MODELGURU: Uma Abordagem Baseada em Métricas para Avaliação de Diagrama de Classes da UML

Cristiano Costa Farias[1]
Kleinner Silva Farias de Oliveira[2]

Resumo: A avaliação de diagrama de classes da UML desempenha um papel fundamental no ensino e na aprendizagem de modelagem de software. Ao elaborar um diagrama de classes da UML, estudantes precisam obter uma avaliação do instrutor para verificar o seu desempenho. Tipicamente, esta avaliação é representada por uma nota, variando de zero a dez. Porém, avaliar os diagramas ainda é uma tarefa subjetiva, sem critérios previamente definidos e compartilhados com os estudantes, exige muito esforço, propensa a erros, sem homogeneidade e transparência sobre os critérios utilizados. Este trabalho, portanto, apresenta o ModelGuru, uma abordagem baseada em métricas para avaliar diagrama de classes da UML. O ModelGuru utiliza métricas de design de software e flexibiliza a avaliação ao tornar configurável os pesos atribuídos as métricas. O ModelGuru foi avaliado através de um estudo de caso para demonstrar sua viabilidade técnica e através do questionário TAM (Modelo de Aceitação de Tecnologia) para entender a percepção de instrutores e alunos em relação aos benefícios da abordagem. No total, 14 participantes responderam o questionário TAM. Os resultados do estudo de caso demonstram a viabilidade do ModelGuru na avaliação de diagramas de classes da UML, gerando notas de forma apropriada. Além disso, os participantes perceberam a facilidade (71,3%) e a utilidade (78,5%) do ModelGuru, bem como mostraram a intenção de uso da abordagem proposta (78,5%). OS resultados indicam que a abordagem é promissora para uso no contexto educacional.

Palavras-chave: Modelagem de Software, Métricas de Projeto, Avaliação, UML, Diagrama de Classes.

Abstract: UML Class Diagram Assessment plays a key role in teaching and learning software modeling. When drawing up a UML class diagram, students need to get an instructor rating to verify their performance. Typically, this assessment is represented by a grade, ranging from zero to ten. However, evaluating diagrams is still a subjective task, without criteria previously defined and shared with students, it requires a lot of effort, is prone to errors, without homogeneity and transparency about the criteria used. This work, therefore, presents the ModelGuru, a metrics-based approach to evaluating UML class diagrams. ModelGuru uses software design metrics and makes evaluation flexible by making the weights assigned to metrics configurable. ModelGuru was evaluated through a case study to demonstrate its technical feasibility and through the TAM (Technology Acceptance Model) questionnaire to understand the perception of instructors and students regarding the benefits of the approach. In total, 14 participants answered the TAM questionnaire. The case study results demonstrate the feasibility of ModelGuru in the evaluation of UML class diagrams, generating grades appropriately. In addition, participants perceived the ease (71.3%) and usefulness (78.5%) of ModelGuru, as well as showing the intention to use the proposed approach (78.5%). The results indicate that the approach is promising for use in the educational context.

Keywords: Software Modeling. Evaluation. UML. Class Diagram.

[1]Graduando em Ciência da Computação pela Unisinos. Email: cristianocf@edu.unisinos.br
[2]Programa de Pós-Graduação em Computação Aplicada, UNISINOS. Email: kleinnerfarias@unisinos.br

1 INTRODUÇÃO

No ensino da modelagem de software a utilização de diagramas é evidente e indispensável, e apesar de na indústria este uso ainda ser exíguo (JÚNIOR; FARIAS; SILVA, 2021) a UML (RUMBAUGH; JACOBSON; BOOCH, 2004) desempenha um papel fundamental no desenvolvimento de software e defende que a utilização da modelagem traz diversos benefícios ao processo da engenharia de software (FERNÁNDEZ-SÁEZ; CHAUDRON; GENERO, 2018). Quando modelos UML são criados pelos alunos ou profissionais da engenharia de software eles têm de ser avaliados afim validar sua consistência e qualidade, no ambiente educacional este *feedback* é mais crítico pois está relacionado diretamente ao processo de aprendizagem do aluno.

A avaliação dos diagramas no ensino da modelagem de software é feita em geral de forma manual pelos instrutores, este processo exige um esforço elevado e é suscetível a erros. Durante a montagem dos testes os instrutores sentem dificuldades para realizar a definição de critérios avaliativos devido à ausência de uma base de testes centralizada, o que torna o processo subjetivo a cada instrutor. Já o aluno ao realizar a modelagem do teste, um procedimento manual que faz parte do processo de aprendizagem, espera receber o *feedback* avaliativo sobre o diagrama que está sendo gerado. A avaliação é realizada pelo instrutor que visita manualmente cada diagrama, que resulta geralmente em uma nota fechada e sem categorização. Não é disponibilizado um relatório apropriado explicando os descontos na nota da modelagem do aluno, assim, o aluno não consegue perceber ou identificar de modo condizente onde estão suas dificuldades na modelagem. Há ausência de um relatório descrevendo de modo claro e com critérios bem definidos a avaliação, dificultando assim o processo de aprendizagem do aluno.

O processo de avaliar um modelo de um aluno é uma tarefa tipicamente fácil para um instrutor, porém, ao realizar o mesmo processo repetidamente para uma turma de alunos torna o processo exaustivo e mais propenso a erros. A engenharia de software nos últimos anos vem sentindo os impactos da alta demanda de trabalhadores no mercado de trabalho, vive-se uma época em que há necessidade de profissionais capacitados e que saibam interagir com variadas etapas do processo de criação de software. A projeção é que nos EUA haverá até o ano de 2022 três vezes mais vagas para graduados em computação no mercado de trabalho do que profissionais que possam ocupá-las (ADAMS, 2014). Estas projeções estimularam o crescimento do número de procura pelos cursos de computação nos últimos anos (SINGER, 2019).

Os instrutores das disciplinas de computação vem enfrentando uma sobrecarga de trabalho com o crescimento das turmas, e isso já é uma preocupação da comunidade acadêmica há anos (HASKER; ROWE, 2011; HASKER, 2011). Hasker propõem a mais de uma década atrás uma ferramenta capaz de realizar avaliação automatizada de diagramas; estudos utilizando *machine learning* (STIKKOLORUM et al., 2019) e noções sintáticas e semânticas (BIAN; ALAM; KIENZLE, 2019) são outros exemplos das alternativas que vem sendo exploradas.

Este trabalho, portanto, apresenta o ModelGuru, uma abordagem baseada em métricas para

avaliação de diagramas de classes da UML, os quais são regularmente utilizados no ensino de modelagem de software (FARIAS et al., 2018; JÚNIOR; FARIAS; SILVA, 2021). A proposta trata-se de uma ferramenta com o objetivo de auxiliar o processo de aprendizagem, que utiliza de métricas de design para classes para definição de notas de modelos criados por estudantes. Foi desenvolvido um protótipo da abordagem utilizando uma arquitetura tradicional *front-end* e *back-end* onde a partir do carregamento de dois modelos: (1) o modelo de referência do instrutor (gabarito); e (2) a modelo desenvolvido pelo aluno como resposta, a ferramenta calcula uma nota ao aluno baseado em 30 métricas categorizadas como "Tamanho", "Herança", "Acoplamento", "Complexidade" e "Diagrama". O protótipo foi testado e avaliado através de uma pesquisa onde os participantes utilizaram a ferramenta que foi disponibilizada em endereço web [3] e depois responderam um questionário com o Modelo de Aceitação de Tecnologia (TAM) (MARANGUNIć; GRANIć, 2014). Os resultados alcançados indicam que existe necessidade e utilidade de uma ferramenta capaz de auxiliar os instrutores em uma avaliação com categorias e homogênea de modelos; este trabalho então introduz uma possibilidade para ser melhorado o processo de aprendizagem na modelagem de software.

O trabalho é estruturado da seguinte forma: Seção 2 introduz o referencial teórico e a motivação do estudo; Seção (3) onde é apresentado as referências já existentes na literatura com os trabalhos relacionados ao tema; Seção (4) que apresentada a ferramenta proposta e como é composto o processo utilizado para realizar o cálculo da nota dos modelos; Seção (5) onde é demonstrado a aplicação do processo de cálculo em um estudo de caso com modelos de soluções de três alunos: A, B e C, e a seção é finalizada com os resultados de um questionário aplicado sobre os padrões do Modelo de Aceitação de Tecnologia (TAM); e pôr fim a Seção (6) que conclui o trabalho.

