UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DION MAICON EUZÉBIO DUARTE

ALODIN: UM MÉTODO DE ALOCAÇÃO DE RECURSOS DIFUSO-INDUTIVO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

DION MAICON EUZÉBIO DUARTE

ALODIN: UM MÉTODO DE ALOCAÇÃO DE RECURSOS DIFUSO-INDUTIVO

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.

Orientador: Prof. MSc. Clayton Kossoski

Coorientadora: Profa. Dra. Simone Nasser Matos

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

ALODIN: UM MÉTODO DE ALOCAÇÃO DE RECURSOS DIFUSO-INDUTIVO
por

DION MAICON EUZÉBIO DUARTE

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 05 de junho de 2018 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.

Prof. MSc. Clayton Kossoski
Orientador

Prof. Dra. Simone Nasser Matos
Coorientadora

Profa. Dra. Simone de Almeida
Membro titular

Prof. MSc. Vinícius Camargo Andrade
Membro titular

Profa. Dra. Helyane B. Borges
Responsável pelo Trabalho de Conclusão de Curso

Prof. MSc. Saulo J. B. Queiroz
Coordenador do curso

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

AGRADECIMENTOS

Agradeço primeiramente a Deus por tudo.

Agradeço a minha família pela educação e por entender meus momentos de ausência, principalmente a minha maior incentivadora, minha mãe Josandra.

Agradeço as políticas públicas sociais que me proporcionaram a oportunidade de estudar em uma das melhores universidades do país.

Agradeço aos meus orientadores Simone Nasser e Clayton Kossoski pela paciência, conhecimento e todo o tempo empenhado neste trabalho.

Agradeço ao núcleo de assistência estudantil da universidade por todo o suporte.

Agradeço a todos os gestos de simplicidade, humildade e generosidade de pessoas que conheci durante essa jornada, foram esses gestos que me encheram de esperança e me fizeram continuar.

“Never forget what you are.
The rest of the world will not.
Wear it like armor, and it can
never be used to hurt you”
Tyrion Lannister (GoT)

DEDICÁTORIA

Ao povo brasileiro, sobre tudo a aqueles que não conseguiram o privilégio do ensino superior no nosso país. Aos meus amigos que sempre acreditaram e me deram força.

RESUMO

DUARTE, Dion Maicon Euzébio. **ALODIN: Um Método de Alocação de Recursos Difuso-Indutivo**. 2018. 83 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

A alocação de recursos humanos em projetos de software é uma atividade fundamental na gerência de projetos. O gerente é o indivíduo melhor qualificado para avaliar a equipe quanto às habilidades e capacidades de cada um de entregar software de melhor qualidade. Na literatura, foram encontradas abordagens de alocação que remetem ao uso de métricas de produto para estimar custos e prazos e outros usam a avaliação de habilidades técnicas para estimar o mesmo, porém as duas formas são feitas isoladamente. Este trabalho criou um método de alocação de recursos, denominado de AloDIn, fundamentado em lógica *fuzzy*, métricas, requisitos de qualidade e conjunto de habilidades trabalhados de forma integrada. O método possui 4 etapas principais: i) determinar o conjunto de habilidade dos recursos usando *fuzzy*, ii) identificar a qualidade de código-fonte de cada recurso usando *fuzzy*, iii) alocar os recursos usando como base os valores obtidos nas etapas anteriores e realizar seu cruzamento por meio da lógica *fuzzy*, iv)finalização do projeto e atualização das habilidades dos recursos. Ao fazer uso dométodo proposto, o gerente terá a disposição métricas que podem lhe ajudar a realizar uma alocação mais próxima ao ideal, evitando apenas a subjetividade muito comum nesta etapa.

Palavras-chave: Alocação de Recursos. Lógica Fuzzy. Requisitos de Qualidade. Métricas. Habilidades.

ABSTRACT

DUARTE, Dion Maicon Euzébio. **ALODIN: A Diffuse-Inductive Resource Allocation Method.** 2018. 83 sheets. Final Paper (Bachelor of Science in Computer Science) - Federal Technological University of Paraná. Ponta Grossa, 2018.

The allocation of human resources in software projects is a fundamental activity in project management. The manager is the best qualified individual to assess the team skills and abilities to deliver better quality software. In the literature were found several allocation approaches that refer to the use of product metrics to estimate costs and deadlines, some authors use the assessment of technical skills to estimate the same, but the two forms are done isolated. This work created a method of allocation of resources, called AloDIn, based on fuzzy logic, metrics, quality requirements and the set of skills working in an integrated mode. The method has 4 main steps: i) determine the skill set of the resources using fuzzy, ii) identify the source code quality of each resource using fuzzy, iii) allocate resources based on the values obtained in previous steps and perform their crossing through fuzzy logic, iv) finalization of the project and updating the abilities of the resources. Using the proposed method, the manager focuses not only over subjectivity of the resource's abilities, but uses metrics that can help him achieve an allocation that is closer to the ideal.

Keywords: Allocation of Resources. Fuzzy Logic. Quality Requirements. Metrics. Skills.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
AC	Aferent Coupling (Acoplamento Aferente)
CBO	Coupling between object classes (Acoplamento entre Classes de Objetos)
CMMI	Capability Maturity Model Integration
COF	Coupling Factor (Fator Acoplamento)
Crisp	Valores numéricos utilizados em operações lógicas
DIT	Depth of Inheritance Tree (Profundidade de Árvore de Herança)
dof	Degree of Fire (Grau de força do disparo da regra)
EC	Eferent Coupling (Acoplamento Eferente)
FCL	Fuzzy Control Language
FIS	Fuzzy Inference System
IETEC	Instituto de Educação Tecnológica
ITIL	Information Technology Infrastructure Library
Java	Linguagem de programação orientada a objetos
LCOM	Lack of Cohesion in Methods (Ausência de Coesão em Métodos)
MHF	Method Hiding Factor (Fator Ocultação de Método)
MLOC	Lines of Code per Method (Linhas de Código por Método)
MOOD	Metrics for Object Oriented Design
NAC	Number of Ancestor Classes (Número de Classes Ancestrais)
NBD	Nested Block Depth (Profundidade de Blocos Aninhados)
NDC	Number of Descendent Classes (Número de Classes Descendentes)
NLM	Number of Local Methods (Número de Métodos Locais)
NOC	Number of Children (Número de Filhos)
PF	Polymorphism Factor (Fator de Polimorfismo)
PMBOK	Project Management Body of Knowledge
PMI	Project Management Institute
RF	Reuse Factor (Fator de Reúso)
NAC	Number of Ancestor Classes (Número de Classes Ancestrais)
RFC	Response for a Class (Resposta de Classe)
RMD	Normalized Distance (Distância Normalizada)
WMC	Weighted Methods Per Class (Métodos Ponderados por Classe)

SUMÁRIO

1 INTRODUÇÃO

O gerenciamento de projetos surgiu da adaptação a novas estratégias de mercado, as quais visam a qualidade de produtos e serviços, por meio do reuso de técnicas de desenvolvimento de projetos (PMBOK, 2013).

De acordo com Heldeman (2005), um projeto é algo não cotidiano e se destina a dar origem a um serviço, produto ou processo, possuindo prazo limitado e natureza temporária. O gerenciamento de projetos é constituído de um conjunto de processos que envolvem o planejamento, a organização e o controle dos aspectos de um projeto (SANTOS; CARVALHO, 2006).

Os projetos são divididos em etapas, chamados de ciclo de vida: Iniciação, Organização e Preparação, Execução e Encerramento (PMBOK, 2013) e em todas há necessidade de gestão de recursos humanos, direta ou indiretamente, seja para a sua readequação, alocação ou remoção. Na gerência de projetos de recursos humanos, o termo recurso é usado para referenciar indivíduo ao longo deste trabalho.

De acordo com o PMBOK (2013), documento de referência do gerenciador de projetos e criado pelo *Project Management Institute* (PMI), um bom gerente de projetos deve possuir um portfólio contendo subportfólios, programas e operações que podem ser usados para a obtenção de resultados precisos e categóricos, moldados para a necessidade de seu plano estratégico. O gerenciamento de portfólios permite, entre outras coisas, autorizar a alocação de recursos humanos do projeto.

A alocação de recursos humanos é um problema estudado na pesquisa operacional. Otero (2009) descreve dois passos no processo de alocação de recursos humanos em projetos de software: o primeiro é obter o número de engenheiros de software necessário e o segundo é que esses engenheiros tenham as habilidades requeridas no projeto. O segundo passo é dependente do primeiro e quando não há uma combinação adequada, a alocação do engenheiro passa a ser analisada recursivamente, na maioria das vezes acaba identificando a necessidade de *outsourcing* (terceirização de recursos), o que torna o projeto caro e demorado.

De acordo com Coffman Jr. et a.l(1997) o problema de alocação de equipes pode ser pensado como um Problema de Empacotamento (*Bin Packing*), classificado como NP-Difícil e pertencente à classe mais geral de Problemas da Mochila (*Knapsack Problems*), o que demonstra a necessidade de uso de técnicas de otimização. Burdett e Li (1995) foram os primeiros a quantificar as capacidades técnicas dos recursos humanos, associá-las com atividades e combiná-las em conjuntos para produzir equipes eficazes em um problema de alocação de recursos humanos.

Silva (2009) propõe a alocação por intermédio de programação linear e dinâmica, com foco em minimização de tempo e de custos, respectivamente. Utilizando a programação linear, modelou a alocação como um Problema de Designação, propondo a utilização de métricas em especial a métrica de software *Lines of Code* (*LOC*), também utilizada por Otero (2009), como fator determinante e que pode indicar a quantidade de tempo que será gasto no projeto.

Santos (2014) apresenta um modelo para a seleção de equipes dinamicamente distribuídas. Algo que pode ser modelado para uso na alocação de indivíduos dentro de uma única equipe, auxiliando em sua formação. Santos (2014) aplica a lógica *fuzzy* (ZADEH, 1965) para caracterizar o conhecimento dos indivíduos, isto é possível porque *fuzzy* é uma lógica capaz de representar melhor o conhecimento, quando comparada a lógica clássica, pois diminui a incerteza sobre a subjetividade de um processo de seleção.

A maioria dos modelos propostos na literatura são voltados para ambientes gerais. Em ambientes de desenvolvimento de software, alguns requisitos não-funcionais do projeto podem ser melhores trabalhados, principalmente durante o planejamento. Sendo assim, alocar recursos de acordo com suas habilidades e metas de qualidade que é capaz de atingir, auxilia ao gerente, pois filtra o conjunto de candidatos para a formação de uma equipe já em seu planejamento. Logo, a alocação de recursos seguindo estes critérios deverá melhorar a qualidade final do projeto ao distribuir tarefas adequadas a capacidade de cada indivíduo.

Este trabalho criou um método de alocação de recursos, denominado de AloDIn, fundamentado em lógica *fuzzy*, métricas, requisitos de qualidade e conjunto de habilidades trabalhados de forma integrada. O método auxilia o gerente de projetos na alocação de recursos humanos em equipes de projeto de software.

Os trabalhos encontrados na literatura de Santos (2014) e Oliveira (2015) apresentaram ferramentas interessantes que foram adaptadas e utilizadas no método. O AloDIn realiza a análise de habilidades dos recursos humanos e seus códigos-fontes, por meio da análise de código fonte é possível estimar a qualidade de software que um recurso pode atingir e por meio da análise das habilidades é possível dar preferência a recursos que tem as habilidades mais impactantes para o projeto.

O método possui 4 etapas principais: i) determinar o conjunto de habilidade dos recursos usando *fuzzy*, ii) identificar a qualidade de código-fonte de cada recurso usando *fuzzy*, iii) alocar os recursos usando como base os valores obtidos nas etapas anteriores e realizar seu cruzamento por meio da lógica *fuzzy*, iv) finalização do projeto e atualização das habilidades dos recursos.

A lógica *fuzzy* é utilizada para que as decisões de alocação sejam menos subjetivas. Por exemplo, ao utilizar a lógica *fuzzy* para avaliação de qualidade, as habilidades do individuo em criar software de fácil reuso podem ser associadas por meio das seguintes métricas: Profundidade da Área de Herança (*DIT*) e Tamanho do Programa em Linhas de Código (*LOC*). Por meio da lógica *fuzzy* são atribuídos para as métricas coletadas valores qualitativos, tais como: ruim, bom, muito bom e excelente. Esses valores são então manipulados mediante o uso de inferência lógica para obter um novo valor que pode ser usado para estimar a qualidade de reusabilidade alcançada.

Utilizando o AloDIn ao avaliar o conjunto de habilidades o gerente pode definir múltiplas combinações de habilidades que são relevantes para a qualidade final do projeto. As habilidades podem ser técnicas e não-técnicas e por meio da lógica *fuzzy* é obtido um valor que indica a qualidade do conjunto de habilidades de cada recurso. O método proposto deve apresentar como resultado a disposição do recurso humano mais qualificado para compor um projeto, de acordo com um conjunto de recursos avaliados.

1.1 OBJETIVOS

O objetivo geral deste trabalho é criar um método de apoio a decisão para alocar recursos humanos para projetos de software, utilizando requisitos não-funcionais, lógica *fuzzy*, conjunto de habilidades e métricas de software. A abordagem visa alocar recursos com habilidades que possam agregar qualidade ao software.

Para alcançar o objetivo geral, se tem os seguintes objetivos específicos:

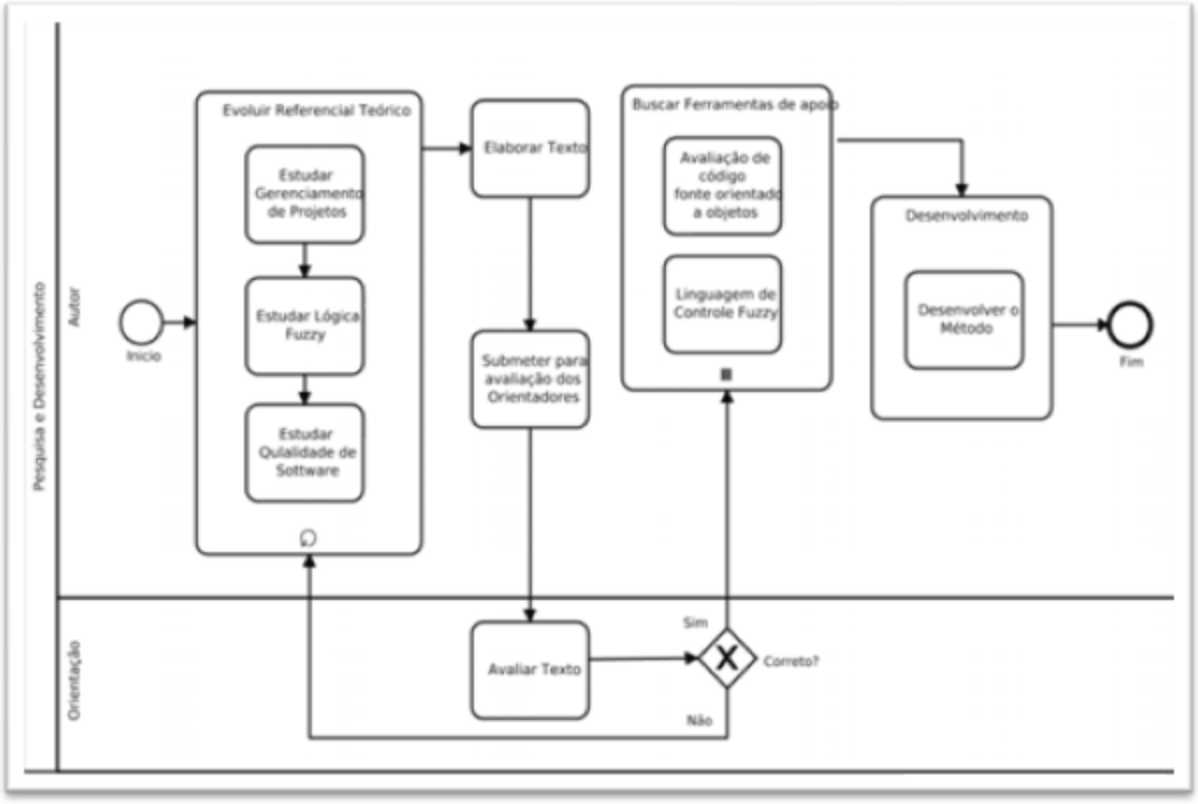
- Analisar o conjunto de habilidades dos recursos, atribuir pesos as habilidades e aplicar políticas de seleção através da lógica *fuzzy.*
- Analisar o código-fonte dos recursos e aplicar políticas de seleção baseadas em qualidade e métricas de software.
- Testar o modelo proposto em casos de teste, para a verificação dos resultados.

1.2 METODOLOGIA DE PESQUISA

O processo metodológico usado para a criação do método proposto está representado na Figura 1 através de um modelo BPMN (*Business Process Model and Notation*). Durante a tarefa de *estudar gerenciamento de projetos* o foco foi nos problemas de alocação de recursos humanos, identificou-se o recorrente problema da subjetividade durante a decisão sobre qual recurso alocar.

A atividade de *estudar lógica fuzzy* foi uma consequência de identificar o seu uso na aplicação desenvolvida por Oliveira (2015). Foi durante está tarefa que a lógica *fuzzy* tornou-se uma ferramenta útil no problema da alocação de recursos humanos. Foi realizado também um estudo sobre o método desenvolvido por Santos (2014) que propunha a avaliação do conjunto de habilidades de equipes, de acordo com requisitos de projeto, por intermédio da lógica *fuzzy.* O resultado apresentado ao fim do método proposto por Santos (2014) é um arranjo de equipes tecnicamente classificadas de acordo com as habilidades de seus integrantes.

Figura 1 – BPMN da metodologia de pesquisa



Fonte: Autoria própria

O *estudo da qualidade de software* foi fundamental devido ao objetivo do método proposto que é alocar recursos tentando atingir o melhor nível na qualidade final do software. Durante está atividade foram estudados os aspectos de qualidade em diferentes perspectivas: Processo; Desenvolvimento e Gerenciamento de Projetos de Software. Nesta atividade se identificou a complexidade de alocação em projetos de software com finalidade de qualidade e como a alocação de recursos humanos interfere na qualidade final de processos e produtos de software.

Após a revisão teórica, foi primordial fazer uma procura nos trabalhos correlacionados *buscando ferramentas de apoio* que seriam utilizadas no método. É importante ressaltar que a aplicação desenvolvida por Oliveira (2015) não será utilizada neste trabalho, mas sua teoria sim. A linguagem de Controle *Fuzzy* (FCL) é a padrão para programas de controle *fuzzy*, por meio dela a *jLogicFuzzy* (CINGOLANI; ALCALÀ-FDEZ, 2012) utilizada por Oliveira (2015) implementa seus procedimentos. No entanto, para este trabalho o funcionamento e os estudos de

caso são demonstrados utilizando a *GNU Octave Platform* e *Octave Fuzzy Logic Toolkit*[1].

O *desenvolvimento do método* foi centrado na seguinte ideia: Se um software pode ser avaliado quanto a sua qualidade, então um desenvolvedor que trabalhou no projeto tem uma participação no desenvolvimento e, portanto um valor pode ser a ele atribuído como conhecimento prático sobre a qualidade computada. O valor da participação do desenvolvedor no projeto ou na implementação de módulos deve ser ponderado pelo gerente ou líder de projeto.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em seis capítulos. O capítulo 2 apresenta os fundamentos e conceitos da gerência de projetos e a alocação de recursos humanos em projetos de software. O capítulo 3 mostra os fundamentos da lógica fuzzy e o trabalho de Oliveira (2015). No capítulo 4 é descrito o processo de desenvolvimento do método proposto e o seu funcionamento. No capítulo 5 são mostrados dois casos de teste utilizando o método proposto. No capítulo 6 é realizada a conclusão do trabalho, a avaliação da contribuição e suas limitações.

[1] Disponível no site: https::://www.gnu.org/software/octave.

2 GERENCIAMENTO DE PROJETOS

Neste capítulo são descritos conceitos fundamentais relacionados ao desenvolvimento deste trabalho. A seção 2.1 apresenta conceitos acerca de gerenciamento de projetos. A seção 2.2 descreve o gerenciamento de recursos humanos de uma forma geral e apresenta os conceitos sobre a alocação de recursos humanos em projetos de software. A seção 2.3 relata sobre a alocação de recursos, bem como o trabalho relacionado ao tema desta pesquisa. Por fim, a seção 2.4 relata as considerações finais do capítulo.

