo.

Conforme é apresentado no Quadro 1, o uso de métricas de software permite a definição quantitativa das características do que se mede. Em essência, uma métrica mede algum atributo (interno ou externo) de um produto, processo, ou recurso.

Quadro 1 – Classificação das métricas

Entidades	**Métricas**	
	Atributos Internos	**Atributos Externos ou Conceito OO**
Processos		
Especificação de construção	tempo, esforço, número de mudanças necessárias	qualidade, custo, estabilidade
Design detalhado	tempo, esforço, número de falhas de especificação encontrado	custo, custo eficácia
Testes	tempo, esforço, número de falhas encontradas no código	custo, custo eficácia, estabilidade
Produtos		
Especificação	tamanho, reuso, modularidade, redundância, funcionalidade	compreensibilidade, manutenibilidade
Design	tamanho, reuso, modularidade, coesão, herança, funcionalidade	qualidade, complexidade
Código	funcionalidade, complexidade algorítmica, controle de fluxo	legibilidade, usabilidade, manutenibilidade, reusabilidade
Recursos		
Pessoal	idade, preço	produtividade, experiência, inteligência
Times	tamanho, nível de comunicação	produtividade, qualidade
Software	preço, tamanho	usabilidade, confiança

Fonte: Adaptado de Fenton e Neil (2000)

2.3 MÉTRICAS DE SOFTWARE ORIENTADO A OBJETOS

Um dos problemas enfrentados em qualquer projeto de software é o de gerenciar conceitos, métodos e dados sem perder o controle na medida em que a complexidade aumenta. Muitos engenheiros de software decidem migrar de um paradigma estruturado para um orientado a objetos por acreditar que a manutenibilidade e extensibilidade do código orientado a objetos é maior que no paradigma procedural (ou estruturado) (ARCHER; STINSON, 1995).

Segundo Booch (1994) programar comportamentos exibidos no paradigma orientado a objetos em muito se difere do paradigma procedural, uma vez que no paradigma orientado a objetos existem conceitos como objetos, classes, atributos, herança, métodos, encapsulamento, polimorfismo, etc. O foco de métricas de software orientado a objetos é o de prover medidas que significativamente ajudem na análise e *design* de sistemas de software.

A aplicação das métricas de software tradicionais não se mostra efetiva quando usada em software orientado a objetos. Um dos principais motivos para isto são: (1) projeções que relacionam o tamanho do programa e a produtividade dos engenheiros de software em sistemas estruturados não se aplicam de forma eficiente ao software orientado a objetos, (2) métricas tradicionais não consideram muitos dos aspectos de um sistema orientado a objetos, (3) a complexidade de um sistema orientado a objetos tem uma escalabidade diferente de sistemas estruturados

(BOOCH, 1994). As métricas orientadas a objetos mais aplicadas no desenvolvimento de sistemas são detalhadas nas próximas seções.

2.3.1 Métricas CK

A fundamentação teórica das métricas orientadas a objetos foi estabelecida principalmente por Chidamber e Kemerer (1994). Eles apresentam uma teoria para medir a complexidade de sistemas segundo o paradigma orientado a objetos. Um conjunto de seis métricas foram definidas e estão relacionadas a seguir:

- **DIT** (Profundidade da árvore de herança, do inglês *Depth of Inheritance Tree*), representa o número de superclasses ou classes ancestrais da classe que se observa. Quanto maior for o valor de DIT, maior será o número de atributos e métodos herdados, o que acarreta em maior complexidade do sistema. A descrição (2.1) representa a definição de como se obter o DIT.

$$DIT = \text{grau de profundade da classe na lista de herança} \tag{2.1}$$

- **NOC** (Número de filhos, do inglês *Number of Children*) essa métrica, assim como a DIT se relaciona com o conceito de hierarquia. Ela mede o número total de filhos de uma classe, isto é, o número imediato de subclasses que possuem dependência da classe observada. Um maior número de filhos representa maior reuso de código dentro do projeto. A descrição (2.2) representa a definição de como se obter o NOC.

$$NOC = \text{número imediato de descendentes de uma classe} \tag{2.2}$$

- **RFC** (Resposta de classe, do inglês *Response For a Class*) representa o número de métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe. Quanto maior for o valor para RFC mais complexa é a classe, logo é mais provável que inclua erros. A descrição (2.3) representa a definição de como se obter o RFC.

$$RFC = NL + NI \tag{2.3}$$

  Onde: NL é o número de métodos locais e NI é o número de métodos invocados pelo método local.

- **CBO** (Acoplamento entre classes de objeto, do inglês *Coupling Between Objects*), representa o número de classes que são utilizadas pela classe observada. Duas classes estão acopladas quando métodos declarados em uma das classe usa métodos ou variáveis de instância definidos pela outra. Múltiplos acessos a uma mesma classe

contam como um acesso apenas. A descrição (2.4) representa a definição de como se obter o CBO.

$$CBO = \sum_{i=1}^{TC} \textit{relação}(\textit{cliente}, C_i) \tag{2.4}$$

Onde: TC é o total de classes do sistema, *cliente* é o objeto analisado e C_i representa cada classe do sistema.

- **LCOM** (Ausência de coesão em métodos, do inglês *Lack of Cohesion in Methods*), esta métrica mede o quão relacionados estão os atributos e os métodos de uma classe. Se todos os métodos e campos se relacionam internamente à classe ou à objetos do mesmo tipo, têm-se nesse caso a situação ideal para a classe. Se LCOM é maior que 1, a classe provavelmente esta violando o princípio de única responsabilidade, isto é, a classe deve ser responsável por uma única atividade. Neste caso, o ideal é dividir a classe em uma ou mais partes. Considerando $M = (m_1, ..., m_n)$ o conjunto de métodos da classe analisada, dois métodos m_i e m_j, estão relacionados se ambos acessam pelo menos um mesmo atributo de uma delas ou se uma das classes invoca o outra (2.5):

$$LCOM = \sum_{i}^{TC} \sum_{j}^{TC} (M_i \times M_j) \tag{2.5}$$

Onde: TC é o total de classes do sistema.

- **WMC** (Métodos ponderados por classe, do inglês *Weighted Methods per Class*), mede a complexidade estática de todos os métodos, isto é, um incremento no número de métodos automaticamente aumenta a complexidade da classe. Devido ao fato de que, um maior número de fluxos possíveis que um método possui, mais difícil é o seu entendimento, e portanto é mais complicada de ser mantida. O WMC é calculado como a soma das complexidades ciclomática de McCabe para cada método local, conforme a fórmula (2.6):

$$WMC = \sum_{i}^{TC} CYCLO(M_i) \tag{2.6}$$

Onde: TC é o total de classes do sistema, $CYCLO$ é a complexidade ciclomática de McCabe, M_i é cada método local à classe analisada.

As métricas descritas anteriormente são frequentemente chamadas de CK devido o fato de os autores destas serem Chidamber e Kemerer, a primeira letra do nome de cada um deles foi escolhida em sua homenagem.

2.3.2 Métricas MOOD

Além das métricas CK, outros conjuntos de métricas dedicadas a software orientado a objetos se destacam. Abreu e Carapuça (1994) definem *Metrics for Object Oriented Design* (MOOD). Frequentemente chamadas de métricas MOOD. Elas foram desenvolvidas com o objetivo de medir as seguintes características da orientação a objetos: encapsulamento, herança, acoplamento e polimorfismo. Essas métricas são adimensionais, ou seja, independentes de tamanho e de linguagem de programação. Cada métrica varia entre 0 e 1:

- **COF** (Fator de acoplamento, do inglês *Coupling Factor*), indica o quão acoplado é o sistema. Seu valor é definido pela fórmula (2.7):

$$COF = \frac{\sum_{i=1}^{TC} AC(C_i)}{TC^2 - TC} \tag{2.7}$$

  Onde: TC é o total de classes no sistema, $AC(C_i)$ é o número de conexões aferentes da classe C_i. O numerador é o total de ligações entre as classes e o denominador é o total de possíveis ligações. Um software fortemente conectado possui baixo grau de independência entre os módulos.

- **MHF** (Fator de ocultação de método, do inglês *Method Hiding Factor*), mede o nível de encapsulamento dos métodos de um sistema. Primeiro é calculada a visibilidade dos métodos com respeito a cada classe. A MHF representa a média de métodos ocultos entre todas as classes da aplicação. A visibilidade de um método privado é sempre zero. Um método protegido age como um método público para as classes que pertencem ao mesmo pacote da classe a qual o método protegido pertence. Métodos públicos são visíveis para todas as classes do sistema. Se todas as classes são privadas o índice de MHF é 1 (ou 100%) e se todos os métodos são públicos, o MHF é igual a 0. A razão MHF pode ser vista em (2.8):

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{Md(Ci)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} Md(C_i)} \tag{2.8}$$

  Onde: $V(M_{mi}) = \frac{\sum_{i=1}^{TC} \text{é-visivel}(M_{mi}, C_j)}{(TC-1)}$ e

$$\text{é-visivel}(M_{mi}, C_j) = \begin{cases} 1, & \text{se } j \neq i \text{ e } C_j \text{ pode referenciar } M_{mi} \\ 0, & \text{caso contrário} \end{cases}$$

  Onde: TC é o total de classes do sistema, M_dC_i representa o número de métodos e construtores. Esses métodos podem ser de qualquer tipo de modificador de acesso. Eles não podem ser abstratos ou herdados.

- **AHF** (Fator de ocultação de atributo, do inglês *Attribute Hiding Factor*), mede o grau de encapsulamento dos atributos no sistema, isto é, a visibilidade dos atributos em uma classe. A invisibilidade de um atributo é calculada a partir do total de classes as quais o atributo não é visível. Um atributo é chamado de *visível* quando ele pode ser acessado por outra classe ou objeto. Os atributos podem ser "escondidos" dentro de uma classe, impedido o acesso de objetos e classes externas. Segue em (2.9) a definição de AHF:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(Ci)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)} \tag{2.9}$$

  Onde: $V(A_{mi}) = \frac{\sum_{i=1}^{TC} \text{é-visivel}(A_{mi}, C_j)}{(TC-1)}$ e

  $$\text{é-visivel}(A_{mi}, C_j) = \begin{cases} 1, & \text{se } j \neq i \text{ e } C_j \text{ pode referenciar } A_{mi} \\ 0, & \text{caso contrário} \end{cases}$$

  Onde: TC é o total de classes do sistema, $A_d(C_i)$ representa o número de atributos definidos. Esses atributos podem ser de qualquer tipo de modificador de acesso. Eles não podem ser herdados.

- **MIF** (Fator de herança de método, do inglês *Method Inheritance Factor*), definido pela razão da soma dos métodos herdados em todas as classes do sistema e o total de métodos para todas as classes. A razão MIF pode ser vista em (2.10):

$$MIF = \frac{\sum_{m=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \tag{2.10}$$

  Onde: TC é o total de classes no sistema, $M_a(C_i) = M_d(C_i) + M_i(C_i)$, $M_i(C_i)$ representa a quantidade de métodos que são herdados. Esses métodos não podem ser sobreescritos, $M_d(C_i)$ representa a quantidade de métodos não abstratos. Esses métodos podem ser de qualquer modificador de acesso, $M_a(C_i)$ representa a quantidade de métodos que podem ser invocados pela classe C_i.

- **AIF** (Fator de herança de atributos, do inglês *Attribute Inheritance Factor*), é definido pela razão da soma dos atributos herdados em todas as classes do sistema e o total de atributos para todas as classes. O numerador representa a quantidade de atributos herdados por todas as classes e o denominador representa a quantidade de atributos locais ou herdados por todas as classes do sistema. Ele expressa o nível de reuso do sistema. É indicado que se mantenha o AIF por volta dos 50%. Valores maiores

podem indicar um nível de herança que pode reduzir a possibilidade de reuso. A razão AIF pode ser vista em (2.11):

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} \tag{2.11}$$

$$\text{Onde: } A_a(C_i) = A_d(C_i) + A_i(C_i)$$

TC é o total de classes no sistema, $A_i(C_i)$ é a quantidade de atributos que são herdados, A_dC_i é a quantidade de atributos definidos. Esses atributos podem ser de qualquer tipo de modificador de acesso e A_aC_i é a quantidade de atributos que podem ser referenciados pela classe C_i.

- **PF** (Fator de polimorfismo, do inglês *Polymorphism Factor*), é definido como a razão de possíveis situações reais de polimorfismo diferente para uma classe e o número máximo de situações possíveis de polimorfismo distintas. A razão POF pode ser vista em (2.12):

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_{i)}}{\sum_{i=1}^{TC} [M_n \times DC(C_i)]} \tag{2.12}$$

Onde: TC é o total de classes no sistema, $M_o(C_i)$ representa a quantidade de métodos na classe C_i, $M_n(C_i)$ refere-se aos novos métodos na classe. C_i, $DC(C_i)$ representa o número de descendentes da classe C_i.

Tanto as métricas CK quanto as MOOD são utilizadas sob a perspectiva de garantia de qualidade do software. O emprego destas métricas pode ser utilizado como um indicador do nível de qualidade do produto. E são os conjuntos de métricas orientadas a objetos para a análise de código. A seguir são descritos os principais motivos para o emprego dessas métricas.

2.3.3 Herança

Em geral, é possível se afirmar que a herança diminui a complexidade dos sistema por reduzir o número de operações e operadores, mas aumenta o nível de abstração dos objetos. As duas métricas usadas para medir a herança são a profundidade e o propagação de herança entre as classes.

1. Profundidade da árvore de herança (DIT) (2.1): um sistema orientado a objetos bem estruturado deve possuir uma floresta de classes ao invés de uma única linhagem de

herança. Conforme a profundidade de uma classe aumenta na hierarquia, maior será o número de métodos que ela deve herdar, portanto, mais complexo se torna a predição de seu comportamento.

2. Número de filhos (NOC) (2.2): classes com muitos filhos são consideradas difíceis de se modificar, e em geral, requerem mais testes por causa dos efeitos que mudanças provocam nos filhos.

2.3.4 Coesão

Coesão se refere ao quão relacionadas as operações realizadas em uma classe estão entre si. Coesão de uma classe é o grau ao qual os métodos locais estão relacionados as variáveis locais à classe. A métrica LCOM (2.5) mede o nível de coesão interna à classe. Um alto nível de coesão indica um bom nível de subdivisão. Um baixo nível de coesão aumenta a complexidade, portanto, aumenta a probabilidade de erros durante o processo de desenvolvimento. Classes com um baixo nível de coesão devem ser subdivididas em duas ou mais classes. A métrica LCOM também avalia a reusabilidade e coesão do sistema (NCSU, 2013).

2.3.5 Acoplamento

Acoplamento é a medida da interdependência entre os módulos da estrutura do software. Por exemplo, dois objetos x e y estão acoplados se um método de um objeto do tipo x invoca um método ou acessa uma variável do objeto y. O fator de acoplamento (COF) (2.7) é uma medida do acoplamento geral do sistema. Sobre o fator de acoplamento é possível afirmar:

1. Um artefato individual é de mais fácil manutenção e permite alterações pontuais sem grandes ajustes ao sistema como um todo para se adequar ao novo artefato.

2. Um sistema com alto grau de acoplamento é, por definição, sensível a mudanças e a propagação de defeitos entre os artefatos.

3. Um sistema com alto grau de acoplamento exige um maior número de testes para garantir a plena funcionalidade do sistema.

Por fim, é possível assegurar que ao se manter o nível de acoplamento reduzido, o sistema como um todo fica mais confiável, visto que cada classe é definida independentemente.

2.3.6 Encapsulamento

A ocultação de informação é um *design* popular dentro da orientação a objetos e a definição de que as propriedades de uma classes deve ser visíveis apenas a ela mesma é aconselhável. O conjunto de métricas MHF (2.8) e AHF (2.9) permitem avaliar o nível de encapsulamento do projeto.

Métodos devem ser encapsulados (escondidos), dentro da classe e não disponibilizados para serem acessados por qualquer objeto do sistema. A ocultação de objetos aumenta o nível de reusabilidade, além de diminuir a complexidade. Portanto, um fator de ocultação de métodos elevado é preferível.

O mesmo conceito aplicado aos métodos se usa para os atributos, isto é, os atributos devem ser escondidos de objetos externos à classe mantedora do atributo. Assim sendo, um fator de ocultação de atributos elevado é preferível.

2.3.7 Outras Métricas Orientadas a Objeto

Além das métricas CK e MOOD, também foram estudadas outras que também são utilizadas com frequência no contexto de avaliação de software:

- **CC** (Número de chamadas, do inglês *Number of Calls*), calcula o número de métodos e atributos internos e externos (à classe observada) usados por um método. Membros externos são os que pertencem a uma classe que não se relaciona com a classe observada hierarquicamente. Os membros internos são os que pertencem a mesma classe.

- **CYCLO** (Complexidade ciclomática, do inglês *Cyclomatic Complexity*), desenvolvida pelo pesquisador McCabe, complexidade ciclomática é utilizada para avaliar a complexidade de um classe, isto é, o número de caminhos linearmente independentes no método analisado. A métrica pode ser expressa na forma $McCabe = E - N + P$. Onde E é o número de arestas no grafo, N é o número de nós no grafo e P é a quantidade de componentes conectados neste grafo.

- **LOC** (Linhas de código, do inglês *Lines of Code*), calcula o número de linhas efetiva dentro de um método ou classe. Comentários ou linhas vazias geralmente não são consideradas.

A métrica LOC é uma das mais importantes métricas criadas, principalmente pelo fato desta ser uma das primeiras métricas desenvolvidas. Esta é utilizada para medir custos e se relaciona com a ideia de tamanho, portanto, seu uso em um projeto pode beneficiar outros

projetos, isto é, se um módulo em um projeto teve um tamanho t, em um próximo projeto que contenha a necessidade de desenvolvimento semelhante deve conter um número similar a t linhas de código (SOMMERVILLE, 2010).

A Figura 1 mostra a relação entre os Requisitos de qualidade e as métricas que podem ser usadas para medi-la.

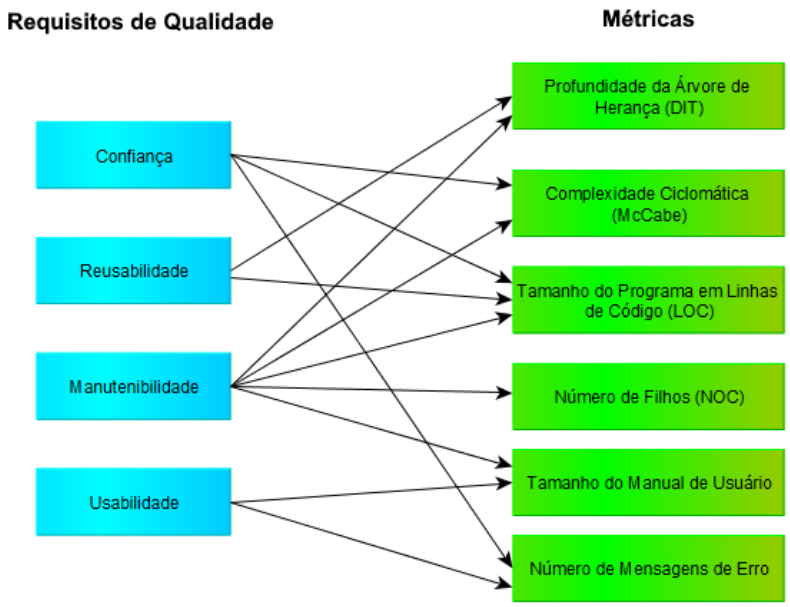

Figura 1 – Relação entre requisitos de qualidade e métricas
Fonte: Adaptado de (SOMMERVILLE, 2010)

As métricas apresentadas neste Capítulo são dedicadas ao software, isto é, este trabalho estuda métricas e sua relação com requisitos de qualidade como base para a avaliação do nível em que um código-fonte orientado a objetos atende aos diferentes requisitos de qualidade. Métricas que a avaliam o processo de desenvolvimento de software, como por exemplo, a produtividade do programador não fazem parte do escopo deste trabalho.

2.3.8 Valores de Referência das Métricas

Os valores de referência para as métricas são importantes porque permitem verificar em um código-fonte se os requisitos de qualidade são bons, regulares ou ruins. Filo (2003) definiu em sua pesquisa valores de referência para o conjunto de métricas CK, além da métrica CCLYO. Esses valores são apresentados no Quadro 2.

Observa-se que no Quadro 2 foram calculados os limites aos quais se aplicam as métricas. Filo (2003) utilizou da notação “MLOC” na tabela, sendo que esta métrica significa "Linhas de Código por Método”, portanto MLOC e LOC podem ser consideradas métricas similares. Os valores de referência para as métricas CBO e RFC não foram definidos por Filo (2003), estas

Quadro 2 – Valores de referência das métricas CK e da métrica CYCLO

Métrica	Bom (Frequente)	Regular (Ocasional)	Ruim (Raro)
CYCLO	CYCLO <7	7 <CYCLO <10	CLYCLO >10
CBO	CBO <7	7 <CBO <14	CBO >14
DIT	DIT <2	2 <DIT <4	DIT >4
MLOC	MLOC <10	10 <MLOC <30	MLOC >30
LCOM	LCOM <0.167	0.167 <LCOM <0.725	LCOM >0.725
NOC	NOC <10	10 <NOC <28	NOC >28
RFC	RFC <5	5 <RFC <15	RFC >15
WMC	WMC <11	11 <WMC <34	WMC >34

Fonte: Adaptado de (FILO, 2003)

foram definidos mediante leitura de outros trabalhos relacionados como, por exemplo, Chidamber e Kemerer (1994).