2 REFERENCIAL TEÓRICO E MOTIVAÇÃO

Esta seção introduz o referencial teórico de conceitos importantes para o entendimento do trabalho, sendo eles o diagrama de classes da UML (Seção 2.1) e as métricas do SDMetrics (Seção 2.2). Também é apresentada nesta seção a motivação para a elaboração deste trabalho (Seção 2.3).

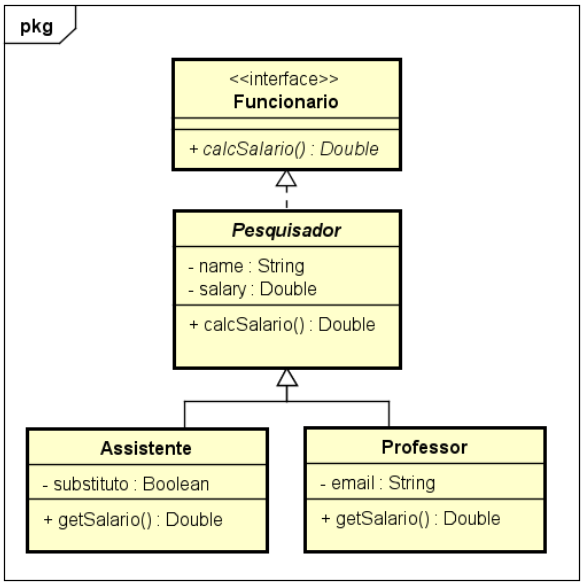
2.1 Diagrama de Classes da UML

A Figura 1 apresenta um exemplo ilustrativo de diagrama de classes. O diagrama de classes é um dos diagramas estruturais da UML (RUMBAUGH; JACOBSON; BOOCH, 2004), são peças fundamentais da modelagem de software e são utilizados para modelar os componentes que formam um sistema. Durante as etapas do desenvolvimento de software é possível converter modelos de diagramas de classes em código, assim como o processo inverso também é válido.

[3] Protótipo ModelGuru: http://modelguru.snotra.com.br/

O diagrama de classes é utilizado na documentação dos sistemas e deve, geralmente, servir como base para o desenvolvimento do software. O principal componente do diagrama de classes é a classe. A Figura 1 apresenta quatro classes, sendo elas: “Funcionario”, “Pesquisador”, “Assistente” e “Professor”. A classe é representada por uma caixa retangular dividida internamente em três compartimentos: o primeiro contém o nome da classe e seu estereótipo (caso exista); no segundo os atributos; e o terceiro os métodos. O diagrama de classes foi escolhido devido a seu grau de utilização tanto no meio educacional quanto na indústria, como apontam estudos realizados (FARIAS et al., 2018; JÚNIOR; FARIAS; SILVA, 2021).

Figura 1 – Exemplo de diagrama de classes da UML



Fonte: Elaborado pelo autor

As classes se conectam através de relacionamentos, tais como Associação, Agregação, Composição, Generalização (Herança) e Realização, cada um com uma característica e uso específico. No exemplo da Figura 1 as classes “Assistente” e “Professor” são generalizações da classe abstrata “Pesquisador”, que por sua vez realiza a interface “Funcionario”. Como na orientação a objeto, os atributos e métodos das classes pais estão disponíveis nas classes que as herdam, salvo casos em que estes forem definidos como privados em sua classe de origem.

2.2 SDMetrics e Métricas

Métricas são um conjunto de medidas quantificáveis independentes entre si que possuem em geral o objetivo de fornecer resultados numéricos de qualquer tipo de processo. O SDMetrics (WüST, 2021) é uma ferramenta que realiza a geração de métricas sobre o diagrama de classes da UML. A ferramenta contabiliza diversas métricas de classe que estão separadas em categorias como: Complexidade, Acoplamento, Herança, Tamanho e Diagrama, e fornece uma lista de métricas de design para nomeação de entidades, estilo, completude entre outros.

Tabela 1 – Métricas de classes do SDMetrics

Categoria	Métrica	Descrição
Tamanho	NumAttr	O número de atributos da classe.
	NumOps	O número de operações em uma classe.
	NumPubOps	O número de operações públicas em uma classe.
	Setters	O número de operações com um nome começando com 'set'.
	Getters	O número de operações com um nome começando com 'get', 'is' ou 'has'.
	Nesting	O nível de aninhamento da classe (para classes internas).
Herança	IFImpl	O número de interfaces que a classe implementa.
	NOC	O número de filhos da classe (generalização UML).
	NumDesc	O número de descendentes da classe (generalização UML).
	NumAnc	O número de ancestrais da classe.
	DIT	A profundidade da classe na hierarquia de herança.
	CLD	Profundidade de classe até a folha.
	OpsInh	O número de operações herdadas.
	AttrInh	O número de atributos herdados.
Acoplamento	Dep_Out	O número de elementos dos quais esta classe depende.
	Dep_In	O número de elementos que dependem desta classe.
	NumAssEl_ssc	O número de elementos associados no mesmo escopo (namespace) que a classe.
	NumAssEl_sb	O número de elementos associados no mesmo ramo de escopo da classe.
	NumAssEl_nsb	O número de elementos associados que não estão no mesmo ramo de escopo da classe.
	EC_Attr	O número de vezes que a classe é usada externamente como tipo de atributo.
	IC_Attr	O número de atributos na classe que têm outra classe ou interface como seu tipo.
	EC_Par	O número de vezes que a classe é usada externamente como tipo de parâmetro.
	IC_Par	O número de parâmetros na classe que têm outra classe ou interface como seu tipo.
	MsgRecv	O número de mensagens recebidas.
	MsgSent	O número de mensagens enviadas.
Complexidade	LLInst	O número de 'lifelines' que representam uma propriedade da qual esta classe é o tipo.
	InstSpec	O número de especificações de instância em que a classe é um classificador.
	Connectors	O número de conectores pertencentes à classe.
	MsgSelf	O número de mensagens enviadas para instâncias da mesma classe.
Diagrama	Diags	O número de vezes que a classe aparece em um diagrama.

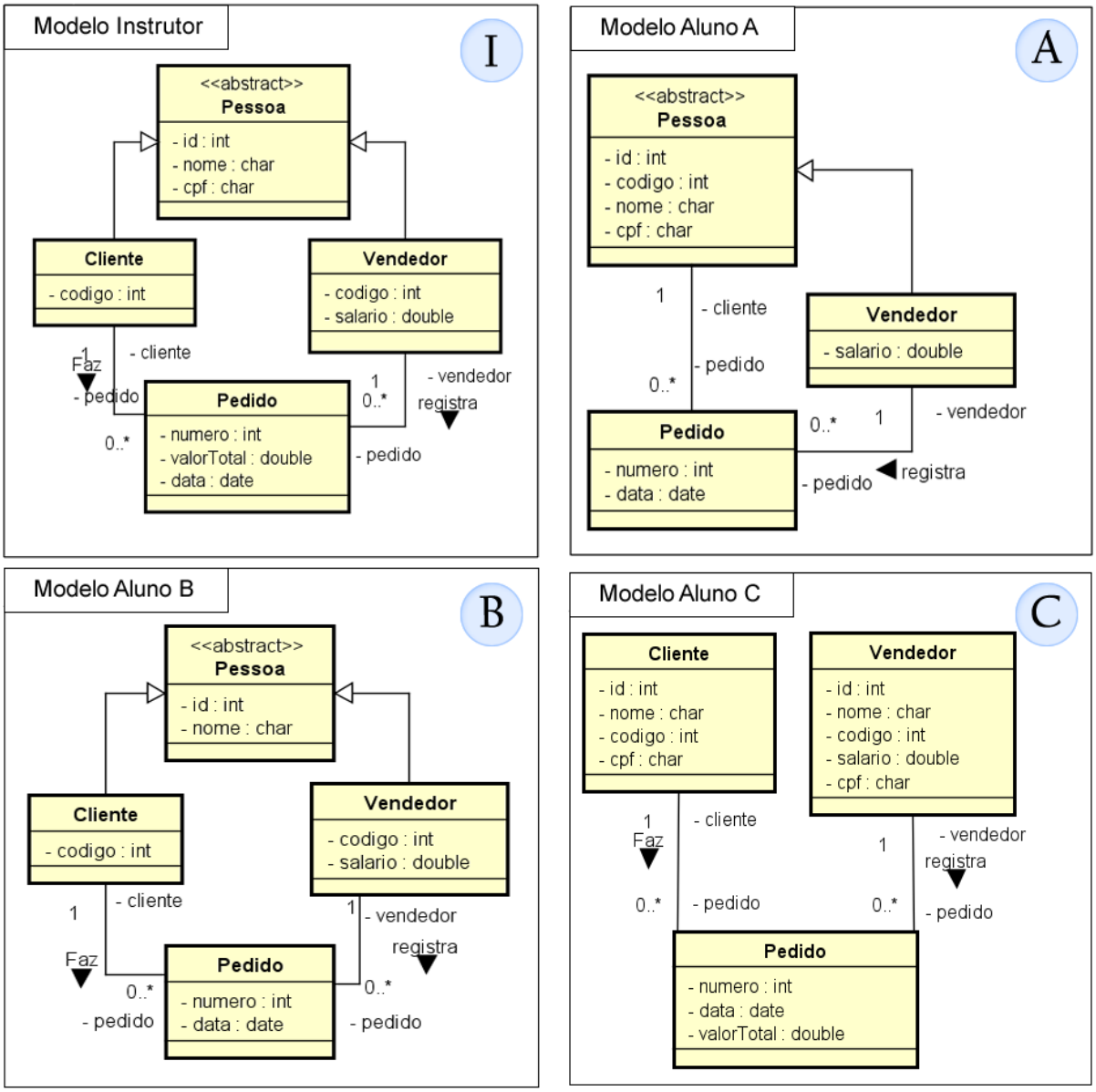
Fonte: Adaptado de (WüST, 2021)