2.1 GERENCIAMENTO DE PROJETOS: UMA VISÃO GERAL

Para Kerzner (2002) o gerenciamento de projetos é um conjunto de atividades que possui um objetivo comum, consome recursos e possui pressões sobre prazos, custos e qualidade. O gerenciamento envolve a aplicação de habilidades, ferramentas, conhecimentos e técnicas às atividades do projeto com finalidade de atingir seus requisitos (DUNCAN, 2003). O gerenciamento de projetos é utilizado pelas empresas que buscam obter melhores resultados (ARAÚJO et al., 2009).

O PMBOK (*Project Management Body of Knowledge*) é um documento de referência para auxiliar a gerência de projetos, criado pelo PMI (*Project Management Institute*). De acordo com PMBOK (2013), um projeto é definido como um conjunto temporário de esforços para a obtenção de um produto único, com prazo e recursos limitados. Existem diversos tipos de projetos que podem fazer uso do PMBOK, como:

- Desenvolvimento de um novo produto, serviço ou processo.
- Alteração da estrutura de processos de uma organização.
- Desenvolvimento ou aquisição de um sistema de informação novo ou modificado.
- Realização de pesquisa.

- Construção de um prédio, planta industrial ou projetos de infraestrutura.
- Implementação, melhoria ou aprimoramento de processos e procedimentos já existentes.

Os projetos têm importância em uma empresa porque compõe o desenvolvimento, planejamento, estruturação ou aprimoramento de produtos, serviços e processos. De acordo com o PMBOK (2013), o gerenciamento de projetos pode ser realizado por meio da aplicação e integração de 47 processos, que estão agrupados em cinco processos principais também chamados de ciclo de vida do projeto:

- Iniciação.
- Planejamento.
- Execução.
- Monitoramento e controle.
- Encerramento.

Um estudo publicado em 2007 pelo itSMF Brasil[2], contando com a participação de 200 empresas no País, revelou que 85% das empresas usam algum *framework* de governança de TI (Tecnologia da Informação). Os *frameworks* mais utilizados foram *Cobit*, ITIL (*Information Technology Infrastructure Library*), CMMI (*Capability Maturity Model Integration*) e PMI (*Project Management Institute*). Durante a pesquisa foram questionados os motivos que levaram a adoção dessas ferramentas de negócios e 40% das empresas responderam que a implementação de metodologias dos *frameworks* representa um alinhamento com as áreas de negócio.

No ano de 2013, outra pesquisa realizada durante o 16º Seminário Nacional de Gestão de Projetos pelo IETEC (Instituto de Educação Tecnológica), que reuniu 94 empresas e 193 profissionais de diversos estados, mostrou que 86% das empresas participantes utilizam os conceitos de gestão de projetos, 80% utilizam priorizar projetos e 82% possuem ou pretendem desenvolver programas formais para a capacitação em gerenciamento de projetos. A pesquisa mostrou ainda, que apesar dos conceitos serem bem difundidos, apenas 39% possuem metodologia de projetos implementada e que entre os problemas mais enfrentados estão, prazos (71%), comunicação (68%) e mudanças no escopo do projeto (56%). Os dados

[2] *IT Service Management Forum Brasil*

divulgados na pesquisa apontam as áreas de tecnologia de informação (46%) e engenharias (60%) como as que mais utilizam a gestão de projetos. O evento levantou a formação acadêmica dos participantes que pode ser observada a seguir (IETEC, 2013):

- Engenharia/ Arquitetura – 51%.
- Administração – 13%.
- Informática – 11%.
- Outros – 25%.

A padronização de processos em algumas das melhores empresas é fundamentada pelos conhecimentos em gerenciamento de projetos de seus gerentes que acumulam conhecimento de sucesso obtido em várias áreas dentro da empresa. O PMI e outros estudiosos de comportamento empresarial reuniram esses conhecimentos e compilaram em um manual de gerenciamento, conhecido por PMBOK. Alguns casos de sucesso aos quais os conhecimentos do PMBOK foram aplicados são documentados e estão disponíveis gratuitamente na página oficial do PMI na internet, tais como GSK (*Global Pharmaceutical Industry*), IATA (*The International Air Transport Association*), IBM CHINA, *Procter & Gamble* e as Agências Governamentais Americanas.

Ainda de acordo com PMBOK, o conhecimento em gerenciamento de projetos pode ser dividido em dez áreas:

- Gerenciamento da Integração.
- Gerenciamento de Escopo.
- Gerenciamento de Custos.
- Gerenciamento de Qualidade.
- Gerenciamento das Aquisições.
- Gerenciamento de Recursos Humanos.
- Gerenciamento das Comunicações.
- Gerenciamento de Risco.
- Gerenciamento de Tempo.
- Gerenciamento das Partes Interessadas.

O escopo desse trabalho está limitado a área de conhecimento que trata do Gerenciamento de Recursos Humanos, as outras áreas embora possam estar interligadas, não serão tratadas de maneira direta no decorrer desta pesquisa.

Segundo Silva (2007), confiar apenas em sua experiência para tomar decisões, raramente dá certo, então um gerente de projetos deve apoiar se possível, suas decisões em projetos similares. Então, quando possuir ferramentas que permitem uma leitura técnica das competências necessárias, tomar decisões que sejam em médias melhores que uma escolha aleatória ou unicamente baseada em experiência (SANTOS, 2014).

Para isso, o gerente de projetos possui ferramentas para gerenciar e elas estão presentes em seu portfólio. Um portfólio contém uma coleção de projetos, programas, subportfólios e operações que devem ser gerenciadas como um grupo para a obtenção de resultados estratégicos (PMBOK, 2013).

A utilização de técnicas e ferramentas de gestão de projetos traz melhoria a alocação de recursos, evitando surpresas na execução de projetos, além de agilizar a tomada de decisão e facilitar as estimativas para projetos futuros (SILVA, 2009).

2.2 GERENCIAMENTO DE RECURSOS HUMANOS

O gerenciamento de recursos humanos envolve um conjunto de processos que mobilizam, organizam, gerenciam e guiam a equipe do projeto. A equipe é composta por pessoas com papéis e responsabilidades necessárias para atender aos requisitos do projeto (PMBOK, 2013; SILVA, 2009). Os processos de gerenciamento de recursos humanos são (PMBOK, 2013):

- Desenvolver o plano de recursos humanos – identificação e documentação de papéis e responsabilidades, habilidades necessárias, relações hierárquicas, além da criação de um plano de gerenciamento de pessoal.
- Mobilizar a equipe do projeto – confirmação da disponibilidade dos recursos humanos e alocação da equipe necessária para terminar as atividades do projeto.
- Desenvolver a equipe do projeto – melhoria de competências, da interação da equipe e do ambiente geral da equipe para aprimorar o desempenho do projeto.

- Gerenciar a equipe do projeto – acompanhamento do desempenho dos membros da equipe, fornecer *feedback*, resolver problemas e gerenciar mudanças para otimizar o desempenho do projeto.

Os processos interagem entre si e com outros processos em outras áreas de conhecimento e podem resultar em um planejamento adicional. Por exemplo:

- Riscos causados por inexperiência.
- Necessidade de mais recursos humanos após a criação da estrutura analítica do projeto.
- Quando as durações das atividades e estimativas são calculadas antes da formação da equipe ou quando os níveis de competência são desconhecidos.

O gerenciamento de recursos humanos consiste em uma tarefa onerosa que envolve a identificação de uma série de combinações de alocação de recursos e fatores, muitas vezes conflitantes (LOPES et al., 2009).

2.2.1 GERENCIAMENTO DE RECURSOS HUMANOS EM PROJETOS DE SOFTWARE

Segundo Pressman (2011), o gerenciamento efetivo de desenvolvimento de software é baseado em 4 Ps: Pessoas, Produto, Processo e Projeto. Sendo essa ordem não arbitrária. Ainda de acordo com o mesmo, o gerente que esquecer que a qualidade de software depende diretamente do esforço humano está fadado ao fracasso neste tipo de projeto.

Ainda de acordo com Pressman (2011), os gerentes de projetos são os profissionais que planejam, monitoram e controlam o trabalho de uma equipe e, para isso, muitos utilizam ferramentas automatizadas que auxiliam na tomada de decisão. Entretanto, um dos maiores desafios dos gerentes de projetos é manter a equipe de desenvolvimento motivada e em alto nível.

Essa dificuldade atrelada aos recursos humanos é muito antiga, e devido a sua importância, o *Software Engeneering Institute* (SEI) desenvolveu um modelo de maturidade e capacidade dos recursos humanos People-CMM *(People Capability and Maturity Model)* que define algumas práticas essenciais a serem

desempenhadas durante os processos de software para o pessoal permanecer motivado e capacitado: formação da equipe, comunicação, ambiente de trabalho, gerenciamento de desempenho, treinamento, compensação, análise de competências e de desenvolvimento, entre outras.

Equipes de software geralmente são grandes e envolvem muitas pessoas e tarefas desenvolvidas em períodos de tempo diferentes. Por isso, as competências tem que ser bem distribuídas. Segundo Pfleeger (2004), o primeiro desafio do gerente de projetos está no planejamento do projeto, nele são definidos os cronogramas e a estimativa de quanto tempo será gasto para desenvolver o sistema e qual o seu custo. Nesta fase, os períodos necessários para realizar cada tarefa/atividade são estimados de acordo com o pessoal disponível. A representação deste cronograma é geralmente feita com auxílio de um diagrama de Gantt.

Na fase de elaboração do cronograma é preciso saber quantas pessoas estarão participando do projeto, que tarefas irão realizar e que habilidades e experiências essas pessoas tem para executar seu trabalho de maneira mais eficiente possível. Existem vários modelos de processo de software que definem como tarefas devem ser executas, porém, independente do modelo de processo de desenvolvimento de software, sempre há atividades comuns e necessárias. As principais atividades de um projeto de software compreendem (PFLEEGER, 2004):

- Análise de requisitos;
- Projeto do sistema;
- Projeto do cronograma;
- Implementação do programa;
- Teste;
- Treinamento;
- Manutenção;
- Garantia de qualidade.

No entanto, até mesmo para as atividades que não são executadas em paralelo são alocados profissionais diferentes, designados por suas habilidades e experiências. Alocar grupos de pessoas diferentes para cada atividade traz a vantagem de poder identificar, antecipadamente, problemas no processo de desenvolvimento. Por exemplo, uma equipe diferente de testes pode identificar erros cometidos por programadores, que poderiam ocorrer caso o mesmo grupo

realizasse ambas as atividades. De acordo com Pfleeger (2004), duas pessoas com o mesmo cargo podem diferir em pelo menos um dos seguintes aspectos:

- Capacidade de desempenhar seu papel;
- Interesse no trabalho;
- Experiência com aplicações semelhantes;
- Experiência com ferramentas ou linguagens semelhantes;
- Experiência com técnicas semelhantes;
- Experiência com ambiente de desenvolvimento semelhante;
- Capacidade para se comunicar com outras pessoas;
- Capacidade para compartilhar responsabilidades com outras pessoas;
- Habilidades de gerenciamento.

Cada um desses aspectos pode afetar a produtividade de um recurso. Esses aspectos sutis podem ter um grande impacto, não somente na estimativa do cronograma, mas também para o sucesso do projeto. Segundo Pfleeger (2004), para entender melhor o desempenho de cada recurso, deve-se conhecer suas capacidades e habilidades para realizar o trabalho. Por exemplo, um profissional que é alocado para uma atividade de codificação pode ter uma habilidade maior em testes e ter um desempenho melhor nesta atividade.

A habilidade de um profissional, algumas vezes, está relacionada à sua confiança em desenvolver a atividade e isso compreende conhecimento e vontade de trabalhar. Entretanto, se for realizado o contrário, isto é, alocação de um profissional para uma atividade em que o mesmo não tem conhecimento, experiência ou habilidade irá desmotivá-lo e até prejudicar o andamento do projeto (PFLEEGER, 2004).

De acordo com Pressman (2011), apesar das principais produtoras de software reconhecer a importância da seleção de equipes e afirmarem que pessoal é o principal fator a ser considerado, ainda não conseguem converter em ações efetivas. Para ser produtiva, uma equipe deve estar organizada para maximizar cada capacidade e habilidade no projeto. Essa organização é uma das tarefas do líder e/ou gerente do projeto.

2.3 ALOCAÇÃO DE RECURSOS HUMANOS

A alocação de recursos humanos envolve a identificação de requisitos do projeto e de habilidades dos recursos humanos necessárias para o projeto (PMBOK, 2013). A qualidade do produto a ser desenvolvido está relacionada às habilidades da equipe que compõe o projeto (SANTOS, 2014).

De acordo com Dyngsoyer (2001), citado por Santos (2014), os processos de gerenciamento de recursos humanos consistem em três etapas:

- Busca de competências necessárias: Definição de competências, obtenção dos recursos com as competências necessárias e seleção dos profissionais.
- Alocação de recursos humanos: Alocação, deslocamento e retirada de profissionais.
- Desenvolvimento de competências: Avaliação de competências adquiridas e capacitação dos profissionais.

A norma NBR ISO 10006 (2006) define as diretrizes de gerência da qualidade de software e da ênfase à gerência de recursos humanos como:

- Definição da estrutura organizacional do projeto: Definição organizacional baseada no entendimento das necessidades do projeto, incluindo a identificação das funções e definição de responsabilidades.
- Alocação da equipe: Seleção e nomeação de pessoal com a competência adequada para atender as necessidades do projeto.
- Desenvolvimento da equipe: Identificação e desenvolvimento de habilidades individuais e coletivas para aperfeiçoar o desempenho do projeto.
- Para Lopes et. al. (2009), alguns projetos, como os de software, tem um grau de dinamismo alto, caracterizado por mudanças constantes em tecnologias e domínio da aplicação. Essa mudança não sistemática requer tomada de decisões que tipicamente são feitas baseadas na experiência pessoal do gerente e não com o uso de modelos sistemáticos (SILVA, 2009).

Santos (2014) propôs um método de alocação de equipes com base em habilidades, em seu método as habilidades são caracterizadas como técnicas e não-

técnicas. Para a definição de habilidades técnicas são coletas todas as informações sobre as tecnologias necessárias para desenvolver módulos de software específicos, por meio dessas informações as equipes são avaliadas sobre suas aptidões e também sobre o conhecimento do domínio. Para coletar as informações sobre as habilidades técnicas dos recursos são elaborados questionários que tem objetivo de identificar os conhecimentos de cada equipe.

Para a classificação das habilidades não-técnicas Santos (2014) define 4 tipos: geográficas, temporais, culturais e de reputação. As habilidades não-técnicas em seu método tentam minimizar problemas de comunicação entre as equipes que implementam módulos em lugares distintos, com tempo estimado e outras atividades paralelas. Para as avaliações são realizadas políticas de seleção que priorizam determinadas habilidades técnicas e não-técnicas, as políticas de seleção são definidas pelo gerente de projetos, baseadas em seu conhecimento e experiência.

O problema da alocação de recursos humanos é muito estudado em pesquisa operacional, alguns autores como Otero (2009) atribuem a alocação o principal entrave na entrega de software planejado, segundo o mesmo, os atrasos ocorrem porque gasta-se muito tempo para treinar pessoal qualificado. Otero (2009) propõe a alocação otimizada para o conjunto de habilidades dos recursos utilizando a métrica SLOC.

Burdet e Li (1995) propõem uma abordagem quantitativa para a construção de equipes, para isso utiliza matemática por meio de recozimento simulado (*Simulated Anneling*) entre três fatores: custo de salário (favorável), habilidades (desejável) e preferências do gerente (desejável). Os algoritmos para a formação de equipes são considerados da classe NP-Difícil, significa que sua execução é exponencial (Otero, 2009).

Silva (2009) propõe a alocação através de programação linear e dinâmica, com foco em minimização de tempo e de custos, respectivamente. Utilizando a programação linear, modelou a alocação como um problema de designação, propondo a utilização de métricas em especial a métrica de software *Lines of Code* (*LOC*).

2.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capitulo foram apresentados fatores importantes na avaliação da alocação de um recurso em uma atividade em projetos de software. As particularidades dos recursos são fatores fundamentais que acabam determinando a qualidade da alocação e são identificadas por meio da caracterização de habilidades que podem ser divididas em técnicas e não-técnicas.

No método proposto por Santos (2014) as diferenças entre as habilidades das equipes são fatores determinantes para a alocação e a política de seleção dessas habilidades ainda é subjetiva. Por isto, este trabalho utiliza as políticas de seleção com a efetiva participação de um ou mais gerentes de projetos, tentando diminuir a subjetividade, adicionando, por exemplo, a avaliação de código-fonte com ênfase em qualidade de software.

3 LÓGICA *FUZZY*

Neste capítulo são descritos conceitos fundamentais relacionados à lógica fuzzy. A seção 3.1 apresenta os conceitos sobre lógica *fuzzy*, sua história e aplicabilidade. Na seção 3.2 são mostradas as operações e a representação de conjuntos *fuzzy*. A seção 3.3 mostra como trabalhar e quais as características que compõe variáveis linguísticas. A seção 3.4 apresenta o processo de fuzzificação e defuzzificação e os conceitos relacionados à máquina de inferência de Mamdani. No fim do capítulo, a seção 3.5 mostra o uso da lógica *fuzzy* para determinar a qualidade de um código-fonte.

3.1 CONCEITOS

Lukasiewicz (1878-1956) foi o primeiro a introduzir em suas pesquisas conjuntos com grau de pertinências combinados a conceitos da lógica clássica booleana. Porém, a teoria dos conjuntos difusos ou lógica *fuzzy*, só foi conhecida depois de estudada e divulgada pelo professor da Universidade da Califórnia Lofti A. Zadeh (1965), sendo ele considerado o pai da lógica *fuzzy* (RIGNEL et al, 2011).

Na lógica clássica, uma proposição pode assumir apenas dois valores, verdadeiro (1) ou falso (0). Desta forma, dado um conjunto Ae um elemento x do conjunto universo U, é possível dizer claramente se $x \in A$ ou se $x \notin A$. Indiscutivelmente, isso resolve muitos problemas em áreas exatas, tal como engenharia, que não se tem dificuldade em classificar elementos e relaciona-los a seus respectivos conjuntos (ORTEGA, 2001).

Zadeh observou a impossibilidade de modelar sistemas com fronteiras mal definidas, utilizando abordagens matemáticas rígidas como, por exemplo, a teoria da probabilidade (ORTEGA, 2001). A lógica *fuzzy* se difere dessas abordagens rígidas, por dar suporte a “conjuntos nebulosos”, possibilitando o uso de raciocínio aproximado ao invés de exato (RIGNEL et al., 2011).

Na lógica *fuzzy*, o valor verdade de uma proposição pode assumir uma infinidade de proposições, chamadas de expressões linguísticas, interpretadas como subconjuntos *fuzzy*. Os predicados por sua vez, são termos subjetivos e imprecisos, tais como: bom, ruim, alto, baixo. Esses dois conceitos tornam mais fácil a incorporação de conhecimento ao sistema, permitindo traduzir o conhecimento de um especialista sem uma abordagem puramente matemática (PACHECO, 2015).

Para Ortega (2011), o foco de Zadeh (1965) é a flexibilização da pertinência de elementos a seus conjuntos, o que acaba introduzindo a ideia de “grau de pertinência”. Seguindo esse conceito, um elemento pode pertencer parcialmente a um conjunto, ou seja, pode ser compatível em certo grau ao conjunto observado.

Em resumo, a lógica *fuzzy* é uma ferramenta que permite tratar informações imprecisas, descritas em linguagem natural, e transformar em formatos numéricos para serem trabalhados por conjuntos *fuzzy*.

3.2 OPERAÇÕES E PROPRIEDADES SOBRE CONJUNTOS *FUZZY*

Os conjuntos *fuzzy* obedecem algumas propriedades que permitem a sua operação. Segundo Pacheco (2010) essas propriedades são extremamente importantes e as mais básicas, originalmente escritas por Zadeh, são definidas a seguir (1):

$$A = \{(x, \mu_A(x)) \ / \ x \in U \}, \mu_A(x) \in [0,1] \quad (1)$$

Propriedades dos conjuntos:

Igualdade (2):

$$\forall x \in U \ / \ \mu_A(x) = \mu_B(x) \quad (2)$$

Se o grau de pertinência de x em A é o mesmo em relação a B, x pertence a ambos os conjuntos.

Inclusão ou Subconjunto (3):

Se $\mu_A(x) \leq \mu_B(x)$, então A é um subconjunto *fuzzy* de B. (3)

União (4):

$$A \cup B = \max [\mu_A(x), \mu_B(x)] \quad (4)$$

A operação de união pode ser expressa em lógica *fuzzy* como o maior valor entre duas funções de pertinência em x, equivale a operação OR da álgebra booleana,

também representadas de forma mais geral por meio das normas S (Rezende, 2005). A representação desta operação pode ser observada na Figura 2(a).