Chauhan *et al.* (2014) pesquisaram os conjuntos de métricas MOOD e CK. A partir do resultado de sua pesquisa foram definidos os valores de referência para estas métricas, conforme apresentado no Quadro 3.

Quadro 3 – Valores de referência das métricas MOOD

Métrica	Bom (%)	Regular (%)	Ruim (%)
MHF	MHF >80	80 >MHF >20	MHF <20
AHF	AHF >80	80 >AHF >20	AHF <20
MIF	MIF <30	30 <MIF <60	MIF >60
AIF	AIF >30	30 <AIF <60	AIF >60
PF	PF >10	10 <PF <25	PF >75
CF	CF >10	10 <CF <25	CF >75

Fonte: Adaptado de (CHAUHAN *et al.*, 2014)

Estes valores de referência são base para a avaliação do código-fonte proposto neste trabalho e utilizados durante o processo de *Fuzzyficação* e *Desffuzyficação* da lógica *fuzzy*, apresentados no próximo Capítulo.

3 LÓGICA *FUZZY*

Este Capítulo apresenta conceitos sobre lógica *fuzzy* importantes para o desenvolvimento deste trabalho. A seção 3.1 relata o histórico da lógica *fuzzy*, bem como pontos em que a lógica *fuzzy* se diferencia da lógica clássica proposicional. A seção 3.2 apresenta o que são variáveis linguísticas. A seção 3.3 descreve como é realizada a modelagem de incertezas por conjuntos *fuzzy*. A seção 3.4 apresenta as regras *fuzzy* e sua estrutura. A seção 3.5 relata sobre os modelos linguísticos *fuzzy*. A seção 3.6 aborda a aplicação da lógica *fuzzy* em métricas de software. A seção 3.7 relata sobre os trabalhos relacionados a esta pesquisa.

3.1 PRINCÍPIOS BÁSICOS DA LÓGICA *FUZZY*

A teoria dos conjuntos *fuzzy* foi apresentada em 1964 por Lofti A. Zadeh (1965), professor do Departamento de Engenharia Elétrica e Ciências da Computação da Universidade da California, em Berkley, quando estudava problemas de classificação de conjuntos, em especial conjuntos cuja fronteiras não são bem definidas, isto é, a transição entre um conjunto e outro é suave. *Fuzzy* difere da lógica clássica por lidar com raciocínio aproximado ao invés de exato, além de tratar de informações imprecisas ou incertas. E a partir de cálculos matemáticos é possível que de um estado de incerteza seja gerado conhecimento e afirma-se que o estudo de lógica *fuzzy* inclui o conhecimento dos conjuntos *fuzzy* além da lógica em si, uma vez que existe uma relação íntima entre ambos.

Zadeh (1965) afirma que na teoria dos conjuntos clássicos, um conjunto contém elementos que, devido as suas características satisfazem precisamente as regras para pertencer ao conjunto. Por sua vez, os conjuntos *fuzzy* contém objetos que são admitidos ao conjunto devido a aproximação. Dessa forma, segundo a lógica clássica, pode-se afirmar que dado um conjunto C e um elemento p pertencente ao universo U, se admite que $p \in C$ ou $p \notin C$. Pode-se, por exemplo, afirmar que um indivíduo de 20 anos de idade *pertence* ao conjunto dos *jovens* e que um indivíduo aos 50 anos de idade *não pertence* a este mesmo conjunto. Ambas as classificações são simples e binárias. Entretanto, pode haver discordância ao se avaliar se a idade 27 deve ou não pertencer ao conjunto dos *jovens*.

Zadeh (1975b) definiu a ideia de flexibilização do conceito de pertinência de elementos a conjuntos, criando assim a noção de *grau de pertinência*. A partir de uma função de pertinência é possível se atribuir o grau de pertinência de um elemento a um conjunto em específico. Isto significa que um elemento pode pertencer parcialmente ao conjunto, isto é, de fato o elemento é considerado compatível, a um certo grau ao conjunto observado. Esse grau de associação entre um elemento p e um conjunto C é chamado de grau de pertinência. Assim, pode-se afirmar que o indivíduo aos 27 anos de idade pertence com certo grau ao conjunto dos *jovens*.

Para a definição dos elementos pertencentes ao conjunto de pessoas *jovens*, existe uma subjetividade intrínseca ao problema de classificação. Por exemplo, dado um conjunto *fuzzy* C_j representante das pessoas *jovens*, o grau de pertinência de um recém-nascido é muito baixo ou inexistente ao conjunto C_j, assim como alguém aos 13 anos de idade têm um grau de pertinência superior ao recém-nascido no conjunto C_j, e um indivíduo aos 17 anos de idade, um grau de pertinência maior que o de 13 anos. Enquanto na lógica clássica a variação entre o pertencimento ou não ao um conjunto é simples, variaando binariamente com valores do tipo "sim ou não", "0 ou 1", na lógica *fuzzy* existem os valores intermediários, admitindo valores dentro do intervalo entre 0 e 1. A Figura 2 representa o grau de pertinência de diferentes idades ao conjunto dos indivíduos jovens.

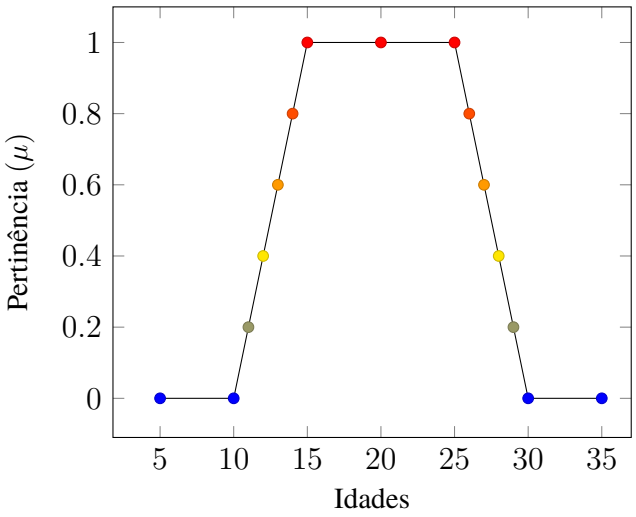

Figura 2 – Grau de pertinência do conjunto de diferentes idades ao conjunto *jovem*
Fonte: Autoria própria

Segundo o conceito de lógica *fuzzy* não existe uma separação abrupta dos elementos entre um conjunto e outro, o que permite a interpretação de que não existe uma dualidade restrita ao elementos de qualquer conjunto (ROSS, 2010). Por exemplo, ao se estabelecer dois grupos de indivíduos, separados de acordo com a idade de seus integrantes, o primeiro grupo é o grupo de jovens C_j e o segundo de adultos C_a. Segundo o lógica clássica qualquer elemento x que pertencesse ao conjunto C_j, não pertence ao conjunto C_a, uma vez que existe uma clara distinção entre os elementos de cada grupo. No entanto, segundo a abordagem *fuzzy* os diferentes conjuntos atuam de maneira complementar, isto é, não existe uma restrição quanto a pertinência dos elementos aos grupos, mesmo pareçam estritamente diferentes (ZADEH, 1965). Por exemplo, ao classificar um indivíduo de 29 anos de idade, ele terá um alto grau de pertinência ao conjunto C_a, embora também pertença à C_j. A variação do grau de pertinência de cada elemento varia com a situação ou domínio de quem modela o sistema (ZADEH, 1978).

Deve-se levar em consideração que os limites de cada conjunto são estipulados pelo especialista do domínio. Dessa forma, embora a Figura 2 mostre um conjunto C_j com um domínio que varia entre entre 10 e 30 anos, um especialista pode definir um outro conjunto para representar

essas faixas de idade ou mesmo alterar o domínio de C_j , uma vez que os limites de um conjunto *fuzzy* variam de pessoa para pessoa.

3.2 VARIÁVEL LINGUÍSTICA

Lee (2005) define variável linguística como uma quíntupla definida em (3.1):

$$\textit{variável linguística} = (x, T(x), \cup, G, M) \tag{3.1}$$

Onde:

x: nome da variável.

$T(x)$: conjunto de valores linguísticos de x.

$\cup$: universo de discurso em que se define $T(x)$.

G: regra sintática para gerar os nomes dos termos relacionados à x.

M: regra semântica para associar significados e valores.

Além dos cálculos matemáticos e análise lógica, ou antes mesmo que estes sejam iniciados, é importante se notar que os conjuntos *fuzzy* necessariamente se relacionam com a linguagem natural. A comunicação, embora constantemente vaga e ambígua, gera poucos casos de problemas no entendimento (ROSS, 2010). Por exemplo, quando se diz que uma pessoa é "alta" não existe uma altura a partir da qual uma pessoa é considerada alta, este termo pode ter diferentes significados de acordo com a região, idade ou nacionalidade das pessoas. Zadeh (1978) afirma que uma proposição lógica *fuzzy* é uma afirmação que envolve uma ideia sem muitas restrições. A variável linguística expressa ideias de forma subjetivas e que podem ser interpretadas de maneira diferente por vários indivíduos.

Zadeh (1965) afirma que a linguagem natural apresenta termos fundamentais chamados átomos. Um conjunto de átomos formam moléculas ou frase da linguagem natural. Termos fundamentais são reconhecidos como termos atômicos, por exemplo, *pouco*, *rápido*, *alto*, *jovem*. Devido ao fato desses termos serem vagos, ele considera ainda que estes podem ser representados como conjuntos *fuzzy*. Termos atômicos são conhecidos também como variável linguística.

A variável linguística *fuzzy* é uma variável que possui um valor que é expressado por um termo linguístico, além de ser representada por um conjunto de termos, em que cada um deles possui uma função de pertinência (ZADEH, 1975a). Ela possui um conjunto de termos linguísticos associados além de um domínio. A Figura 3 exemplifica o conceito de variável linguística, considerando que a variável seja *idade*. O termos linguísticos associados a *idade* são: *criança*, *jovem*, *adulto* e *idoso*. Para cada termo linguístico é definida uma função que representa quantitativamente o conceito do termo. Termos linguísticos são também chamados de *termos difusos*.

Zadeh (1978) acrescenta que o principal objetivo da lógica *fuzzy* é o de produzir uma

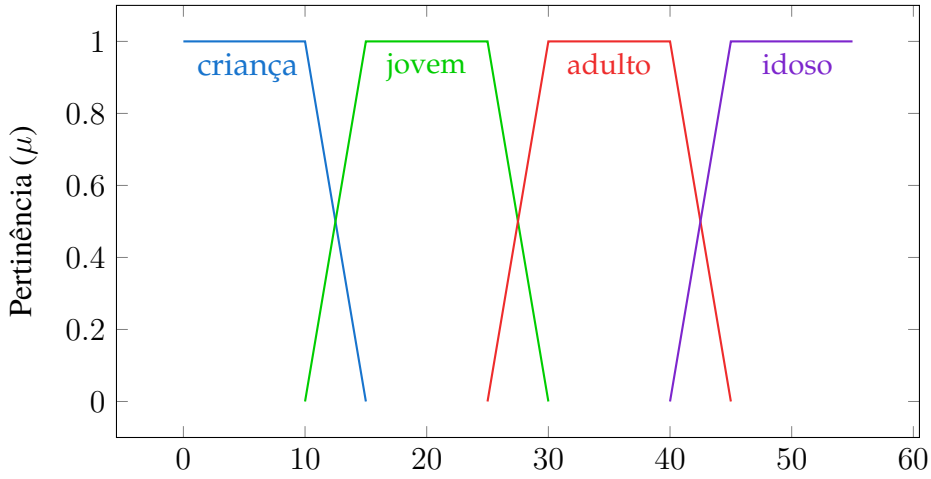

Figura 3 – Exemplo de variável linguística
Fonte: Autoria própria

fundamentação teórica para racionalizar proposições imprecisas. Uma restrição *fuzzy* pode ser entendida como um conjunto de valores que são atribuídos a uma variável. Para qualquer elemento x é feita uma avaliação de suas características. A partir desta é possível declarar se esse elemento pertence ao conjunto. A sentença (3.2) pode ser traduzida para *Idade (Ana)* = *jovem*, considerando que o conjunto *Idade* admita valores entre o intervalo $(0, 100)$, *jovem* pode ser interpretado como um subconjunto pertencente ao universo das idades.

$$\textit{Ana é } \underline{\textit{jovem}} \tag{3.2}$$

O termo linguístico *jovem* é avaliado por uma função de pertinência. A Figura 3 exibe graficamente a função relativa à este termo. Considerando que a idade de Ana seja igual a 28, pode-se afirmar que essa idade tem um grau de pertinência no valor de 0.4 ao termo *jovem*. Conforme sua idade avance, Ana pertencerá menos ao grupo jovem, transitando para o conjunto *adultos*, para depois de alguns anos pertencer ao conjunto dos *idosos*.

Para a definição da quíntupla definida anteriormente, onde a variável linguística seja *idade*, obtém-se:

$$\textit{variável linguíistica} = (\textit{idade}, \textit{T(idade)}, \cup, G, M) \tag{3.3}$$

Onde:

idade: *jovem*.

T(idade): {criança, jovem, adulto, idoso}.

$\cup$: $[0, 100]$ anos de idade.

G: define a ordem com a qual os termos serão associados á variável.

M(criança): varia entre 0 e 15.

M(jovem): varia entre 10 e 30.

M(adulto): varia entre 25 e 45.

M(idoso): a partir dos 40 anos.

3.3 OPERADORES *FUZZY*

A lógica clássica bivalente utiliza os operadores booleanos *E*, *Ou* e *Complemento* para realizar as operações de intersecção, união e complemento entre conjuntos (LEE, 2005). Esses operadores estão apresentados no Quadro 4.

Quadro 4 – Operadores lógicos

x	y	$x \wedge y$	$x \vee y$	$\neg x$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Fonte: Adaptado de (LEE, 2005)

O Quadro 4 são aplicados operadores booleanos para a representar a relação entre dois conjuntos distintos x e y. A lógica *fuzzy* utiliza conjuntos *fuzzy* (também chamados de conjuntos difusos) e também atua sobre os operadores união, interseção e complemento. A principal diferença da lógica *fuzzy* em se tratando da operação entre conjuntos é que são reconhecido valores reais entre 0 e 1.

Considerando dois conjuntos *fuzzy* X e Y, ao se aplicar o operador "união" ($\vee$) o resultado será equivalente a pertinência do maior valor de pertinência dos dois conjuntos. O operador união é geralmente representado na forma: *max*. Ao se aplicar o operador "interseção" ($\wedge$) nos conjuntos A e B o resultado será equivalente a pertinência do menor valor de pertinência dois dois conjuntos. O operador interseção é geralmente representado na forma *min*. Na operação "complemento"($\neg$) deve-se tomar o valor que complementa 1. Sendo assim, foram definidos operadores lógicos capazes de representar conjuntos *fuzzy* a partir de uma generalização dos operadores tradicionais. Essa generalização é exibida no Quadro 5 .

Quadro 5 – Operadores lógicos *fuzzy*

Operador tradicional	Operador *fuzzy*
$x \wedge y$	*min*($\mu_A(x)$,$\mu_B(y)$)
$x \vee y$	*max*($\mu_A(x)$,$\mu_B(y)$)
$\neg x$	$1 - \mu_A(x)$

Fonte: Adaptado de (LEE, 2005)

No Quadro 5 os operadores *fuzzy min* e *max* utilizam da notação μ_x e μ_y, ambas se referem ao grau de pertinência que um elemento têm a um dado termo, o símbolo μ representa o grau de pertinência do elemento à função de pertinência.

A aplicação dos operadores *fuzzy* permite a descrição de qualquer valor no intervalo do domínio do problema. Considerando por exemplo, $x = 0.2$ e $y = 0.5$, é possível se obter os seguintes resultados aplicando os operadores *fuzzy*, conforme mostra o Quadro 6.

Quadro 6 – Operadores lógicos *fuzzy*

Operador *fuzzy*	Dedução
min($\mu_A(x)$,$\mu_B(y)$)	*min*($\mu_A(x)$,$\mu_B(y)$) = *min*$(0.2, 0.5) = 0.2$
max($\mu_A(x)$,$\mu_B(y)$)	*max*($\mu_A(x)$,$\mu_B(y)$) = *max*$(0.2, 0.5) = 0.5$
$1 - \mu_A(x)$	$1 - x = 1 - 0.2 = 0.8$

Fonte: Autoria própria

Considerando dois conjuntos *fuzzy* A e B pode-se afirmar que a união entre esses conjuntos pode ser expressa na forma: $A \cup B = \{\max(\mu_A(x), \mu_B(y))\}$, a interseção é dada por: $A \cap B = \{\min(\mu_A(x), \mu_B(y))\}$, e o complemento é expresso por:$\bar{A} = \{1 - \mu_A(x)\}$.

3.4 REGRAS *FUZZY*

O emprego de regras é popular quando se utiliza lógica *fuzzy* e servem para descrever uma situação em específico a partir da qual é realizado uma inferência para se produzir um resultado. Ela atua de forma a mapear as entradas e indicar a saída correta. A partir da regra *fuzzy* é possível a modelagem direta do conhecimento desejado em termos de condições específicas. As regras são baseadas em duas partes, uma condição e uma consequência. A estrutura básica é apresentada em (3.4):

Se <*premissa*>, **Então** <*consequência*> (3.4)

Os fatores que descrevem a condição são normalmente chamados de premissa, enquanto que a segunda corresponde a resposta do sistema a um estímulo ou uma conclusão de um estado, que é denominado consequência (LEE, 2005). A *premissa* e a *consequência* se relacionam mediante uma *implicação* lógica. Esses elementos formam uma relação entre duas variáveis, essa relação entre x e y derivam na forma de uma regra apresentada no Quadro 7:

Quadro 7 – Elementos de uma implicação *fuzzy*

Premissa:	x é A
Implicação:	**Se** x é A, **Então** y é B
Consequência:	y é B

Fonte: Autoria própria

A premissa descreve a situação de uma das variáveis de entrada do sistema, isto é, ela representa a condição para que se determine um resultado ou resposta. A implicação pode ser interpretada como uma relação lógica entre x e y. Enquanto que a consequência é a descrição de uma parte dentro do domínio das variáveis de saída. Portanto, para a definição das consequências se faz necessário o domínio geral do sistema (ORTEGA, 2001).

Após definido o conjunto de regras *fuzzy* é necessária a criação de uma "máquina de inferência" para se obter a resposta final. Existem diversas formas de se definir o método de inferência, isto depende da situação em que se trabalha. Todavia, a inferência mais utilizada, principalmente para o controle de sistema é o Método Mamdani (LEE, 2005), descrito na seção 3.5.2.

Uma regra *fuzzy* geralmente assume a forma (ORTEGA, 2001):

$$R : \textbf{Se } x \text{ é } A, \textbf{Então } y \text{ é } B \tag{3.5}$$

Onde x é uma variável linguística de entrada com domínio X, y é uma variável linguística de saída com domínio Y, eles formam a parte antecedente e consequente, respectivamente. A e B são subconjuntos *fuzzy* dos conjuntos X e Y.

3.5 MODELOS LINGUÍSTICOS *FUZZY*

Os sistemas *fuzzy* são uma extensão dos sistemas clássicos, isto é, com a adição de conceitos como imprecisão e probabilidade. A utilização de conjuntos *fuzzy* possibilita que seja modelada a imprecisão dentro de um sistema. Considerando que apesar de haver conceitos nebulosos, estes são tratados de forma a simular o pensamento humano. Utilizando de técnicas como heurísticas e base de conhecimento capazes de compreender o estado atual do sistema (ORTEGA, 2001).

Segundo Ortega (2001) os modelos *fuzzy* são normalmente separados em duas categorias distintas, cada uma delas tem um propósito específico. O primeiro grupo é definido pelos *Modelos Linguisticos* (MLs), baseado no conjunto de regras do tipo *Se-Então* onde os predicados são vagos. Este modelo apresenta as quantidades associadas aos termos linguísticos, simplificando assim as relações entre a modelagem e a linguagem natural.

A segunda categoria de modelo *fuzzy* é baseada no modelo de *Takagi-Sugeno-Kang* (TSK), proposta pelos pesquisador Sugeno e sua equipe em 1974. Este método não utiliza apenas de conceitos *fuzzy* para a resolução de problemas (ROSS, 2010).

3.5.1 Modelos Linguísticos (MLs)

Estes modelos são estruturados a partir de uma base de conhecimentos e se utilizam de um conjunto de regras definidas a partir de conhecimentos heurísticos. O poder de decisão do sistema é relativo às regras que foram elaboradas e de como o sistema *fuzzy* foi definido. Existem diversos modelos de regras e raciocínios que podem ser aplicados. É possível se definir diversos tipos de inferências para relacionar as entradas e saídas de um conjunto de regras.