É apresentado na Tabela 1 as 30 métricas de classe do SDMetrics utilizadas neste trabalho. A métrica "NumAttr", em exemplo, corresponde ao número de atributos da classe, já a métrica "Setters" corresponde ao número de métodos (também chamados de operações ou funções) que tem prefixo "set", que na orientação a objeto representa uma convenção para métodos que inserem valores a atributos da classe; ambas estas métricas citadas pertencem a categoria "Tamanho". Já em "Acoplamento", por exemplo, à métricas como "Dep_Out" que indica o número de elementos de que esta classe depende, e a "Dep_In" que indica o número de elementos que dependem desta classe.

2.3 Motivação

A Figura 2 apresenta quatro exemplos de diagrama de classes da UML. Os exemplos representam uma atividade avaliativa aplicado por um instrutor a três alunos: Aluno A, Aluno B e Aluno C. O modelo do instrutor (Figura 2.I) consiste do gabarito para o exercício e contém quatro classes, sendo elas: "Pedido", que possui os atributos "numero", "data" e "valorTotal"; "Cliente" com apenas o atributo "codigo"; "Vendedor" que também possui o atributo "codigo" e o atributo "salario"; e pôr fim a classe abstrata "Pessoa" com atributos "id", "nome" e "cpf".

Figura 2 – Exemplos de diagrama de classes



Fonte: Elaborado pelo autor

A classe “Pedido” do diagrama possui associação com “Cliente” e “Vendedor”, e estas últimas são classes generalizadas da classe abstrata “Pessoa”.

O modelo de solução apresentado pelo Aluno A (Figura 2.A) possui divergências visíveis em relação ao do instrutor. A classe “Cliente” não está presente e o atributo “codigo” foi colocado diretamente na classe abstrata “Pessoa”, o diagrama apresenta uma solução incorreta pois observando o modelo de referência é perceptível que o “codigo” de “Cliente” e “Vendedor” são diferentes, e o Aluno A ao colocá-lo na superclasse o torna igual para as duas subclasses. Há também a ausência do atributo “valorTotal” na classe “Pedido”. O modelo do Aluno B (Figura 2.B) por outro lado, não possui muitas divergências, apenas a ausência do atributo “cpf” na classe “Pessoa” e do atributo “valorTotal” na classe “Pedido”, os demais atributos e relacionamentos estão corretos. O Aluno C (Figura 2.C) apresentou uma solução com bastante divergência ao modelo do instrutor, o aspecto principal é a falta da classe abstrata “Pessoa”, tendo todos os atributos que deveriam pertencer a ela colocados de modo repetido nas classes “Cliente” e “Vendedor”.

Observado os três modelos dos alunos e o gabarito do instrutor, nota-se que existe uma grande variedade possível de respostas, o que resulta em dificuldades para gerenciar os critérios de notas no momento da avaliação e assim permitir que o instrutor a mantenha justa para todos os alunos. Então, algumas questões surgem: (1) como saber qual seria o valor justo de desconto que o Aluno A deveria receber por não ter colocado a classe “Cliente”?; (2) o Aluno B que apenas esqueceu de inserir dois atributos deve receber qual desconto?; e (3) o Aluno C que não criou a generalização necessária e adicionou os atributos de forma repetida deve ser mais penalizado?. Como e de que forma então o instrutor pode realizar uma avaliação justa para todos?

Avaliar os diagramas utilizando critérios objetivos e gerar um feedback rápido vem sendo explorado por estudos anteriores (HASKER, 2011; HASKER; ROWE, 2011; BIAN; ALAM; KIENZLE, 2019). Não possuir uma forma automatizada de avaliação de diagramas faz com que os instrutores tendam a criar exercícios com modelos extremamente simples, para que assim se tenha tempo e a correção possa ser viável, o que causa uma limitação no ensino. A ausência de uma abordagem que contribua positivamente para o processo de aprendizagem na avaliação de modelos, utilizando de critérios e que retorne ao aluno um *feedback* construtivo é a motivação central para elaboração deste trabalho.

3 SELEÇÃO DOS TRABALHOS

A seguir é apresentado os trabalhos relacionados ao tema que foram selecionados durante a revisão da literatura.

3.1 Trabalhos Relacionados

Para a seleção dos trabalhos relacionados foram realizadas duas etapas: (1) foi adicionado o trabalho de Farias e Silva (FARIAS; SILVA, 2020) considerado como referência principal junto com seus trabalhos referenciados; (2) foi definida a *STRING* de pesquisa:

- ((UML OR "unified modeling language" OR "software modeling" OR "class diagram" OR "modelling language") AND (grading OR evaluation OR "automated grading" OR "students assessment"))

A *STRING* foi executada nos repositórios digitais da *IEEE* e *ACM Digital Library*, e conforme orienta Dyba el tal. (DYBA; DINGSOYR; HANSSEN, 2007), Kitchenham e Brereton (KITCHENHAM; BRERETON, 2013), também foi selecionada a base de indexação *Scopus*.

Ao final do processo de pesquisa foram reunidos um total de 2403 artigos que foram revisados e classificados de acordo com seu título, resumo, palavras chaves e área de atuação, resultado em 8 trabalhos selecionados.

3.2 Análise dos Trabalhos Relacionados

(FARIAS; SILVA, 2020). Apresentam o UMLGrade, uma proposta de ferramenta que deve atender a demanda de avaliação de modelos UML através de aspectos semânticos, sintáticos, regras de design, legibilidade e princípios de orientação a objetos. São especificadas as etapas que devem ser seguidas no processo de avaliação que são elas: Preparação do teste, definição de critérios, execução do teste, avaliação do UML e relatório de entrega, o trabalho porém não realizou implementação de nenhuma parte do sistema proposto.

(BIAN; ALAM; KIENZLE, 2019). Apresentam uma solução para a avaliação de modelos UML através de um algoritmo de classificação que utiliza correspondência semântica, sintática e estrutural. O trabalho se preocupa com a questão de redistribuição de pesos pós análise permitindo o instrutor de realizar ajustes necessários quando identifica algum padrão de erros ou acertos. O algoritmo classifica e coloca as notas de cada elemento do diagrama separadamente em um metamodelo criado sobre o modelo, o algoritmo foi testado em uma tarefa na qual foi constatado uma diferença média 14% na nota em relação a correção dos mesmos modelos por um professor. O trabalho não menciona a utilização de métricas para avaliação e o algoritmo de classificação desenvolvido isoladamente não pode ser considerado exatamente uma aplicação.