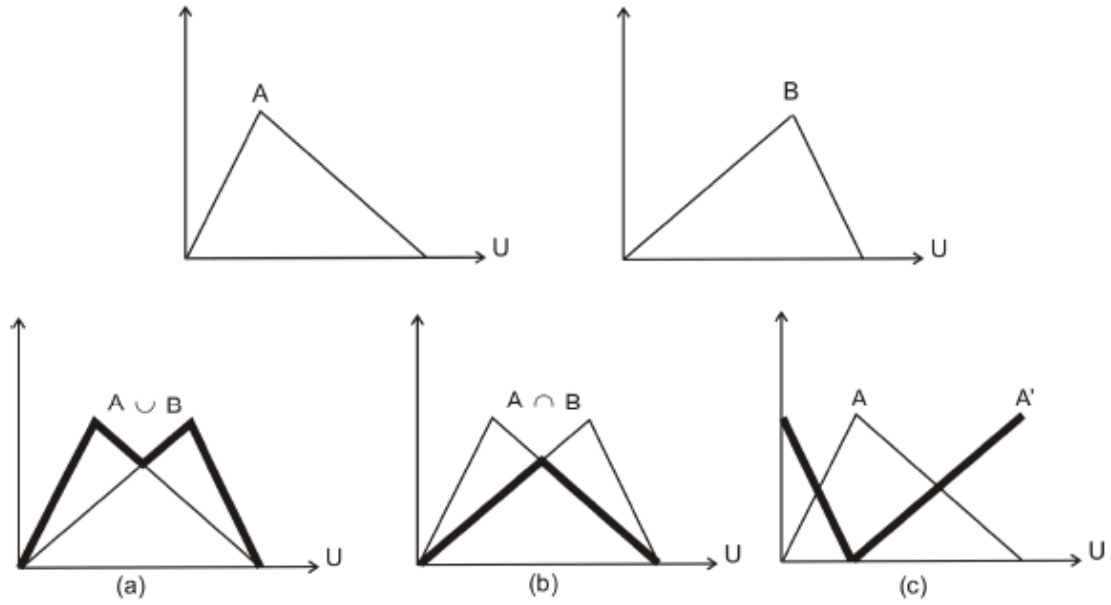
Interseção ou Conjunção (5):

$$A \cap B = \min[\mu_A(x), \mu_B(x)] \tag{5}$$

A operação de intersecção pode ser expressa em lógica *fuzzy* como o menor valor entre duas funções de pertinência em x, equivale a operação AND da álgebra booleana, também representadas de forma mais geral por meio das normas T (Rezende, 2005). A representação desta operação pode ser observada na **Erro! Fonte de referência não encontrada.** (b).

A representação gráfica das operações de conjuntos em números *fuzzy* pode ser observada na Figura 2.

Figura 2 – Operações de conjuntos *fuzzy* e sua representação.



Fonte: Castanho et al. (2009)

Na lógica *fuzzy* também é definido o $\alpha - cut$ (ou corte de nível α) de um conjunto. Rentería (2006) interpreta como o conjunto *fuzzy* que apresenta uma restrição ou limite imposto ao domínio do conjunto baseado no valor α, assim o conjunto resultante contém todos os elementos do domínio que possuem um grau de pertinência $\mu(x)$ superior ou igual ao valor de α. Para o conjunto *A,* por exemplo, são os subconjuntos dos números reais, definidos por (6):

$$[A]^{\alpha} = x \in U;\ \mu_A(x) \geq \alpha \tag{6}$$

Um conjunto *fuzzy* A é chamado de número *fuzzy* quando o conjunto universo, onde A está definido, é o conjunto dos números reais e satisfaz as seguintes condições:

- Todos os $\alpha - cuts$ de A, são não vazios com $0 \leq \alpha - cuts \leq 1$.
- Todos os $\alpha - cuts$ *de* A, são intervalos fechados em $\mathbb{R}$.
- O suporte de A, é um conjunto limitado.

De acordo com Pacheco (2010), existem vários tipos de números *fuzzy*, sendo a sua escolha apenas uma opção de projeto. Os números *fuzzy* mais comuns são triangulares e trapezoidais, sendo que cada modelo tem sua forma particular de calcular a função de pertinência de x ao número. Um conjunto *fuzzy* chamado *Singleton* se seu suporte é um único ponto em U com grau de pertinência igual a 1, $\mu(x) = 1$ (RENTERÍA, 2006).

Por exemplo, um número *fuzzy* é dito Triangular se sua função de pertinência é como visto em (7).

$$\mu_A(x) = \begin{cases} 0, se\ x, = a, \\ \dfrac{x-a}{u-a}, se\ a \leq x \leq u, \\ \dfrac{x-b}{u-b}, se\ u \leq x \leq b, \\ 0, se\ x \geq b. \end{cases} \tag{7}$$

Um número *fuzzy* Trapezoidal é assim chamado se sua função de pertinência tem a forma de trapézio, é dada por (8):

$$\mu_A(x) = \begin{cases} \dfrac{x-a}{b-a}, se\ a \leq x \leq b, \\ 1, se\ b \leq x \leq c, \\ \dfrac{d-x}{d-c}, se\ c \leq x \leq d, \\ 0, caso\ contrário. \end{cases} \tag{8}$$

Outro número muito utilizado é em forma de Sino, sendo aplicado para a representação quando a função de pertinência for suave e simétrica em relação a um número real. A seguinte função mostra estas propriedades (9).

$$\mu_A(x) = \begin{cases} \exp\{\dfrac{-(x-u)^2}{a}\}, se\ u - \delta \leq x \leq u + \delta, \\ 0, caso\ contrário. \end{cases} \tag{9}$$

Neste trabalho será utilizada uma ferramenta (detalhada em outro capítulo) que realiza as operações e mostra graficamente a representação dos conjuntos

fuzzy, portanto a teoria acima é apenas para complementar a teoria que envolve a lógica *fuzzy*.

3.3 VARIÁVEIS LINGUÍSTICAS

Segundo Ortega (2001), a lógica *fuzzy* pode ser considerada uma das ferramentas matemáticas mais poderosas para lidar com imprecisões, incertezas e verdades parciais. Ademais, antes do surgimento da lógica *fuzzy*, as informações vagas não tinham como ser processadas.

Portanto, levando em consideração o princípio de uma lógica capaz de utilizar o conceito de incerteza, torna-se necessária a obtenção de entidades que representem de alguma maneira valores imprecisos. Rezende (2005) define uma variável linguística como uma entidade capaz de representar de modo impreciso um conceito ou uma variável de um determinado problema.

Variáveis linguísticas se diferenciam de variáveis de domínio numérico (*crisp - são* valores em formato numérico comumente utilizado em operações matemáticas) por admitirem apenas valores que já foram definidos previamente. Possuem como função apresentar uma aproximação de valores que são relacionados a domínios complexos. Além disso, essas variáveis podem sofrer ações de modificadores. Os modificadores, de forma análoga aos advérbios no âmbito linguístico, podem intensificar ou amenizar os valores *fuzzy* (MARRO et al., 2000).

Em complemento, Souza (2010) caracteriza que uma determinada variável linguística em um universo U é aquela cujos valores assumidos por ela são subconjuntos *fuzzy* de U. Os valores de uma variável podem ser tanto termos atômicos quanto sentenças em uma linguagem específica, sendo este o motivo da variável ser considerada linguística.

Uma variável linguística, expressa um valor em termos de linguagem natural e pode ser definida como formalmente como uma quíntupla (LEE, 2005) (10):

$$Variável\ linguística = (x, T(x), U, G, M) \tag{10}$$

Onde:

x: nome da variável.

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

U: universo de discurso.

G: regra sintática para gerar nomes.

M: regra semântica para associar significados.

Outros autores definem variáveis linguísticas como quádruplas ou sêxtuplas, o que explica a definição como um critério de projeto, em que sintaticamente e semanticamente é preciso obter resultados diferentes, embora o princípio seja o mesmo.

De acordo com Rezende (2005), as propriedades sintáticas definem o formato em que serão armazenadas as informações linguísticas *fuzzy,* proporcionando a obtenção de uma base de conhecimento contendo sentenças estruturadas. A obtenção desta base acaba sistematizando os processos de armazenamento, busca e processamento de dados existentes.

De outra forma, as propriedades semânticas especificam de que modo é extraído e processado o conhecimento, armazenado na forma de declarações condicionais *fuzzy*, ou regras de produção *fuzzy* e contido na estrutura definida pelas propriedades sintáticas.

3.3.1 Regras de Produção *Fuzzy*

A maneira mais comum de armazenar informações em uma Base de Conhecimento *fuzzy* é a representação por meio de regras de produção *fuzzy* (REZENDE, 2005). Desta forma, a representação de conhecimento humano pode ser formada por duas partes principais (11):

$$\boldsymbol{IF} < antecedente >, \boldsymbol{THEN} < consequente >. \quad (11)$$

O antecedente é composto por um conjunto de condições que, quando satisfeitas, determinam o processamento do consequente da regra por um mecanismo de inferência *fuzzy.* Este processo é chamado de disparo da regra.

Por sua vez, o consequente é formado por um conjunto de ações ou diagnósticos que são gerados pelo disparo da regra. Os consequentes são processados em conjunto para gerar uma resposta determinística para cada variável de saída do sistema (REZENDE, 2005).

A construção dos antecedentes (região *fuzzy* no espaço de entrada) muitas vezes resulta em um trabalho de classificação, enquanto a elaboração dos

consequentes (região *fuzzy* no espaço de saída) exige um conhecimento, ainda que empírico, sobre a dinâmica do sistema. Portanto, espera-se que a elaboração dos consequentes de uma regra, seja mais complexa do que a dos antecedentes (ORTEGA, 2001). O Quadro 1 exibe a representação canônica de uma base de regras *fuzzy.*

Quadro 1 – Representação Canônica de Base de Regras *Fuzzy*

$Regra\ 1 - IF\ condição\ C1, THEN\ restrição\ R1.$
$Regra\ 2 - IF\ condição\ C2, THEN\ restrição\ R2.$
...
$Regra\ n - IF\ condição\ Cn, THEN\ restrição\ Rn.$

Fonte: Adaptado de Sivanandam et al. (2007)

Depois de construído o conjunto de regras é necessário uma máquina de inferência, para extrair delas a resposta final. Existem vários métodos de inferência e a escolha deles depende do sistema que está sendo analisado. No entanto, o método de inferência mais comumente utilizado é o Método de Mamdani que será detalhado na subseção 3.4.1 (ORTEGA, 2001).

3.4 SISTEMA DE INFERÊNCIA *FUZZY*

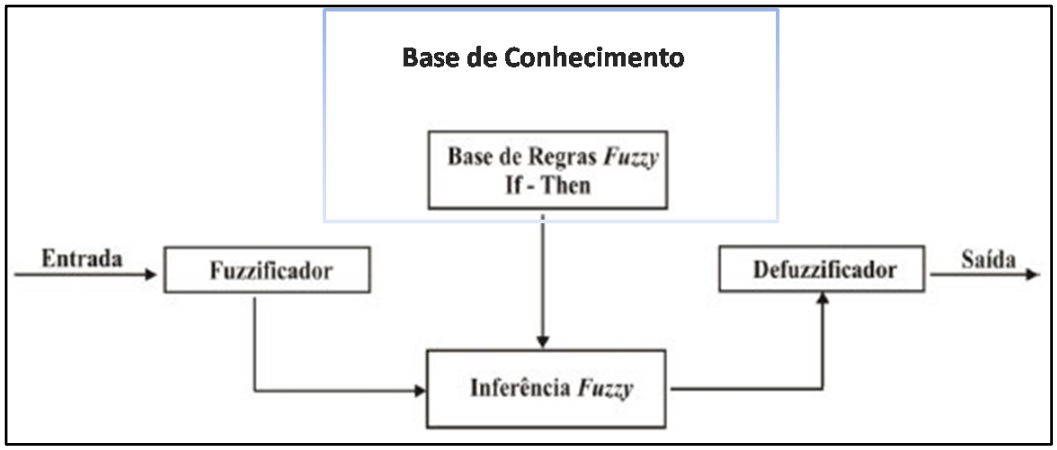
A principal tarefa de um sistema de inferência *fuzzy* é a tomada de decisão. A semântica define o mecanismo de inferência e como serão processados os antecedentes, quais são os indicadores de disparo das regras e quais os operadores utilizados sobre os conjuntos *fuzzy* existentes para executar o processamento de conhecimento (REZENDE, 2005). De acordo com Rezende (2005), existem alguns modelos de inferência *fuzzy* que são escolhidos de acordo com as propriedades sintáticas definidas, ou seja, o modelo de processamento definido para o sistema de conhecimento depende da forma de armazenamento de informações escolhida.

Características de um sistema de inferência *fuzzy* (FIS) incluem: i) a saída da FIS será sempre um conjunto *fuzzy* independentemente de sua entrada ser *fuzzy* ou c*risp*; ii) é preciso ter uma saída *fuzzy* quando utilizado um controlador e também

uma unidade de defuzzificação na saída que seja capaz de converter variáveis *fuzzy* em variáveis *crisp*. Na Figura 3 são mostradas as unidades de um FIS:

- *Base de Regras* – Está presente na Base de Conhecimento, contêm as regras *fuzzy IF-THEN* fornecidas por especialistas ou extraídas de dados numéricos.
- *Database* – Também está presente na Base de Conhecimento e define as funções de pertinência usadas pelos conjuntos *fuzzy* de acordo com as suas regras.
- *Inferência Fuzzy ou Unidade de Decisão Fuzzy*– Determina como as regras são ativadas e combinadas, ou seja, é a unidade de execução das regras.
- *Fuzzificador ou Unidade de Fuzzificação*– Converte valores *crisp* em valores *fuzzy.*
- *Defuzzificador ou Unidade de Defuzzificação* – Converte valores *fuzzy* em valores *crisp.*

Figura 3 – Diagrama de Blocos de um Sistema de Inferência *Fuzzy* (*FIS*)



Fonte: Adaptado de Cherri (2011)

De acordo com este modelo de FIS, a unidade de fuzzificação deve suportar um bom número de métodos de fuzzificação que serão utilizados para a conversão de entradas *crisp* em entradas *fuzzy*. A base de conhecimento deve conter um conjunto de regras e o banco de dados formado após a conversão de entrada *crisp* em entrada *fuzzy.* Por fim, a unidade de defuzzificação converte os valores *fuzzy* em valores *crisp* na saída.

Os dois métodos mais utilizados de FIS são *Mamdani Fuzzy Inference System e Takagi-Sugeno Fuzzy Model (TS Model),* sendo a principal diferença entre eles a maneira como o conjunto de regras do consequente será formado. Este documento explica somente o modelo Mamdani, já que esse será utilizado no desenvolvimento do método proposto.

3.4.1 Modelo de Mamdani

Em 1974, Ebhasim Mamdani propôs um método de inferência que foi por muitos anos o padrão para a utilização dos conceitos da lógica *fuzzy* em processamento de conhecimento (REZENDE, 2005). Em 1975 aplicou seu método esperando controlar uma combinação de máquinas a vapor e caldeiras sintetizando um conjunto de regras difusas obtidas de pessoas que trabalhavam no sistema.

Para calcular a saída com este modelo são necessários seis passos (CHERRI, 2016):

1) Determinar um conjunto de regras fuzzy.
2) Fuzzificar as entradas usando as funções de pertinência apropriadas.
3) Combinar as entradas fuzzificadas de acordo com as regras *fuzzy* para estabelecer a força da regra.
4) Procurar o consequente da regra pela combinação da força da regra e a saída da função de pertinência.
5) Combinar os consequentes para obter uma distribuição de saída adequada.
6) Defuzzificar as distribuições de saídas (Fundamental apenas se uma saída *crisp* for necessária).

Uma regra típica do modelo inclui relações tanto em seus antecedentes como em seus consequentes.

A regra semântica tradicionalmente utilizada para o processamento de inferências com o modelo de Mamdani é chamada de inferência Máx-Min que faz uso das operações de união e intersecção entre conjuntos da mesma forma que Zadeh (REZENDE, 2005).

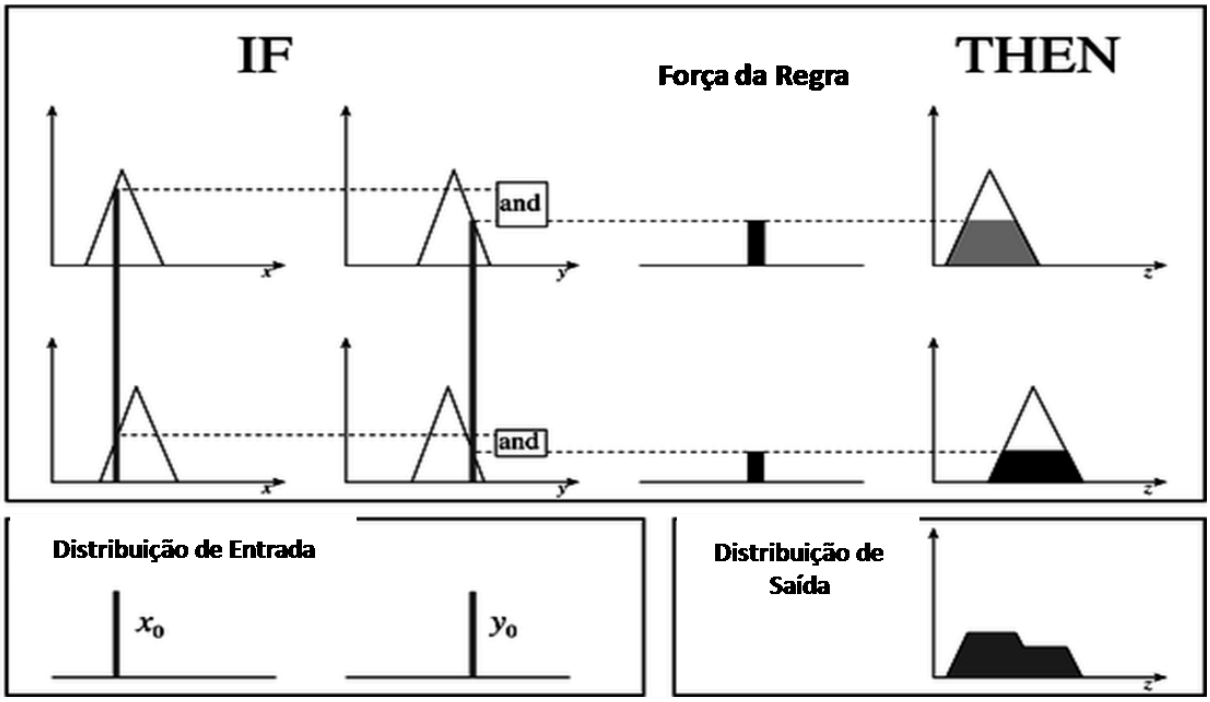
Durante o processo de conversão $crisp \rightarrow fuzzy$ que transforma informações quantitativas em qualitativas e descritas por Rezende (2005), os

antecedentes de cada regra são processados por meio da operação min (Interseção) entre os graus de pertinência das entradas. Este processo transforma informação qualitativa em outra informação qualitativa que pode ser usada para tomada de decisão.

O processo gera um grau de pertinência para cada uma das regras, isto é, são gerados os graus de ativação, também chamado de *DoF* (acrônimo do inglês para*: Degree of Fire*). Todo este processo é comumente chamado de fuzzificação.

. A Figura 4 mostra o processo para duas regras e três entradas, x_i e y_i antecedentes e z_i consequentes. A força da regra é equivalente ao grau de disparo da regra obtido através da operação de min.

Figura 4 – Exemplo de duas entradas *fuzzy* e duas regras *Mamdani FIS*



Fonte: Adaptado de E. Nartey (2016)

Por fim, os coeficientes de disparo vão limitar os valores máximos dos conjuntos *fuzzy* de saída gerados por todas as regras, aplicando uma operação global de união que vai compor um conjunto *fuzzy* para cada variável de saída.