As regras podem ser definidas a partir do princípio de única entrada e única saída ou

SISO (do inglês, *Single Input, Single Output*) (ORTEGA, 2001). Todavia, o conjunto de regras pode seguir o estilo de múltiplos valores de entrada e múltiplos valores de saída ou MIMO (do inglês, *Multiple Input, Multiple Output*) (ORTEGA, 2001). Neste modelo são definidas as inferência para relacionar as entradas e saídas do conjunto de regras. A forma mais comum de sua representação, em se tratando de um sistema SISO é descrita no Quadro 8 (ORTEGA, 2001):

Quadro 8 – Sistema do tipo Single Input, Single Output

Se x é A_1 **Então** y é B_1 **Ou**
Se x é A_2 **Então** y é B_2 **Ou**
...
Se x é A_m **Então** y é B_m

Fonte: Adaptado de (ORTEGA, 2001)

Onde x é uma variável linguística de entrada com domínio X, y é uma variável linguística de saída com domínio Y e A_i e B_i são subconjuntos *fuzzy* dos conjuntos X e Y, respectivamente. Os valores das variáveis A_i e B_i são em muitos casos associados a termos linguísticos, como por exemplo: *muito*, *muito pouco*, *pequeno*, *alto*, *grande*. É importante lembrar que no modelo linguístico todas as condições são avaliadas. Este sistema busca mapear para cada valor de entrada um resultado correspondente para a saída. Assim, o método de inferência define a transformação de um valor *fuzzy* de entrada em um valor de saída.

3.5.2 Mecanismos de Inferência

Assumindo que uma regra *fuzzy* pode ser descrita como uma adaptação do modelo de regra definido no Quadro 8, é possível se obter uma representação mais simplificada desta regra *fuzzy* pela seguinte expressão (3.6) (LEE, 2005).

$$R : A \rightarrow B \tag{3.6}$$

Baseado na interpretação de produtos cartesianos, R pode ser considerado uma relação bidimensional entre as funções de pertinência (3.7) (ORTEGA, 2001):

$$\mu_{Rel}(x, y) = f(\mu_A(x), \mu_B(y)) \tag{3.7}$$

Onde a função f, chamada de "função de implicação *fuzzy* ", realiza a operação de transformação entre o grau de pertinência de x em A e y em B para a relação de (x, y) em $A \times B$. Assim é introduzida a noção de função de uma implicação *fuzzy*.

O Modelo Mamdani é utilizado no ramo da engenharia e automação de sistemas e oferece uma proposta de implicação *fuzzy* simples e frequentemente utilizada. Neste modelo

linguístico as regras, são descritas na forma (3.8) (ORTEGA, 2001):

$$\mu_{Rel}(x, y) = f(\mu_A(x) \wedge \mu_B(y)) = (min\{\mu_A(x), \mu_B(y)\} \tag{3.8}$$

O operador $\wedge$ é representado pela operação de conjunção *fuzzy min*. A agregação entre as regras de um conjunto é realizada pelo operador "União", isto é, se aplica o operador de disjunção *fuzzy* (*max*) entre as regras. Esta relação é chamada composição *min-max*. A equação (3.9) representa a disjunção entre as regras, onde se busca o maior valor possível por meio do operador *max* (MIRANDA; JUNIOR; KRONBAUER, 2003).

$$\sum_{i=1}^{n} max[x_i \wedge y_i] \tag{3.9}$$

Em se tratando de um conjunto de regras do tipo SISO é possível se mapear para cada termo em x um valor correspondente em y.

3.5.3 Modelo Linguístico de Múltiplas Variáveis

O modelo linguístico de múltiplas variáveis de entradas e uma única variável de saída, ou MISO (do inglês, *Multiple Input, Single Output*) é o tipo de modelo linguístico mais utilizado pela capacidade de representação de problemas. Esse modelo têm como regra o Quadro 9 (ORTEGA, 2001):

Quadro 9 – Sistema do tipo Muiltiple Input, Single Output

Se x é A_1 **E** y é B_1 **Então** z é C_1 **Ou**
Se x é A_2 **E** y é B_2 **Então** z é C_2 **Ou**
...
Se x é A_m **E** y é B_m **Então** z é C_m

Fonte: Adaptado de (ORTEGA, 2001)

Onde x e y são as variáveis linguísticas de entrada e z é a variável linguística de saída. A_i, B_i e C_i são os subconjuntos *fuzzy* dos universos X, Y e Z, respectivamente. Assim como no modelo SISO, cada regra possui uma relação *fuzzy* R que pode ser representada na forma (3.10) (ORTEGA, 2001):

$$\mu_{Rel}(x, y, z) = [\mu_A(x) \wedge \mu_B(y)] \rightarrow \mu_C(z) \tag{3.10}$$

A Figura 4 exemplifica a aplicação do método de inferência Mamdani aplicando uma composição *min-max* do tipo MISO onde uma regra *fuzzy* passa por um sistema de inferência. O sistema é composto por duas variáveis de entrada x e y e uma variável de saída z (CHEN, 2013).

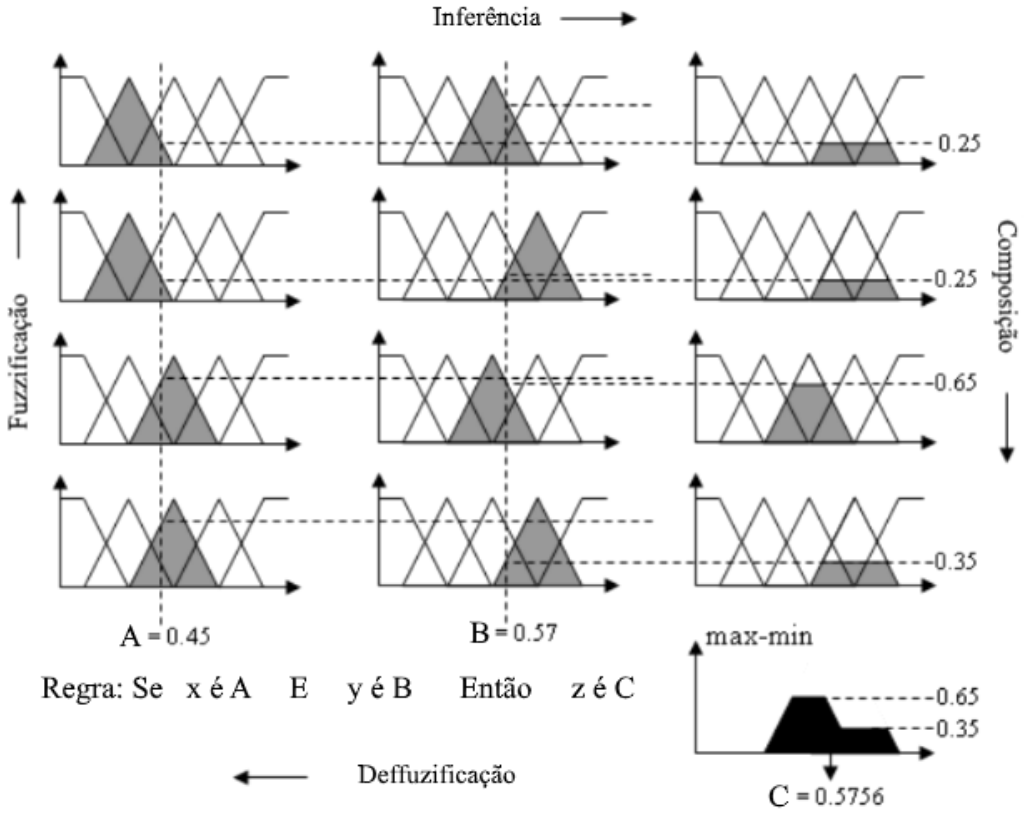

Figura 4 – Método de inferência Mamdani utilizando *min-max*
Fonte: Adaptado de (CHEN, 2013)

A associação entre as variáveis formam as regras do sistema. Para se obter os valores de z é inicialmente aplicado o operador *min* em cada uma das relações entre A e B, para cada variável linguística admitida em A e B é definido obtido o valor correspondente em C, isto é, o operador *min* nas variáveis de entrada de A possui duas funções de pertinência compatíveis ao valor 0.45. A variável B também possui duas funções de pertinência compatíveis ao valor 0.57. Assim, é realizado o produto cartesiano entre as variáveis de entrada para gerar os valores admitidos em C. Neste caso, o produto cartesiano $A \times B = 4$, portanto, serão geradas 4 funções em C. Aos valores obtidos em C é aplicado o operador *max*, esta etapa é chamada de *composição*. Por fim, é definida a região *fuzzy* correspondente aos valores de entrada indicados e a área gerada pela operação *max* deve ser realizada na etapa de *Deffuzyficação*, isto é, é calculado um valor *crisp* a partir de um conceito *fuzzy*.

Uma outra forma de aplicação de implicação *fuzzy* popular é método Larsen, que pode ser compreendido como uma adaptação do modelo *min-max*. Neste modelo se aplica a composição *max-produto* para se obter z, ocorre uma adaptação da relação descrita em (3.10), substituindo o operador *fuzzy* ***E*** ($\wedge$) pela operação multiplicação entre as variáveis de entrada: $\sum_{i=1}^{n} max[x_i \cdot y_i]$ (CHEN, 2013).

O processo de inferência gera um valor linguístico para a variável de saída e este termo deve ser interpretado, pois a saída de um sistema *fuzzy* que utiliza o método de inferência Mamdani é um conjunto *fuzzy*. A tradução de um termo *fuzzy* para um valor real (*crisp*) é chamada de

Defuzzyficação (ORTEGA, 2001).

3.5.4 Métodos de *Defuzzyficação*

A *Defuzzyficação* é uma forma de interpretação de um termo vago para um valor físico, na maioria dos casos esse valor é numérico. Embora o termo linguístico esteja definido dentro das variáveis de saída, ainda é possível uma variação dos valores reais de acordo com a estratégia seguida. A Figura 5 exibe as estratégias de *Deffuzyficação* que podem ser utilizadas na obtenção da saída *crisp* (CHEN, 2013).

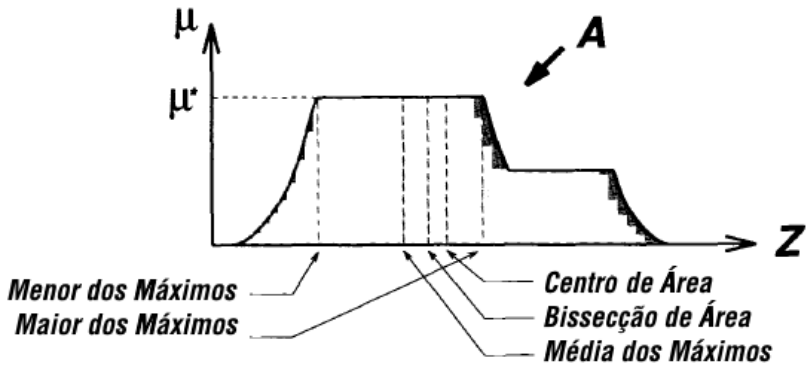

Figura 5 – Métodos de *Defuzzyficação*
Fonte: Adaptado de (CHEN, 2013)

Segue uma breve explicação das estratégias mais comuns de *Desffuzzyficação*:

- **Centro de Área**, esta é a técnica de *Defuzzyficação* mais popularmente utilizada. Esse método considera toda a distribuição de possibilidade de saída do modelo expresso por (3.11):

$$z_{CA} = \frac{\int_Z \mu_A(z) z dz}{\int_Z \mu_A(z) dz} \tag{3.11}$$

  Onde: $\mu_A(z)$ representa a agregação das saídas da função de pertinência.

- **Média dos Máximos**, esta técnica calcula a média de todos os valores de saída que tenham os maiores graus de possibilidade (3.12).

$$z_{MM} = \frac{\int_{Z'} z dz}{\int_{Z'} dz} \tag{3.12}$$

  Onde: $z' = z|\mu_A(z) = \mu*$. Em particular, se $\mu_A(z)$ possui um único máximo em z, então $z_{mm} = z*$. Entretanto se $\mu_A(z)$ alcança seu máximo em qualquer z $\in [z(esquerda), z(direita)]$, então $z_{mm} = (z(esquerda) + z(direita))/2$.

3.5.5 Arquitetura de um Sistema *Fuzzy*

A estrutura de um controlador *fuzzy* é apresentada na Figura 6, composta pelas variáveis reais, representadas pelo termo *valores crisp*, são utilizadas como entrada de um sistema *fuzzy*. A *Entrada do Sistema* representa os valores numéricos com os quais é possível se inicializar o sistema *fuzzy*. Essas medidas do ambiente passam por um processo de *Fuzzyficação* para que possam ser representadas na forma de variáveis e termos linguísticos *fuzzy*. Nesta etapa também são definidas as relações entre as funções de pertinências e os diferentes valores admitidos dentro do domínio do sistema, definindo assim o conjunto de regras que formam a *Base de Regras*.

Mediante um sistema de inferência atuando em conjunto com a *Base de Regras*, se produz um valor. Esse valor de saída, no entanto não representa um valor numérico, mas um "gráfico" formado pela emprego de um operado lógico do tipo *OU*, o que provoca na "soma" de todas as regras da base de regras aplicando como entrada as funções de pertinência. Este conjunto *fuzzy* por não representar um valor numérico necessita ser interpretado dentro de um conjunto *fuzzy*, portanto, à variável *fuzzy* de saída é aplicado o processo de *Deffuzyficação* para que o resultado seja representado por um valor preciso.

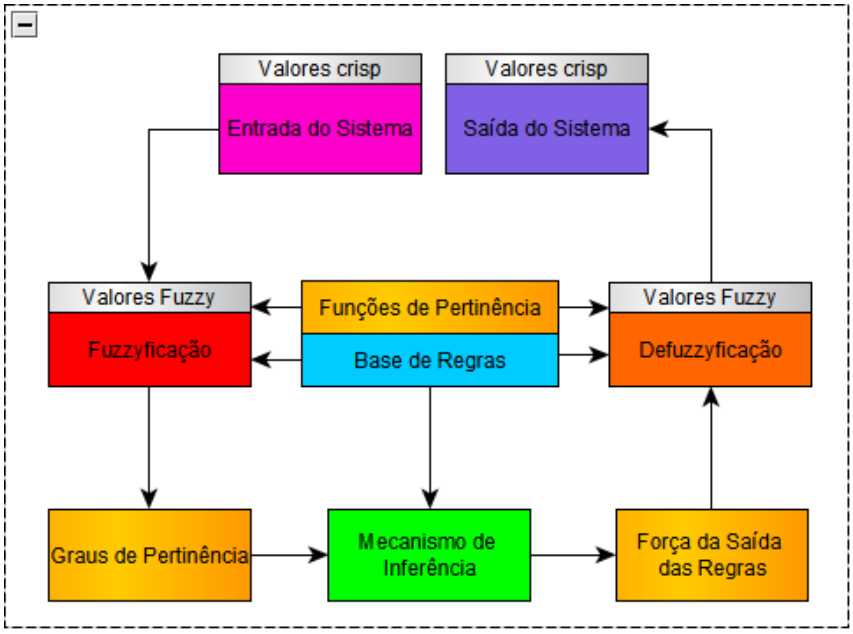

Figura 6 – Estrutura conceitual de um controlador *fuzzy*
Fonte: Adaptado de (MIRANDA; JUNIOR; KRONBAUER, 2003)

De acordo com os valores de entrada do sistema, algumas regras podem ter uma maior influência para o caso específico, isto é, valores *crisp* que se aproximam de uma determinada regra tornam essa regra mais "forte". Caso a entrada do sistema seja diferente de uma regra x, essa regra terá uma menor influência no resultado final do sistema. Esta etapa é chamada na Figura 6 de *Força da Saída das Regras*, ou seja, é calculada a importância individual de cada regra.

Um sistema *fuzzy* atua por meio de inferência *fuzzy* que são obtidas a partir da modelagem do conhecimento de um especialista no domínio. A Base de Regras é então, um conjunto de informações primordiais para que um sistema *fuzzy* atue de maneira coerente. Um sistema de inferência *fuzzy* é normalmente chamado de FIS (do inglês, *Fuzzy Inference System*. Em um FIS são definidos os conjuntos *Fuzzy*, os operadores lógicos e a base de conhecimento.

Os *Mecanismos de Inferência* é o processo de formulação do mapeamento entre uma dada entrada e uma saída utilizando lógica *fuzzy*. O mapeamento permite que decisões sejam tomadas. Os mecanismos de inferência envolvem a relação entre as funções de pertinência, operadores lógicos e o conjunto de regras.

3.6 SITUAÇÃO-EXEMPLO DO USO DE LÓGICA *FUZZY*

O processo de avaliar quantitativamente o software com valores precisos é custoso, principalmente pelo fato de envolver um conhecimento sobre o domínio que muitos profissionais não possuem. Além de ser uma atividade complexa, a classificação de partes específicas de um software é uma atividade subjetiva e deve ser realizada cuidadosamente para garantir que haja um diagnóstico preciso.

Um exemplo de como pode ser difícil mensurar quantitativamente um software é a definição de um método “longo”. Quantas linhas teria tal método? Mesmo que para um determinado projeto se defina qual o tamanho de um método longo, talvez num próximo projeto um método longo tenha o dobro do número de linhas, uma vez que na média desse segundo projeto, os métodos são mais extensos. É difícil também definir se um requisito de qualidade está sendo atendido adequadamente, uma vez que um ele é composto por diversos fatores diferentes.

O emprego de lógica *fuzzy* pode ser uma solução para este tipo de situação, porque pode tratar de problemas que envolvem incerteza e imprecisão. Dessa forma, não faz-se necessário que hajam “verdades absolutas” quanto ao tamanho de um método por exemplo, um método pode ser “quase longo”, ou parte “longo” e parte “muito longo”. A incerteza do problema de “qual deve ser o tamanho de um método muito longo” é levada em conta dentro de um sistema *fuzzy* de tal forma que são admitidos valores parciais dentro dos conjuntos *fuzzy*.

Além da capacidade de adequação ao problema, outro ponto positivo do emprego de lógica *fuzzy* é a possibilidade de alterar o sistema de forma bastante simplificada. Uma vez que o sistema esteja estabelecido e haja a necessidade de alteração dos valores, será necessário somente uma atualização destes (conjuntos ou método de Deffuzificação) dentro do sistema, entretanto, a lógica permanecerá a mesma, ou seja, o sistema pode ser facilmente adaptado para diferentes situações.

Desenvolvedores tentam criar códigos efetivos e limpos. Um código limpo é desejável pois este é de fácil manutenibilidade, legibilidade e reuso. Ao serem aplicadas técnicas para

a classificação do código, é possível atingir um padrão de qualidade no desenvolvimento do projeto. O grande desafio em se aplicar as métricas é definir qual é o valor ideal para cada uma das métrica selecionadas.

3.6.1 Aplicação da Lógica *Fuzzy* utilizando Métricas de Software

A lógica *fuzzy* possibilita a representação linguística do conhecimento para uma execução de uma estratégia de controle. Por exemplo, em uma empresa, o gerente de projetos estabelece que para se desenvolver um qualidade os programadores devem seguir as estratégias de:

1. Escrever métodos de forma concisa.

2. Definir uma quantidade reduzida de parâmetros para um método.

A fim de automatizar a avaliação da qualidade do código-fonte, considerando os dois objetivos descritos anteriormente, pode-se utilizar lógica *fuzzy* e os seguintes de qualidade seriam: *tamanho* e *número de parâmetros de um método*, respectivamente. É possível definir algumas regras do tipo *Se-Então* utilizando o conectivo lógico E como uma estratégia de controle para a avaliação automatizada da qualidade do código produzido. A partir da definição feita por um especialista no domínio, no caso, a partir de uma consulta ao gerente do projeto, é possível derivar algumas regras, conforme apresenta o Quadro 10.

Quadro 10 – Regras para a avaliação da qualidade dos métodos

Regra	SE	E	ENTÃO
Situação 1	Tamanho do método α é *longo*	Número de parâmetros β é *alto*	A avaliação do método γ é *preocupante*
Situação 2	Tamanho do método α é *curto*	Número de parâmetros β é *médio*	A avaliação do método γ é *bom*
Situação 3	Tamanho do método α é *curto*	Número de parâmetros β é *muito baixo*	A avaliação do método γ é *excelente*
Situação 4	Tamanho do método α é *médio*	Número de parâmetros β é *médio*	A avaliação do método γ é *bom*

Fonte: Autoria própria

As regras definidas no Quadro 10 são baseadas em termos linguísticos. A variável *Tamanho do método* tem como termos associados: *curto*, *médio*, *longo* e *muito longo*. Enquanto que a variável *Número de parâmetros* possui os seguintes termos associados: *baixo*, *médio*, *alto* e *muito alto*. A parte consequente da regra gera a variável *Qualidade do método*, que por sua vez possui as seguintes funções de pertinência: *excelente*, *muito bom*, *bom*, *regular*, *preocupante*. A Figura 7 ilustra as funções de pertinência referentes ao conjunto *fuzzy* de qualidade do método.