(STIKKOLORUM et al., 2019). Propõem a utilização de aprendizagem de máquina para realizar a avaliação automatizada de modelos UML. Foi realizado um estudo exploratório onde a abordagem proposta foi testada sobre dois experimentos classificando níveis diferentes de graduação, os resultados mostraram que a classificação não foi precisa o suficiente, o melhor dos casos que utilizou uma graduação de 5 pontos alcançou uma precisão de 69,42%, não sendo ideal para uma avaliação automatizada de nota, mas pode ser utilizada como indicadora da qualidade do modelo.

(**VESIN et al., 2018**). O trabalho apresenta a proposta de integração de diferentes ferramentas já existentes utilizadas no ensino da engenharia de software com objetivo de possibilitar mais agilidade no processo de avaliação. A arquitetura tem foco no ambiente web, porém para que o objetivo proposto seja alcançado é necessário de investimentos e esforços na elaboração das funcionalidades ausentes nas ferramentas existentes e que são essenciais para o processo.

(**HASKER; ROWE, 2011**). É apresentada a ferramenta UMLint que tem como objetivo a identificação de erros comuns em diagramas UML. A ferramenta avalia os modelos de diagramas inseridos pelos alunos e retorna um feedback adequado relacionando e indicando os erros cometidos na modelagem. A ferramenta não possui características de avaliação comparativa entre modelos.

(**HASKER, 2011**). Utilizando dos recursos do UMLint na identificação de falhas comuns, Hasker criou a ferramenta UMLGrader que tem como objetivo realizar a análise comparativa de um modelo de referência do instrutor com o modelo criado pelo aluno, apresentando o feedback de erros do UMLint e uma resposta relacionada a comparação dos modelos, indicando ausência de componentes. Para garantir maior eficácia é fornecido aos alunos uma tarefa simples que diminui em muito o espaço da solução tornando a ferramenta um pouco limitada para o ensino de modelagem de software.

(**COFFEY, 2014**), Apresentam uma técnica de avaliação utilizando métricas conforme orientações de (YI; WU; GAN, 2004). Estabelecem pesos para várias características dos diagramas que após a comparação entre modelos quanto menor for os valores resultantes mais semelhantes estão os modelos. Porém a técnica é apresentada sobre pequenos projetos de software e a avaliação é realizada de forma manual.

(**CANAL; FARIAS; GONCALES, 2018**). Descrevem um algoritmo capaz de medir a distância entre diagramas de sequência, esta analise possibilita observar quanto um modelo se difere de outro, para a execução é utilizado o algoritmo de Levenshtein (MILLER; VANDOME; MCBREWSTER, 2009) que calcula qual o menor número possível de operações para se transformar uma *String* em outra. Para a execução foi utilizado a ferramenta MoCoTo (FARIAS et al., 2015), Portanto não foi identificado a presença de uma aplicação ou de um processo de avaliação flexível.

3.3 Análise Comparativa e Oportunidade de Pesquisa

Para realizar a análise comparativa dos trabalhos relacionados foram definidos seis Critérios de Comparação (CC) que caracterizam a oportunidade de pesquisa e desenvolvimento para o trabalho proposto.

Abaixo são descritos os critérios de comparação:

- **Múltiplas Categorias de Avaliação (CC1):** Capacidade de avaliação dos modelos em critérios como Complexidade, Herança, Acoplamento e Tamanho. A comparação baseada em critérios já foi utilizada em outros trabalhos (RUBERT; FARIAS, 2022; JÚNIOR;

FARIAS, 2021; MENZEN; FARIAS; BISCHOFF, 2021) os quais, demonstraram sua utilidade.

- **Contexto Educacional (CC2):** O objetivo do trabalho deve estar interessado a fins educacionais e de aprendizagem e não diretamente a indústria.
- **Métricas e Comparação (CC3):** Realiza a comparação de métricas para realizar a avaliação de modelos.
- **Flexibilidade de Avaliação (CC4):** O trabalho apresenta mecanismos que flexibilizam a avaliação através de distribuição de pesos.
- **Feedback Eficiente (CC5):** Apresenta um *feedback* rápido construtivo ao aluno que auxilia no processo de aprendizagem.
- **Aplicação (CC6):** Foi desenvolvida uma aplicação real para apresentar a proposta do trabalho.

Tabela 2 – Análise comparativa dos trabalhos relacionados selecionados

Trabalho Relacionado	**Critério de Comparação**					
	CC1	**CC2**	**CC3**	**CC4**	**CC5**	**CC6**
Trabalho Proposto	●	●	●	●	●	●
(FARIAS; SILVA, 2020)	◐	●	◐	○	●	○
(BIAN; ALAM; KIENZLE, 2019)	◐	●	◐	◐	●	◐
(STIKKOLORUM et al., 2019)	○	●	○	○	◐	◐
(VESIN et al., 2018)	○	●	○	○	●	◐
(HASKER; ROWE, 2011)	○	●	○	○	●	●
(HASKER, 2011)	○	●	◐	○	●	●
(COFFEY, 2014)	◐	●	●	○	○	○
(CANAL; FARIAS; GONCALES, 2018)	○	◐	○	○	◐	◐

Legenda: ● Atende Completamente | ◐ Atende Parcialmente | ○ Não Atende

Oportunidades de pesquisa. A análise dos trabalhos relacionados é demonstrada na Tabela 2 onde é evidenciado que: (1) apenas o trabalho proposto contempla todos os critérios de comparação; (2) apenas o trabalho de Coffey (COFFEY, 2014) utiliza completamente o uso de métricas porém é aplicado de forma manual; (3) nenhum trabalho explora a possibilidade de distribuição de pesos pelos critérios; e (4) dos poucos trabalhos que desenvolveram uma aplicação real nenhum deles ainda possui a ferramenta disponível para uso e/ou deram sequência no projeto. Portanto foi identificado a oportunidade de pesquisa: desenvolvimento de uma ferramenta com uma abordagem baseada em métricas para a avaliação de modelos que possibilite a distribuição de pesos e readequação de notas auxiliando o processo de aprendizagem na modelagem de software. Oportunidade que será explorada nas seções seguintes.

4 ABORDAGEM PROPOSTA

Nesta seção é apresentada a abordagem proposta ModelGuru.

4.1 Visão Geral da Abordagem Proposta

A Figura 3 ilustra os passos do processo utilizado no ModelGuru para realização da tarefa de avaliação de modelo. Será descrito a seguir cada um destes passos:

- **Passo 1: Montagem do Teste.** Na primeira etapa, o instrutor fornece para a ferramenta os requisitos da modelagem informando-os através de um formulário de texto. Após o instrutor fornecer os requisitos ele deverá submeter 1 ou N modelos de soluções possíveis para o exercício considerando que estes são os modelos de referência (gabarito) para a avaliação das soluções apresentadas pelos alunos. Assim, o instrutor finaliza a etapa e prossegue para o próximo passo.

- **Passo 2: Definição de Critérios.** Com as regras e os modelos de resposta do teste estabelecidos agora o instrutor deve informar o peso de cada critério de avaliação, que são os pesos correspondentes para as métricas de: Complexidade, Herança, Acoplamento, Diagrama e Tamanho. Deve ser informado um percentual correspondente para cada critério. Esta etapa pode ser ajustada novamente pelo instrutor a qualquer momento, inclusive após os alunos já terem executado o teste, com o objetivo de adaptar a correção e as notas de acordo com algum padrão ou situação identificada pelo instrutor.