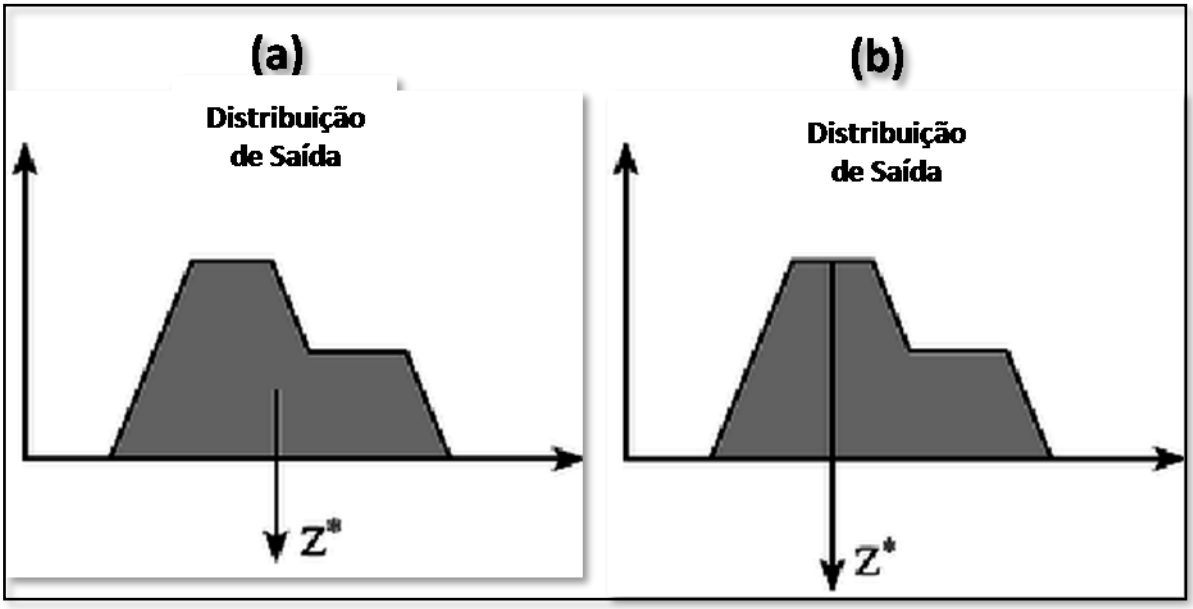
Este processo transforma informação qualitativa em outra informação qualitativa que pode ser usada para tomada de decisão. Porém em sistemas com

atuadores convencionais é preciso gerar um valor *crisp* para a saída, portanto uma etapa a mais é necessária conhecida como defuzzificação (REZENDE, 2005).

A conversão $fuzzy \rightarrow crisp$ transforma informações qualitativas em quantitativas, sendo um processo de especificação. Este processo é comumente chamado de defuzzificação. Há dois métodos que são mais utilizados o Centro de Massa e Média dos Máximos (REZENDE, 2005).

O método do Centro de Massa usa a distribuição de saída e busca seu centro associando ao seu valor *crisp* correspondente, conforme mostra a **Erro! Fonte de referência não encontrada.**.

Figura 5 – Representação de defuzzificação com centro de massa (a) e média dos máximos (b)



Fonte: Adaptado de Nartey (2016)

3.5 LÓGICA FUZZY APLICADA PARA DETERMINAR QUALIDADE EM CÓDIGO-FONTE

Conforme relatado no capítulo anterior, Santos (2004) usa da subjetividade para alocar recursos a um projeto de software. Uma forma de diminuir esta subjetividade é o uso de métodos que possam avaliar a habilidade de um individuo analisando seu código-fonte.

Oliveira (2015) propôs a avaliação da qualidade de software em código-fonte orientado a objetos. Com o intuito de manipular problemas de incerteza e imprecisão foi utilizada a lógica *fuzzy*. Utiliza uma relação entre requisitos de qualidade e métricas de software, em que cada requisito de qualidade tem uma ou mais métricas associadas, como pode ser visto no Quadro 2. Note que para se avaliar manutenibilidade em um código-fonte utiliza-se as seguintes métricas DIT (Profundidade da Árvore de Herança), McCabe (Complexidade de McCabe), LOC (Linhas de Código) e NPM (Número de Métodos Públicos).

Quadro 2 – 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: Oliveira (2015)

Para a avaliação do código-fonte é fornecido um conjunto de classes Java compiladas (.class). Após a entrada dessas classes, o usuário deve selecionar um conjunto de qualidades que deseja avaliar e as métricas associadas a cada requisito sãoautomaticamente apresentadas. Após a seleção dos requisitos de qualidade, o usuário deve criar um conjunto de regras de produção que serão utilizadas no método defuzzificação.

A avaliação de código-fonte é realizada com uma ferramenta auxiliar *CKJM Extended* que trabalha com métricas CK (CHIDAMBER; KEMERER, 1994) e MOOD (HARRISON; COUNSELL; NITHI, 1998), (ABREU; CARAPUÇA, 1994) sendo essas duas as mais populares métricas de software orientado a objetos (OLIVEIRA, 2015). Após a avaliação do código-fonte pelas ferramentas, são atribuídos valores *crisp* a cada uma das métricas avaliadas.

Na etapa final do processo utiliza o *jFuzzyLogic* (CINGOLANI; ALCALA-FDEZ, 2012) para a Fuzzificação dos conjuntos fuzzy de métricas de acordo as regras de produção criadas. Por fim, um resultado é apresentado com um valor *fuzzy.*

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo relatou os conceitos sobre a lógica *fuzzy*, definições de variáveis linguísticas e o método Mamdani usado inferência de lógica *fuzzy*. O trabalho realizado por Oliveira (2015), apresentado neste capítulo, é base para o método proposto para avaliação de habilidades de recursos humanos envolvidos em tarefas de programação e sua capacidade de atingir qualidade.

Oliveira (2015) avaliou a qualidade de código-fonte utilizando a lógica *fuzzy* devido a subjetividade em definir o que é representa um software com qualidade. Para isso utilizou das políticas de qualidade, tal como Santos (2014) fez com habilidades.

4 AloDIn: MÉTODO DE ALOCAÇÃO DE RECURSOS DIFUSO-INDUTIVO

Este capítulo apresenta a o funcionamento do método de alocação de recursos humanos baseado em requisitos de qualidade, métricas e habilidades individuais. A seção 4.1 detalha proposta do método. A seção 4.2 mostra a avaliação das habilidades dos recursos com uso de lógica *fuzzy*. A seção 4.3 mostra a avaliação de projetos através de métricas de software com uso de lógica *fuzzy.* A seção 4.4 mostra a alocação do recurso de modo prático. A seção 4.5 mostra as recomendações de encerramento do projeto e do método de alocação. A seção 4.6 mostra uma simulação de funcionamento do método e por fim a seção 4.7 apresenta as considerações finais do capítulo.

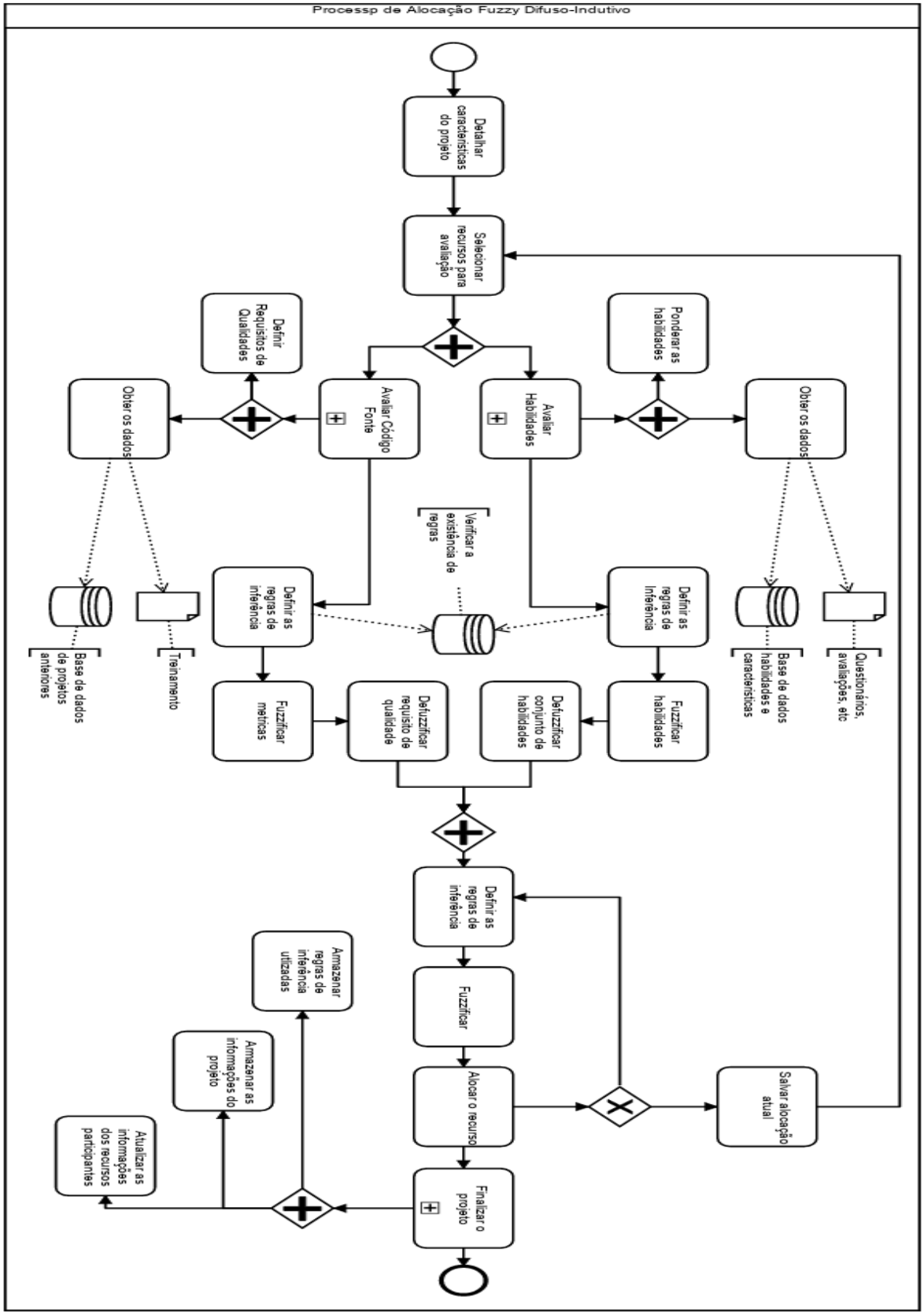
4.1 FUNCIONAMENTO DO MÉTODO ALODIN

O método proposto, denominado AloDIn, utiliza a lógica fuzzy para alocação de recursos humanos a um projeto de software. Requisitos de entrada para o método são:conjunto de habilidades que se deseja do recurso, e módulos ou projeto já desenvolvidos pelo recurso. A Figura 6 mostra o processo de funcionamento do método usando um diagrama BPMN (*Business Process Model and Notation*).

O método é divido em 4 processos: Avaliar as habilidades; Avaliar o código-fonte; Alocar o recurso e Finalizar o projeto. As avaliações são realizadas usando lógica *fuzzy*. Para melhores detalhes da notação do modelo BPMN[3] consulte a Wikipédia.

[3] https;//pt.wikipedia.org/wiki/Business_Process_Model_and_Notation

Figura 6 – BPMN Método de Alocação Difuso-Indutivo



Fonte: Autoria própria

Para a simulação do método foi utilizada a GNU Octave uma linguagem de programação científica capaz de gerar gráficos 2D e 3D. Um de seus *plugins* o *Octave Fuzzy Logic Toolkit* permite utilizar o modelo de inferência de Mamdani. Por meio dessa ferramenta foi possível simular todos os dados de alocação, com operações de matrizes e definição de variáveis linguísticas. Por exemplo, é possível definir as variáveis linguísticas de um conjunto *fuzzy* de forma declarativa tal como é feito em uma linguagem de programação orientada a objetos.

A *Fuzzy Logic Toolkit* também disponibiliza os métodos de defuzzificação e impressão 2D para os conjuntos de pertinência.

O processo de avaliar as habilidades será detalhado na seção 4.2, assim como o processo de avaliar qualidade através de código-fonte na seção 4.3. Na seção 4.4 é apresentada uma simulação do método para melhor entendimento. Por fim, a seção 4.5 apresenta algumas recomendações para o uso do método.

4.2 AVALIAÇÃO DAS HABILIDADES DOS RECURSOS COM *FUZZY*

O método propõe a obtenção de um conjunto de habilidades por meio de dados oriundos de um questionário que contemple o conhecimento de um indivíduo usando uma escala, conforme apresenta o Quadro 3.

Quadro 3 – Exemplo de um modelo de questionário para o método

Qual é o seu nível de conhecimento e aplicação para as áreas/tecnologias citadas nesse formulário?	
Habilidade X	9
Habilidade Y	6
Habilidade Z	9
Habilidade W	10
	Escala Conhecimento **Excelente** **10** **Muito Bom** **9** **Bom** **8** **Médio** **7** **Regular** **6**

Fonte: Autoria própria

O método não especifica um modelo padrão de questionário, sendo assim a definição do tipo de questionário fica a cargo do gerente de projetos. Espera-se somente que o questionário tenha uma estrutura tal como o exemplo mostrado no Quadro 3.

O gerente de projetos é o profissional que após a definição dos requisitos do projeto, escolhe quais habilidades serão primordiais ou não para o sucesso do projeto. Logo, é o responsável por selecionar quais habilidades devem ser contempladas no questionário.

O questionário é uma boa ferramenta, mas sofre com a subjetividade do recurso na sua autoavaliação. Sendo assim, a melhor forma de obtenção do conjunto de habilidades para o método proposto é aquela que foi realizada através da avaliação de experiências ou testes. Por exemplo, para obter valores para as quatro habilidades do Quadro 3 poderiam ser realizadas provas teóricas.

Na utilização do método a experiência do gerente acerca da administração, organização e produção de software pode mensurar as habilidades dos recursos, tais como: produtividade, comunicação, liderança, conhecimento de domínio, conhecimento sobre técnicas, etc. Sendo assim, o gerente deve ser capaz de reconhecer as habilidades dos recursos e considerar fatores como tempo, disponibilidade e domínio do problema para atualizá-las.

As habilidades dos recursos são levadas em consideração na alocação de projetos de software e podem ser medidas de acordo com nível de conhecimento e experiência, entre outros. Para avaliação das habilidades 4 passos sequenciais são executados:

1) Definir as habilidades que são determinantes.
2) Ponderar de acordo com o objetivo.
3) Definir o conjunto de regras.
4) Aplicar lógica *fuzzy.*

No primeiro passo o gerente deve escolher quais são as habilidades importantes para a alocação de recurso. No segundo passo o gerente atribuí pesos para cada habilidade, pois uma habilidade pode ser mais importante que outra para um determinado projeto. O terceiro passo é a definição do conjunto de regras que estão associadas a melhor alocação do ponto de vista do gerente. Por fim, no quarto

passo é aplicado a lógica *fuzzy* e como isto tem-se como resultado um valor de *crisp* para cada recurso que é usado na etapa de alocação de recurso.

4.3 AVALIAÇÃO DO CÓDIGO-FONTE UTILIZANDO *FUZZY*

Para avaliar a capacidade do individuo em desenvolver software com qualidade, o método proposto precisa de um projeto ou módulo desenvolvido por ele. Na falta deste, uma alternativa é que o gerente proporcione um treinamento onde o indivíduo gere um código-fonte que possa ser avaliado por uma ferramenta de métricas. Conforme relatado por Oliveira (2015), uma ou mais métricas são usadas para avaliar um requisito de qualidade e determinar o quão bom um software foi projetado. A relação de métricas e quais requisitos de qualidade estão associados está ilustrado no Quadro 2 localizado na página 41.

Portanto, a avaliação do código-fonte segue os princípios de Oliveira (2015), em que o gerente inicialmente deve determinar quais requisitos de qualidade deseja avaliar no projeto e em seguida executa o método descrito por Oliveira (2015) que como tem como resultado um valor de *crisp* para o código-fonte.

4.4 ALOCAR O RECURSO

Conforme relatado nas seções 4.2 e 4.3 no final da avaliação é gerado um valor de *crisp* para o conjunto de habilidades e para o código-fonte considerando os requisitos de qualidade que foram selecionados. Estes valores são a entrada para esta etapa que tem como finalidade realizar a alocação efetiva do recurso.

Da mesma forma que nas etapas anteriores, é necessário a definição de uma política de alocação para a escrita das regras de inferência e por fim a aplicação da lógica *fuzzy.* Ao final deste processo, os recursos melhores qualificados estarão a disposição para a alocação. Esta tarefa pode ser repetida várias vezes até que a política de alocação do gerente seja satisfeita.

.

4.5 FINALIZAÇÃO DO MÉTODO

O método é finalizado efetivamente apenas após o fechamento do projeto, quando o gerente deve gravar todas as informações possíveis sobre sua conclusão e todos os recursos são liberados de suas funções no projeto.

Espera-se que ao final do projeto o gerente ou líder de projetos, que observou o uso de tecnologias pelas equipes, atualize uma base de conhecimento para que estes dados possam posteriormente ser extraídos e utilizados em outro processo de alocação.

A atualização da base de conhecimento deve ajustar os valores de habilidades. Os recursos podem ter suas habilidades reajustadas durante o projeto, processo, desenvolvimento, etc. Logo, os gerentes devem estar atentos capturando qualquer ação que possua relevância no ambiente de desenvolvimento. Todas as regras de inferência e as políticas de seleção devem ser salvas para uso em futuros processos de alocação.

4.6 SIMULAÇÃO DO MÉTODO PROPOSTO

Nesta seção é descrito um exemplo simulando o uso do método AloDIn. A primeira avaliação a ser realizada é sobre o conjunto de habilidades. A Tabela 1 mostra um conjunto de valores (gerados aleatoriamente) atribuídos para todos os recursos em relação ao conjunto de habilidades. É importante lembrar que os valores podem ser oriundos de questionários de avaliação conforme relatado na seção 4.2.

A primeira coluna representa os indivíduos e as demais são as habilidades. As colunas que não tem valores representam os recursos que não tem habilidades estimadas e que podem ficar de fora durante o processo de alocação, caso o gerente decidir.

Tabela 1 – Conjunto de habilidades dos recursos gerados aleatoriamente

Recurso	Produtividade	Domínio do Problema	Orientação a Objetos	Padrões de Projeto
A	8,0	10,0	10,0	9,0
B	10,0	9,0	9,0	8,0
C	8,0	6,0		
D	9,0	10,0	7,0	7,0
E	8,0	9,0		
F	8,0		6,0	6,0
G			7,0	8,0
H	9,0			
I		9,0	7,0	6,0

Fonte: Autoria própria

Para o exemplo da simulação, as habilidades de *Domínio do problema* e *Produtividade* são consideradas. Assumindo que o gerente opte por desconsiderar os recursos que possuem valores faltantes, somente os recursos A, B, C, D e E são selecionados.

O próximo passo é elaborar uma tabela em que constem os recursos escolhidos considerando um peso para cada conjunto de habilidade. A Tabela 2 mostra as habilidades que já estão devidamente ponderadas, em que foi considerado um peso 0.9 (90%) para *Produtividade* e 1.0 (100%) para o *Domínio do Problema*. Ponderar as habilidades é um meio de o gerente valorizar as habilidades mais importantes para o projeto.

Tabela 2 – Valores ponderados para habilidades determinantes

Recurso	Produtividade	Domínio do Problema
A	7,2	10,0
B	9,0	9,0
C	7,2	6,0
D	8,1	10,0
E	7,2	9,0

Fonte: Autoria própria

As políticas de seleção proposta pelo método são fundamentadas na experiência e conhecimento do gerente de projetos. A única maneira de diminuir a subjetividade no método é aumentar o número de gerentes que definem as políticas de seleção, sendo assim, a definição de políticas de alocação depende do grau de conhecimento do especialista. Um exemplo de políticas de seleção pode ser observada no Quadro 4. Estas também foram geradas de forma aleatória.

Quadro 4 – Política de seleção para habilidades dos recursos

		Domínio do Problema				
		Ruim	**Regular**	**Bom**	**M. Bom**	**Excelente**
Produtividade	**Ruim**	Ruim	Ruim	Regular	Bom	Bom
	Regular	Ruim	Regular	Bom	Bom	Bom
	Bom	Regular	Bom	Bom	M. Bom	M. Bom
	Muito Bom	Regular	Bom	Bom	M. Bom	Excelente
	Excelente	Bom	Bom	Bom	Excelente	Excelente

Fonte: Autoria própria

Após a definição das políticas as regras de produção *fuzzy* são escritas no formato mostrado no Quadro 5. Todas as regras são escritas de acordo com a política de seleção do gerente.