Observa-se que segundo a definição de qualidade do método exibida na Figura 7, para que se obtenha uma boa avaliação é necessário que o valor seja o mais baixo possível. Onde 0 é o

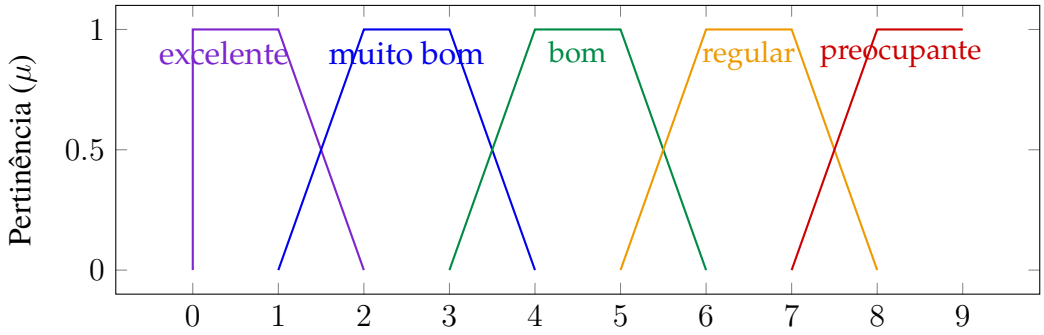

Figura 7 – Representação do conjunto *fuzzy*: Qualidade do método
Fonte: Autoria própria

valor ideal, e a partir de 7 é considerado que a qualidade do método é preocupante. Esses valores são obtidos mediante a entrada do tamanho e número de parâmetros do método e de acordo com os valores de entrada é definida a qualidade do método. Considerando que são duas as variáveis de entrada e uma variável de saída, neste caso, utiliza-se um modelo linguístico de múltiplas variáveis, tal como MISO (seção 3.5.1). Considerando o modelo definido na Expressão 3.13, pode-se estabelecer regras do tipo:

$$\textbf{Se } \textit{tamanho} \text{ é } A \textbf{ E } \textit{número de parâmetros} \text{ é } B \textbf{ Então } \textit{qualidade} \text{ é } C \tag{3.13}$$

Regras tal como a definida em (3.13), com apenas duas variáveis de entrada, são mais fáceis de serem desenvolvidas, facilitando o processo de desenvolvimento de regras.

Um conjunto *fuzzy* pode ser sintetizado como uma coleção de funções de pertinência que se relacionam entre si, dispostas em um domínio compreendido de valores reais. A forma mais simplificada de representar as diferentes funções de pertinência se dá pela convenção de expressá-las como Figuras geométricas como por exemplo: *triangular*, *trapezoidal*, *triangular*, *gaussiana*, entre outras, ilustradas na Figura 8.

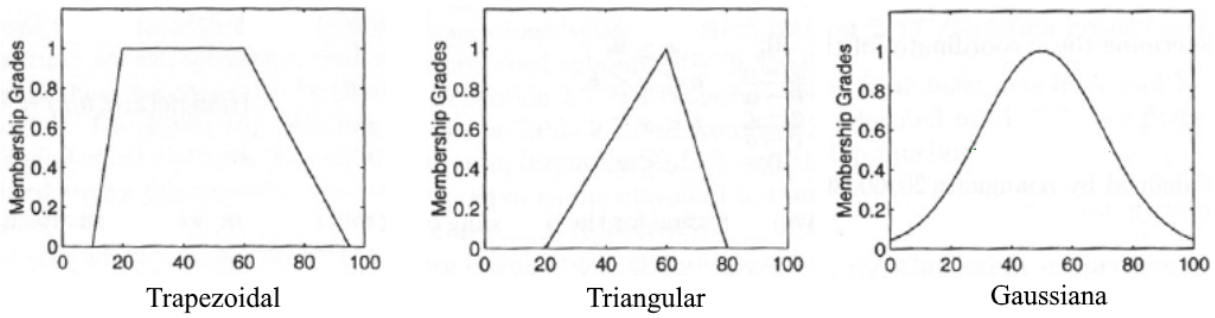

Figura 8 – Formatos de função de pertinência populares
Fonte: Adaptado de (KLINKHACHORN, 2004)

As fórmulas de 3.14 à 3.17 representam as diferentes funções de pertinência relativas a cada um dos termos associadas á variável *Tamanho do método*. Essas funções tem o formato

trapeziodal.

$$\text{Tamanho curto} = \mu_{curto}(x) = \begin{cases} \mu_c(x) = 1, \text{se } x \leq 5 \\ \mu_c(x) = \frac{10-x}{5}, se\ 5 \leq x \leq 10 \\ \mu_c(x) = 0, \text{se } x > 10 \end{cases} \tag{3.14}$$

$$\text{Tamanho médio} = \mu_{medio}(x) = \begin{cases} \mu_m(x) = 0, \text{se } x < 5 \text{ ou } x > 20 \\ \mu_m(x) = 0.2(x-5), \text{se } 5 \leq x \leq 10 \\ \mu_m(x) = 1, \text{se } 10 \leq x \leq 15 \\ \mu_m(x) = \frac{20-x}{5}, \text{se } 15 \leq x \leq 20 \end{cases} \tag{3.15}$$

$$\text{Tamanho longo} = \mu_{longo}(x) = \begin{cases} \mu_l(x) = 0, \text{se } x < 15 \text{ ou } x > 30 \\ \mu_l(x) = 0.2(x-15), \text{se } 15 \leq x \leq 20 \\ \mu_l(x) = 1, \text{se } 20 \leq x \leq 25 \\ \mu_l(x) = \frac{30-x}{5}, \text{se } 25 \leq x \leq 30 \end{cases} \tag{3.16}$$

$$\text{Tamanho muito longo} = \mu_{muitoLongo}(x) = \begin{cases} \mu_{ml}(x) = 0, \text{se } x < 25 \\ \mu_{ml}(x) = \frac{x-25}{5}, \text{se } 25 \leq x \leq 30 \\ \mu_{ml}(x) = 1, \text{se } x > 30 \end{cases} \tag{3.17}$$

É possível avaliar que as regras são compostas de termos linguísticos não muito precisos, como *muito longo*, *alto*, etc. Informações vagas e imprecisas são utilizadas porque não existe nenhuma definição precisa das palavras usadas nas regras do Quadro 10. Um conjunto *fuzzy* é capaz de representar fatos linguísticos incertos como um método *muito longo*. As Figuras 9 e 10 apresentam as funções de pertinência para variáveis linguísticas de entrada do controle de qualidade de código-fonte.

A Figura 9 representa as funções de pertinência para os termos associados à variável linguística *Tamanho do método*. Quanto menor for o número de linhas de código internos ao método avaliado, maior será a qualidade do método. Portanto, o termo *curto* representa a função de pertinência desejada para que se obtenha um bom nível na avaliação do método. Enquanto que o termo *muito longo* representa uma situação não indicada, isto é, o tamanho do método não é o mais indicado.

A Figura 10 representa as funções de pertinência para os termos associados à variável linguística *Número de Parâmetros*. Enquanto menor for a quantidade de parâmetros utilizados no método avaliado, maior será a qualidade do método. Portanto o termo *baixo* representa a função de pertinência desejada para que se obtenha um bom nível na avaliação do método. Enquanto

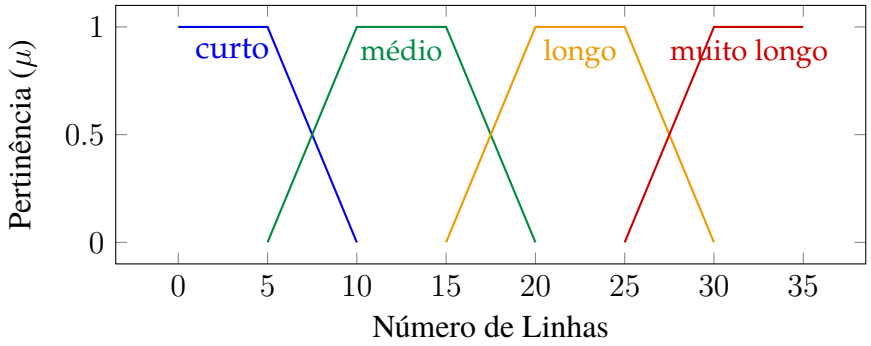

Figura 9 – Representação do conjunto *fuzzy*: Tamanho do método
Fonte: Autoria própria

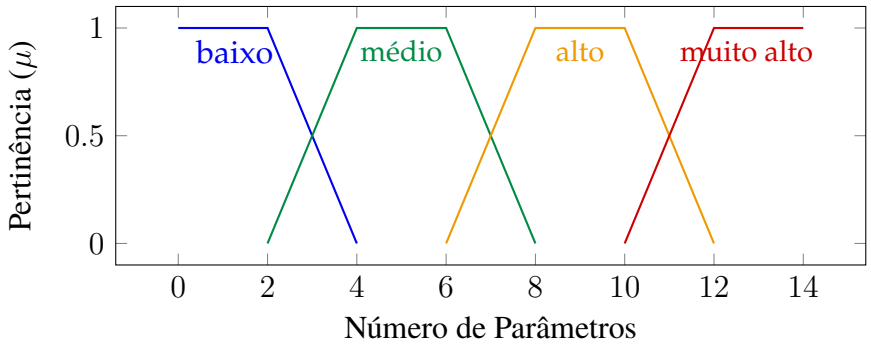

Figura 10 – Representação do conjunto *fuzzy*: Número de parâmetros do método
Fonte: Autoria própria

que o termo *muito alto* representa uma situação não indicada, isto é, o número de parâmetros é o ideal.

Com base nas regras definidas no Quadro 10 é possível se criar uma regra como apresenta o Quadro 11.

Quadro 11 – Exemplo de regra *fuzzy*

Se	tamanho do método	é	*longo*	**E**
	número de parâmetros	é	*médio*	
Então	avaliação do método	é	*regular*	

Fonte: Autoria própria

Esta regra é composta pelo termo linguístico *longo* da variável linguística tamanho do método, e o termo linguístico *médio* da variável linguística número de parâmetros, combinados pelo operador lógico ***E***. Uma forma para a simplificação da definição das regras é a criação da base completa de regras linguísticas, isto é, como esta situação apresenta apenas duas variáveis de entrada e uma única saída é possível se desenvolver a base com todas as regras possíveis na forma de uma matriz, onde são representadas as regras linguísticas do problema conforme mostra o Quadro 12.

Quadro 12 – Base completa de regras linguísticas - Mapa de regras

E		Número de Parâmetros			
		Baixo	**Médio**	**Alto**	**Muito Alto**
Tamanho do Método	**Curto**	Excelente	Muito Bom	Bom	Regular
	Médio	Muito Bom	Bom	Regular	Regular
	Longo	Bom	Regular	Regular	Preocupante
	Muito Longo	Regular	Regular	Preocupante	Preocupante

Fonte: Autoria própria

As combinações entre linhas e colunas descrevem as situações possíveis para a avaliação qualitativa do código-fonte. A qualidade que será atribuída ao método é determinada pelo termo linguístico da interseção da linha com a coluna. O Quadro 13 apresenta um exemplo quando o número de parâmetros do método é *baixo* e o tamanho do método é *curto*.

Quadro 13 – Exemplo de regra em que a avaliação é excelente

Se	tamanho do método	é	*curto*	**E**
	número de parâmetros	é	*baixo*	
Então	avaliação do método	é	*excelente*	

Fonte: Autoria própria

3.7 TRABALHOS RELACIONADOS

O trabalho usado como base para o desenvolvimento deste trabalho foi o de Avidagic, Boskovic e Causevic (2008) em que o grupo desenvolveu uma aplicação que utiliza o modelo da lógica *fuzzy* para avaliação de um aspecto da qualidade de software: manutenibilidade do código. A avaliação de código é realizada mediante classificação do código *bad smelss* como entrada.

Bad smells são os "erros" no código que prejudicam sua leitura e compreensão. Apenas a refatoração não garante que o software será melhorado, pois para tal é importante que se saiba quando é ou não aconselhável se refatorar o software. Os autores utilizaram a lista proposta por Fowler (2000) para a identificação de *bad smells*, esta lista contém 22 itens. Os autores selecionaram 4 *bad smells* e uma métrica para avaliar a complexidade ciclomática. Os *bad smells* considerados foram:

- **Código duplicado**, considerando que um código esteja escrito em mais do que um lugar, é importante que se encontre uma maneira de implementar o código sem que seja necessária a sua repetição.

- **Método longo**, se assume que um método longo é aquele que é de difícil entendimento, alteração ou extensão. No paradigma orientado a objetos deve-se sempre priorizar por métodos curtos e com uma única função.
- **Lista de parâmetros longa**, indica que o método possui muitos parâmetros, o que dificulta o entendimento.
- **Campos temporários**, uma variável membro de uma classe é utilizada apenas ocasionalmente ou redundante.

Os autores consideram que *bad smells* estão intimamente ligados à manutenibilidade do sistema, uma vez que o código perde em facilidade de manutenção quando ele apresenta um alto nível de *bad smells*, isto é, um código de baixa qualidade é dificilmente mantido e deve ser evitado ao máximo.

Eles definem um modelo *fuzzy* que avalia a manutenibilidade de uma classe em uma escala que varia entre *muito ruim* e *excelente*. As entradas para o modelo são os valores de código duplicado e campos temporários conforme apresenta o Quadro 14.

Quadro 14 – Métricas utilizadas

Métrica	Descrição
LOC	Linhas de código
V(G)	Complexidade ciclomática de McCabe
NOP	Número de parâmetros

Fonte: Adaptado de (AVIDAGIC; BOSKOVIC; CAUSEVIC, 2008)

Os autores definiram a ordem pela qual o processo de inferência *fuzzy* é baseado, isto é, uma entrada *crisp* é inicialmente tratada no processo de *Fuzzyficação*, onde os valores numéricos são convertidos em variáveis linguísticas. As funções de pertinência são utilizadas para a associação entre as variáveis e o termo linguístico. A próxima etapa utiliza a Base de Regras *fuzzy* em conjunto com os fatos obtidos na *Fuzzyficação* para a etapa de *Defuzzyficação*, isto é, a produção de uma saída precisa, gerada de acordo com os valores de entrada.

Depois definiram as funções de pertinência para cada um dos elementos da entrada de dados. Como por exemplo, a variável *código duplicado* é representado por cinco funções de pertinência: *muito pouco*, *pouco*, *médio*, *alto*, *muito alto*.

Regras *fuzzy* são utilizadas para definir a qualidade do software dependendo dos valores de entrada. Os autores definiram regras *fuzzy* utilizando o operador *E* entre os valores de entrada, conforme o exemplo apresentado no Quadro 15.

O emprego da técnica *fuzzy* geralmente produz um grande número de regras, que variam de acordo com o número de variáveis de entrada e saída, além do número de funções de pertinência. Conforme o número de regras providas pelo usuário cresce, aumenta também a precisão geral do sistema. Assim, o modelo *fuzzy* foi aplicado no nível de classe, porque a intenção dos autores era a de desenvolver um modelo capaz de avaliar métodos em uma classe localmente.

Quadro 15 – Regras geradas para a avaliação do código sob o critério de *bad smells*

Se	código duplicado	é	*pouco*	**E**
	método longo	é	*baixo*	**E**
	número de parâmetros	é	*baixo*	**E**
	complexidade ciclomática	é	*simples*	**E**
	lista de parâmetros	é	*muito baixa*	**E**
	campos temporários	é	*muito baixo*	
Então	avaliação do código	é	*excelente*	

Fonte: Adaptado de (AVIDAGIC; BOSKOVIC; CAUSEVIC, 2008)

Os autores concluíram que o uso de lógica *fuzzy* em conjunto com as métricas para a avaliação de código orientado a objetos é uma atividade promissora. Porém a avaliação de código-fonte, apesar de importante, é também uma atividade subjetiva porque requer uma análise humana para a tomada de decisões e alteração de prioridades dentro de um projeto de software.

No trabalho de Avidagic, Boskovic e Causevic (2008) foi definido um modelo para a avaliação do código sob a perspectiva de *bad smells*, porém esta pesquisa se limita a avaliação de um único requisito de qualidade, no caso Manutenibilidade. A diferença do trabalho proposto com a pesquisa de Avidagic, Boskovic e Causevic (2008) é elencada pelos seguintes motivos:

1. Pesquisa sobre métricas aplicadas a software orientada a objetos.
2. Estabeleceu relação entre métricas e os requisitos de qualidade.
3. A ferramenta proposta para a avaliação é capaz de avaliar múltiplos requisitos de qualidade.
4. A avaliação proposta comporta a adição de novos requisitos de qualidade e novas métricas, além de permitir a criação e alteração da Base de Regras.
5. É possível avaliar um conjunto de classes.

4 AVALIAÇÃO DE CÓDIGO-FONTE BASEADO EM ORIENTAÇÃO A OBJETO

Este Capítulo apresenta informações sobre a concepção da avaliação proposta para a identificação da qualidade do arquivo Java compilado *.class* a partir da perspectiva dos requisitos de qualidade desejados. A seção 4.1 relata o processo de funcionamento da avaliação. A seção 4.2 descreve o tratamento do grau de qualidade de um conjunto de arquivos *.class*. A seção 4.3 apresenta o processo de construção da ferramenta desenvolvida para avaliação de um projeto, composto por um conjunto de arquivos Java compilados.

4.1 PROCESSO GERAL PARA A AVALIAÇÃO

O processo utilizado para o desenvolvimento deste trabalho consistiu inicialmente na divisão em sete atividades, apresentadas na Figura 11 por meio de uma diagrama de atividades, sendo que dessas, as três primeiras ocorreram em paralelo. É importante notar que durante estas etapas iniciais, há uma distinção entre as atividades, sendo que o entendimento de lógica *fuzzy* atua de maneira quase que independente das outras duas etapas realizadas em paralelo. O estudo de métricas, qualidade de software e requisitos de qualidade se relacionam entre si como componentes da Engenharia de Software. O estudo de lógica *fuzzy a priori* não se relaciona com as outras atividades, porém com o decorrer das pesquisas, esta se mostra como uma técnica promissora para a realização das atividades propostas neste trabalho.

Na atividade *Estudar Métricas de Software* foram realizadas pesquisas para compreensão da definição, importância e utilização de métricas nas diversas etapas do desenvolvimento de software. Devido ao fato deste trabalho ter como objetivo a avaliação de um código orientado a objetos "pronto", optou-se pelo estudo de métricas empregadas para a avaliação de um produto terminado, isto é, não foram estudadas métricas utilizadas durante o processo de escrita do código, por exemplo, controle de produtividade ou contagem de erros. Nesta etapa também foram estudadas as principais métricas de código orientado a objetos: métricas CK (CHIDAMBER; KEMERER, 1994) e MOOD (ABREU; CARAPUçA, 1994), conforme a seção 2.3.

Concomitantemente ao estudo de métricas de software, a atividade *Estudar Qualidade de Software* teve como finalidade o levantamento do referencial teórico importante sobre as características de um software de qualidade, além da descrição de Requisitos de Qualidade e a sua importância para a elucidação dos atributos prioritários dentro do desenvolvimento do projeto. Isto é, a partir do momento em que se escolhe um requisito de qualidade como prioridade no desenvolvimento de um produto, ocorre uma troca em que de um lado há o ganho específico na requisito selecionado, porém do outro lado, requisitos podem sofrer queda na qualidade. Por exemplo, quando se separa a funcionalidade em classes diferentes, ganha-se em manutenibilidade e perde-se em desempenho.

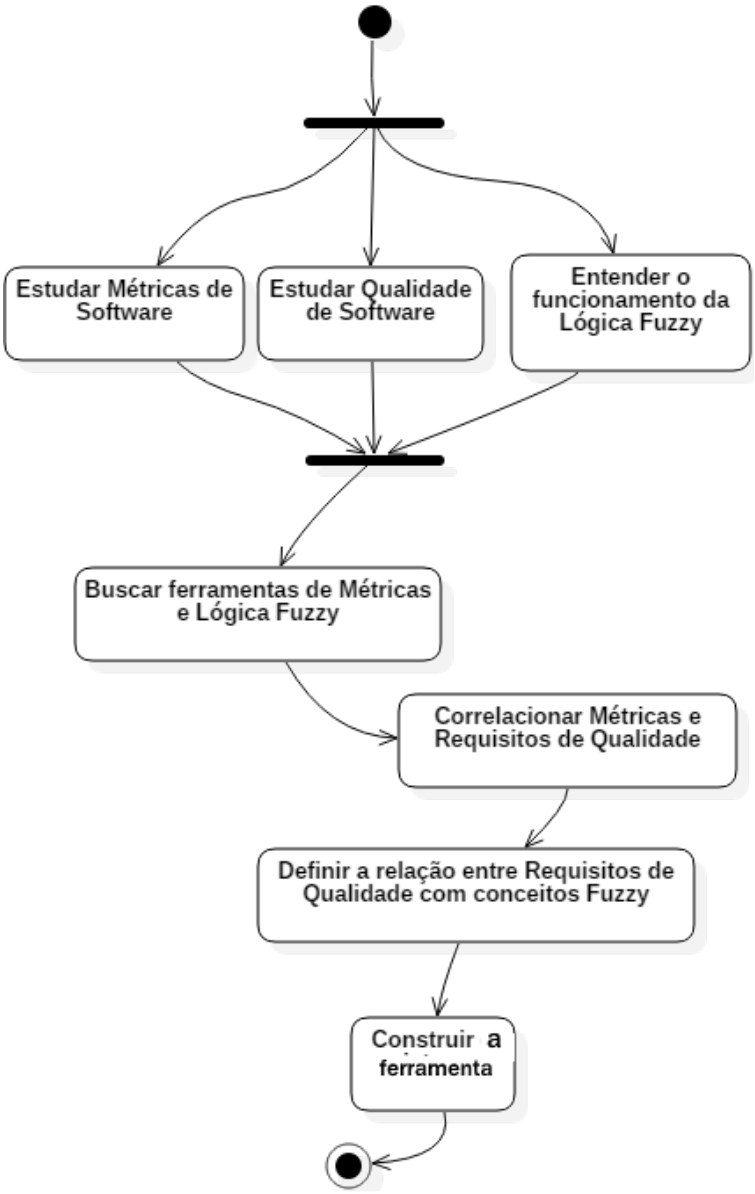

Figura 11 – Diagrama de atividades para o processo de funcionamento da avaliação
Fonte: Autoria própria

Após o estudo sobre métricas e qualidade de software foram pesquisados conceitos da Engenharia de Software que não tinham sido bem compreendidos, ou foram considerados importantes de serem mencionados, principalmente para se obter uma melhor definição sobre elementos como Requisitos de Qualidade, processos e produtos.