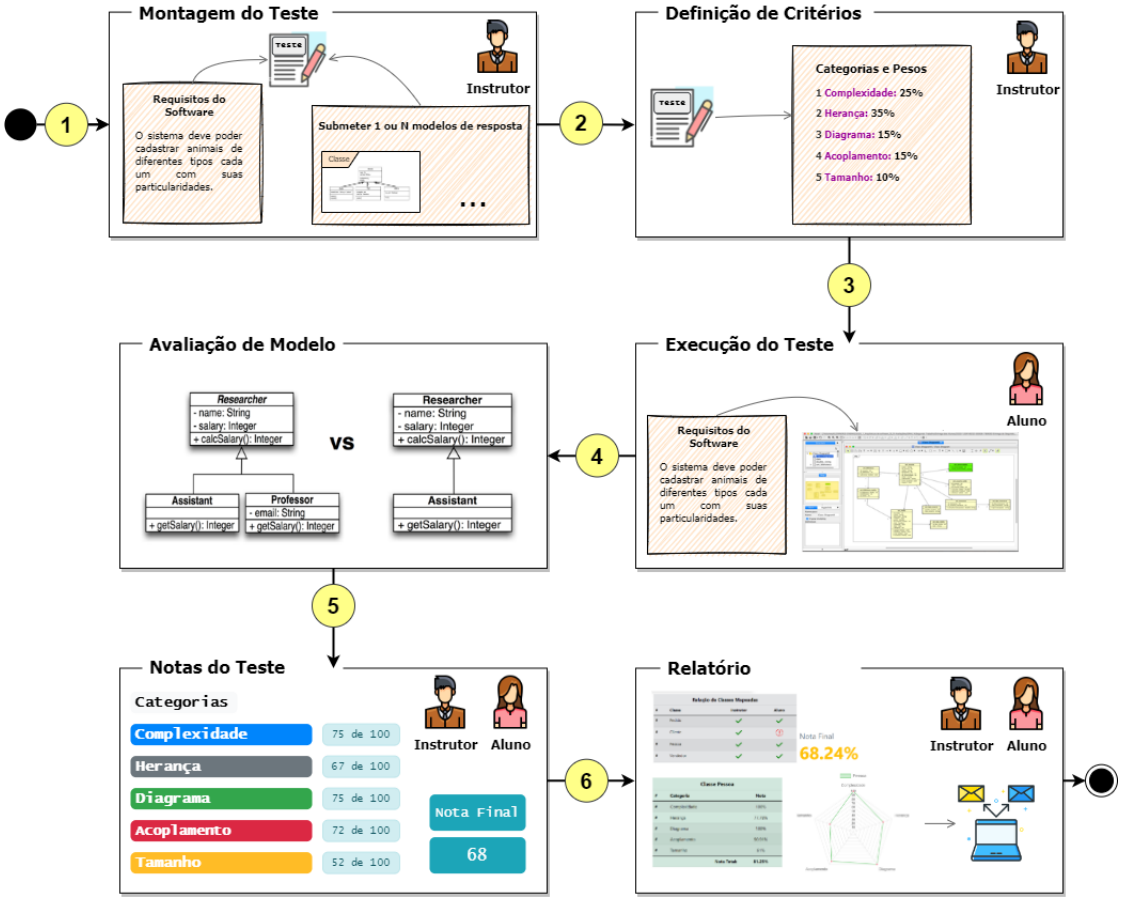
- **Passo 3: Execução do Teste.** Nesta etapa o aluno executa a solução do exercício elaborado pelo instrutor. Poderá ser utilizado pelo aluno a ferramenta que for de sua preferência para criar um modelo (diagrama de classes) que responda os requisitos estabelecidos pelo instrutor, mas deve estar atento aos padrões de entrada definidos pelo ModelGuru para que a resposta possa ser submetida no sistema. Será descrito mais sobre estes padrões de entrada na Seção 4.2. Após o aluno submeter o modelo imediatamente a ferramenta inicia o processo de avaliação descrita no próximo passo.

- **Passo 4: Avaliação de Modelo.** Nesta etapa com o modelo de referência do instrutor e o modelo de solução submetido pelo aluno o sistema ModelGuru realiza a avaliação e comparação das métricas para identificar o percentual de acertos do aluno. A técnica de avaliação será mais bem descrita na seção subsequente (4.2).

- **Passo 5: Notas do Teste.** Com os valores obtidos na etapa anterior o sistema ModelGuru apresenta agora a tela de notas do teste realizado. É mostrado o percentual de acerto para cada categoria e métricas, esta nota final é apresentada ao aluno e ao instrutor ao consultar o resultado do teste. As notas de todos os alunos que executarem o teste podem ser alvas na base de dados para que possam ser acessadas individualmente, ou serem analisadas de modo coletivo futuramente.

- **Passo 6: Relatório.** Através de relatórios gráficos será realizado a demonstração dos dados. Todos os resultados dos testes feitos pelos alunos podem ser monitorados através

Figura 3 – Visão geral do ModelGuru



Fonte: Elaborado pelo autor

de gráficos e painéis pelo instrutor possibilitando uma análise mais analítica sobre o progresso do conhecimento e aprendizado de uma turma ou grupo de alunos. A evolução desta etapa caracteriza o conceito de observabilidade onde de posse dos dados de todos os testes seria possível uma série de análises de desempenho dos alunos. Através destas informações o instrutor poderia também realizar ajustes de pesos nos critérios com o objetivo de diminuir penalidades em situações que forem identificadas falhas comuns entre os alunos, o que pode sinalizar um ponto que deve ser mais bem abordado pelo instrutor em aula.

4.2 Processo e Algoritmos Propostos

Nesta seção é explicado o processo proposto e os algoritmos propostos para executar o cálculo da nota na avaliação de modelos. É importante ser introduzido primeiro o Algoritmo 1 que representa o *main* (fluxo principal) da abordagem proposta.

O Algoritmo 1 apresenta na entrada da função principal dois parâmetros, sendo eles o *models* que simboliza os modelos do instrutor e do aluno, e o *configs* que representa as configu-

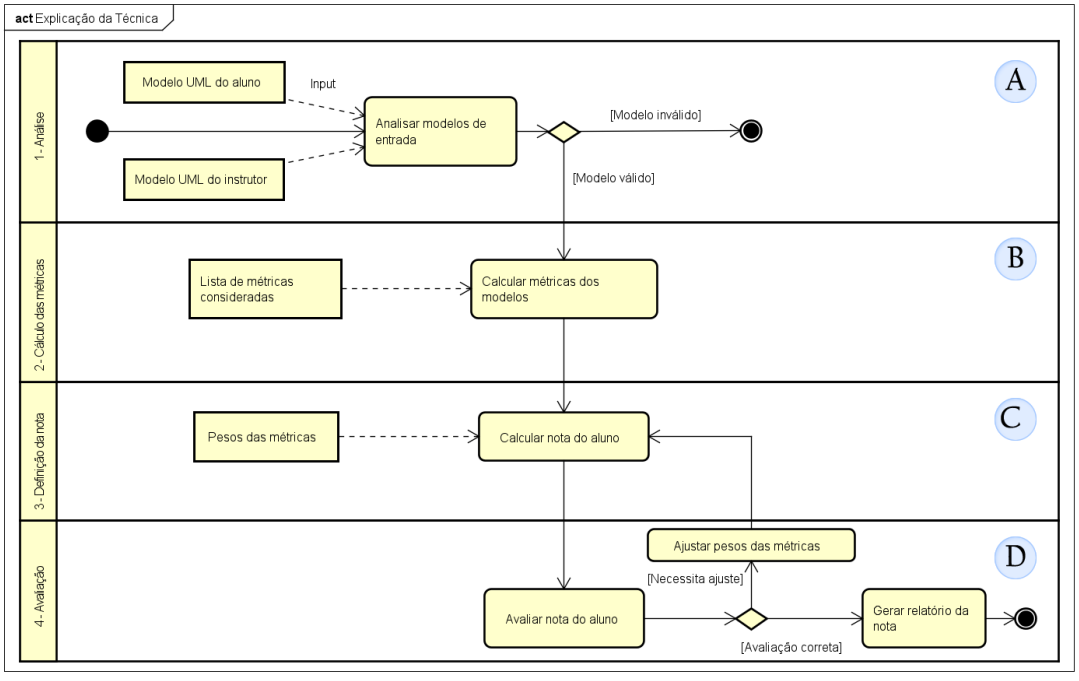
rações para o processo que podem ser editadas antes da avaliação. No corpo da função existe uma sequência de chamadas de métodos que formam o processo de avaliação, sendo eles na ordem: *buildProcessConfigs(configs)*, *validateModels(models)*, *loadMetrics(models, configs)*, *calculateResult(metrics, configs)* e *generateReport(result, configs)*. Cada um será explicado separadamente a seguir juntamente com a explicação das etapas da proposta.

Algorithm 1: main (fluxo principal)

```
1  function main (models, configs)
2      configs ← buildProcessConfigs(configs);
3      validModels ← validateModels(models);
4      if validModels = false then
5          return;
6      end
7      metrics ← loadMetrics(models, configs);
8      result ← calculateResult(metrics, configs);
9      report ← generateReport(result, configs);
10     return report;
11 end
```

O diagrama de atividades na Figura 4 apresenta os passos descritos na Seção 4.1 agrupados dentro de quatro etapas (raias do diagrama) sendo elas: Análise, Cálculo das métricas, Definição da nota e Avaliação. A seguir será demonstrado a explicação de cada uma destas etapas e seus algoritmos correspondentes.