Quadro 5 – Exemplo de regras de produção *fuzzy* para avaliação das habilidades

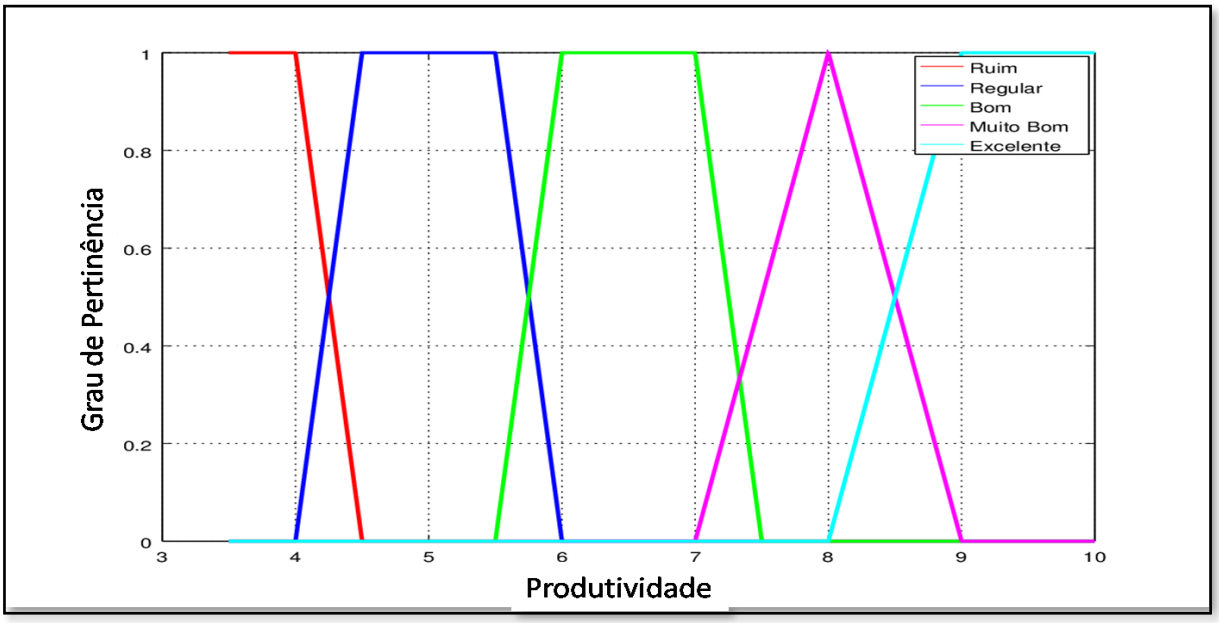
$$\textbf{IF}(Produtividade\ é\ Ruim)\ \textbf{AND}(Domínio\ do\ Problema\ é\ Ruim),$$
$$\textbf{THEN}\ (Conjunto\ de\ Habilidades\ é\ Ruim),$$
$$\textbf{IF}(Produtividade\ é\ Ruim)\ \textbf{AND}(Domínio\ do\ Problema\ é\ Regular),$$
$$\textbf{THEN}(Conjunto\ de\ Habilidades\ é\ Ruim).\ .....\ ,$$
$$\textbf{IF}(Produtividade\ é\ Regular)\ \textbf{AND}(Domínio\ do\ Problema\ é\ M.Bom),$$
$$\textbf{THEN}\ (Conjunto\ de\ Habilidades\ é\ Bom).\ .\ .\ .$$

Fonte: Autoria própria

Depois de construídas as regras, aplica-se a lógica *fuzzy* usando o método Mamdani. O método foi executado na ferramenta *GNU Fuzzy Logic Toolkit*. A

representação das funções de pertinências para as habilidades de produtividade e domínio do problema são mostradas na Figura 7 e Figura 8, respectivamente. As funções de pertinência são definidas de acordo com a necessidade, nesta simulação foram usados números trapezoidais e triangulares.

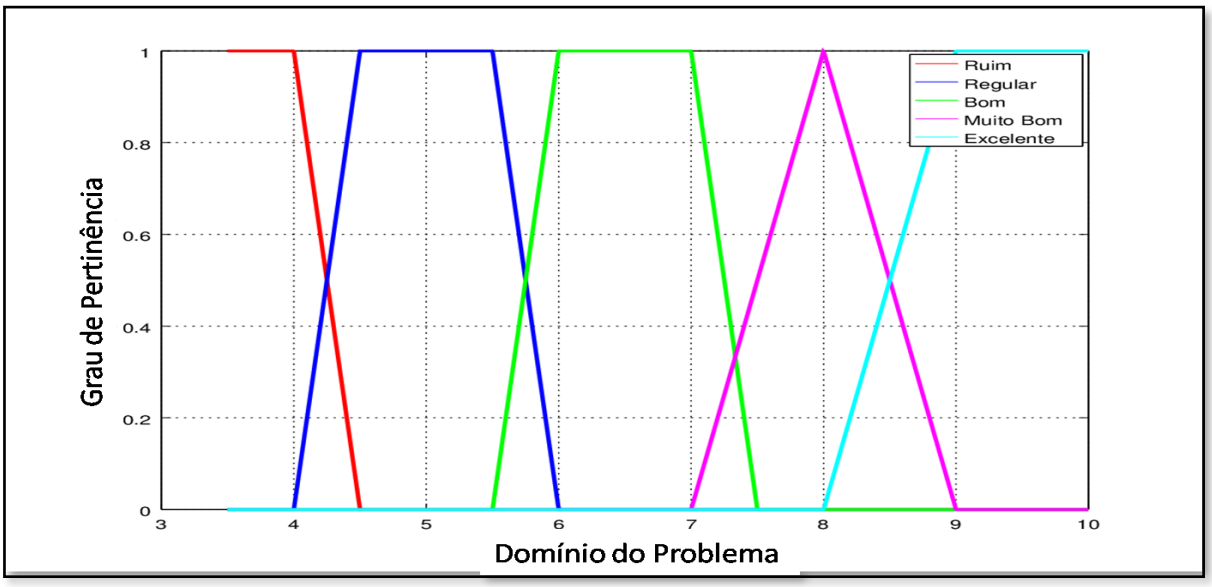
Figura 7 – Função de pertinência para produtividade



Fonte: Autoria própria

Na Figura 7 é possível observar que há 5 variáveis linguísticas mapeadas em suas respectivas funções de pertinência. Para a geração destas funções deve-se informar para a *GNU Fuzzy Logic Toolkit* qual é o intervalo de cada função de pertinência e o tipo de número *fuzzy* que será utilizado na representação. A Figura 7 possui quatro representações trapezoidais e uma triangular.

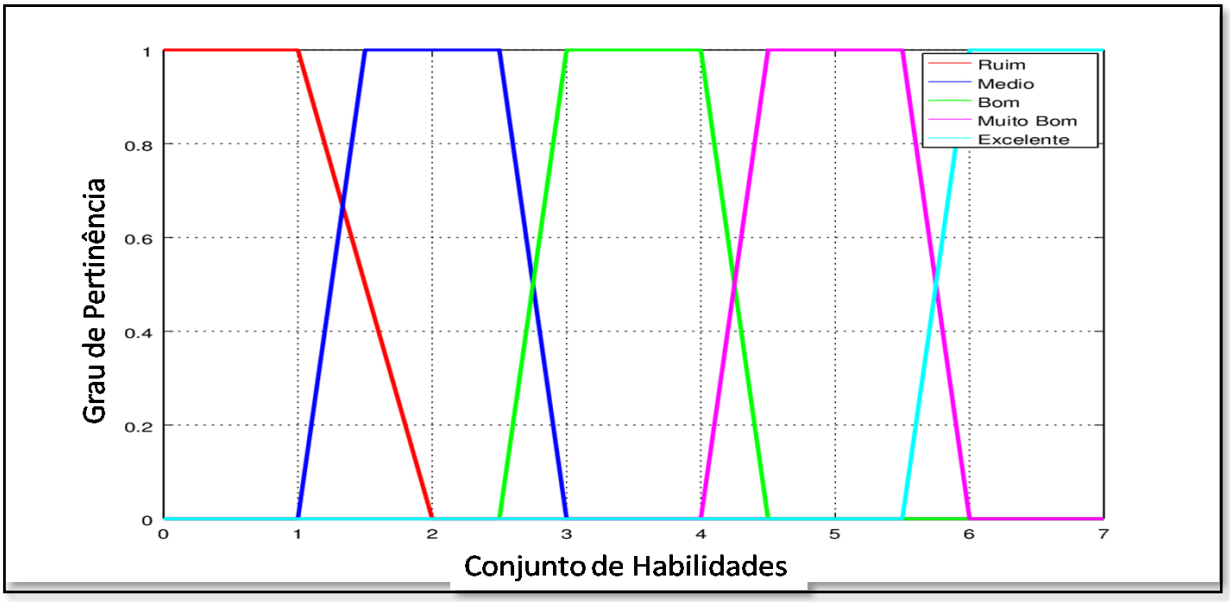
Figura 8 – Função de pertinência para domínio do problema



Fonte: Autoria própria

O conjunto de saída considerando o conjunto de habilidade é exibido na Figura 9. Este é o conjunto dos consequentes da regra, ou seja, o resultado obtido com as operações entre o conjunto de produtividade e domínio do problema será calculado de acordo com o grau de disparo da regra no conjunto de habilidades.

Figura 9 – Função de pertinência conjunto de saída



Fonte: Autoria própria

Após a aplicação do método são gerados valores *crisp* que estão entre o valor inicial e final do conjunto de saída (eixo x da Figura 9). O resultado das operações de conjuntos *fuzzy* para os conjuntos já apresentados podem ser observados na Tabela 3. Os valores são obtidos de acordo com os processos apresentados no capítulo 3.

Tabela 3 – Valores crisp para as habilidades defuzzificadas

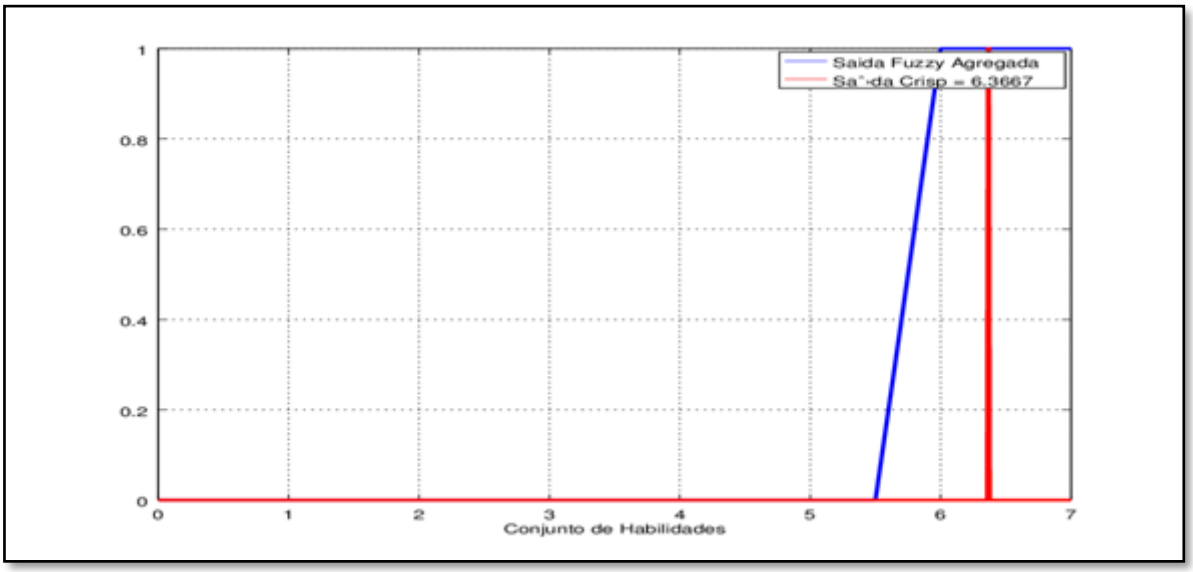
Recurso	Produtividade	Domínio do Problema	Valor crisp
A	7,2	10,0	5,0
B	9,0	9,0	6,4
C	7,2	6,0	3,5
D	8,1	10,0	5,1
E	7,2	9,0	5,0

Fonte: Autoria própria

Considerando a Tabela 3, o recurso mais bem qualificado é o B. É importante notar que ele não tem o melhor valor para o atributo que tem o maior peso, mas suas habilidades disparam uma das regras que o avaliam entre muito bom e excelente para ambas as habilidades. Neste exemplo, a força do disparo da regra e as regras de inferência foram fundamentais e cumpriram o propósito da lógica *fuzzy* não desprezando o grau de pertinência dos conjuntos associados.

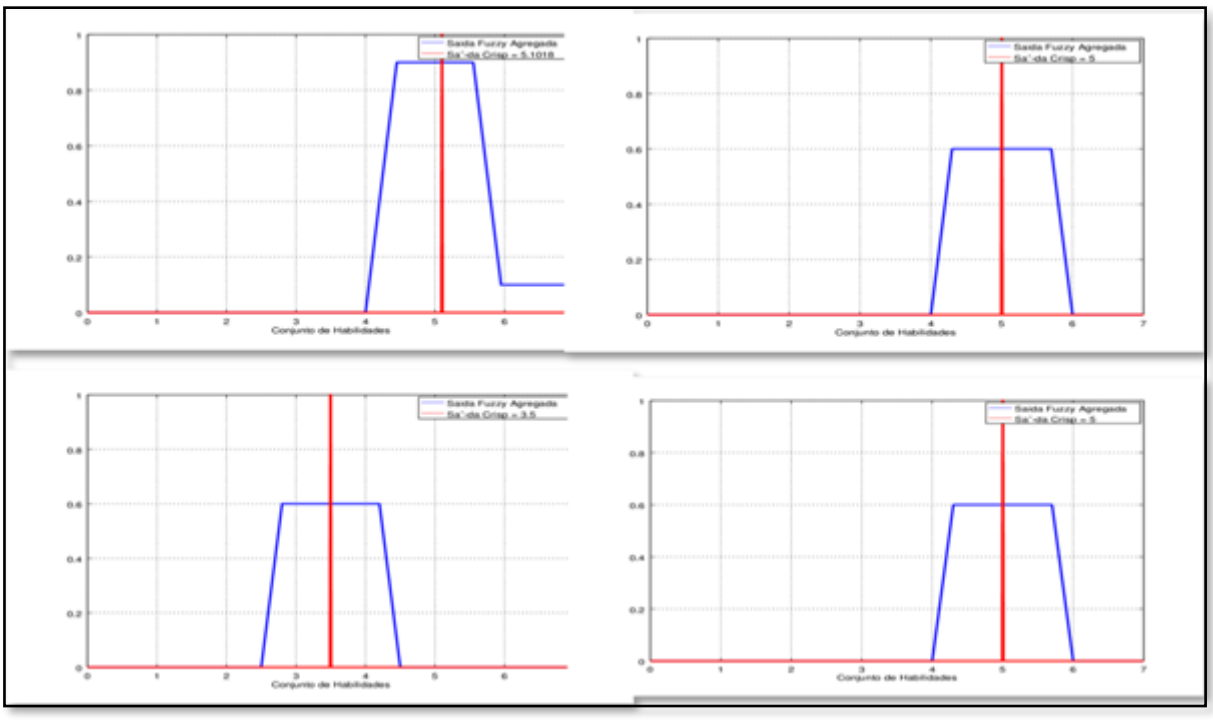
A saída *fuzzy* agregada para o recurso mais apto é mostrada na Figura 10. Para os demais recursos as saídas são mostradas na Figura 11. O método de defuzzificação utilizado é o centro de massa.

Figura 10 – Saída agregada para o recurso melhor qualificado B



Fonte: Autoria própria

Figura 11 – Saídas agregadas para os recursos D, A, C e E



Fonte: Autoria própria

Depois de gerado os valores de *crisp* para o conjunto de habilidades, deve-se avaliar os códigos-fontes produzidos pelos recursos. Nesta simulação, a avaliação de qualidade usa códigos de aplicações reais, *Qualitas Corpus* Terra et. al (2013), uma coleção de sistemas mantida por engenheiros de software para uso

empírico em pesquisa de artefatos de código. A *Qualitas.Class Corpus* é uma versão compilada de *Qualitas Corpus* que oferece 111 sistemas Java. Também avalia esses sistemas com 23 métricas a nível de classe. A avaliação das métricas é realizada com *Google Code Pro Analytix e Metrics* (*Plugin Eclipse*).

As métricas utilizadas na simulação são VG (Complexidade de McCabe) e WMC (Métodos Ponderados por Classe) e o requisito de qualidade a ser avaliado é a Complexidade a sua relação está expressa no Quadro 2. Para definição dos termos linguísticos serão utilizados os valores do catálogo de referência de Filó (2014), disponíveis no Quadro 6.

Quadro 6 – Catálogo de valores referencia para métricas orientadas a objetos

Métrica	Bom/Frequente	Regular/Ocasional	Ruim/Raro
CA	*CA* <= 7	7 <= *CA* <= 39	*CA* > 39
CE	*CE* <= 6	6 < *CE* <= 16	*CE* > 16
MLOC	*MLOC* <= 10	10 < *MLOC* <= 30	*MLOC* > 30
NOC	*NOC* <= 11	11 < *NOC* <= 28	*NOC* > 28
NOF	*NOF* <= 3	3 < *NOF* <= 8	*NOF* > 8
NOM	*NOM* <= 8	8 < *NOM* <= 14	*NOM* > 14
NORM	*NORM* <= 2	2 < *NORM* <= 4	*NORM* > 4
NSC	*NSC* <= 1	1 < *NSC* <= 3	*NSC* > 3
NSF	*NSF* <= 1	1 < *NSF* <= 5	*NSF* > 5
NSM	*NSM* <= 1	1 < *NSM* <= 3	*NSM* > 3
PAR	*PAR* <= 2	2 < *PAR* <= 4	*PAR* > 4
SIX	*SIX* <= 0,019	0,019 < *SIX* < 1,333	*SIX* > 1,333
VG	*VG* <= 2	2 < *VG* <= 4	*VG* > 4
WMC	*WMC* <= 11	11 < *WMC* <= 34	*WMC* > 34
DIT	*DIT* <= 2	2 < *DIT* <= 4	*DIT* > 4
LCOM	*LCOM* <= 0,167	0,167 < *LCOM* =< 0,725	*LCOM* > 0,725
NBD	*NBD* <= 1	1 < *NBD* <= 3	*NBD* > 3
RMD	*RMD* <= 0,467	0,467 < *RMD* <= 0,750	*RMD* > 0,750

Fonte: Adaptado de Filo (2016).

A métrica de VG de McCabe pode estar relacionada a facilidade de compreensão do software a nível de métodos. Ao manter os métodos com uma complexidade ciclomática razoável, espera-se que o código seja mais fácil de entender, menos suscetível a erros, mais fácil de mudar e reutilizar (FILO, 2014). De acordo com Sommerville (2010), quanto maior o valor da métrica de WMC mais complexa será a classe de objeto. A política de qualidade que será adotada para avaliar o requisito de qualidade de Complexidade está apresentada no Quadro 7.

Quadro 7 – Política de avaliação requisito de qualidade complexidade

		VG		
		Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
WMC	**Bom/Frequente**	Excelente	Alto	Médio
	Regular/Ocasional	Médio	Médio	Baixo
	Ruim/Raro	Baixo	Péssimo	Péssimo

Fonte: Autoria própria

Os recursos avaliados são os mesmos: A, B, C, D, E. Para cada um deles foi atribuído um dos projetos disponíveis no *Qualitas.Class Corpus.* Ao recurso A foi atribuído o projeto *jfreechart,* biblioteca para gráficos 2D. Ao recurso B foi atribuído o projeto *jgraphpad* também uma biblioteca para gráficos 2D. O recurso C recebeu o projeto *jgraph* biblioteca para a visualização de grafos. O recurso D foi atribuído o projeto *jgrapth* outra biblioteca para visualização de grafos. E o recurso E recebeu o projeto *jhotdraw*.

Os valores coletados para cada um dos projetos referente a cada métrica é apresentado na Tabela 4.

Tabela 4 – Valores coletados métricas de complexidade

Recurso	McCabe (VG)	WMC
A	2,023	22,972
B	2,458	11,829
C	2,558	24,383
D	1,788	8,080
E	2,197	20,569

Fonte: Autoria própria

Logo após as regras de inferência *fuzzy* são escritas de acordo com o critério de escolha do gerente, portanto, o fator experiência do gerente tem um peso fundamental nas escolhas. O Quadro 8 mostra exemplos de regras que foram construídas nesta simulação para as métricas VG e WMC.