A terceira atividade é definida como o estudo da lógica *fuzzy*, que envolve o estuda das etapas fundamentais para o uso e aplicação dessa lógica, bem como as características que à distingue dos demais modelos de raciocínio lógico foram abordados. Por isso, foi estipulada a criação da atividade *Entender o funcionamento da Lógica Fuzzy*, na qual é possível se notar que a lógica *fuzzy* é útil para a representação de situação em que há alto grau de incerteza, como é o caso da definição de termos críticos referentes à classificação qualitativa dos elementos que compõem um software. Nesta etapa também foi desenvolvido um exemplo para determinar se

um método é longo usando a lógica *fuzzy*, descrito na seção 4.1.

Terminadas estas três primeiras etapas que formam a base do referencial teórico desta pesquisa, iniciou-se um novo ciclo de atividades que se fundamentou na busca por ferramentas úteis para o desenvolvimento da ferramenta proposta neste trabalho, realizada na atividade *Buscar ferramentas de Métricas e Lógica Fuzzy*. Esta atividade foi baseada em duas partes: Métricas de código-fonte e ferramentas para Lógica *fuzzy*. Para a primeira parte foram procuradas ferramentas capazes de avaliar atributos específicos de código-fonte, indicando os valores quantitativos de métricas para o código-fonte orientado a objetos. Em particular, buscou-se por software capazes de obter métricas de código orientado a objeto, destacando-se entre estas as métricas definidas por Chidamber e Kemerer (1994) e Abreu e Carapuça (1994). Isto se deu principalmente pelo fato destes serem os autores das métricas CK e MOOD explicadas nas sub-seções 2.3.1 e 2.3.2, respectivamente, sendo estas uma das primeiras e mais populares métricas de software orientado a objetos.

A segunda parte foi constituída pela busca por ferramentas capazes de representar os conceitos da lógica *fuzzy*, como por exemplo: conjuntos, termos e variáveis linguísticas, assim como as etapas de *Fuzzyficação* e *Deffuzyficação*. As ferramentas capazes de representar a lógica *fuzzy* são normalmente chamadas de *engine*. Foram encontradas várias *engines* que realizam a operação desejada e para esta pesquisa optou-se pelo emprego da *jFuzzyLogic* (CINGOLANI; ALCALà-FDEZ, 2012) para a realização das rotinas *fuzzy*. Esta ferramenta foi escolhida porque implementa a linguagem *Fuzzy control language* (FCL) e é a mais popular citada no sitio: *www.sourceforge.net*.

Logo após à busca por ferramentas, foi necessária a definição de como se realizaria a relação entre as Métricas de software orientado a objetos e os requisitos de qualidade de um software. Esta atividade é chamada de *Correlacionar Métricas e Requisitos de Qualidade*.

O inicio da execução desta atividade consistiu no uso da definição do que vem a ser um requisito de qualidade definido pela ISO/IEC no padrão 9126, que basicamente constitui na construção de um modelo que visa padronizar a criação de software de qualidade. O ISO/IEC definiu um modelo constituído de seis características detalhadas na seção 3.2, são elas: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. O ISO/IEC denomina essas características como atributos de "*External and Internal Quality*". Este trabalho agrupa esse conjunto de características como "Requisitos de Qualidade" em virtude de simplificação do termo.

Os elementos da Qualidade de Software se relacionam com os conceitos da lógica *fuzzy*. Optou-se por seguir uma definição onde a ferramenta desenvolvida é capaz de representar abstrações e conceitos como *variáveis linguísticas*, *conjunto de termos*, *termos*, *variáveis de entrada e saída*.

O Quadro 16 estabelece o critério de que um Requisito de Qualidade é formado por um conjunto de métricas. Por tratarem de variáveis linguísticas as métricas devem possuir termos que estão ligados a variável linguística em si. Assim, foi definida uma relação que encadeia

Quadro 16 – Relação entre conceitos da lógica *fuzzy* e Qualidade de Software

Qualidade de Software	**Conceito *fuzzy***
Métrica	Entrada da ferramenta
Valores atribuídos à métricas	Conjunto de termos linguísticos que podem ser atribuídos
Requisito de Qualidade	Variável de Saída

Fonte: Autoria própria

os elementos da Qualidade de Software da seguinte maneira: um Requisito de Qualidade R é constituído por um conjunto de métricas C_m. As métricas individualmente não são capazes de representar qualitativamente o sistema, mas agrupadas tem um grande poder de significado, ou seja, conforme o número de métricas utilizadas referentes a um requisito de qualidade, mais significativo é o resultado da avaliação deste conjunto. Tomando como exemplo a métrica *DIT*, definida na fórmula 2.1, exclusivamente esta não é capaz de representar muito sobre a qualidade de um software, porém quando utilizada em conjunto com outras métricas é capaz de indicar o nível de qualidade de um requisito de um software. Um conceito como herança, pode ser manipulado por métricas como a *DIT*. Considerando que um conjunto de métricas C_m esteja relacionado à uma métrica como R, é possível se estabelecer a seguinte relação (4.1):

$$\begin{aligned} &\textbf{Se } C_m \text{ apresenta valores } = \underline{\textit{bons}} \\ &\textbf{Então } \text{A avaliação do Requisito } R \text{ é } \underline{\textit{positiva}} \end{aligned} \tag{4.1}$$

Onde: a avaliação do conjunto C_m implica no nível de qualidade de um requisito de qualidade R, em que os valores correspondentes às métricas determinam diretamente sua avaliação, isto é, se a maioria dos elementos de C_m é considerado como bom ou excelente, o requisito R terá uma avaliação boa ou excelente, respectivamente.

Observa-se que a definição de um conjunto como *bom* ou *regular* é difícil, tanto pela subjetividade do assunto quanto pela definição do termo linguístico adotado. Exatamente por esses motivos é que a lógica *fuzzy* é adotada como uma solução para problemas que tratam de alto grau de incerteza.

Uma vez que cada métrica é avaliada individualmente, pode-se estabelecer uma variável linguística capaz de representar em que nível o conjunto C_m ela se encontra. A partir da definição de qual é o conceito do conjunto de métricas, se estabelece um paralelo entre as métricas e o requisito de qualidade. Assim, é possível afirmar que, a partir de um conjunto de métricas pode-se diagnosticar a qualidade geral dos requisitos de qualidade, bem como do software como um todo. O Quadro 17 é uma síntese da criação das relações entre as métricas e os requisitos de qualidade.

A relação entre a atividade de classificação de código-fonte e a lógica *fuzzy* ocorre no momento em que se transformam os diferentes elementos do âmbito da programação em conceitos e conjuntos *fuzzy*. Assim, deve-se mapear os diferentes elementos do ambiente estudado para o conceito de conjuntos *fuzzy*, termos, variáveis linguística, funções de pertinência, etc.

O Quadro 17 apresenta a relação entre as métricas orientada a objetos e os requisitos de

Quadro 17 – Relação de Requisitos de Qualidade x Métricas

Autores	Métrica		Avaliação
	Sigla	**Descrição**	**Requisito de Qualidade (RQ) Conceito OO (C)**
Chidamber & Kemerer - CK	WMC	Métodos Ponderados por Classe	Complexidade (RQ)
	DIT	Profundidade da Árvore de Herança	Herança (C) Manutenibilidade (RQ)
	NOC	Número de Filhos	Herança (Q)
	CBO	Acoplamento entre Classes de Objetos	Acoplamento (RQ)
	RFC	Resposta de Classe	Acoplamento (RQ)
	LCOM	Ausência de Coesão em Métodos	Coesão (RQ)
Abreu - MOOD	MHF	Fator de Ocultação de Método	Encapsulamento (RQ)
	AHF	Fator de Ocultação de Atributo	Encapsulamento (RQ)
	MIF	Fator de Herança de Método	Herança (C)
	AIF	Fator de Herança de Atributo	Herança (C)
	COF	Fator de Acoplamento	Acoplamento (RQ)
	PF	Fator de Polimorfismo	Polimorfismo (C)
	RF	Fator de Reuso	Reuso (C)
McCabe	McCabe	Complexidade de McCabe	Complexidade (RQ) Manutenibilidade (RQ)
	LOC	Linhas de Código	Manutenibilidade (RQ)
	NPM	Número de Métodos Públicos	Manutenibilidade (RQ)

Fonte: Adptado de (FILO, 2003)

qualidade que foram mapeados neste trabalho.

A última etapa do processo, *Construir a ferramenta*, consistiu na implementação dos conceitos de qualidade de software e Lógica *fuzzy* com o objetivo de criar uma ferramenta capaz de diagnosticar qual é o nível em que um requisito de qualidade está sendo atendido em um projeto orientado a objeto. O diagnóstico da qualidade é obtido a partir do emprego da lógica *fuzzy*.

Para este trabalho foram selecionados os Requisitos de Qualidade listados no Quadro 17, exceto Polimorfismo, uma vez que para a avaliação deste conceito OO não foram identificadas métricas relacionadas. A partir da definição de quais são os conjuntos C_m relacionados a cada um dos requisitos selecionados é possível a aplicação de um sistema *fuzzy* onde as variáveis de entrada são as métricas, e os requisitos de qualidade são as variáveis de saídas. A relação entre o Requisito de Qualidade *acoplamento* é dada por (4.2):

$$\begin{aligned} &\textbf{Requisito de Qualidade} : R_a = \underline{\text{Acoplamento}} \\ &C_m(R_a) = \underline{\textit{CBO, RFC, COF}} \end{aligned} \tag{4.2}$$

A relação entre o Requisito de Qualidade *manutenibilidade* dada por (4.3):

$$\begin{aligned} &\textbf{Requisito de Qualidade} : R_m = \underline{\text{Manutenibilidade}} \\ &C_m(R_m) = \underline{\textit{DIT, McCabe, LOC, NPM}} \end{aligned} \tag{4.3}$$

Para a utilização satisfatória da ferramenta faz-se necessário que o usuário tenha domínio sobre os conceitos abordados na ferramenta, como requisitos de qualidade e métricas para a interação com o software para a entrada destes dados.

4.2 TRATAMENTO DO GRAU DE QUALIDADE DE UM CODIGO FONTE ORIENTADO A OBJETO

A ferramenta desenvolvida utiliza o conceito de lógica *fuzzy* para a definição de qual é o grau de qualidade de um código-fonte orientado a objetos. O valor resultante admitido é referente à qualidade do código-fonte sob a perspectiva dos requisitos de qualidade, pois realiza o diagnóstico levando em consideração um conjunto de métricas, por exemplo, DIT, como ilustra o Quadro 18 para o requisito de manutenibilidade.

Quadro 18 – Definição de um requisito de qualidade

Requisito de qualidade: Manutenibilidade
Conjunto de métricas associadas ao requisito de qualidade: *DIT*
Valor de entrada da métrica (DIT): *3*

Fonte: Autoria própria

O comportamento esperado pela ferramenta é definir qual será o termo linguístico associado ao valor das variáveis de entrada. Considerando que a métrica tenha como funções de pertinência a Figura 12, pode-se estabelecer que a função de pertinência *fuzzy* que está associada a este valor, no caso DIT = 3 será a função do termo "médio".

Depois de avaliados os termos que se relacionam com a variável de entrada, deve-se aplicar a entrada à avaliação mediante cada uma das regras da Base de Regras, para se definir quais são as regras que "disparam" com esse valor de entrada. Devido ao fato deste caso ser composto por apenas uma variável de entrada e uma variável de saída, este representa um modelo linguístico do tipo SISO (seção 3.5.1), portanto: $M_i \rightarrow Q$, onde cada uma das métricas M_i influenciam diretamente no valor da qualidade Q. Neste caso, como M_i é *médio*, a qualidade do requisito manutenibilidade dever ser atribuída em um nível correspondente, neste caso, *médio*.

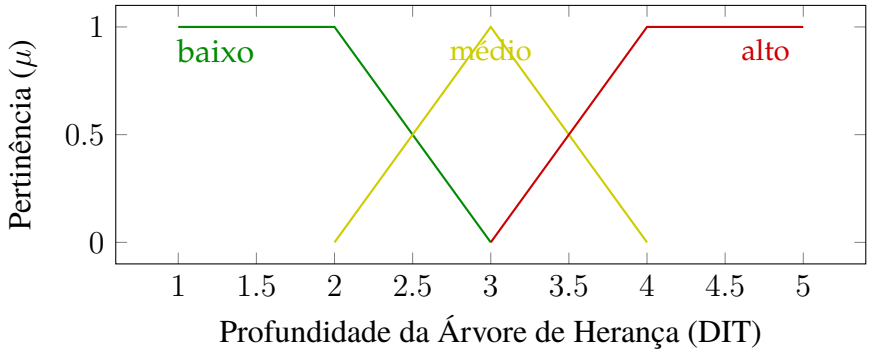

Figura 12 – Representação dos termos associadas à métrica DIT
Fonte: Autoria própria

4.3 PROCESSO DE FUNCIONAMENTO DA FERRAMENTA PARA A AVALIAÇÃO QUALIDADE DE CÓDIGO-FONTE

A ferramenta proposta trabalha com a ideia de que o usuário deve informar os arquivos Java compilados pertencentes a um projeto. Assim, realiza o processo automatizado para executar um programa que calcula os valores para cada métrica de software. O programa de cálculo das métricas adotado neste trabalho é o *CKJM Extended*, escolhido porque implementa as métricas citadas no Quadro 17, exceto as MOOD.

O fluxograma ilustrado na Figura 13 exibe o processo de uso da ferramenta. Ela trabalha integrando as partes de lógica *fuzzy* com métricas e requisitos de qualidade. Ao iniciar a ferramenta, a operação *Buscar Requisitos de Qualidade, Termos e Métricas* é executada com a finalidade de buscar os requisitos, termos e métricas que estão cadastrados no banco de dados e que representam o domínio da aplicação. O usuário cria um projeto e informa os dados: nome do projeto, autor e pasta de origem.

Após preenchidas as informações básicas do projeto, o usuário seleciona os requisitos de qualidade que deseja avaliar em seu código-fonte, podendo posteriormente criar novos requisitos, métricas e termos.

A ferramenta estabelece um conjunto de associações entre métricas, termos e requisitos para automatizar o uso do software, realizadas na etapa *Relacionar Requisitos e Métricas* (*Fuzzyficação*). Porém, novas associações podem ser estabelecidas pelo usuário.

Após finalizadas as possíveis alterações no projeto, o usuário entra com a pasta onde está localizado o projeto com os arquivos que deseja avaliar. A ferramenta procura dentro da pasta do projeto os arquivos *.class* para serem avaliados (build/classes). A avaliação das métricas ocorre da seguinte maneira: copiar os arquivos *.class* para dentro da pasta do projeto informada no inicio do processo, chamar o software que realiza o cálculo das métricas (*CKJM Extended*) e salvar os valores obtidos pelo software de métricas na pasta do projeto por meio de um arquivo *.xml*. O arquivo *.xml* contém o nome da classe, a métrica e o valor obtido.

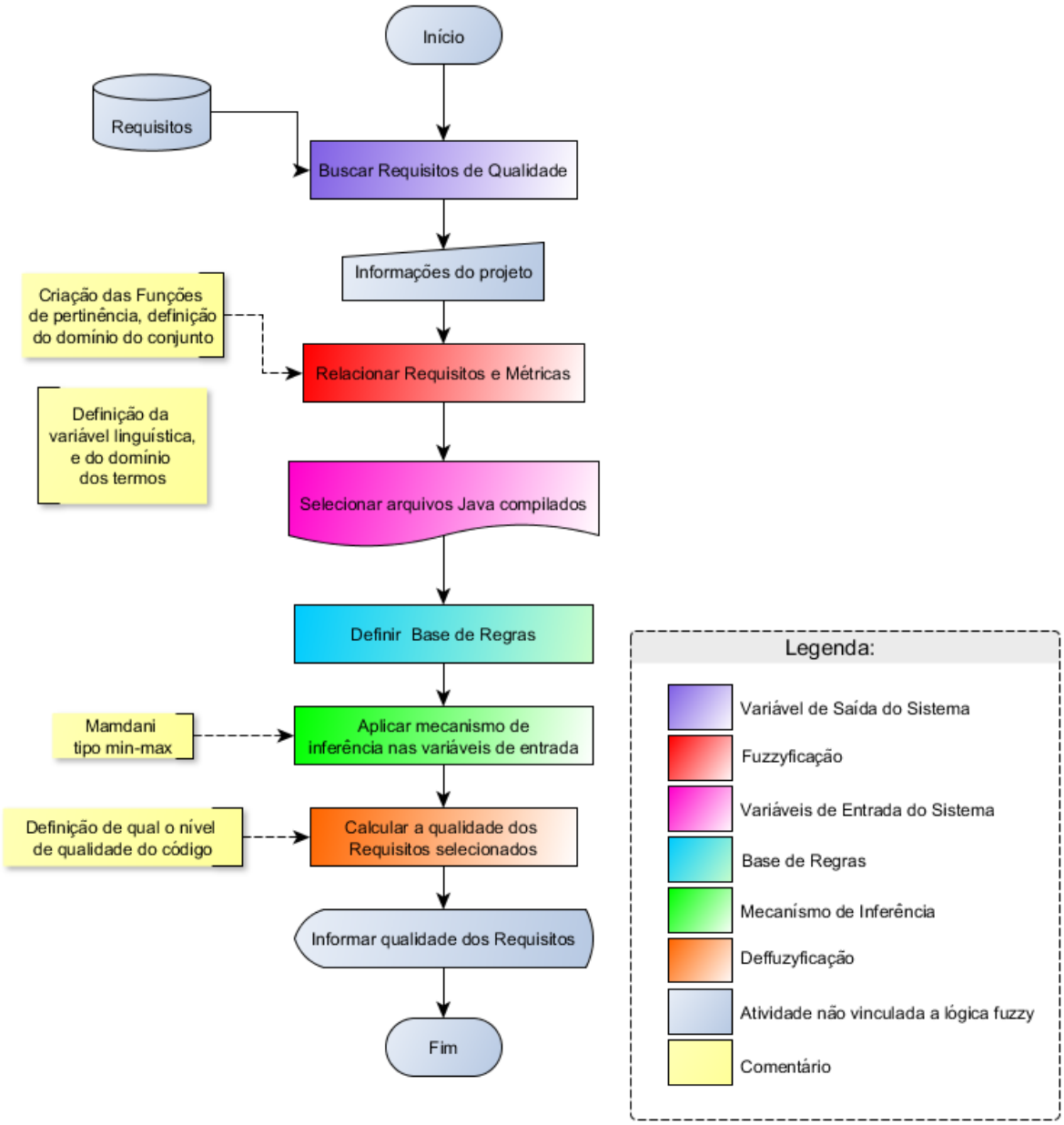

Figura 13 – Fluxograma do uso da ferramenta proposta
Fonte: Autoria própria

Logo após, o usuário deve definir sua Base de Regras que contém as regras de produção em que o antecedente contém as métricas e os termos correspondentes e o consequente é formado pelos requisitos de qualidade e os termos correspondentes. A geração das regras definidas pelo usuário é gravada em um arquivo do tipo *.fcl* para que possa ser utilizado como entrada no próximo processo. Para cada requisito é gerado um arquivo *.fcl*. Conforme afirma Avidagic, Boskovic e Causevic (2008) a automatização da etapa de geração de regras é complexa é foge do escopo desta ferramenta.

Depois de definidas as regras na ferramenta, o usuário solicita o cálculo da qualidade do código-fonte e internamente é iniciada a atividade do controlador *fuzzy* (*jFuzzyLogic*), em que

a partir os dados obtidos dos cálculos das métricas são utilizados como entrada para o cálculo da qualidade de requisitos selecionados. Estas operações iniciam-se na atividade *Aplicar mecanismo de inferência nas variáveis de entrada*, no caso desta ferramenta utiliza-se o mecanismo definido por Mamdini do tipo *min-max* (conforme a seção 3.5), uma vez que esta é a forma com que mais se utiliza lógica *fuzzy*, além de garantir resultados confiáveis.

A tarefa *Calcular a qualidade dos Requisitos selecionados* pode ser definida como a etapa de *Deffuzyficação* dentro da sequência da lógica *fuzzy*, isto é, a soma dos valores obtidos pela relação entre a entrada do sistema e o conjunto de regras, aplica-se um cálculo matemático para se obter um valor *crisp* novamente. Este valor, dentro do contexto da ferramenta proposta varia entre 0 a 10, onde 0 é um resultado extremamente satisfatório e 10 é um resultado que indica que o código provavelmente precisa ser melhorado.

Após finalizados os cálculos, o resultado da qualidade dos requisitos selecionados são apresentado ao usuário, isto é realizado durante a etapa *Informar qualidade dos Requisitos*, terminando o fluxo de uso da ferramenta. Portanto, uma visão geral da ferramenta proposta é exibida no Quadro 19.