Figura 4 – Diagrama de atividades do processo



Fonte: Elaborado pelo autor

Análise. A etapa de Análise inicia o processo. A responsabilidade inclui receber os modelos de entrada do instrutor e do aluno e analisar a validade destes modelos. Ao ser identificado algum modelo inválido o processo da técnica é abortado imediatamente, caso contrário avança para a próxima etapa. Esta etapa está descrita na Figura 4. A e são propostos os Algoritmos 2 e 3.

O Algoritmo 2 é responsável por carregar as configurações padrões definidas estaticamente na aplicação e sobrescrevê-las com as configurações feitas pelo instrutor na entrada do ModelGuru. Dentro das configurações estão regras e parâmetros que conduzem a forma como a avaliação deve ser realizada, as configurações utilizadas atualmente são os pesos das métricas. Na sequência as *configs* devem ser retornadas ao fluxo principal e então é executado o Algoritmo 3.

Algorithm 2: buildProcessConfigs

```
1  function buildProcessConfigs (configs)
2      defaultConfigs ← getDefaultConfigs();
3      configs ← mergeConfigs(defaultConfigs, configs);
4      return configs;
5  end
```

O Algoritmo 3 é responsável pela entrada de arquivos. A entrada de modelos é através de um formato de interoperabilidade, no caso da UML seria o UML eXchange Format (UXF) (SUZUKI; YAMAMOTO, 1999) ou XML Metadata Interchange (XMI) (XMI®, 2015). O XMI foi escolhido por ser o padrão atual, permitindo tanto o instrutor quanto ao aluno utilizar de uma grande variedade de softwares de modelagem que são adaptados para este formato de exportação de modelos. Com o arquivo carregado é verificado se o formato é XMI e depois se seu tamanho é menor que o estipulado em nossa constante "MAX_SIZE", uma medida de segurança para evitar que grandes arquivos travem o processo de avaliação.

Algorithm 3: validateModels

```
1  function validateModels (models)
2      size ← models.length;
3      while size > 0 do
4          check ← isXMI(models[size − 1]);
5          check ← check and smallerThan(models[size − 1], MAX_SIZE);
6          if check = false then
7              return false;
8          end
9          size ← size − 1;
10     end
11     return true;
12 end
```

Após a execução do Algoritmo 3 o processo retoma o fluxo principal onde é verificado se os

modelos foram validados com sucesso, caso algum dos modelos seja sinalizado como inválido o processo é abortado imediatamente.

Cálculo das Métricas. Nesta etapa descrita na Figura 4.B ocorre a geração das métricas dos modelos carregados, tanto do aluno quanto do instrutor. O Algoritmo 4 é responsável por este processo. Na linha 2 é criado o objeto *metrics*, que após o laço de repetição *while* é preenchido com as métricas de todos os modelos através da linha 5 onde é chamado o SDMetrics. Na linha 9 é chamado a função *classMatching* que recebe o objeto de métricas e a informação de qual implementação da interface de comparação será usada. A interface de comparação define regras para realizar as combinações entre as classes no modelo do instrutor e o modelo do aluno, deste modo permitimos evoluir de modo mais prático o mecanismo de comparação sem trazer danos a estrutura. O protótipo usa atualmente a implementação básica que tenta realizar a combinação através dos nomes mais próximos entre os modelos do aluno e do instrutor, considerando assim como pares válidos durante a etapa de seleção.

Algorithm 4: loadMetrics

```
1  function loadMetrics (models, configs)
2      metrics ← getMetricsStart();
3      size ← models.length;
4      while size > 0 do
5          metricsObject ← executeSDMetrics(models[size − 1], configs);
6          addMetrics(metrics, metricsObject, models[size − 1]);
7          size ← size − 1;
8      end
9      classMatching(metrics, configs.getMatchImplementation());
10     return metrics;
11 end
```

Definição da nota. Na definição das notas (Figura 4.C) é recebido como entrada as métricas recém calculadas pela etapa anterior e os pesos das métricas presentes nas *configs*. Este processo está descrito no Algoritmo 5 e é a parte central do processo de cálculo da nota com base nas métricas. As variáveis *m*, *a* e *i* representam o índice atual da métrica, do modelo do aluno e do modelo do instrutor respectivamente, *Pm* é o peso da métrica extraída das *configs*, *Im* e *Am* são os valores das métricas calculadas pelo SDMetrics para o modelo do instrutor e aluno, *ABSm* é o valora absoluto da subtração da métrica do instrutor pela métrica do aluno, *final* é a variável que representa a nota final calculada para a métrica atual e *Dm* representa o percentual de desconto que a nota desta métrica deve receber após o cálculo. Após a execução destes cálculos os valores resultantes são colocados no objeto *result* que ao final é retornado para a função principal.

Avaliação. A Avaliação é a última etapa e é aqui que as notas calculadas do modelo do aluno são disponibilizadas para que o instrutor possa visualizar, neste momento ele pode escolher por realizar um ajuste nos pesos das métricas após identificar algum problema ou critério avaliativo que necessite de alguma correção. Realizando ajuste de peso o processo retorna para a etapa

Algorithm 5: calculateResult

```
1  function calculateResult (metrics, configs)
       /* m = índice atual do array de métricas                    */
       /* a = índice atual do modelo de aluno                      */
       /* i = índice atual do modelo do instrutor                  */
2      Pm ← configs.getMetrics()[m];
3      Im ← metrics[i].getMetrics()[m];
4      Am ← metrics[a].getMetrics()[m];
5      ABSm ← |Im − Am|;
6      if ABSm = 0 then
7          final ← Pm;
8      else
9          if Im = 0 then
10             final ← 0;
11         else
12             Dm ← ((ABSm ∗ 100)/(Im > Am ? Im : Am));
13             final ← (Pm − ((Pm ∗ Dm)/100));
14         end
15     end
16     setProportion(final, configs);
17     addResult(result, metrics[a], final);
18     return result;
19 end
```

anterior de “Definição da nota” e o processo é calculado novamente agora sobre os novos parâmetros de pesos das métricas. Quando a avaliação está correta o sistema libera a geração do relatório da nota onde é descrito todos os critérios e pode ser visualizado quais que influenciaram desconto de nota. Ao final o aluno então recebe acesso a este relatório finalizando o fluxo. Este processo é realizado na linha 9 do Algoritmo 1, e retorna o relatório para apresentação utilizando o objeto de resultados (*result*) recebido do cálculo das notas.