Quadro 8 – Exemplo de regras de produção *fuzzy* para avaliação das habilidades

$$\textbf{IF}(VG \text{ é } Bom/Frequente)\textbf{AND}(WMC \text{ é } Bom/Frequente), \textbf{THEN}\ (Avaliação\ da\ Complexidade \text{ é } Excelente).$$

$$\textbf{IF}(VG \text{ é } Regular/Ocasional)\ \textbf{AND}(WMC \text{ é } Ruim/Raro), \textbf{THEN}\ (Avaliação\ da\ Complexidade \text{ é } Baixa).$$

....

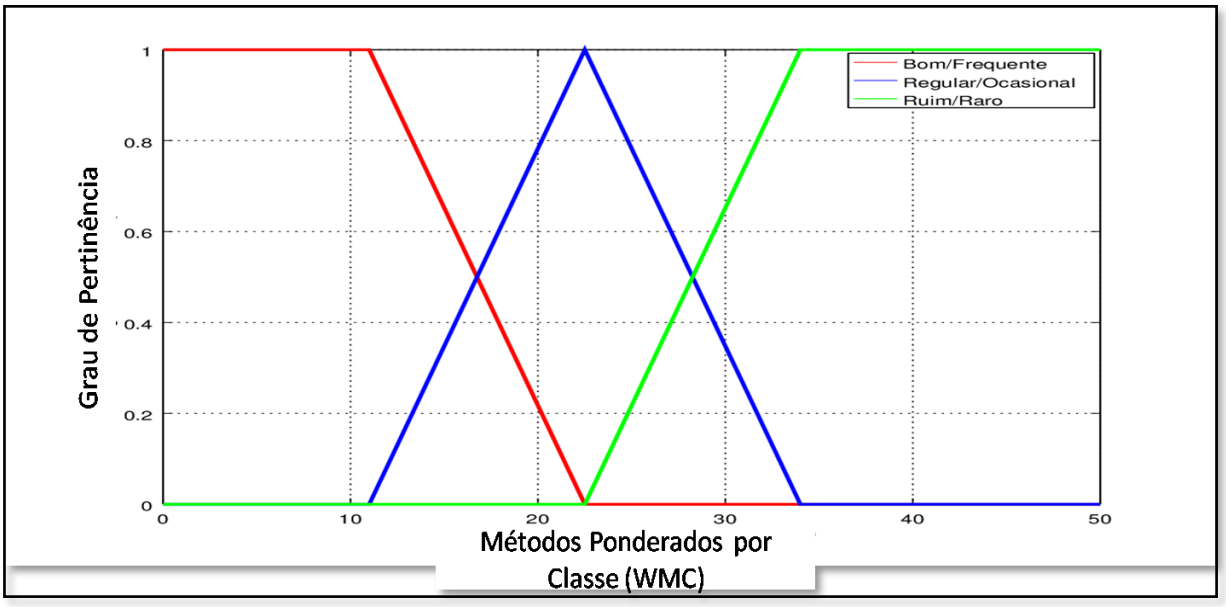
$$\textbf{IF}(VG \text{ é } Ruim/Raro)\textbf{AND}(WMC \text{ é } Ruim/\ Raro), \textbf{THEN}\ (Avaliação\ da\ Qualidade \text{ é } Péssima).$$

. . . .

Fonte: Autoria própria

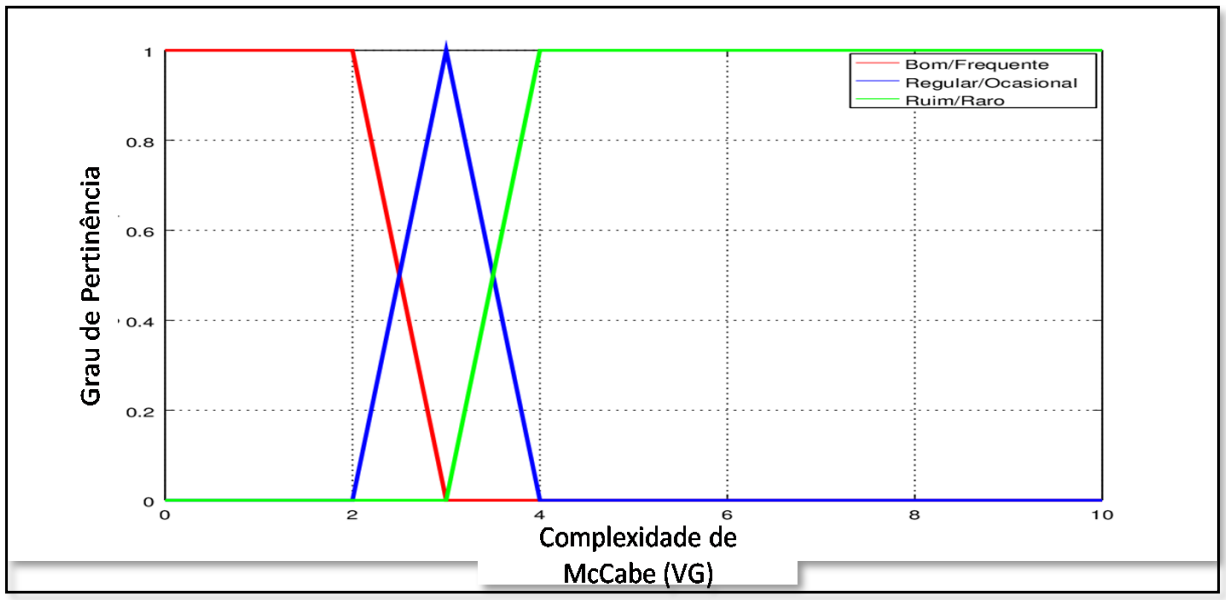
Com base nos valores coletados e no catálogo de referências de Filo (2014) as funções de pertinência são definidas. Os valores coletados nas métricas servem como fronteiras para as funções de pertinência, uma vez que se tem os valores máximos e mínimos, pode-se definir os limites das funções de pertinências. Como pode ser visto na Figura 12 e na Figura 13, em que os valores máximos são 50 e 10 respectivamente.

Figura 12 – Função de pertinência Métodos Ponderados por Classe (WMC)



Fonte: Autoria própria

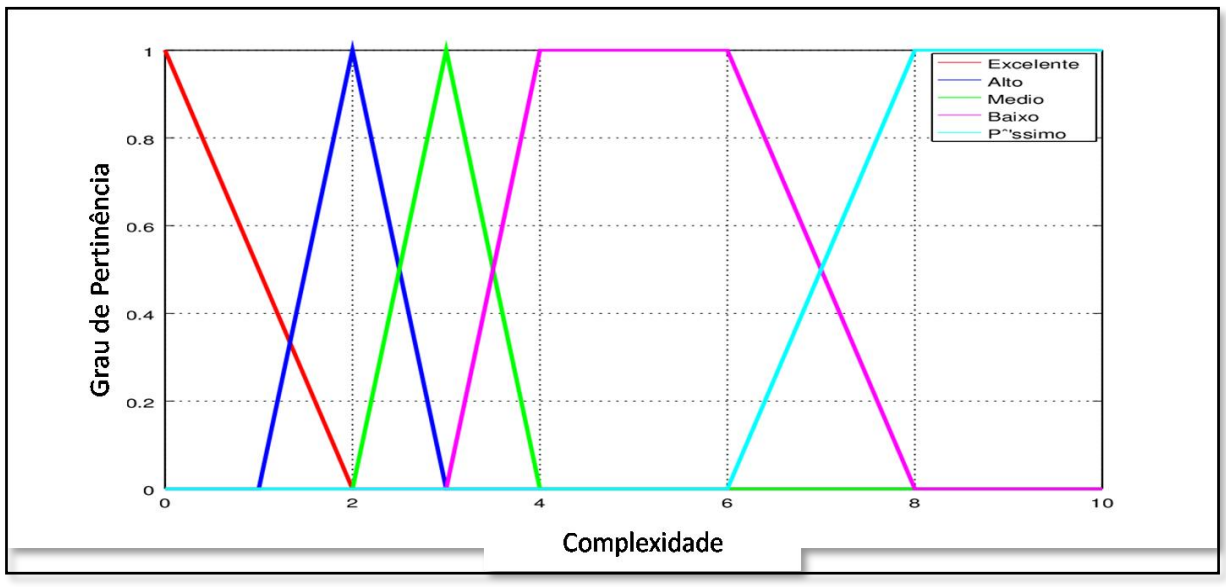
Figura 13 – Função de pertinência Complexidade de McCabe (VG)



Fonte: Autoria própria

A função de pertinência do conjunto de saída por sua vez, é definida de acordo com as políticas do Quadro 7, como pode ser visto na Figura 14.

Figura 14 – Função de pertinência qualidade do requisito qualidade Complexidade



Fonte: Autoria própria

Após a aplicação das funções de pertinência as saídas desfuzzificadas de cada um dos projetos avaliados podem ser observadas na Tabela 5.

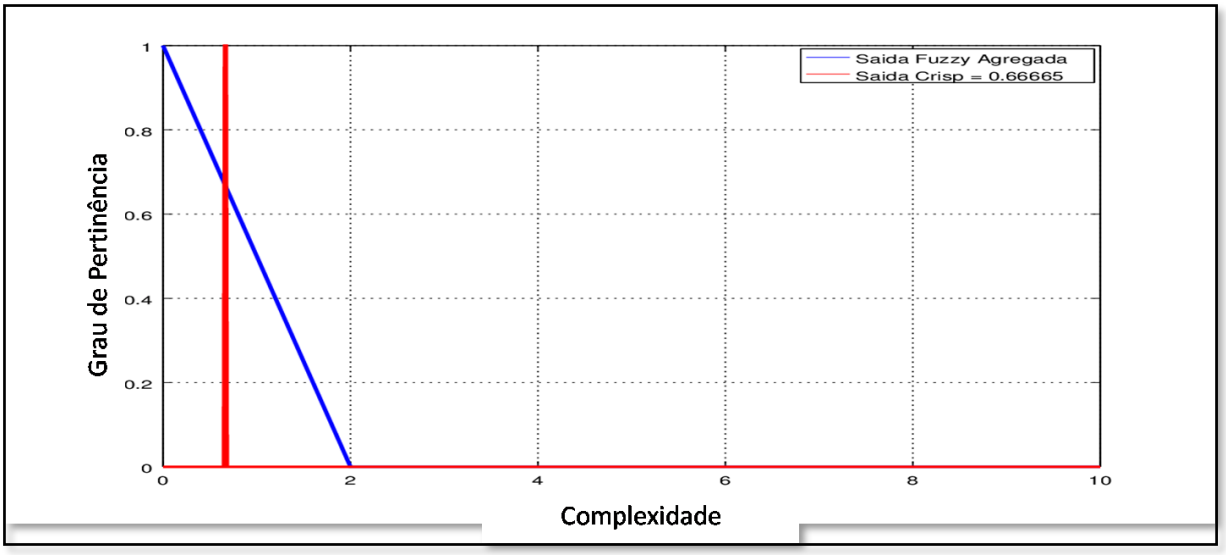
Tabela 5 – Valores *crisp* para a saída da avaliação dos requisitos de qualidade Complexidade

Recurso	McCabe (VG)	WMC	Saída Crisp
A	2,023	22,972	3,63205
B	2,458	11,829	1,44379
C	2,558	24,383	5,18966
D	1,788	8,080	0,66665
E	2,197	20,569	2,47756

Fonte: Autoria própria.

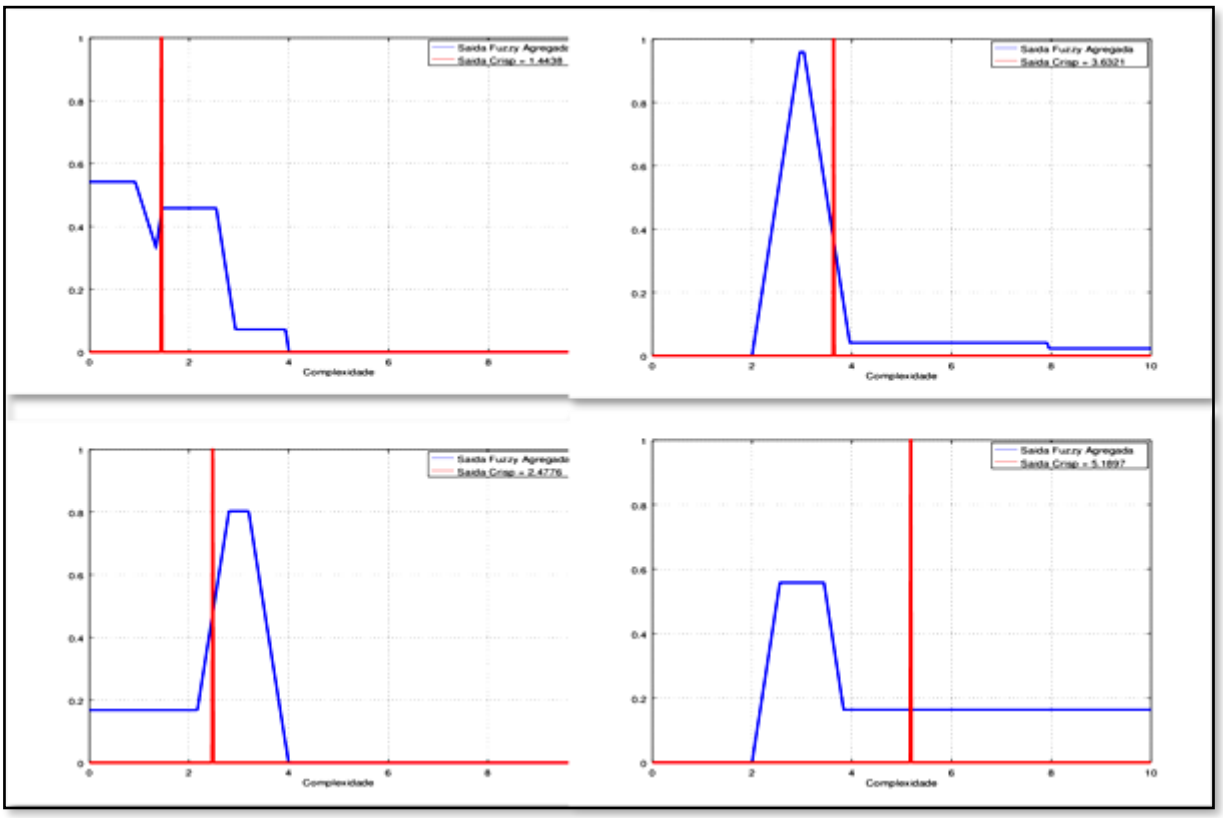
Na Tabela 5 os recursos com o menor valor crisp associado ao seu código são considerados os melhores. Como pode ser visto o recurso mais bem qualificado é o D. A saída *fuzzy* agregada para o recurso melhor qualificado é mostrada na Figura 15. Para os demais recursos as saídas são mostradas na Figura 16.

Figura 15 – Saída agregada para o recurso com projeto melhor qualificado D



Fonte: Autoria própria

Figura 16 – Saída agregada de todos os recursos avaliados B, A, E e C



Fonte: Autoria própria

Após terem sido gerados os valores de *crisp* para o conjunto de habilidade e código-fonte, inicia-se a etapa de Alocação de Recursos. A Tabela 6 ilustra os resultados dos conjuntos usados nesta simulação que foram obtidos nos passos anteriores.

Tabela 6 – Valores *crisp* do conjunto de habilidades e complexidade para cada recurso

Recurso	Conjunto de Habilidades	Complexidade
A	5,0000	3,63205
B	6,3600	1,44379
C	3,5000	5,18966
D	5,1012	0,66665
E	5,0000	2,47756

Fonte: Autoria própria

Como nas etapas anteriores, os valores *crisp* gerados tinham uma representação em termos linguísticos que foram definidas em ambos os exemplos anteriores elas serão reutilizadas (funções de pertinência Figura 9 na página 51 e Figura 14 na página 58). Portanto, os consequentes das regras anteriormente utilizadas, serão os antecedentes desta etapa, apenas é preciso definir uma nova política, agora voltada para a alocação direta como pode ser visto no Quadro 9.

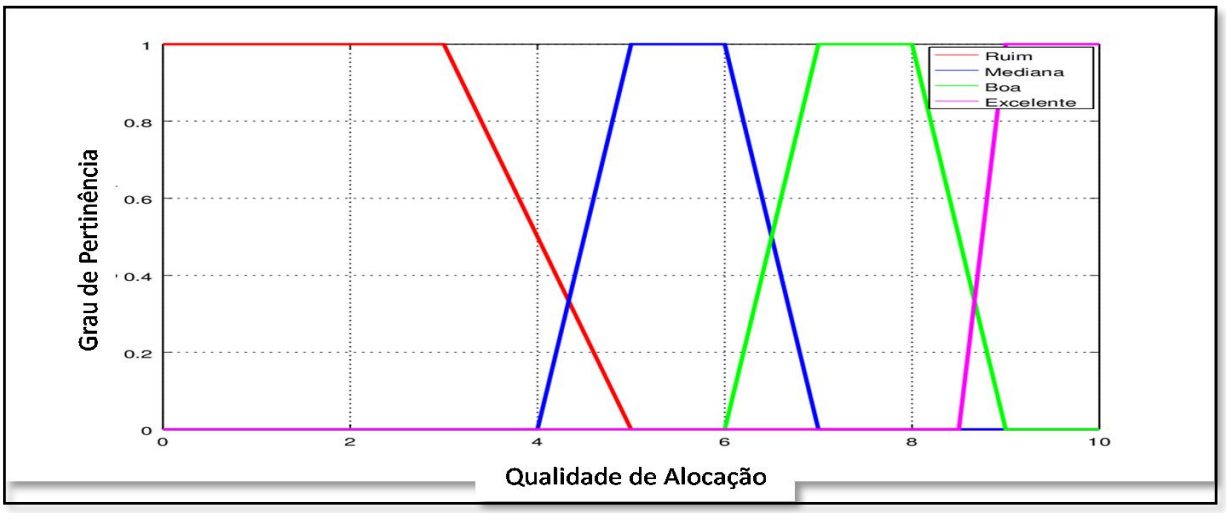
Quadro 9 – Política de seleção final habilidades x requisito de qualidade

		Capacidade do Recurso de Reduzir Complexidade				
Conjunto de Habilidades		**Excelente**	**Alta**	**Média**	**Baixa**	**Péssima**
	Ruim	Mediana	Mediana	Ruim	Ruim	Ruim
	Mediano	Boa	Boa	Mediana	Ruim	Ruim
	Bom	Boa	Boa	Mediana	Ruim	Ruim
	Muito Bom	Excelente	Boa	Mediana	Ruim	Ruim
	Excelente	Excelente	Excelente	Boa	Mediana	Ruim

Fonte: Autoria própria

Para realizar a fuzzificação nesta etapa, apenas é preciso definir uma nova função de pertinência para a saída baseada na política de seleção do Quadro 9. A função de pertinência gerada pode ser vista na Figura 17.

Figura 17 – Função de pertinência qualidade de alocação



Fonte: Autoria própria

A Tabela 7 apresenta os valores finais para o método de alocação para os recursos e políticas adotadas. A Figura 18 mostra as saídas agregadas para os recursos melhores qualificados. De acordo com a Figura 16 entende-se que os recursos qualificados com o método têm valores associados entre 0 e 10. Quanto maior o grau de saída melhor é a alocação do recurso.