Quadro 19 – Visão geral da ferramenta proposta

Entrada	**Processos da ferramenta proposta**	**Saída**
Arquivo de projeto contendo um conjunto de códigos-fontes Java compilados	Gravar os arquivos em disco na pasta escolhida do projeto	Criar o arquivo *.fcl* para cada um dos requisitos de qualidade selecionados.
	Executar a ferramenta *CKJM Extended* para a obtenção dos valores das métricas relativas aos arquivos compilados	Exibir os gráficos oriundos da ferramenta *jFuzzyLogic* .
	Redirecionar a saída do *CKJM Extended* para a geração de um arquivo *.xml* que consta o valor obtido para cada métrica de cada classe	
	Exibir do arquivo *.xml* na ferramenta	
	Executar a ferramenta *jFuzzyLogic* para execução da lógica fuzzy	

Fonte: Autoria propópia

5 RESULTADOS

Este Capítulo apresenta os resultados alcançados por este trabalho de pesquisa. A seção 5.1 relata um exemplo de aplicação do uso de lógica *fuzzy*. A seção 5.2 apresenta a estrutura da ferramenta proposta, bem como o seu funcionamento. A seção 5.3 relata a análise dos resultados obtidos a partir da utilização da ferramenta proposta em dois estudos de caso. A seção 5.4 descreve uma comparação entre a avaliação proposta e os trabalhos relacionados.

5.1 EXEMPLO DA APLICAÇÃO DE LÓGICA *FUZZY* PARA DETERMINAR SE MÉTODO É LONGO

Esta seção relata um resultado obtido da etapa do processo *Entender o Funcionamento da Lógica Fuzzy*, ilustrado na Figura 11, aplicada para determinar se um método é ou não longo.

Dado um método com as seguintes características: tamanho do método = 8 linhas de código e número de parâmetros = 11 elementos. Deseja-se obter qual é o nível de qualidade de um método, levando em conta as duas variáveis expostas. É possível utilizar um modelo linguístico do tipo MISO e o Mecanismo de Inferência Mamdani *min-max* para a solução deste problema. A Figura 14 representa os valores correspondentes às variáveis que são abordadas neste método. O gráfico à esquerda (Figura 19a)) correspondente ao tamanho do método e o da direita (Figura 19b)) ao número de parâmetros do método avaliado. Estas variáveis são utilizadas como variáveis de entrada no modelo linguístico. O objetivo desta atividade é determinar a Qualidade do método, sendo que os termos e o gráfico referentes a este objetivo é representado na Figura 7.

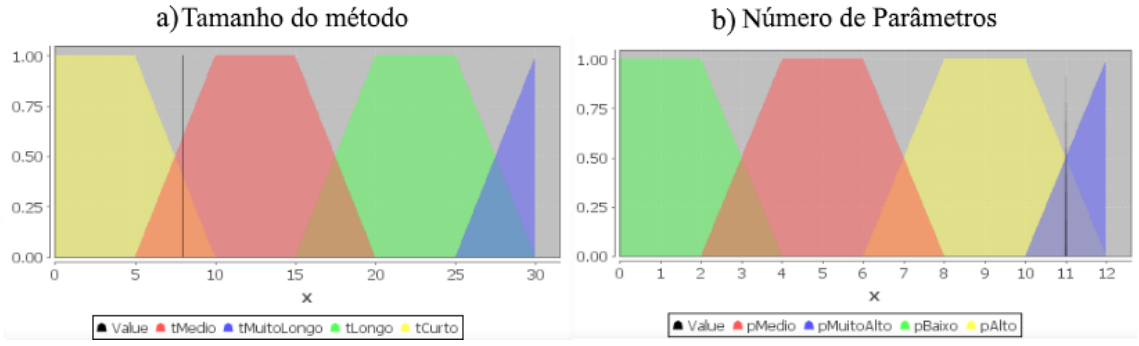

Figura 14 – Exemplo do processo de avaliação do código-fonte
Fonte: Autoria própria

Para o exemplo ilustrado na Figura 14 se define que as duas variáveis de entrada *Tamanho do método* e *Número de Parâmetros* do método serão referenciadas por t e p, respectivamente. E que a variável de saída Qualidade do Método será representada por q. Os termos linguísticos de todas as variáveis tanto de entrada quanto de saída deste exemplo são definidos como funções de

pertinência de formato *trapezoidal*, conforme demonstram as Figuras 7, 9 e 10. O Quadro 20 apresenta a relação de termos relacionados às variáveis definidas na situação do exemplo.

Quadro 20 – Conjuntos relativos ao domínio do problema

$C_t = \{$*Curto, Médio, Longo, Muito Longo*$\}$
$C_p = \{$*Baixo, Médio, Alto, Muito Alto*$\}$
$C_q = \{$*Excelente, Muito Bom, Bom, Regular, Preocupante*$\}$

Fonte: Autoria própria

Considerando uma entrada $E = \{t, p\} = \{8, 11\}$, deve-se aplicar os valores da entrada à Base de Regras para se determinar *força* h_i de cada regra R_i dentro da Base de Regras. A Base de Regras foi definida anteriormente, conforme o Quadro 12. Neste caso, o conjunto de regras que forma a Base de Regras tem o tamanho igual a dezesseis, uma vez que este é o número de relações possíveis considerando que o conjunto t possui 4 termos linguísticos e o conjunto p possui 4 termos linguísticos. Entende-se por *força* a relação entre os valores de entrada e a regra R_i avaliada. Um exemplo de regra criado é para a situação exemplo é exibido no Quadro 21.

Quadro 21 – Regras geradas pelo para determinar se um método é longo

R_1:	**Se** t é *curto* **E** p é *baixo* **Então** q é *excelente*
R_2:	**Se** t é *curto* **E** p é *médio* **Então** q é *muito bom*
...	
R_{16}:	**Se** t é *muito longo* **E** p é *muito alto* **Então** q é *preocupante*

Fonte: Autoria própria

Ao se avaliar a variável linguística t percebe-se que o termo "8 linhas de código" é compreendido tanto pelo termo linguístico *curto* bem como o termo *médio*. A variável linguística p, por se tratar de 11 parâmetros, comporta os termos *alto* e *muito alto*. Então, os valores referentes às funções de pertinência de cada termo podem ser expressos no Quadro 22.

Quadro 22 – Valores relativos aos termos de cada conjunto *fuzzy*

$C_t(8) = \{$*curto* $\rightarrow \mu = 0.4$, *medio* $\rightarrow \mu = 0.6$, *longo* $\rightarrow \mu = 0.0$, *muito longo* $\rightarrow \mu = 0.0\}$
$C_p(11) = \{$*baixo* $\rightarrow \mu = 0.0$, *medio* $\rightarrow \mu = 0.0$, *alto* $\rightarrow \mu = 0.5$, *muito longo* $\rightarrow \mu = 0.5\}$

Fonte: Autoria própria

Após a avaliação de cada um dos termos relacionados aos valores de entrada C_t e C_p, inicia-se a etapa de avaliação das regras R_i da Base de Regras. A força de cada uma das regras f_i (coeficiente de disparo) é computada em com base nos valores antecedentes e então se cria uma relação entre as regras e os valores de entrada para a designação de uma saída difusa da

regra. Então, para o cálculo da força de cada regra, geralmente se aplica operador *fuzzy min* em cada uma das regras definidas. O Quadro 22 apresenta a sequencia de avaliação das regras computadas.

Quadro 23 – Sequência da avaliação das variáveis de entrada

Regra_1 : **Se** x é A_1 **E** y é B_1 **Então** z é D_1
Regra_2 : **Se** x é A_2 **E** y é B_2 **Então** z é D_2
...
$\text{Força}(\text{Regra}_1) = \textit{min}(x, y)$
$\text{Força}(\text{Regra}_2) = \textit{min}(u, v)$
$D_1 = \text{Força}(\text{Regra}_1)$
$D_2 = \text{Força}(\text{Regra}_2)$
$z = \textit{max}(\text{Força}(\text{Regra}_1), \text{Força}(\text{Regra}_2))$
$z = \textit{max}(\textit{min}(x, y), \textit{min}(u, v))$

Fonte: Adaptado de (MIRANDA; JUNIOR; KRONBAUER, 2003)

Onde: x, y, u e v são as entradas do sistema, A_1, A_2, B_1 e B_2 são os termos para valores das variáveis de entrada (variáveis linguísticas), e D_1 e D_2 são os termos associados a variável de saída do sistema z.

Considerando x_0 e y_0 os valores *crisp* de entrada associados aos termos x e y pertencentes aos conjuntos *fuzzy* A_i e B_i, o valor de disparo das regras pode ser calculado como: $\alpha_i = \mu_{Ai}(x_0) \wedge \mu_{Bi}(y_0)$, o operador de conjunção, é realizado pela lógica *fuzzy* na forma do operador *min*, logo: $\alpha_i = \textit{min}\ (\mu_{Ai}(x_0), \mu_{Bi}(y_0))$. Os valores α_i são denominados graus de pertinência de disparo ou coeficientes de disparo.

A Figura 15 representa o cálculo do coeficiente de disparo da terceira regra provinda da Base de Regras definida no Quadro 12. Considerando que as variáveis $\mu_{Ai}(x_0) = C_t(8)$ e $\mu_{Bi}(y_0) = C_p(11)$. Verifica-se que para esta entrada obtém-se $\alpha_3 = 0.4$.

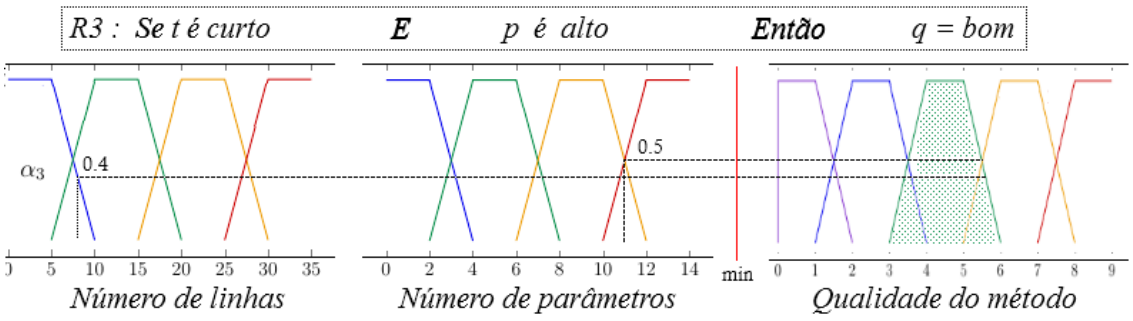

Figura 15 – Avaliação das variáveis de entrada segundo a regra 3

Fonte: Autoria própria

Os valores descritos no Quadro 24 representam a relação entre a entrada $E = \{t, p\} = \{8, 11\}$ e o conjunto de Regras definido na Base de Regras, gerando os valores correspondentes ao coeficiente de disparo α_i.

Quadro 24 – Funcionamento da etapa de Fuzzyficação

Regra$_1$ $\rightarrow$	$\alpha_1 = \mathit{min}(\mu_{curto}(8), \mu_{baixo}(11)) = \mathit{min}(0.4, 0.0) = 0.0$
Regra$_2$ $\rightarrow$	$\alpha_2 = \mathit{min}(\mu_{curto}(8), \mu_{medio}(11)) = \mathit{min}(0.4, 0.0) = 0.0$
Regra$_3$ $\rightarrow$	$\alpha_3 = \mathit{min}(\mu_{curto}(8), \mu_{alto}(11)) = \mathit{min}(0.4, 0.5) = 0.4$
Regra$_4$ $\rightarrow$	$\alpha_4 = \mathit{min}(\mu_{curto}(8), \mu_{muitoAlto}(11)) = \mathit{min}(0.4, 0.5) = 0.4$
...	
Regra$_7$ $\rightarrow$	$\alpha_7 = \mathit{min}(\mu_{medio}(8), \mu_{alto}(11)) = \mathit{min}(0.6, 0.5) = 0.5$
Regra$_8$ $\rightarrow$	$\alpha_8 = \mathit{min}(\mu_{medio}(8), \mu_{muitoAlto}(11)) = \mathit{min}(0.6, 0.5) = 0.5$
...	
Regra$_{16}$ $\rightarrow$	$\alpha_{16} = \mathit{min}(\mu_{muitoLongo}(8), \mu_{muitoAlto}(11)) = \mathit{min}(0.0, 0.5) = 0.0$

Fonte: Autoria própria

As regras cujo os coeficientes de disparo obtidos são maiores que zero são chamadas de regras que *disparam*, ou seja, para dados valores de x_0 e y_0 representam valores cuja saída z é positiva. Portanto, essa regra será utilizada para o cálculo da saída deste sistema *fuzzy*.

Depois de definidos os valores de disparo de cada uma das regras, deve-se relacionar os coeficientes de disparo α_i com os graus de pertinência μ_i da parte consequente da regra: $\mu_{Di}(z) = \mathit{min}(\alpha_i, \mu_{Di}(z))$, conforme mostra o Quadro 25.

Quadro 25 – Conjunto de saída para determinar se um método é longo

Regra$_1$ $\rightarrow$	$\mu'_{D1}(y) = \mathit{min}(\alpha_1, \mu_{C1}(y)) = \mathit{min}(0.0, \mu_{excelente}(y))$
Regra$_2$ $\rightarrow$	$\mu'_{D2}(y) = \mathit{min}(\alpha_2, \mu_{C2}(y)) = \mathit{min}(0.0, \mu_{muitoBom}(y))$
Regra$_3$ $\rightarrow$	$\mu'_{D3}(y) = \mathit{min}(\alpha_3, \mu_{C3}(y)) = \mathit{min}(0.4, \mu_{bom}(y))$
Regra$_4$ $\rightarrow$	$\mu'_{D4}(y) = \mathit{min}(\alpha_4, \mu_{C2}(y)) = \mathit{min}(0.4, \mu_{regular}(y))$
...	
Regra$_7$ $\rightarrow$	$\mu'_{D7}(y) = \mathit{min}(\alpha_7, \mu_{C7}(y)) = \mathit{min}(0.4, \mu_{regular}(y))$
Regra$_8$ $\rightarrow$	$\mu'_{D8}(y) = \mathit{min}(\alpha_8, \mu_{C8}(y)) = \mathit{min}(0.4, \mu_{regular}(y))$
...	
Regra$_{16}$ $\rightarrow$	$\mu'_{D16}(y) = \mathit{min}(\alpha_{16}, \mu_{C16}(y)) = \mathit{min}(0.0, \mu_{preocupante}(y))$

Fonte: Autoria própria

Após definidas as áreas referentes ao conjunto D'_i é realizada a operação global de união para compor o conjunto *fuzzy* para cada variável de saída, isto é, para cada valor de D'_i é somada

a área para a criação de uma figura geométrica que representa o conjunto *fuzzy* na variável de saída. Esta operação pode ser representada na forma $D' = \sum i = 1^n max(\alpha_i, \mu Ci)$. A Figura 16 mostra os resultados obtidos a partir das entradas indicadas, utilizando do método de Mamdami min-max, obtém-se uma região B', representada pela cor acinzentada. Esta área em específico representa apenas um conjunto *fuzzy* e para a descoberta do valor *crisp* relativo as entradas t e p realiza-se a etapa de *Defuzzyficação*. No caso da Figura 16, aplicando o método de centro de área para a obtenção do valor numérico, o valor *crisp* equivalente à entrada $E = \{t, p\} = \{8, 11\}$ é igual a 5.7, isto é, este método pode ser classificado como tendo uma qualidade em nível *regular*.

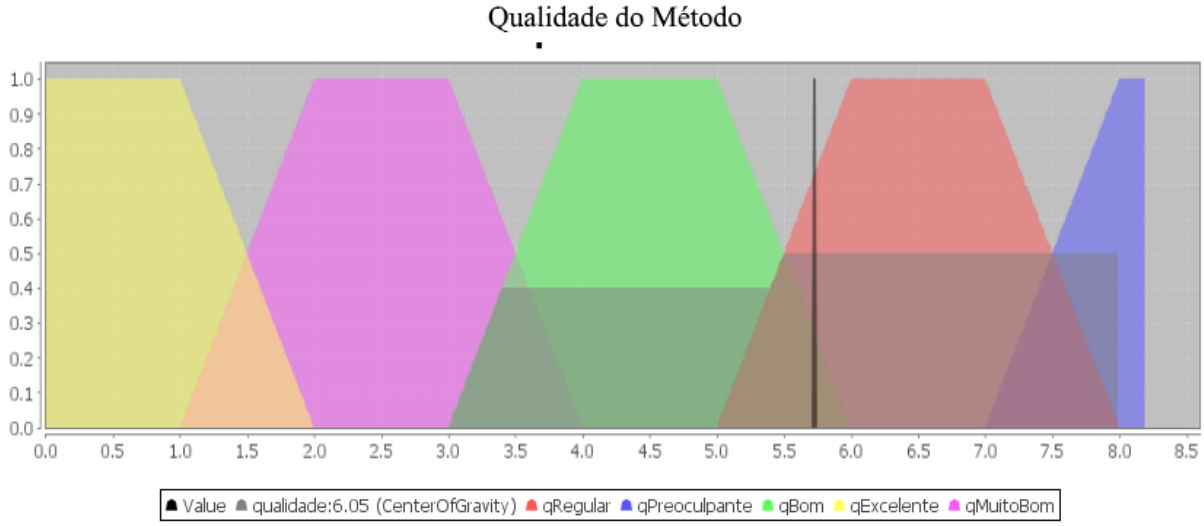

Figura 16 – Etapa de *Defuzzyficação* para determinar se um método é longo
Fonte: Autoria própria

5.2 FUNCIONAMENTO DA FERRAMENTA PROPOSTA

A ferramenta foi desenvolvida em uma linguagem orientada a objetos (Java) e o banco de dados foi o SQLite (SQLITE, 2015), uma vez que este banco é de fácil manutenção e não exige que seja instalado um serviço para que funcione corretamente.

A Figura 17 representa uma parte do Diagrama de Modelo de Dados que foi utilizado para a criação do banco da ferramenta proposta. É apresentado apenas a parte de maior relevância para este projeto. Nesta Figura existe uma relação entre as tabelas “Requisito” e a tabela “Métrica” do tipo *many-to-many*, essa associação tem como paralelo na lógica *fuzzy* o fato de que uma variável de saída pode ter uma ou mais variáveis de entrada, portanto, uma mesma métrica pode estar associada a vários requisitos, assim como um requisito tem o seu nível de qualidade estipulado por um conjunto de métricas.

Outra relação da tabela “Requisito” ocorre com a tabela “Termo”, uma vez que um Requisito quando avaliado deve possuir um termo correspondente ao seu nível de qualidade.

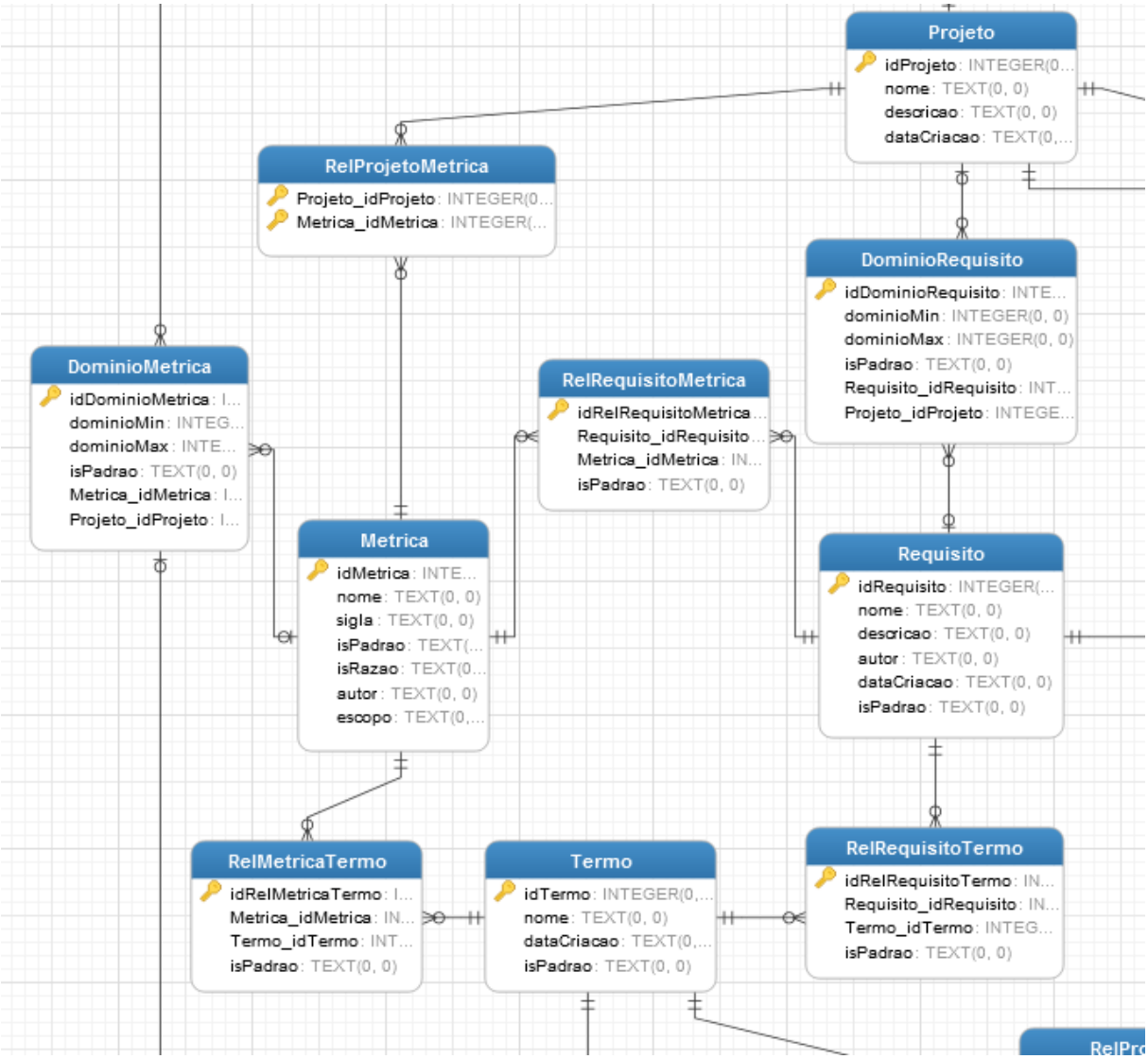

Figura 17 – Fragmento do diagrama de modelo de dados da ferramenta propósta
Fonte: Autoria própria

Na tabela "Termo" estão mantidos termos aos quais os conjuntos *fuzzy* se relacionam, isto é, considerando que o requisito avaliado seja *Coesão*, pelo fato de existirem diferentes níveis de coesão admitidos para um software, o Requisito possui um conjunto com vários termos que lhe estão associado. A partir da definição de variável linguística definida na seção 3.2 se obtém a definição da variável linguística Coesão em (5.1).

$$variável\ linguíistica = (x, T(x), \cup, G, M) \tag{5.1}$$

Onde *Manutenibilidade*: nome da variável linguística.