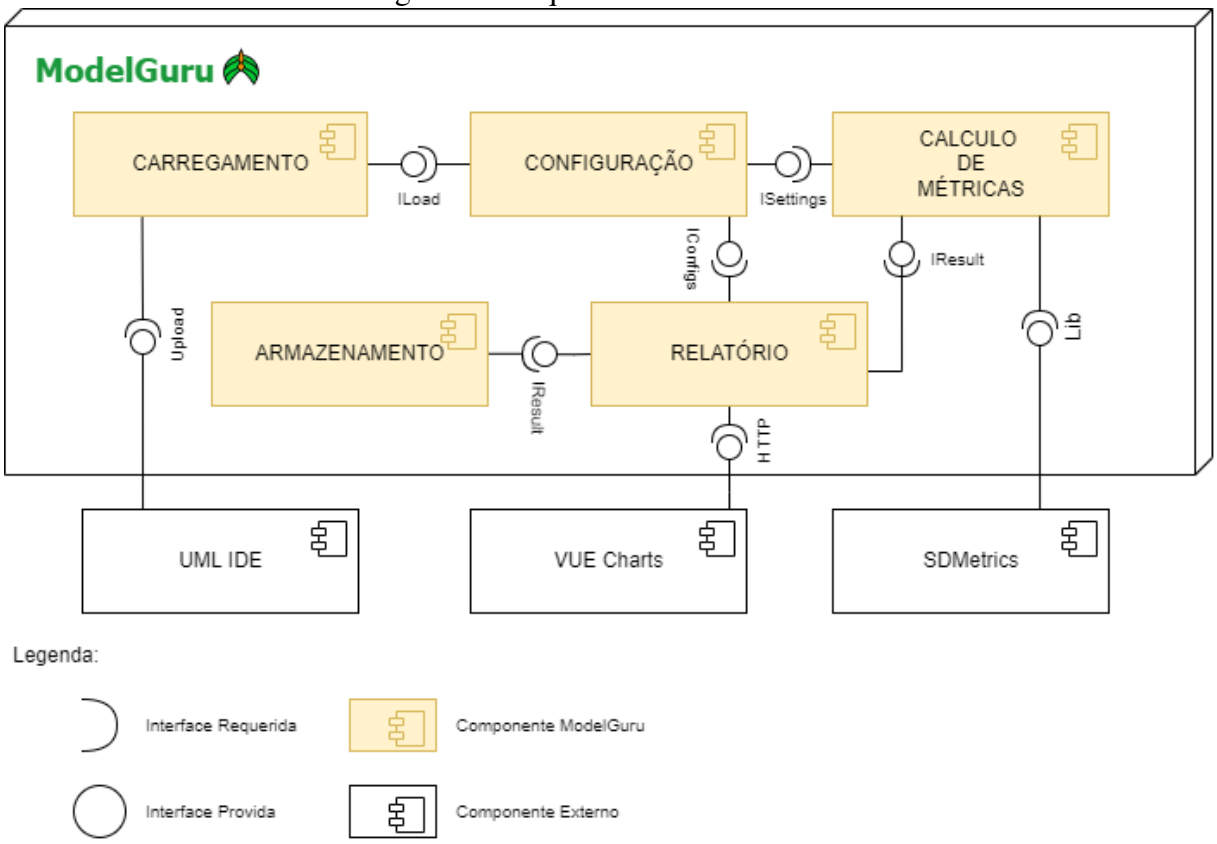
4.3 Arquitetura da Abordagem

A Figura 5 demonstra a arquitetura da ferramenta proposta, onde é observado todos os componentes existentes, sendo eles: CARREGAMENTO, responsável pelo carregamento dos modelos e validação do arquivo XMI, CONFIGURAÇÃO, repensável pela gerencia e montagem dos testes, sendo nele também a definição dos pesos das métricas, CALCULO DE MÉTRICAS, repensável pelo calculo da nota com bases nestas métricas coletadas através do SDMetrics, ARMAZENAMENTO, responsável por guardar e gerir as informações dos cálculos e modelos, e o RELATÓRIO que é repensável por montar as saídas e *feedback* dos resultados dos cálculos para serem apresentados ao instrutor e os alunos.

Há também componentes fora da estrutura do ModelGuru, sendo eles o UML IDE que corresponde as ferramentas que serão utilizadas pelos alunos e instrutores para geração dos

modelos, o VUE Charts, componente do VUE JS usado como ferramenta visual para realizar a plotagem dos dados em diagramas e *dashboards* e o SDMetrics que é utilizado na geração das métricas.

Figura 5 – Arquitetura do ModelGuru



Fonte: Elaborado pelo autor

4.4 Aspectos de Implementação

A implementação do protótipo do ModelGuru[4] esta separada em *front-end* e *back-end* e será descrito agora ambas as camadas separadamente.

Back-end. Na camada de *back-end* onde é realizado o processo e o calculo das métricas foi utilizada a linguagem Java na versão 12, para organização da estrutura e arquitetura foi utilizado o *framework* Spring (JOHNSON et al., 2022) que nos disponibiliza uma série de módulos organizados pelo módulo gerencial Spring Boot, que foi utilizado na versão 2.7.0. Dentre as dependências principais estão o Spring MVC que é responsável pela organização entre as camadas de *Model*, *View* e *Controller*, o Spring WEB que disponibiliza a camada WEB onde podem ser usados *webservices RESTful* e também é disponibilizo o container de aplicação Java Apache Tomcat embutido para facilitar a implantação da aplicação e o Spring Data JPA/JDBC, que fornece os recursos para o uso de banco de dados dentro do ecossistema

[4] Protótipo ModelGuru: http://modelguru.snotra.com.br/

Figura 6 – Tela de Carregamentos dos Modelos no Protótipo Desenvolvido

Fonte: Elaborado pelo autor

que apesar de ainda não estar sendo realizada na versão de protótipo atual já esta programada para começar a ser utilizada nas próximas versões. A etapa de geração das métricas no *back-end* são realizadas utilizando a biblioteca do SDMetrics que foi adicionada ao projeto em formato JAR, após a geração das métricas os cálculos e avaliação são executadas diretamente no código Java.

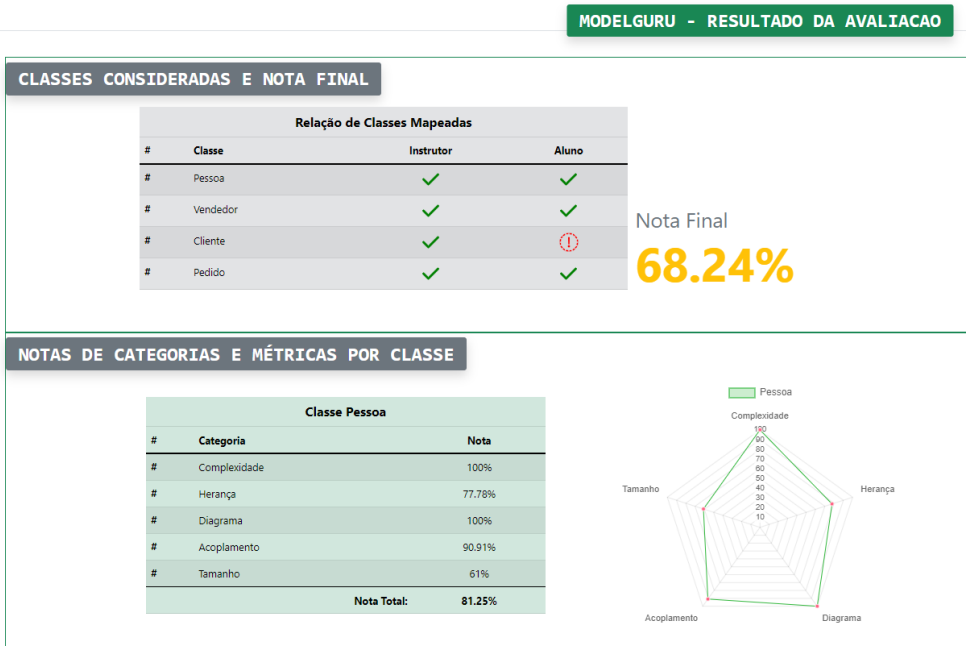
Front-end. A Camada de *front-end* foi desenvolvida utilizando o framework Javascript Vuejs na versão 3 (YOU, 2022). O Vue fornece um ecossistema reativo e componentizado que facilita a organização e o desenvolvimento da camada de apresentação, internamente são utilizados plugins do próprio Vue e de terceiros para conceder ao protótipo alguns recursos desejados, dentre eles a biblioteca Axios para possibilitar chamadas REST e a biblioteca Charts.js para geração de gráficos dos relatórios do processo que é utilizada através do componente vue-charts também adicionado ao projeto.

A comunicação entre a camada de *back-end* e *front-end* é feita através de requisições REST, o fluxo se inicia no *front-end* com a submissão dos modelos pelo website como pode ser observado na Figura 6, após serem enviados ao *back-end* é realizado o processo já exemplificado na Seção 4.2, que por fim retorna para o *front-end* como resposta o relatório da avaliação do modelo que é mostrado para o usuário na forma de gráficos e tabelas que é demonstrado na Figura 7 onde é visualizado na parte superior do relatório, na figura nos mostra a relação de classes consideradas e a nota final, e logo abaixo é possível ser observada a primeira classe avaliada com suas notas categorizadas e demonstradas separadamente, é possível ver logo a direita também o gráfico de radar utilizado do Charts.js.

5 AVALIAÇÃO

Nesta seção será apresentada a avaliação da abordagem proposta através de: (1) Um estudo de caso explicando o processo de calculo; e (2) os resultados da aplicação do questionário TAM ao grupo de participantes da pesquisa.

Figura 7 – Tela de Relatório da Avaliação do Modelo no Protótipo Desenvolvido



Fonte: Elaborado pelo autor

5.1 Estudo de caso

Será demonstrado agora a partir de um exemplo a aplicação da técnica, para isso é considerado os modelos apresentados na Figura 2, onde há o exemplo do instrutor e dos alunos A, B e C respectivamente. Neste exemplo será considerado que as métricas “NumAttr”, “NumOps” e “NumDesc” correspondem a 20% da nota cada uma, totalizando 60%, os demais 40% são divididos igualmente pelas demais métricas.