Tabela 7 – Valores crisp para qualidade da alocação

Recurso	Conjunto de Habilidades	Complexidade	Qualidade da Alocação
A	5,0000	3,63205	2,9972
B	6,3600	1,44379	9,3036
C	3,5000	5,18966	2,0417
D	5,1012	0,66665	9,3297
E	5,0000	2,47756	6.5356

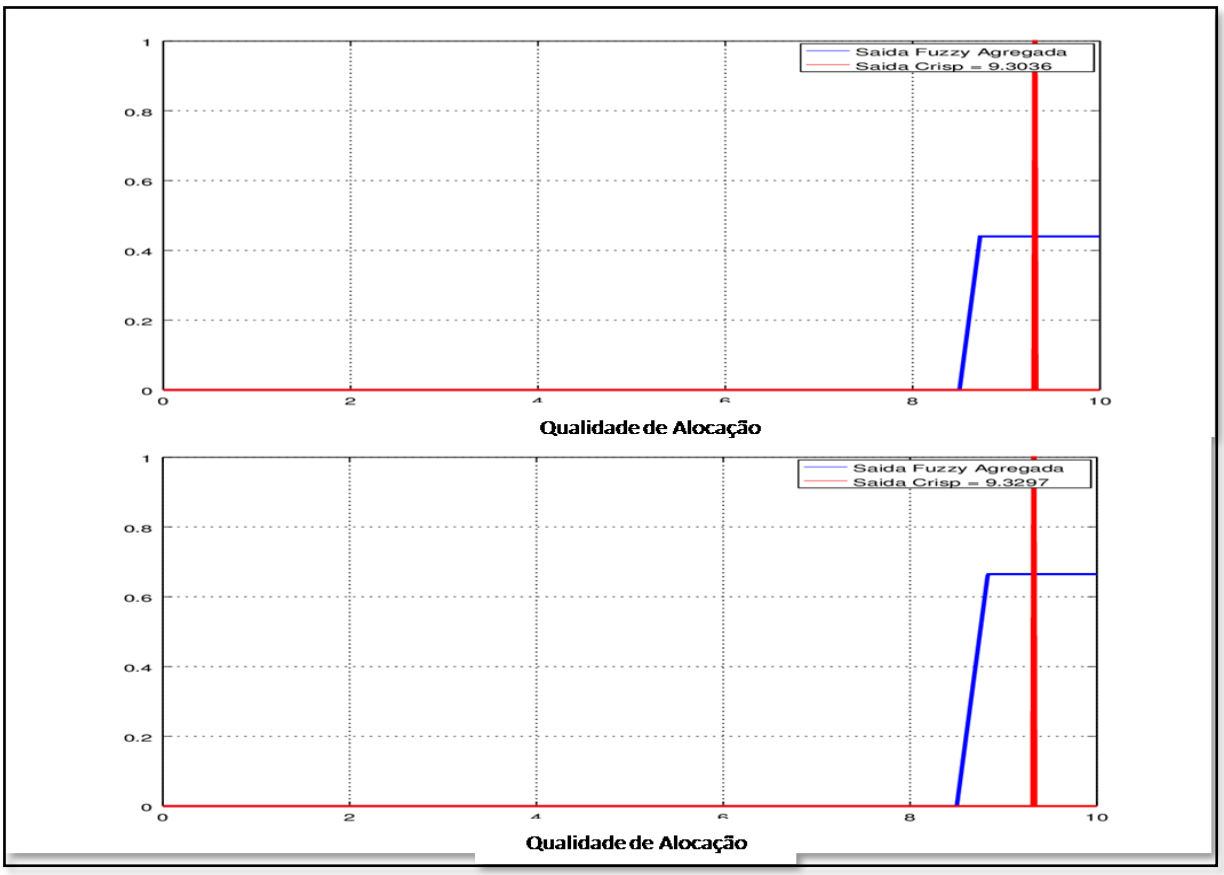
Fonte: Autoria própria

Ao analisar a Tabela 7, é possível perceber que os valores *crisp* dos recursos ficaram bem distribuídos com exceção dos dois mais bem qualificados (B e D). Isso demonstra que a lógica *fuzzy* e as políticas de qualidade podem ajudar no processo de alocação de recursos em projeto de software. Por exemplo, nesta

simulação observa-se que os recursos A e C poderiam receber um treinamento orientado a reduzir a complexidade de seus códigos.

As variáveis linguísticas são muito convenientes quando não há uma relação matemática explícita entre dois fatores decisivos, como acontece em situações diárias. Por exemplo, sem as variáveis linguísticas possivelmente falharíamos ao tentar relacionar qualidade da alocação a dois valores brutos como os da Tabela 6.

Figura 18 – Saídas agregadas para os recursos B e D



Fonte: Autoria própria.

Neste exemplo foi possível identificar os dois recursos mais aptos à alocação B e D. A escolha de alocação agora passa a ser do gerente que pode retomar o processo quantas vezes for preciso, a fim de testar outras políticas de seleção.

Para que o método seja bem aproveitado, devem-se armazenar as regras utilizadas ao confirmar a alocação de um recurso. Assim é possível reutilizar regras que deram certo e eliminar políticas de alocação que resultaram em falhas.

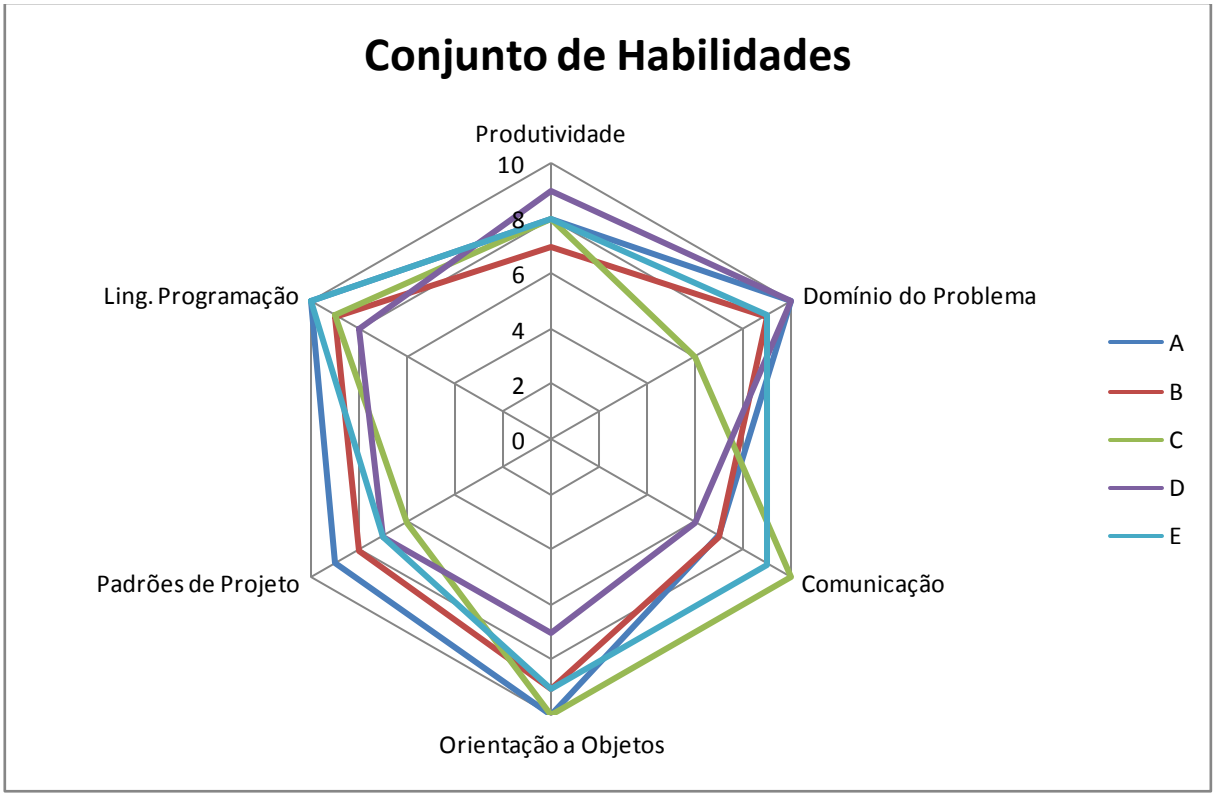
4.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo o método de alocação difuso-indutivo (AloDIn) foi descrito e seu funcionamento detalhado por meio de um exemplo simulado. A avaliação realizada sobre o conjunto de habilidades mostrou como um recurso pode ser avaliado de acordo com os objetivos de projeto, foram atribuídos pesos para as habilidades mais determinantes. A etapa de alocação mostrou como usar os valores coletados das etapas anteriores para outra avaliação com uma política mais direta.

5 CASOS DE TESTE

Este capítulo apresenta dois casos de testes com situações que hipoteticamente podem ocorrer num ambiente de desenvolvimento de projetos. As seções 5.1 e 5.2 mostram exemplos para o mesmo conjunto de recursos. Para os dois casos de testes são considerados os mesmos conjuntos de habilidades, conforme mostra o Gráfico 1.

Gráfico 1 – Conjunto de habilidades dos recursos para os casos de teste



Fonte: Autoria própria

O caso de teste 1 teve a intenção de realizar a análise do método proposto considerando que todos os recursos possuíam projetos já desenvolvidos em um domínio em que se deseja alocar recurso. Por sua vez, o caso de teste 2 analisou o método quando os recursos não haviam desenvolvido projetos no domínio específico da alocação, mas já tinham experiência em criação de código-fonte ou módulos em outros domínios.

5.1 CASO DE TESTE 1

O projeto ao qual o gerente de projetos deseja alocar recursos, tem domínio a área de gráficos 2D, então a análise de código-fonte requer que sejam informados projetos de preferência no mesmo domínio ou que sejam semelhantes.

Para análise de requisitos da qualidade são utilizados os valores da Tabela 4 (página 56) para avaliar a capacidade dos recursos em desenvolver projetos com baixa complexidade.

O gerente deve selecionar as habilidades e ponderá-las conforme sua política de alocação. Deseja-se avaliar as habilidades sobre: Produtividade no domínio, Domínio do Problema e Comunicação. Então, irá analisar sobre essas três habilidades, ponderando com um peso maior na produtividade e aplicando a lógica *fuzzy* sobre sua política de seleção, os valores de saída são mostrados na Tabela 8.

Tabela 8 – Conjunto de habilidades dos recursos destaque em produtividade

Recurso	Produtividade	Domínio do Problema	Comunicação	Conj. Hab. maior Produtividade
A	8,0	9,0	5,6	5,0000
B	7,0	8,1	5,6	3,5000
C	8,0	5,4	8,0	3,5000
D	9,0	9,0	4,8	5,0000
E	8,0	8,1	7,2	3,8864

Fonte: Autoria própria

Desejando ter uma noção melhor sobre as habilidades, o gerente pondera novamente as habilidades iniciais com um peso maior sobre domínio do problema como pode ser viso na Tabela 9 e faz o mesmo para comunicação como pode ser observado na Tabela 10.

Como os requisitos de qualidade sobre complexidade são analisados sobre um mesmo domínio, o gerente define apenas uma política de seleção, para este caso de teste são utilizados os valores da Tabela 9.

Tabela 9 – Conjunto de habilidades dos recursos destaque em domínio do problema

Recurso	Produtividade	Domínio do Problema	Comunicação	Conj. Hab. melhor Domínio do Problema
A	7,2	10	5,6	3,8864
B	6,3	9,0	5,6	3,5000
C	7,2	6,0	8,0	3,8864
D	8,1	10	4,8	5,0000
E	7,2	9,0	7,2	3,8864

Fonte: Autoria própria

Tabela 10 – Conjunto de habilidades dos recursos destaque em comunicação

Recurso	Produtividade	Domínio do Problema	Comunicação	Conj. Hab. melhor Comunicação
A	9,0	9,0	7,0	3,5000
B	5,6	8,1	7,0	3,5000
C	6,4	5,4	10	3,5000
D	7,2	9,0	6,0	3,8864
E	6,4	8,1	9,0	5,0000

Fonte: Autoria própria

A Tabela 11 mostra as saídas para o método com ênfase na produtividade, a Tabela 12 mostra a saída quando a importância maior é o domínio do problema.

Tabela 11 – Distribuição de saída da alocação: habilidades x complexidade, destaque em produtividade

Recurso	Conjunto de Habilidades	Complexidade	Melhor Alocação Produtividade
A	5,0000	3,63205	2,9972
B	3,5000	1,44379	7,5000
C	3,5000	5,18966	2,0417
D	5,0000	0,66665	9,3297
E	3,8864	2,47756	6,5356

Fonte: Autoria própria

Tabela 12– Distribuição de saída da alocação: habilidades x complexidade, destaque em domínio do problema

Recurso	Conjunto de Habilidades	Complexidade	Melhor Alocação Domínio do Problema
A	3,8864	3,63205	2,9972
B	3,5000	1,44379	7,5000
C	3,8864	5,18966	2,0417
D	5,0000	0,66665	9,3297
E	3,8864	2,47756	6,5356

Fonte: Autoria própria

Por fim a Tabela 13 mostra a saída quando é dado maior destaque para comunicação.

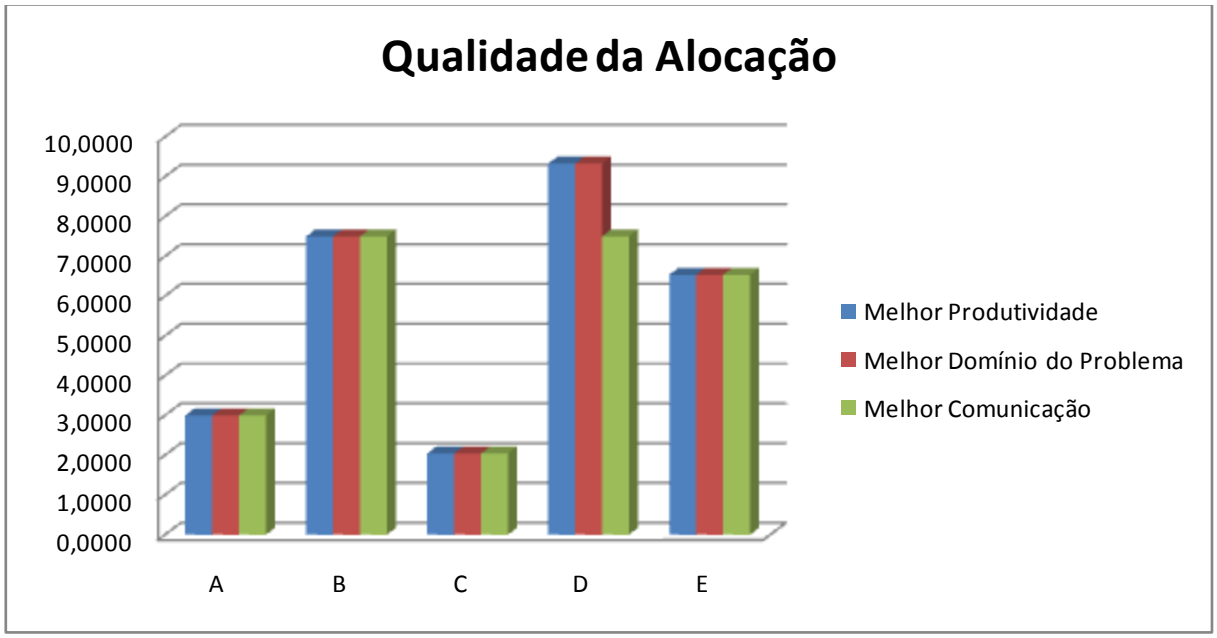
Tabela 13– Distribuição de saída da alocação: habilidades x complexidade, destaque em comunicação

Recurso	Conjunto de Habilidades	Complexidade	Melhor Alocação Comunicação
A	3,5000	3,63205	2,9972
B	3,5000	1,44379	7,5000
C	3,5000	5,18966	2,0417
D	3,8864	0,66665	7,5000
E	5,0000	2,47756	6,5356

Fonte: Autoria própria

O resultado da política de seleção adotada com atribuições de pesos diferentes para cada habilidade pode ser observadas no Gráfico 2. Neste caso de estudo o gerente necessita saber qual é o melhor recurso a ser alocado sob diferentes perspectivas. É importante notar que mesmo mudando os pesos para as habilidades a política de seleção privilegia o requisito de qualidade, mas isso acontece apenas quando os requisitos de qualidade e o domínio são bem definidos.

Gráfico 2 – Distribuição final de alocação para diferentes habilidades



Fonte: Autoria própria.

No caso de estudo 2 será mostrado um exemplo em que os requisitos estão bem definidos, porém nem todos os recursos têm projetos no mesmo domínio da aplicação.

5.2 CASO DE TESTE 2

Para este casos de estudos serão utilizados 5 projetos presentes no *Qualita.Class*: JMoney (Organizador de finanças pessoais), Apache JMeter (Aplicação para realização de testes de funcionalidade), Trove (Biblioteca Java que promete um melhor desempenho para trabalhar com *Collections*), jEdit (Editor de textos para programador Java) e Apache Jena (Framework para construção de redes semânticas).

O requisito de qualidade avaliado será reusabilidade, para avaliar este requisito serão coletadas as seguintes métricas: média de linhas de código por método MLOC (Linhas de Código por Método) e DIT (Profundidade na Árvore de Herança).

Os projetos citados acima são distribuídos aos recursos A, B, C, D e E, como pode ser observado na Tabela 14.

Tabela 14 – Valores coletados métricas de reusabilidade.

Recurso	DIT	MLOC
A	3,20500	9,76800
B	2,69700	6,66600
C	1,61100	8,35400
D	2,14300	10,89400
E	1,86400	3,93800

Fonte: Autoria própria

As políticas de qualidade para DIT e MLOC podem agora ser escritas, a definição está favorecendo um pouco mais a métrica de MLOC o que pode ser observado no Quadro 10.

A partir da definição das políticas são escritas as regras de inferência e os termos linguísticos utilizados na fuzzificação pelas funções de pertinência.

Quadro 10 – Política de qualidade para requisito de reusabilidade

		DIT		
		Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
MLOC	**Bom/Frequente**	Excelente	Alto	Médio
	Regular/Ocasional	Médio	Médio	Baixo
	Ruim/Raro	Baixo	Péssimo	Péssimo

Fonte: Autoria própria

O processo de fuzzificação é então realizado, os valores defuzzificados para as políticas adotadas são observados na Tabela 15.

.

Tabela 15 – Valores crisp para a avaliação do requisito de qualidade reusabilidade

Recurso	DIT	MLOC	Reusabilidade
A	3,20500	9,76800	4,64574
B	2,69700	6,66600	1,58507
C	1,61100	8,35400	6,00277
D	2,14300	10,89400	0,66665
E	1,86400	3,93800	4,04831

Fonte: Autoria própria

Como os dados coletados nos projetos fornecidos não tem um domínio específico, as habilidades avaliadas são mais técnicas com uma perspectiva diferente do caso de estudo anterior. As habilidades avaliadas serão: Orientação a Objetos, Padrões de Projeto e Linguagem de Programação. As habilidades são ponderadas primeiramente priorizando orientação a objetos a **Erro! Auto-referência de indicador não válida.** ilustra os valores obtidos.

Tabela 16 – Conjunto de habilidades dos recursos destaque em orientação a objetos

Recurso	Orientação a Objetos	Padrões de Projetos	Ling. Programação	Conj. Hab. O.O
A	9,00	9,00	8,00	5,1018
B	8,10	8,00	7,20	3,8864
C	9,00	6,00	7,20	3,8864
D	6,30	7,00	6,40	3,5000
E	8,10	7,00	8,00	5,0000

Fonte: Autoria própria

A Tabela 17 mostra os valores para o conjunto de habilidades priorizando padrões de projeto e a Tabela 18 os valores priorizando linguagem de programação.

Tabela 17 – Conjunto de habilidades dos recursos destaque em padrões de projeto

Recurso	Orientação a Objetos	Padrões de Projetos	Ling. Programação	Conj. Hab. Padrões de Projeto
A	9,00	9,00	8,00	6,3667
B	8,10	8,00	7,20	3,8864
C	9,00	6,00	7,20	3,8864
D	6,30	7,00	6,40	3,5000
E	8,10	7,00	8,00	5,0000

Fonte: Autoria própria.

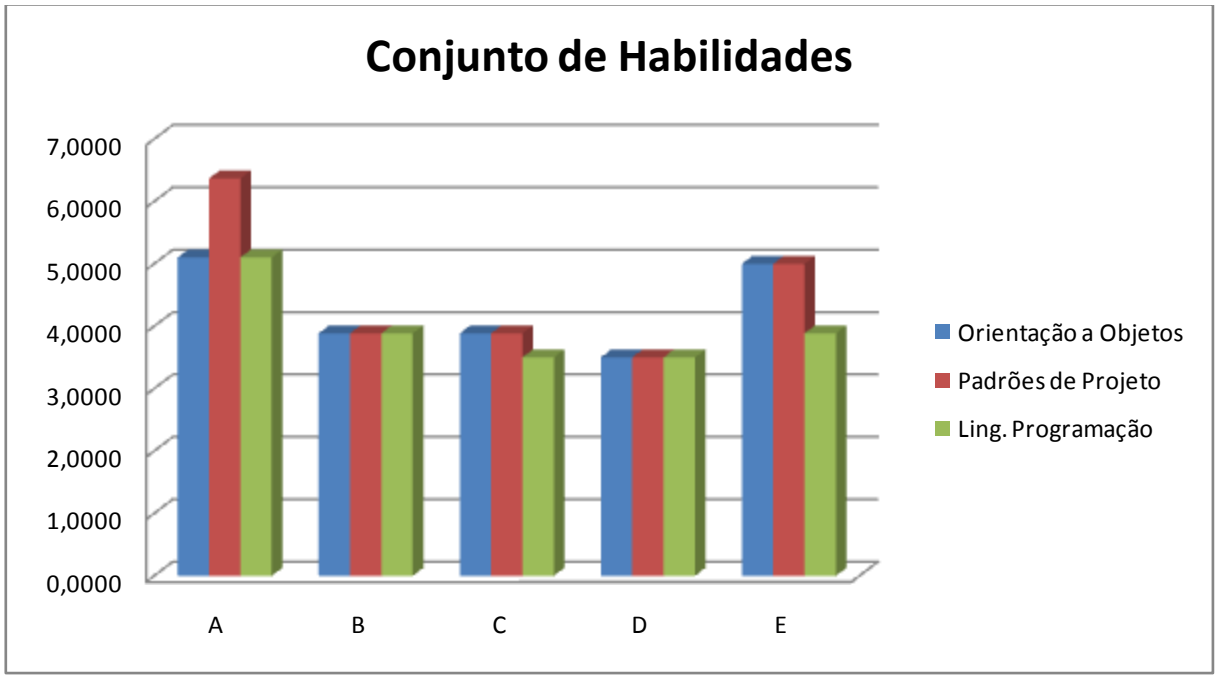
Tabela 18 – Conjunto de habilidades dos recursos destaque em linguagem de programação

Recurso	Orientação a Objetos	Padrões de Projetos	Ling. Programação	Conj. Hab. Ling. de Programação
A	8,00	8,10	10,00	5,1018
B	7,20	7,20	9,00	3,8864
C	8,00	5,40	9,00	3,5000
D	5,60	6,30	8,00	3,5000
E	7,20	6,30	10,00	3,8864

Fonte: Autoria própria.