T(Manutenibilidade) = {*baixo, médio, alto*}.

$\cup$: $[0, 10]$.

M(baixo): entre 0 e 5.

M(médio): entre 3 e 7.

M(alto): entre 5 e 10.

A tabela “Requisito” ainda se relaciona com “DomínioRequisito”. Esta tabela foi criada com o intuito de definir quais são os limites inferiores e superiores do Requisito, isto é, quais são os valores aceitos para um requisito ser válido. Optou-se por se definir que todos os requisitos possuem o mesmo domínio. O limite inferior é 0 e o superior é 10. Os termos que estão vinculados aos Requisitos devem, consequentemente, ser distribuídos dentro deste domínio.

A tabela “Metrica” possui relacionamentos com outras tabelas de maneira semelhante à tabela “Requisito”. Uma vez que uma métrica possui vários termos que estão associados ao seu nível, conforme mostra a Figura 12, e também existe um domínio aceitável para uma métrica. Foram definidas as relações “Metrica” $\times$ “Termo” e “Metricas” $\times$ “DominioMetricas”.

Dado o requisito de qualidade *Manutenibilidade* é possível se definir um esquema para a aplicação da lógica *fuzzy*. Dessa forma, as variáveis de entrada são as métricas que se relacionam com o Requisito: *DIT*, *CYCLO*, *MLOC*, e a variável de saída é o próprio requisito: *Manutenibilidade*. Os termos difusos associados as variáveis de entrada são: *Baixo* (B), *Médio* (M) e *Alto* (A).

Os termos difusos associados ao requisito de qualidade são: *Excelente* (EX), *Muito Bom* (MB), *Bom* (BM), *Regular* (R) e *Preocupante* (P). Por esta ferramenta ser do tipo MISO, não existe a necessidade de indicar quais variáveis de entrada se relacionam com a variável de saída, uma vez que há apenas uma variável de saída. O Quadro 26 apresenta as relações entre estes elementos.

Quadro 26 – Variáveis e termos difusos para o controlador *fuzzy*

Variável	Entrada/Saída	Termo Difuso	Inicío	Términio
CYCLO	Entrada	Baixo	0	2
		Médio	1	4
		Alto	3	10
DIT	Entrada	Baixo	0	2
		Médio	1	4
		Alto	3	10
MLOC	Entrada	Baixo	0	10
		Médio	8	30
		Alto	28	35
Manutenibilidade	Saída	Excelente	0	2
		Muito Bom	1	4
		Bom	3	6
		Regular	5	8
		Preocupante	7	9

Fonte: Autoria própria

Com as informações do Quadro 26 é possível criar as relações entre as variáveis, definindo assim a Base de Regras. O Quadro 27 apresenta um exemplo de regra.

Quadro 27 – Regras geradas para a avaliação do código sob o critério de *bad smells*

Se	CYCLO	é	*Baixo*	**E**
	DIT	é	*Médio*	**E**
	MLOC	é	*Baixo*	**E**
Então	Manutenibilidade	é	*Muito Bom*	

Fonte: Autoria própria

5.2.1 Aplicação da Ferramenta

Esta seção apresenta um exemplo prático do uso da ferramenta proposta. Ao iniciar o programa o usuário tem a opção de criar um novo projeto. A Figura 18 mostra a tela principal desenvolvida que é composta por duas partes: no canto inferior à esquerda ficam dispostas as variáveis de entrada (métricas) e as variáveis de saída do sistema (requisitos). Antes que um novo projeto seja criado o sistema não possui nenhuma variável selecionada, logo estes campos estão vazios.

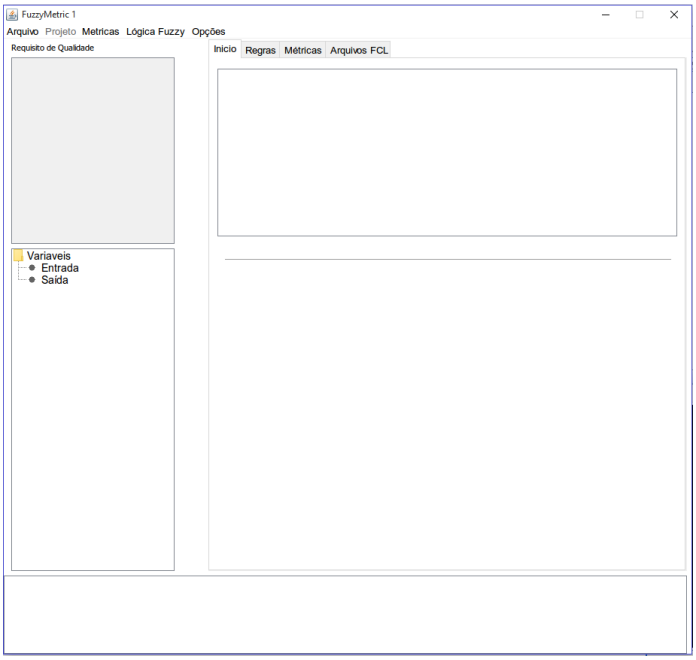

Figura 18 – Tela inicial da ferramenta

Fonte: Autoria própria

Para iniciar o uso da ferramenta é necessário criar um novo projeto, selecionando a opção *Arquivo* na barra de ferramentas, seguido por *Novo Projeto*. O sistema abrirá uma tela responsável

pela adição de novos projetos, conforme ilustra a Figura 19. O usuário deve informar o nome do projeto, além de escolher a pasta em que deseja salvar os arquivos da saída da ferramenta. Para a utilização da ferramenta de forma mais segura, é indicado que a pasta onde se cria um novo projeto esteja vazia. Devido a este fato, após o usuário clicar no botão *Próximo* a ferramenta irá perguntar ao usuário se ele permite que os arquivos presentes na pasta selecionada sejam deletados.

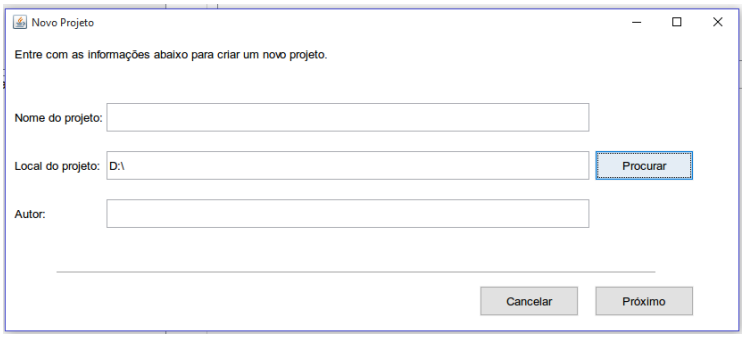

Figura 19 – Tela novo projeto
Fonte: Autoria própria

Depois de definidas as informações básicas do projeto o usuário seleciona os requisitos de qualidade que deseja utilizar para a avaliação do código-fonte. Conforme mostra a Figura 20, nesta tela é exibida uma lista com os Requisitos de Qualidade disponíveis para serem avaliados. Esses requisitos foram extraídos do banco de dados logo ao iniciar o uso da ferramenta. Nota-se que estão disponíveis seis requisitos de qualidade. O usuário deve clicar no botão *Adicionar* para incluir o requisito selecionado na lista dos requisitos que serão avaliados. Na Figura 20 os requisitos selecionados foram: Acoplamento e Manutenibilidade.

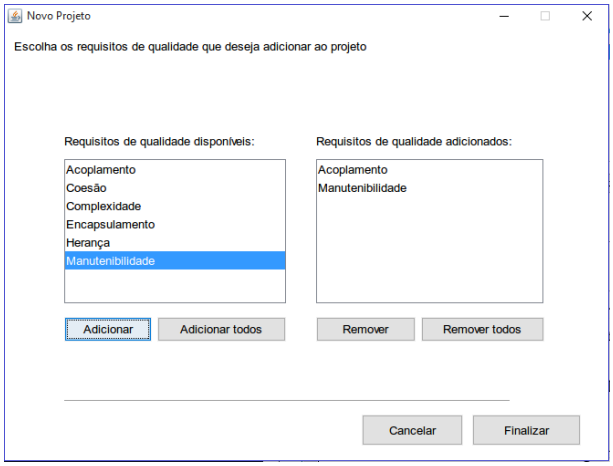

Figura 20 – Tela selecionar requisito de qualidade
Fonte: Autoria própria

Depois dos Requisitos serem selecionados, o usuário pode confirmar a operação clicando no botão *Finalizar* ou pode cancelar a criação do projeto como um todo clicando no botão *Cancelar*.

Depois de adicionados os requisitos o fluxo do programa retorna a tela principal. Onde é possível que o usuário veja as métricas que estão associadas aos requisitos selecionados, disponíveis no canto inferior-direito da ferramenta. Também é exibido um botão por onde o usuário pode entrar com os arquivos Java compilados necessários para a avaliação do código-fonte. Conforme ilustra a Figura 21

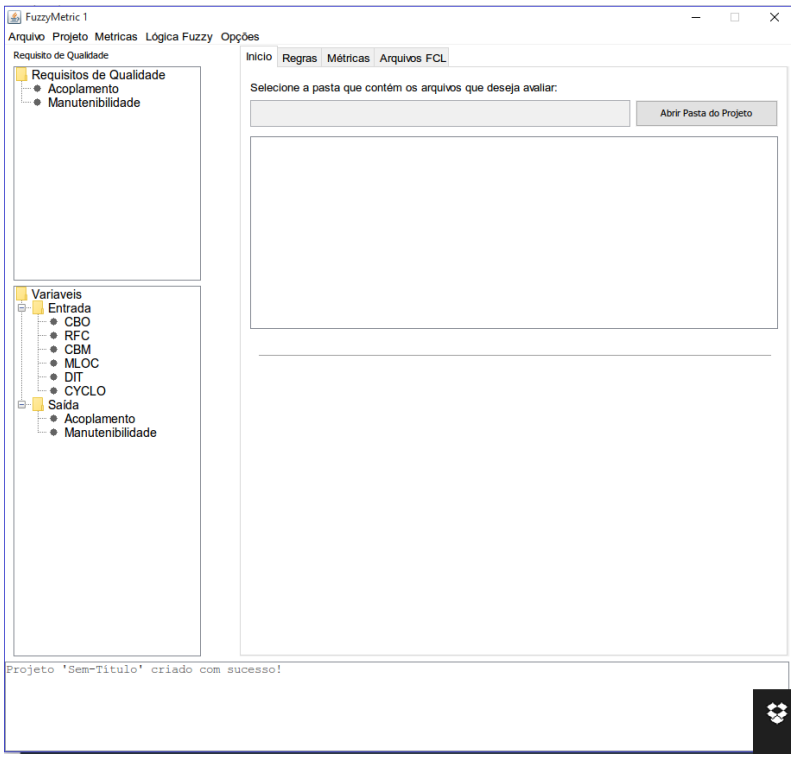

Figura 21 – Tela inicial da ferramenta, apresentando os requisitos selecionados
Fonte: Autoria própria

Ao usuário clicar em *Abrir Pasta do Projeto* deve selecionar a pasta onde encontram-se os arquivos compilados Java (.class) do projeto. A ferramenta busca dentro da pasta selecionado os arquivos compilados.

A ferramenta foi concebida para avaliação de projetos que contenham a pasta *build*, dentro desta procura a pasta *classes* e copia todos os arquivos do tipo *.class* para a pasta do projeto, definida anteriormente.

A Figura 22 mostra a ferramenta após o processo de abertura dos arquivos compilados. É exibida uma lista com todos os arquivos .class identificados, neste exemplo foram obtidos 32 arquivos dentro da pasta que será avaliada.

Após a copia, o usuário pode realizar a operação de avaliação (botão *Avaliar arquivos*). A ação vinculada a este botão consiste na execução do software *Extended Tool for Calculating Chidamber and Kemerer Java Metrics* (*CKJM extended*), disponível no endereço https://github.com/mjureczko/CKJM-extended. Essa ferramenta consiste em um programa desenvolvido com o intuito de calcular métricas do *bytecode* de arquivos Java compilados.

O funcionamento da ferramenta *CKJM Extended* consiste nas seguintes ações: o usuário

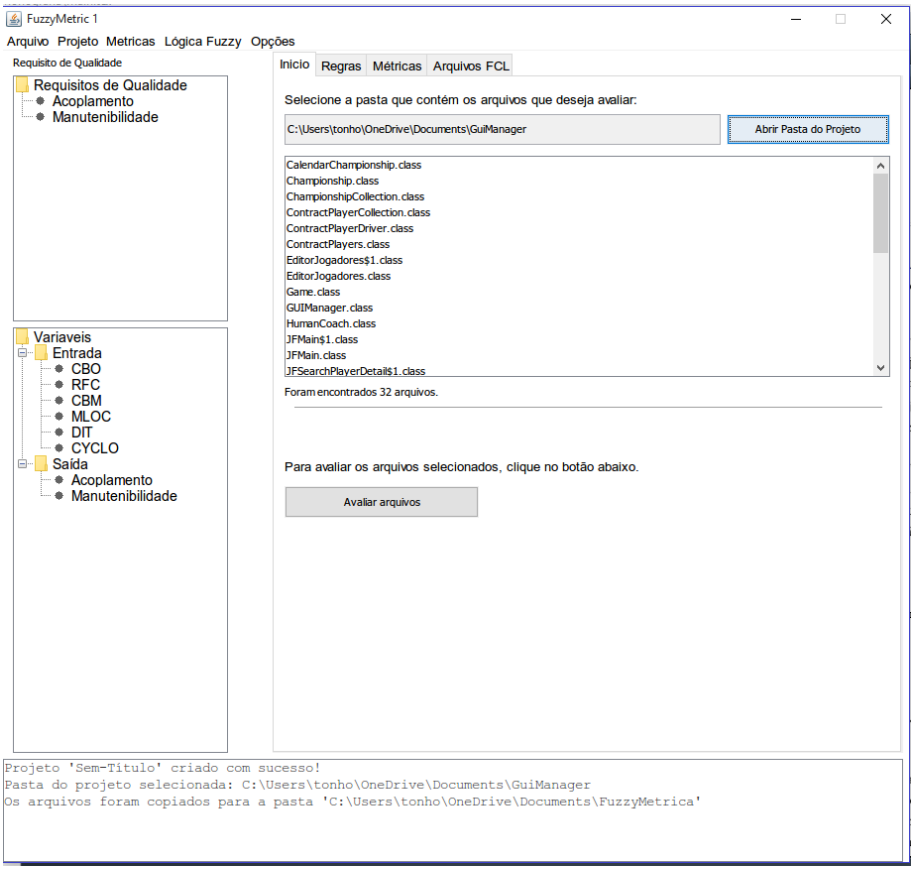

Figura 22 – Arquivos compilados abertos

Fonte: Autoria própria

determina o caminho para os arquivos desejados via linha de comando, a ferramenta mostra os resultados via saída padrão para cada um dos arquivos avaliados e a saída consiste no nome completo da classe e seus valores correspondentes. A ferramenta proposta redireciona a saída do *CKJM Extended* gerando um arquivo no formato *.xml* que contém informação referente aos arquivos selecionados pelo usuário. Esse arquivo *.xml* é salvo na pasta do projeto.

A Figura 23 mostra a execução da ferramenta *CKJM Extended* e é disponibilizado ao usuário uma saída informando qual é o comando utilizado para execução da ferramenta. É informado ao usuário ainda ,qual é a localização do arquivo que contém os valores das métricas obtidas.

Conforme exibido na Figura 23, após a etapa de avaliação das métricas, são disponibilizados ao usuário dois novos botões: *Ver XML* e *Definir Regras*. No primeiro é possível que o usuário visualize o arquivo gerado pela ferramenta *CKJM Extended*. No segundo, é disponibilizado ao usuário a definição das regras. A definição das regras consiste em uma etapa vinculada a elaboração da Base de Regras da lógica *fuzzy*.

A Figura 24 mostra como as regras são definidas dentro da ferramenta proposta. O usuário pode selecionar os diferentes elementos pertencentes ao sistema *fuzzy* para a criação das regras. Considerando que uma regra é formada por requisitos de qualidade, métricas e termos, nesta tela estão disponíveis ao usuários informações importantes para a definição das regras.

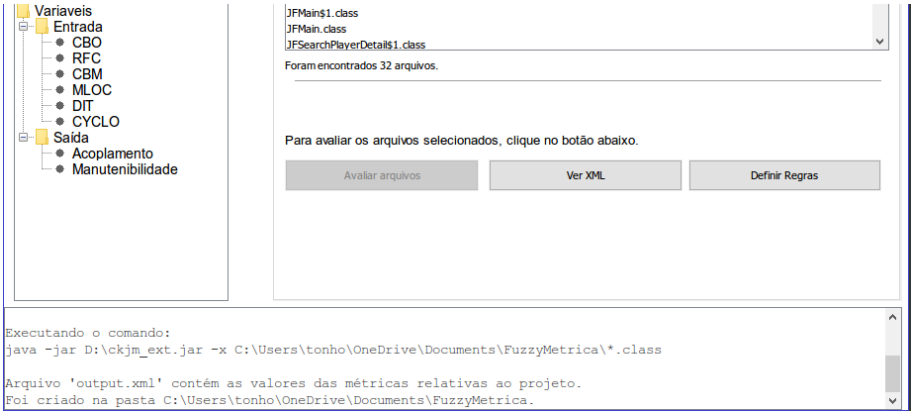

Figura 23 – Execução da ferramenta *CKJM Extended*
Fonte: Autoria própria

Após a escrita da regra o usuário deve clicar no botão *Adicionar Regra* para vincular a regra à Base de Regras.

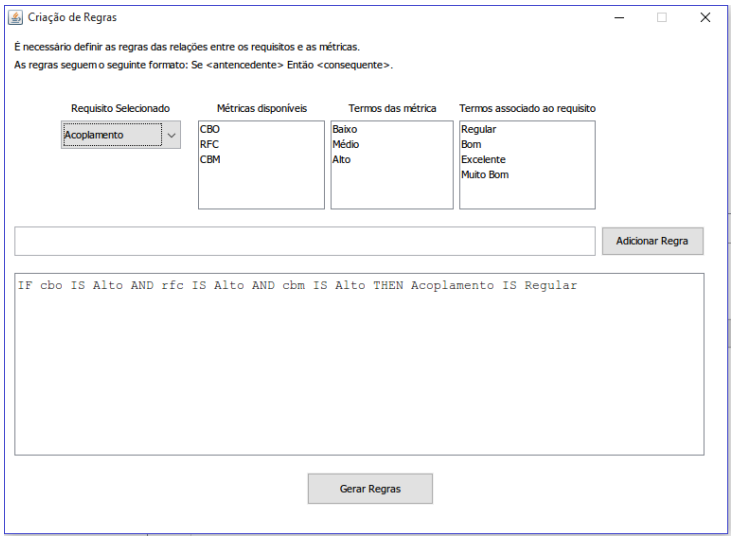

Figura 24 – Criação das regras
Fonte: Autoria própria

A Figura 25 ilustra os elementos utilizados para a formação de uma regra. Para cada elemento foi atribuída uma legenda numérica. Para o requisito (No caso, foi selecionado *Acoplamento*) foi atribuído o número 1. Estes estão dispostos em uma caixa de combinação, assim o usuário pode apenas ver e definir as regras para um único requisito por vez. As métricas relacionadas ao requisito selecionado na caixa de combinação estão disponíveis em uma lista, essa recebe o número 2. Os termos vinculados as métricas selecionadas são mostrados na lista identificada pelo número 3. Os termos vinculados ao requisito selecionado são exibidos na lista identificada pelo número 4. Caso o requisito seja alterado os itens das listas 2, 3 e 4 também serão atualizados.

Após o usuário terminar a definição das regras o sistema gera os arquivos *.fcl* e executa a ferramenta *jFuzzyLogic* para aplicação da lógica *fuzzy* e exibe o resultado final da avaliação.