O Gráfico 3 apresenta os valores obtidos somente na avaliação das habilidades dos recursos.

Gráfico 3 – Visão das habilidades sobre diferentes perspectivas



Fonte: Autoria própria

A Tabela 11 mostra as saídas para o método com ênfase na orientação a objetos, a Tabela 20 apresenta a saída quando a importância maior é padrões de projeto e por fim a Tabela 21 exibe a saída em destaque para linguagem de programação.

Tabela 19 – Distribuição de saída da alocação: habilidades x reusabilidade, destaque em orientação a objetos

Recurso	Conjunto de Habilidades	Reusabilidade	Melhor Alocação Orientação a Objetos
A	5,1018	4,64574	2,0417
B	3,8864	1,58507	7,5000
C	3,8864	6,00277	2,0422
D	3,5000	0,66665	7,5000
E	5,0000	4,04831	2,0417

Fonte: Autoria própria

Tabela 20 – Distribuição de saída da alocação: habilidades x reusabilidade, destaque em padrões de projeto

Recurso	Conjunto de Habilidades	Reusabilidade	Melhor Alocação Padrões de Projeto
A	6,3667	4,64574	2,9972
B	3,8864	1,58507	7,5000
C	3,8864	6,00277	2,0417
D	3,5000	0,66665	7,5000
E	5,0000	4,04831	2,0417

Fonte: Autoria própria.

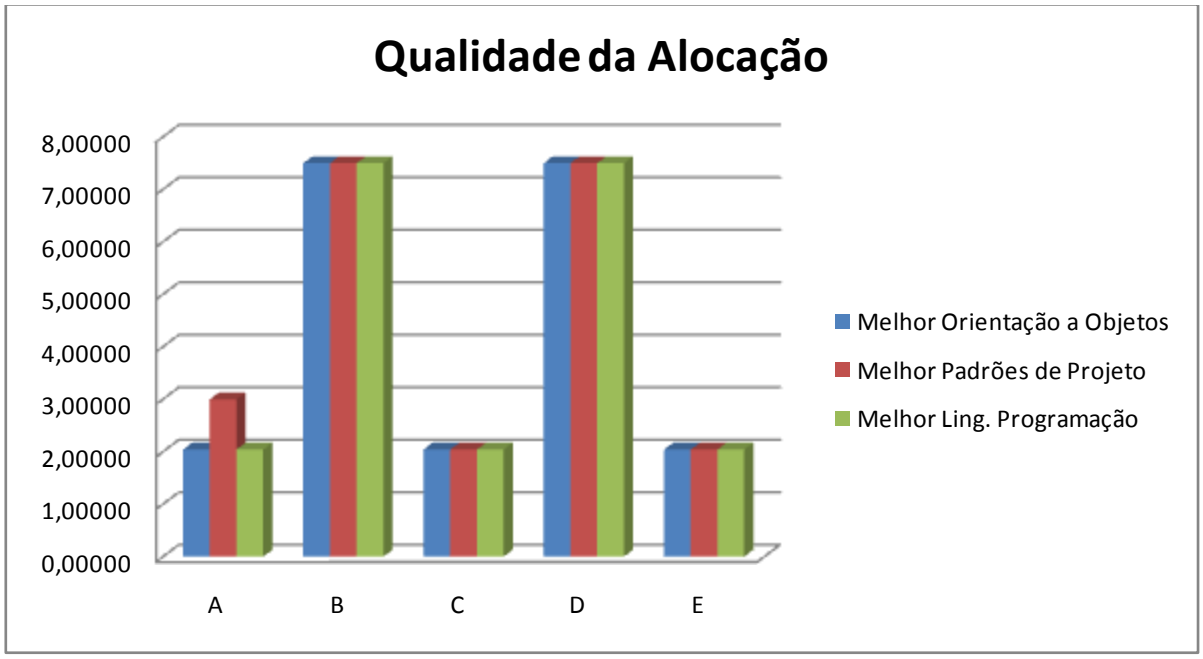
Tabela 21 – Distribuição de saída da alocação: habilidades x reusabilidade, destaque em linguagem de programação

Recurso	Conjunto de Habilidades	Reusabilidade	Melhor Alocação Ling. de Programação
A	5,1018	4,64574	2,0417
B	3,8864	1,58507	7,5000
C	3,5000	6,00277	2,0417
D	3,5000	0,66665	7,5000
E	3,8864	4,04831	2,0417

Fonte: Autoria própria.

Os resultados finais obtidos nesta avaliação podem ser observados no Gráfico 4. Note que ao comparar os valores finais antes da última etapa do caso de teste 2 no Gráfico 2, foi avaliado que os recursos A e E eram os que tinham as melhores habilidades, porém ao aplicar a última política de seleção entre habilidades e reusabilidade foi possível perceber que os recursos A e E não conhecem bem suas habilidades. Todavia os recursos B e D subestimaram suas habilidades e o recurso C é aquele que tem uma noção melhor de suas habilidades.

Gráfico 4– Distribuição final de alocação para diferentes habilidades



Fonte: Autoria própria.

Neste caso de estudo o método identificou possíveis falhas na obtenção no conjunto de habilidades, mesmo quando não havia código-fonte de mesmo domínio da aplicação. Um treinamento pode ser realizado, principalmente sobre o recurso C e os recursos A e E devem ser submetidos a um ajuste em suas habilidades até que realizem novos testes. Os recursos B e D são os melhores recursos para a alocação neste caso de teste.

Ao final do projeto o mesmo deve ser analisado sobre sua qualidade para que os recursos alocados tenham suas habilidades e requisitos de qualidade atualizados na base de dados, indicada pelo método.

5.3 ANÁLISE DA APLICAÇÃO DO MÉTODO

A proposta inicial do método evoluiu de forma que diversos aspectos da alocação de recursos humanos foram considerados. Esses aspectos são muito estudados em gerência de projetos, nessa abordagem foram relacionadas as habilidades à capacidade de atingir qualidade de software. A importância de incluir fatores humanos na alocação é fundamental, porque em grandes ambientes há

muitas equipes e tanto equipes como projetos tem características que são melhores adaptas a recursos humanos específicos, com suas particularidades e qualidades.

Deste modo, a avaliação de habilidades por lógica *fuzzy* se mostrou muito superior a outras abordagens, pois foram eliminados fatores como subjetividade que estão relacionados intimamente a estimar habilidades por valores discretos. As abordagens falham principalmente por não conseguirem identificar a nossa realidade atual que faz do ambiente de trabalho altamente dinâmico, um lugar em que a caracterização de habilidades é meramente temporária.

Para os exemplos mostrados nesse trabalho, foram utilizadas métricas de software orientado a objetos porque são as que tinham um catálogo de referência mais amplo e confiável, o método de alocação proposto é agnóstico de linguagem de programação. Os princípios aqui utilizados já foram utilizados na literatura, sendo assim associar métricas a requisitos de qualidade não é algo novo, todavia como afirma Filó (2014), apesar de serem conhecidos os benefícios da qualidade de software são poucas as métricas de referência reconhecidas e confiáveis para essa associação menos subjetiva de métricas com qualidade de software.

É importante lembrar que há métricas internas e externas relacionadas aos recursos, nesse sentido as habilidades mensuradas durante o método de alocação são privadas. Apenas o gerente de projeto deve ter acesso as habilidades dos recursos, mesmo após a aplicação de testes. Essa medida visa manter o ambiente sadio, com menos competitividade e risco de ruído, uma vez que sabendo do processo de avaliação, pode acontecer de um recurso apenas tentar aumentar sua pontuação decorando testes e não efetivamente aprendendo no processo.

De acordo com Rezende (2002), a medição resulta em mudança cultural, em modificações de comportamento, em vontade de produzir software confiável. Ainda segundo o mesmo deve-se criar um linha de base (banco de dados) que coleta, armazena, realiza computação de métricas e avaliação dos dados, para que de posse desses dados os engenheiros de software tenham uma melhor visão do trabalho que realizam. Essa é a principal ideia do método de alocação difuso-indutivo, é que ele possa ser melhorado continuamente com uma participação ativa da equipe de desenvolvimento e da equipe de gerenciamento.

5.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou exemplos de uso do método em situações distintas, no caso de teste 1 todos os recursos tinham projetos (código-fonte) relacionados ao domínio do sistema, por isso o método focou mais nas habilidades não técnicas na tentativa de saber qual o melhor recurso com habilidades não técnicas que tem o perfil melhor qualificado para o projeto.

No caso de teste 2, os recursos tinham projetos em áreas de domínios diferentes, logo é difícil estimar a qualidade de software, pois os requisitos não são claros. Portanto, o foco foi nas habilidades técnicas na tentativa de identificar bons desenvolvedores.

O método identificou que os recursos podem não saber dimensionar seus conhecimentos, utilizando o método foi possível perceber que os recursos que tinham a melhor avaliação do conjunto de habilidades técnicas não tiveram seus projetos (códigos-fontes) bem avaliados.

6 CONCLUSÃO

Este trabalho criou um método de alocação de recursos composto por várias etapas, porém em cada etapa, exceto a última, a lógica fuzzy é utilizada. A primeira etapa permite avaliar um conjunto de habilidades de um recurso produzindo um valor de *crisp*. Na segunda etapa para não ficar na subjetividade, é proposto uma avaliação do código-fonte produzido pelo recurso e como saída se tem também um conjunto *crisp*. Os valores de *crisp* gerados na primeira e segunda etapa são parâmetros de entrada para a terceira etapa em que se realiza a alocação efetivamente. Isto mostra que as etapas trabalham de forma integrada. Por fim, a última etapa sugere que todos os dados gerados sejam armazenados para que possam ser reaproveitados em um novo processo de alocação.

Fatores positivos do AloDIn é que permite ao gerente ter uma percepção sobre os parâmetros de alocação, tais como atribuir pesos diferentes para cada conjunto de habilidades ou métricas. Trabalhar com pesos diferentes permite uma melhor adaptação a necessidade do projeto.

Outro ponto relevante do método proposto é que os recursos são avaliados individualmente quanto a sua capacidade de desenvolver software com qualidade. Isto pode influenciar na equipe que aprende mais dividindo o ambiente de desenvolvimento com pessoal bem capacitado. Além de identificar falhas e necessidade de treinamento, conforme relatado dos casos de teste.

Há algumas limitações no método proposto quanto a escassez de recursos para a avaliação, o número de regras de inferência tende a ser grande, o gerente deve ter conhecimento sobre qualidade de software e o desconhecimento sobre o domínio da aplicação. Para reduzir as limitações é proposto registrar a participação do recurso no projeto, desde o momento da sua entrada até sua saída, inclusive o motivo. As regras de inferência devem ser armazenadas e se preciso avaliadas junto a um especialista.

6.1 TRABALHOS FUTUROS

A partir deste trabalho novas pesquisas e projetos podem ser desenvolvidos. São elas:

- Identificação de métricas a nível de código, para outras linguagens de programação. E sua relação com requisitos de qualidade.
- Criar a ferramenta para automatizar o processo.
- Utilização de *Data Science* sobre os dados coletados no método para melhorar os registros e reconhecer políticas de alocação de qualidade.
- Ampliar a avaliação para alocação voltada a testes e engenharia de requisitos.
- Ampliar a avaliação de qualidade além do código-fonte, estabelecendo parâmetros confiáveis para alocar de acordo com nível de usabilidade e acessibilidade atingida. Por exemplo, utilizar leitores de tela, para coletar medidas e correlacionar a qualidade utilizando lógica *fuzzy*.

REFERÊNCIAS

ABREU, F.B., CARAPUÇA, R. Objected-oriented software engineering: measuring and controlling the development processs. **Proceedings of 4th International Conference on Software Quality**, out 1994.

ARAÚJO, Ernesto. Lógica Difusa (Fuzzy) e Raciocínio Aproximado: Conceitos e Aplicações. **ERMAC**. Pato Branco: 2009, UTFPR. Disponível em: <https://www.researchgate.net/publication/40741668_Logica_Difusa_Fuzzy_e_Raciocinio_Aproximado_Conceitos_e_Aplicacoes> Acesso em: 10 dez . 2017.

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

BURDETT, Greg; LI, Raymond K.-Y. A quantitative approach to the formation of workgroups. **SIGCPR**, 1995. p. 202 – 212. Disponível em: <http://dl.acm.org/citation.cfm?doid=212490.212599>. Acesso em: 03 abr. 2017.

CINGOLANI, Pablo; ALCALÀ-FDEZ, Jesus. jfuzzylogic: a robustandflexiblefuzzy-logicinference system languageimplementation. **IEEE InternationalConference**, 2012. p. 1–8.

CHERRI, Adriana Cristina; ALEM JUNIOR, Douglas José; SILVA, Ivan Nunes da. Inferência fuzzy para o problema de corte de estoque com sobras aproveitáveis de material. **Pesquisa Operacional**, Rio de Janeiro , v. 31, n. 1, p. 173-195, Abril. 2011 . Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382011000100011&lng=en&nrm=iso>. Acessoem 10 Mar. 2018. http://dx.doi.org/10.1590/S0101-74382011000100011.

CHIDAMBER, S., KEMERER, C. A metric suite for object oriented design, IEEE Transactions On Software Engineering, jun 1994.

COFFMAN Jr., E. G.; GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin packing: a survey**. PWS Publishing Co**., Boston, 1997. Disponível em: <http://portal.acm.org/citation.cfm?id=241938.241940>. Acesso em: 10 abr. 2017.

E. NARTEY, Isaac Owusu-Nyarko. Feasibility Study on Intelligent Evaluation of Marine Traffic Congestion Degree for Restricted Water Using Fuzzy Expert System with AIS Report. **Journal of Electrical and Electronic Engineering**. Vol. 4, No. 6, 2016, pp. 150-156. doi:10.11648/j.jeee.20160406.12

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

FILÓ, Tarcízio Guerra Savino.**Identificação de valores referência para métricas de softwares orientados por objetos**. 2014. 197 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Minas Gerais, Minas Gerais, 2014.

HARRISON, R., COUNSELL S. J.; NITHI V. An Evaluation of MOOD Set of Object-Oriented Software Metrics. **IEEE Transactionson Software Engineering**, jun 1998.

HELDEMAN, Kim. **Gerência de Projetos – fundamentos: um guia prático para quem quer certificação em gerência de projetos.** Tradução de Luciana do Amaral Teixeira. 5 ed. Rio de Janeiro: Elsevier, 2005.

IETEC. Pesquisa realizada com os participantes do 16º Seminário Nacional de Gestão de Projetos. **Instituto de Educação Tecnológica**. Disponível em: <http://techoje.com.br/pdf/pesquisa-seminario-projetos-2013.pdf>. Acessoem: 13 jun. 2017.

ISO IEC 91261. Software Engineering product quality part 1: Quality model, 2001.

ISO IEC 25n. Square (System and Software Quality Requirements and Evaluation), .Disponível em: <http://iso25000.com>. Acesso em: 05 jun. 2017.

KERZNER, Harold. **Gerenciamento de Projetos: Uma abordagem sistêmica para planejamento, programação e controle.** Tradução de João Gama Neto e Joyce I. Prado. 10 ed. São Paulo: Blucher, 2011.

LEE, Kwang H. **First Course on Fuzzy Theory and Applications.**3. ed. Berlim: Springer, 2005.

MARRO, Alessandro A.; et al. **Lógica Fuzzy: Conceitos e Aplicações** 2010.

NBR ISO 10006. Gestão da qualidade – Diretrizes para a Qualidade no gerenciamento de Projetos. **Associação Brasileira de Normas Técnicas**. 2000. Disponível em: <http://qualidade.ipen.br/Normas_Iso/nbriso10006.pdf>. Acessoem: 13 jun. 2017.

OLIVEIRA, João Antônio Carvalho de. **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.

ORTEGA, Neli Regina S. **Aplicação da Teoria de Conjuntos Fuzzy a Problemas da Biomedicina.** 2001. 166 f. Tese (Doutorado em Ciências) – **Universidade de São Paulo**,São Paulo, 2001.

OTERO, L. D. et al. A systematic approach for resource allocation in software projects. **Computers& Industrial Engineering**,v.56, n.4, mai. 2009 p. 1333 – 1339, 2009. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0360835208001708>. Acesso em: 10 abr. 2017.

PARK, R. E., GOETHERT, W.B., FLORAC W. A. **Goal Driven Measurement – A Guidebook.**Software Eng. Institute, Carnegie MellonUniversity, 1996. Disponível em < https://resources.sei.cmu.edu/asset_files/Handbook/1996_002_001_16436.pdf > Acesso em: 17 mar. 2018.

PMBOK. **Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK ®)**. 5 ed. Pennsylvania: PMI, 2013.

PRESSMAN, Roger S. **Engenharia de Software. Uma abordagem profissional.** Tradução de Ariovaldo Griesi. 7 ed. Porto Alegre: AMGH, 2011.

RENTERÍA, Alexandre Roberto. **Estimação de probabilidade *fuzzy* a partir de dados imprecisos**. 2006. 94 f. Tese (Doutorado em Engenharia Elétrica) – Pontificia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. 2 ed. Rio de Janeiro: Brasport, 2002.

REZENDE, Solange Oliveira. **Sistemas Inteligentes: Fundamentos e Aplicações**. Barueri: Manole, 2005.

RIGNEL, D. G. de S. et al. Uma introdução a Lógica Fuzzy. **Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica**, v. 1, n. 1, p. 17-28, 2011.

ROCHA, Ana Regina Cavalcanti da. **Qualidade de Software: Teoria de Prática.** São Paulo: Prentice Hall, 2001.

ROCHA, Ítalo Mendonça. **Uma Abordagem Otimizada para o Problema de Alocação de Equipes e Escalonamento de Tarefas para a Obtenção de Cronogramas Eficientes.** 2011. 121 f. Dissertação (Mestrado em Ciência da Computação) –Universidade Estadual do Ceará,Fortaleza, 2011.

ROSS, Timothy J. **Fuzzy Logic With Engineering Applications.**3 ed. New Mexico: Wiley, 2010.

SANTOS, J. A; CARVALHO, H. G. **Referencial Brasileiro de Competências em Gerenciamento de Projetos.** Curitiba: ABGP, 2006.

SANTOS, Vinícius Souza dos. **Uma abordagem para a seleção de equipes tecnicamente qualificadas para implementação de projetos de software.** 2014. 154 f. Dissertação (Mestrado em Informática) – Universidade Federal da Paraíba, João Pessoa, 2014.

SILVA, Lúcio Camara e. **Modelos de decisão para a alocação de recursos humanos em projetos de sistemas de informação**. 2009. 79 f. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Pernambuco, Recife, 2009.

SIVANANDAM S.N, S. SUMATHI, S.N. DEEPA**. Introduction to Fuzzy Logic using MATLAB**. 1 ed. New York, Springer, 2007.

SOMMERVILLE, Ian**. Software Engineering**. 9. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2010.

TERRA, R; MIRANDA, L. F.; VALENTE, M. T.; BIGONHA, R. Qualitas.class Corpus: A compiled version of the Qualitas Corpus. **Software Engineering Notes**, pages 1–4, 2013.

ZADEH, Lofti A**.** Fuzzy sets. **Information and Control,** v. 8, n. 3, p. 338 – 353, 1965. Disponível em: <http://www.sciencedirect.com/science/article/pii/S001999586590241X>. Acessoem: 03 abr. 2017.