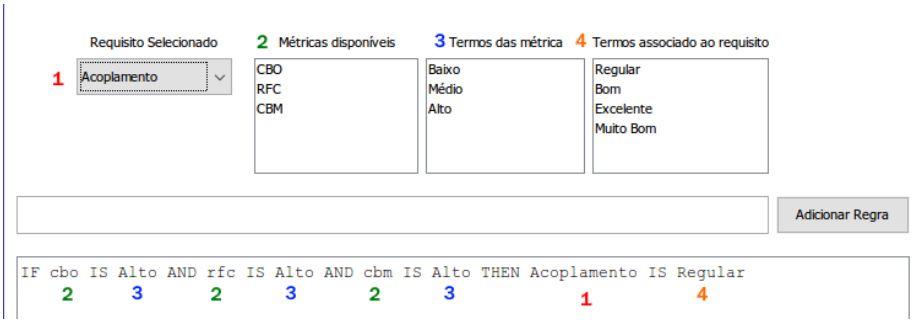

Figura 25 – Definição de uma regra

Fonte: Autoria própria

5.3 ANÁLISE DOS RESULTADOS

Para análise da execução da ferramenta foram utilizados dois projetos orientados a objetos: *Insulun Pump* e *Adapter*.

5.3.1 *Insulin Pump*

O pesquisador Ian Sommerville em seu livro *Software Engineering* (SOMMERVILLE, 2010) desenvolveu um estudo de caso chamado *Insulin Pump*, no qual ilustra vários aspectos que devem ser observados ao se criar sistemas que envolvem alta precisão e garantia de segurança ao usuário. Esse estudo de caso é composto por 11 classes e foi utilizado como base para o primeiro experimento.

O requisito de qualidade usado na avaliação foi a *Manutenibilidade*. Este possui duas métricas associadas no banco de dados e a partir da média dos valores obtidos em cada uma das métricas é possível determinar a qualidade geral do requisito avaliado. A ferramenta *CKJM Extended* gerou um arquivo *.xml* que contém as métricas de cada uma das classes. O Quadro 27 mostra a média para as métricas obtidas.

Quadro 28 – Média das métricas do sistema *Insulin Pump*

Métrica	Valor Observado
CYCLO	5.45
DIT	1.72

Fonte: Autoria própria

Depois de identificadas as métricas do sistema avaliado cabe ao usuário a definição das Base de Regras que deseja utilizar para a aplicação da lógica *fuzzy*. Para a avaliação da ferramenta foram definidas as seguintes regras:

Regra 1 : **SE** (DIT é Alto) **E** (CYCLO é Alto) **ENTÃO** Manutenibilidade **é** Regular

Regra 2 : **Se** (DIT é Baixo) **E** (CYCLO é Alto) **ENTÃO** Manutenibilidade **é** Bom

Regra 3 : **SE** (DIT é Alto) **E** (CYCLO é Baixo) **ENTÃO** Manutenibilidade **é** Bom

Regra 4 : **SE** (DIT é Baixo) **E** (CYCLO é Baixo) **ENTÃO** Manutenibilidade **é** Excelente

A partir da definição das regras, a ferramenta criou o arquivo .fcl e executou o *jFuzzyLogic*. Após as etapas de *Fuzzyficação* e *Deffuzyficação* as regras são analisadas de acordo com a entrada das métricas definidas no Quadro 27. De acordo com a ferramenta é possível classificar a qualidade do requisito selecionado tendo como base os termos CYCLO e DIT, conforme mostra a Figura 26.

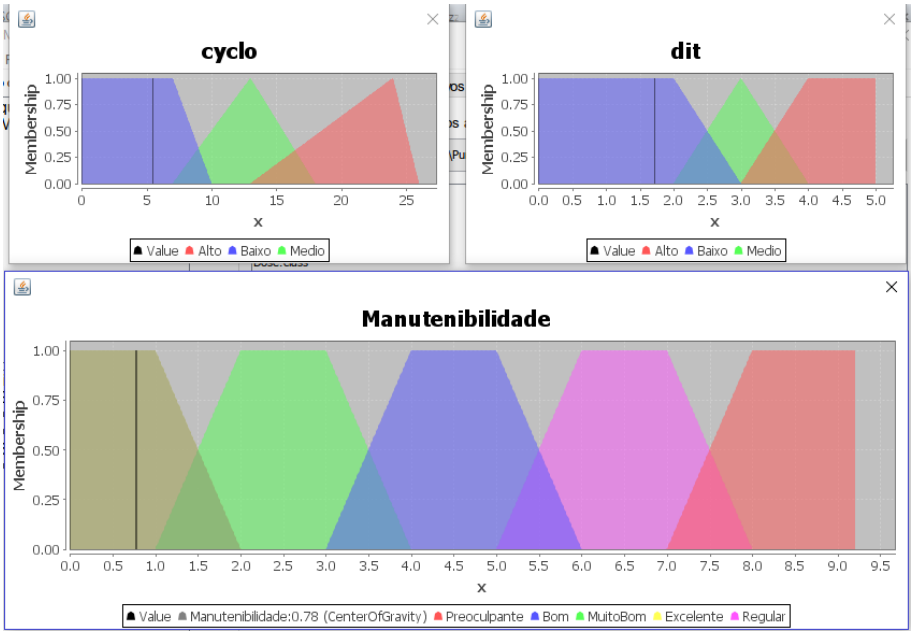

Figura 26 – Qualidade do requisito Manutenibilidade para *Insulin Pump*
Fonte: Autoria própria

Na Figura 26 observa-se que o valor da métrica CYCLO é igual a 5.45, este valor numérico está associado ao termo *Baixo*. O valor obtido para a métrica DIT é de 1.72, o que correspondeu ao termo *Baixo*. Com os dois valores para métricas termo *Baixo*, a Regra 4 foi executada e o valor do requisito de qualidade foi *Excelente*.

5.3.2 *Adapter*

No endereço https://github.com/iluwatar/java-design-patterns estão disponíveis exemplos de implementações de *design patterns* desenvolvidos para a linguagem Java. Foi escolhida a aplicação do design pattern: *Adapter* para ser avaliado. O projeto *Adapter* contém cinco classes e elas foram compiladas, testadas e avaliadas pela ferramenta proposta.

Para este caso, optou-se pela avaliação do requisito de qualidade *Acoplamento*. Este possui três métricas associadas no banco de dados e a partir da média dos valores obtidos em cada uma das métricas é possível se determinar a qualidade geral do requisito avaliado. A ferramenta *CKJM Extended* gerou um arquivo *.xml* que contém as métricas de cada uma das classes. O Quadro 28 apresenta a média para as métricas obtidas.

Quadro 29 – Média das métricas do sistema *Adapter*

Métrica	Valor Observado
CBM	0
CBO	2.5
RFC	5.1

Fonte: Autoria própria

Depois de identificadas as métricas do sistema avaliado, definiu-se a seguinte Base de Regras:

Regra 1 : **SE** CBO **é** Alto **E** RFC **é** Alto **E** CBM **é** Baixo **ENTÃO** Acoplamento **é** Regular

Regra 2 : **SE** CBO **é** Alto **E** RFC **é** Alto **E** CBM **é** Alto **ENTÃO** Acoplamento **é** Regular

Regra 3 : **SE** CBO **é** Alto **E** RFC **é** Médio **E** CBM **é** Médio **ENTÃO** Acoplamento **é** Bom **é** Bom

Regra 4 : **SE** CBO **é** Baixo **E** RFC **é** Baixo **E** CBM **é** Alto **ENTÃO** Acoplamento **é** MuitoBom

Regra 5 : **SE** CBO **é** Baixo **E** RFC **é** Alto **E** CBM **é** Baixo **ENTÃO** Acoplamento **é** Bom

Regra 6 : **SE** CBO **é** Baixo **E** RFC **é** Baixo **E** CBM **é** Baixo **ENTÃO** Acoplamento **é** Excelente

De acordo com os valores das métricas obtidas, a Regra 6 será selecionada. Ao se realizar a etapa de *Deffuzificação* a qualidade do requisito avaliado é *Excelente* de acordo com a ferramenta proposta, conforme mostra a Figura 27.

A partir do teste dos dois estudos de caso é possível observar que os códigos-fonte selecionados apresentam um bom nível em relação ao requisito de qualidade escolhido para a avaliação. Nota-se que um código-fonte de alta qualidade geralmente apresenta bons níveis em várias métricas. Como nos dois estudos de caso, em que para as métricas avaliadas os valores foram baixos, e o resultado foi positivo.

Apesar de serem apresentados resultados com dois estudos de caso, a ferramenta deve ser utilizada para a identificação de qualidade em projetos mais complexos. Ressalta-se que ela já esta preparada para avaliar qualquer tipo de requisito de qualidade.

A grande dificuldade para realização dos experimentos foi a criação das regras, pois um conjunto de regras devem ser elaboradas de forma a atender todo o domínio. Nos estudos de casos foram criadas algumas das regras.

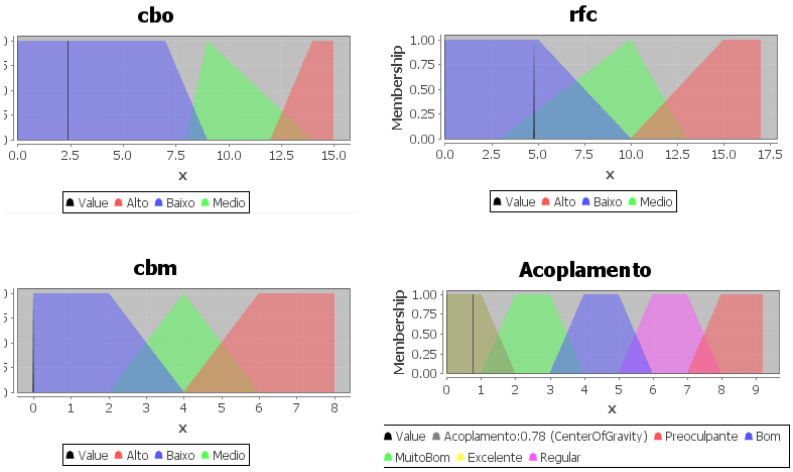

Figura 27 – Qualidade do requisito Acoplamento
Fonte: Autoria própria

5.4 COMPARAÇÃO ENTRE OS TRABALHOS RELACIONADOS

O trabalho desenvolvido por Avidagic, Boskovic e Causevic (2008) avaliou o código-fonte orientado a objeto para analisar o requisito de qualidade Manutenibilidade, utilizando como entrada os *bad smells* (códigos mal desenvolvidos que dificultam a leitura e compreesão). Os *bad smells* foram escolhidos por estarem relacionados diretamente com o requisito de qualidade e a lista dos *bad smells* foram identificados usando como base o trabalho desenvolvido por Fowler (2000). O trabalho proposto também realiza a avaliação de código-fonte orientado a objeto, porém não considera somente um requisito de qualidade a ser avaliado, permitindo ao usuário ter uma visão mais ampla de quais requisitos de qualidade em seu projeto estão com a qualidade esperada e quantos ainda precisam ser melhores aperfeiçoados usando, por exemplo, as técnicas de Fowler (2000).

Assim como no trabalho de Avidagic, Boskovic e Causevic (2008) , a avaliação proposta também aplicou a lógica *fuzzy* para identificar a qualidade não somente do requisito de Manutenibilidade quanto também de outros requisitos, pois realizou a correlação entre requisitos com suas respectivas métricas.

O processo de *Fuzzyficação* e *Defuzzyficação* proposta por Avidagic, Boskovic e Causevic (2008) é realizado manualmente, por sua vez na avaliação proposta o mecanismo foi automatizado usando como *engine* a ferramenta *jFuzzyLogic*. As associações entre métricas, termos e requisitos são geradas automaticamente quando a ferramenta é inicializada. Além disto, oferece ao usuário a possibilidade de inserção de novos termos, requisitos de qualidade e métricas, pois todos os dados são armazenados em um banco de dados e podem ser reusados em novos projetos.

Outro diferencial deste trabalho em relação ao de Avidagic, Boskovic e Causevic (2008) está em que os valores das métricas são obtidas por uma ferramenta automatizada *CKJM Extended*

e geradas para um arquivo *.xml* o qual é lido pela *engine* de lógica *fuzzy*. Tanto no trabalho Avidagic, Boskovic e Causevic (2008) quando neste a elaboração de regras ocorre de forma manual, porém a diferença é que na avaliação proposta ela deve ser inserida dentro da ferramenta para que avaliação do código-fonte seja realizada.

O Quadro 27 ilustra de forma resumida as diferenças e semelhanças entre o trabalho de Avidagic, Boskovic e Causevic (2008) e o proposto que foram descritas anteriormente.

Quadro 30 – Comparação entre trabalhos relacionados

Critérios de Comparação	**Avidagic, Boskovic e Causevic (2008)**	**Avaliação Propósta**
Processo de *Fuzzyficação* de *Deffuzyficação*	Manual	Automatizado
Criação de Regras	Manual	Manual
Uso de código-fonte orientado a objeto	Sim	Sim
Quantidade de requisitos de qualidade avaliados	=1	>= 1
Suporte a um ambiente automatizado para avaliação de código-fonte	Não. Porém usa software para obter os valores das métricas	Sim
Uso de valores *default* para métricas, termos e requisitos de qualidade em um projeto	Não	Sim
Trabalho com conjunto de classes	Não	Sim

Fonte: Autoria própria

Apesar das diferenças entre o trabalho proposto e o que foi desenvolvido por Avidagic, Boskovic e Causevic (2008), pode-se afirmar que este foi a base para o desenvolvimento desta pesquisa.

6 CONCLUSÃO

A avaliação de um projeto orientado a objetos foi proposta neste trabalho com a finalidade de verificar se um requisito de qualidade para o código-fonte é classificado, por exemplo, como: excelente, muito bom, bom, regular ou preocupante. Como a faixa de classificação que se pode ter para o código-fonte não é 0 e 1, utilizou-se a lógica fuzzy para a manipulação de incerteza e é na literatura uma das mais usadas para esta situação.

Inicialmente foram identificadas métricas de software aplicadas ao código-fonte orientado a objeto, tais como: CBO (Acoplamento entre Classes de Objetos), RFC (Resposta de Classe), COF (Fator de Acoplamento) as quais foram correlacionadas com o requisito de qualidade Acoplamento, conforme exibido no Quadro 17. Foram pesquisadas ferramentas automatizadas capazes de calcular os valores das métricas de software e executar a lógica *fuzzy*, dentre elas, optou-se pelo uso da *CKJM Extended* e *jFuzzyLogic*, respectivamente.

O processo de avaliação proposto permite ao usuário criar um projeto, abrir um projeto contendo os arquivos compilados Java, escolher ou criar métricas, termos ou requisitos de qualidade e de forma automatizada é capaz de: criar a relação entre os requisitos de qualidade e métricas, chamar o software *CKJM Extended* - que gera para cada métrica um valor quantitativo - para gravar os valores numéricos para métrica em um arquivo *.xml*, gerar o arquivo *.fcl* que é entrada para a (*jFuzzyLogic*) a qual realiza o processo de *Fuzzyficação* e *Deffuzificação* e devolve o resultado para a ferramenta que será exibido ao usuário. A implementação deste processo permitiu a criação da ferramenta para avaliação de código-fonte baseado em lógica fuzzy, métricas e requisitos de qualidade.

A ferramenta foi aplicada em dois estudos de caso, *Insulin Pump* e *Adapter*, nos quais foi constatado que para cada métrica os termos identificados foram baixos, o que resultou em uma análise positiva para o requisito de qualidade analisado, a saber, Manutenibilidade e Acoplamento.

A avaliação proposta neste trabalho difere do estabelecido por Avidagic, Boskovic e Causevic (2008) porque fornece uma forma automatizada para avaliar o código-fonte, correlaciona requisitos de qualidade com métricas, avalia um conjunto de classes e um conjunto de requisitos.

6.1 TRABALHOS FUTUROS

A partir deste trabalho novas pesquisas podem ser desenvolvidas. São elas:

- Refatoração da ferramenta proposta para aumentar sua reusabilidade, manutenibilidade e flexibilidade.
- Criação automatizadas da Base de Regras.

- Identificação de novas correlação entre os requisitos de qualidade e métricas.
- Realização novos testes com estudos de casos mais complexos para verificar eficiência da ferramenta proposta.
- Adição de novas ferramentas para calcular métricas e lógica *fuzzy*.

REFERÊNCIAS

ABREU, Fernando B. Candidate métrics for object-oriented software within a taxonomy framework. **Journal of Systems of Software**, Elsevier Science, North-Holland, v. 26, n. 1, 1994.

ARCHER, Clark; STINSON, Michael. **Object-Oriented Software Measures**. [S.l.], 1995.

AVIDAGIC, Zikrija; BOSKOVIC, Dusanka; CAUSEVIC, Aida. Code evaluation using fuzzy logic. **Anais**... WSEAS Conference Fuzzy 2008, Sofia, Bulgaria, n. 1, 2008.

BOOCH, Grady. **Object-oriented Analysis and Design with Applications**. 2 ed. Redwood City: Benjamin-Cummings Publishing Co., Inc. 1994.

CHAUHAN, Ritu et al. Estimation of software quality using object oriented design metrics. In: . India: IJIRCCE, 2014. **International Journal of Innovative Research in Computer and Communication Engineering**, p. 2581 – 2586. ISBN 2320-9801.

CHEN, Bindi. **Mamdani Fuzzy Models**. 2013. <www.bindchen.co.uk/post/AI/mamdani-fuzzy -model.html>. The Fundamentals - Fuzzy System. Acesso em: 26 ago. 2015.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 20, n. 6, p. 476–493, 1994.

CINGOLANI, Pablo; ALCALÀ-FDEZ, Jesus. jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. **In:** IEEE. Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on. In:. [S.l.], 2012. p. 1–8.

FENTON, Norman E.; NEIL, Martin. Software metrics: Roadmap. **Proceedings** of the Conference on The Future of Software Engineering. New York, NY, USA: ACM, 2000. (ICSE ’00), p. 357–370.

FENTON, Norman E.; PFLEEGER, Shari L. **Software Metrics - A Rigorous & Practical Approach**. 2 nd. ed. Boston, MA, US: PWS Publishing Company, 1997.

FILO, Tarcísio Guerra Savino. **Identificação de valores referência para métricas de softwares orientados por objetos**. 2014. 197 f. Dissertação — Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 2003. Disponível em: <www.dcc.ufmg. br/pos /cursos/defesas/1849M.pdf> . Acesso em: 25 out. 2015.

FOWLER, M. **Improving the Design of Existing Code**. 1. ed. New York: Addision-W esley Publishing, 2000.

GALIN, Daniel. **Software Quality Assurance**. 1. ed. London: Pearson, 2004. ISO/IEC9126-1.

ISO IEC 9126-1. **Software Engineering - product quality - part 1: Quality model**. [S.l.], 2001. Disponível em: . Acesso em: 24 abr. 2015.

KITCHENHAM, Barbara. Measuring software development. **IEEE Software**, Elsevier, Amsterdam, p. 303–31, 1990.

KLINKHACHORN, Professor Powsiri. **Class Notes**. 2004.<www.csee.wvu.edu/classes/cpe521/ presentations/Membership.pdf> . CpE521 - Fuzzy Logic. Acesso em: 01 nov. 2015.

LEE, Kwang H. **First Course on Fuzzy Theory and Applications**. 3. ed. [S.l.]: Springer, 2005. MILLS, Everald E. Software Metrics - SEI Curriculum Module SEI-CM-12-1.1. [S.l.], 1988.

MIRANDA, Pedro; JUNIOR, Mauro Barbosa Vilela; KRONBAUER, Diego. **Sistema de Controle Difuso de Mamdani - Aplicações Pêndulo Invertido e outros**. 2003. 49 f. Monografia — Departamento de Computação e Estatística, Universidade Federal de Mato Grosso do Sul, Campo Grande, 2003. Disponível em: <www.dct.ufms.br/~mzanusso/producao/ PedroMir.pdf>. Acesso em: 20 ago. 2015.

NCSU, Realsearch Group at. **An Introduction to Object-Oriented Metrics**. 2013. . An Introduction to Object-Oriented Metrics. Acesso em: 23 out. 2015.

ORTEGA, Neli Regina Siqueira. **Aplicação da Teoria de Conjuntos Fuzzy a Problemas da Biomedicina**. 2001. 166 f. Tese (Doutorado) — Instituto de Física, Universidade de São Paulo, São Paulo, 2001. Disponível em: <www.ime.usp.br/~tonelli/verao-fuzzy/neli/principal.pdf>. Acesso em: 11 set. 2015.

PRESSMAN, Roger S. **Software Engineering: A practitioner's aprroach**. 5. ed. [S.l.]: Mc Graw Hill, 2001.

ROSS, Timothy J. **Fuzzy Logic With Engineering Applications**. 3 rd. ed. [S.l.]: Wiley, 2010.

SOMMERVILLE, Ian. **Software Engineering**. 9. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2010. SQLITE. SQLite. 2015. . SQLite. Acesso em: 01 nov. 2015.

SQLITE. SQLite. 201 <www.sqlite.ortg> SQLite. Acesso em 01 nov. 2015.

ZADEH, Lofti A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338 – 353, 1965.

_____. The concept of a linguistic variable and its application to approximate reasoning. **In:**. Journal of Information Science, p. 199, 1975.

_____. **Fuzzy Sets and Their Applications to Cognitive and Decision Processes**. 3 rd. ed. Berkeley: Wiley, 1975. Disponível em: . Acesso em: 14 mai. 2015. . Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, p. 1 – 28, 1978. Acesso em: 14 mai. 2015.