Como já foi levantado anteriormente na Seção 2.3 é notável que ambas as soluções dos alunos possuem algumas divergências ao modelo do instrutor, e estas diferenças que devem ser identificadas e avaliadas de modo justo de acordo com a aplicação do processo de cálculo.

Ao executar os modelos no SDMetrics será obtido as métricas que estão informadas na tabela 3 onde é observado a métrica “NumAttr”, que mostra que o numero de atributos possui valor 3 na classe “Pessoa” do modelo do instrutor, o que corresponde com o diagrama do instrutor na figura 2, pode-se validar assim que as informações geradas pelo SDMetrics estão condizentes com os modelos.

Agora deve ser calculada o valor absoluto ABSm = | Im - Am |, pegando como exemplo a métrica “NumAttr” considerando a classe “Pessoa”, no modelo do instrutor consta o valor 3, já no modelo do Aluno A esta o valor 4, executando a fórmula para estes valores: ABSm = |3-4| se resulta em 1 absoluto, agora é preciso transformar este valor em um percentual válido chamado “Dm”, neste caso Dm = ((1 * 100)/ 3), resultando em 33,33% de desconto de nota para esta métrica desta classe.

Agora basta multiplicar ’Dm’ pelo peso estabelecido pelo instrutor para esta métrica, como

Tabela 3 – Métricas calculadas para os modelos

Modelo	Instrutor				Aluno A				Aluno B				Aluno C			
Métricas \ Classes	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido
NumAttr	3	2	1	3	4	1	-	2	2	2	1	2	-	5	4	3
NumOps	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
NumPubOps	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Setters	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Getters	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Nesting	0	0	0	0	1	0	-	0	0	0	0	0	-	0	0	0
IFImpl	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
NOC	2	0	0	0	1	0	-	0	2	0	0	0	-	0	0	0
NumDesc	2	0	0	0	1	0	-	0	2	0	0	0	-	0	0	0
NumAnc	0	1	1	0	0	1	-	0	0	1	1	0	-	0	0	0
DIT	0	1	1	0	0	1	-	0	0	1	1	0	-	0	0	0
CLD	1	0	0	0	1	0	-	0	1	0	0	0	-	0	0	0
OpsInh	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
AttrInh	0	3	3	0	0	4	-	0	0	2	2	0	-	0	0	0
Dep_Out	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Dep_In	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
NumAssEl_ssc	0	1	1	2	0	1	-	1	0	1	1	2	-	1	1	2
NumAssEl_sb	0	1	1	2	1	1	-	2	0	1	1	2	-	1	1	2
NumAssEl_nsb	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
EC_Attr	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
IC_Attr	0	0	0	1	0	0	-	1	0	0	0	1	-	0	0	1
EC_Par	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
IC_Par	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Connectors	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
InstSpec	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
LLInst	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
MsgSent	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
MsgRecv	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
MsgSelf	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0
Diags	0	0	0	0	0	0	-	0	0	0	0	0	-	0	0	0

Fonte: Elaborado pelo autor

mencionado anteriormente nesta métrica será considerado a correspondência total de 20% na nota final, então calculando notaFinal = (20.0 - ((20.0 * 33.3) / 100)); resultando em 13,34, informando que o aluno acertou 13,3% dos 20% totais desta métrica.

O mesmo cálculo é executado para cada métrica em cada classe e no final é realizado um somatório entre os resultados para montar a nota final do aluno.

Na Tabela 4 é mostrado todo o processo de calculo da nota do Aluno A baseada nas métricas etapa por etapa, note que na coluna correspondente a classe “Cliente” nas métricas do aluno esta informado “null”, isso por que o Aluno A não adicionou a classe “Cliente” em seu modelo, sendo assim é considerado o desconto total em todas as métricas para esta classe.

Nas Tabelas 5 e 6, os resultados dos cálculos para os Alunos B e C são mostrados respectivamente, onde é observado que o Aluno B que possui poucos desvios em relação ao modelo do instrutor obteve a nota mais alta, cerca de 96,42%, já os alunos A e C que cometeram mais erros, tiveram 65,62% e 62,78% de acertos.

Com os resultados dos cálculos é possível observar que foi possível estabelecer um padrão de descontos de nota justo aos alunos, uma vez que tanto o Aluno A quanto o C esqueceram de acrescentar uma classe no modelo e ambos ao final possuíram uma nota muito próxima, evidenciando a homogeneidade da avaliação.

5.2 Avaliação TAM

Para realizar a avaliação do protótipo ele foi disponibilizado na *web* através do endereço http://modelguru.snotra.com.br/. Foi criado na *homepage* do sistema um tutorial explicando

Tabela 4 – Nota calculada pra o Aluno A

Pi	Métricas (i)	Instrutor (X)				Aluno (Y)				f(Xi, Yi) = |Xi-Yi|				Di				nf(Di, Pi) = Pi - ((Di/100) * Pi)			
		Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido
20	**NumAttr**	3	2	1	3	4	1	null	2	1	1	null	1	33.33	50.00	100.00	33.33	13.33	10.00	0.00	13.33
20	**NumOps**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	20.00	20.00	0.00	20.00
1.48	**NumPubOps**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Setters**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Getters**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Nesting**	0	0	0	0	1	0	null	0	1	0	null	0	100.00	0.00	100.00	0.00	0.00	1.48	0.00	1.48
1.48	**IFImpl**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**NOC**	2	0	0	0	1	0	null	0	1	0	null	0	50.00	0.00	100.00	0.00	0.74	1.48	0.00	1.48
20.00	**NumDesc**	2	0	0	0	1	0	null	0	1	0	null	0	50.00	0.00	100.00	0.00	10.00	20.00	0.00	20.00
1.48	**NumAnc**	0	1	1	0	0	1	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**DIT**	0	1	1	0	0	1	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**CLD**	1	0	0	0	1	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**OpsInh**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**AttrInh**	0	3	3	0	0	4	null	0	0	1	null	0	0.00	33.33	100.00	0.00	1.48	0.99	0.00	1.48
1.48	**Dep_Out**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Dep_In**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**NumAssEl_ssc**	0	1	1	2	0	1	null	2	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**NumAssEl_sb**	0	1	1	2	1	1	null	2	1	0	null	0	100.00	0.00	100.00	0.00	0.00	1.48	0.00	1.48
1.48	**NumAssEl_nsb**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**EC_Attr**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**IC_Attr**	0	0	0	1	0	0	null	1	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**EC_Par**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**IC_Par**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Connectors**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**InstSpec**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**LLInst**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**MsgSent**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**MsgRecv**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**MsgSelf**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
1.48	**Diags**	0	0	0	0	0	0	null	0	0	0	null	0	0.00	0.00	100.00	0.00	1.48	1.48	0.00	1.48
															Nota Final por Classe			**79.63**	**89.51**	**0.00**	**93.33**
															Nota Final do Aluno			**65.62**			

Fonte: Elaborado pelo autor

Tabela 5 – Nota calculada pra o Aluno B

Pi	Métricas (i)	Instrutor (X)				Aluno (Y)				f(Xi, Yi) = |Xi-Yi|				Di				nf(Di, Pi) = Pi - ((Di/100) * Pi)			
		Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido
20	**NumAttr**	3	2	1	3	2	2	1	2	1	0	0	1	33.33	0.00	0.00	33.33	13.33	20.00	20.00	13.33
20	**NumOps**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	20.00	20.00	20.00	20.00
1.48	**NumPubOps**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Setters**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Getters**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Nesting**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**IFImpl**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**NOC**	2	0	0	0	2	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
20.00	**NumDesc**	2	0	0	0	2	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	20.00	20.00	20.00	20.00
1.48	**NumAnc**	0	1	1	0	0	1	1	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**DIT**	0	1	1	0	0	1	1	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**CLD**	1	0	0	0	1	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**OpsInh**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**AttrInh**	0	3	3	0	0	2	2	0	0	1	1	0	0.00	33.33	33.33	0.00	1.48	0.99	0.99	1.48
1.48	**Dep_Out**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Dep_In**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**NumAssEl_ssc**	0	1	1	2	0	1	1	2	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**NumAssEl_sb**	0	1	1	2	0	1	1	2	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**NumAssEl_nsb**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**EC_Attr**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**IC_Attr**	0	0	0	1	0	0	0	1	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**EC_Par**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**IC_Par**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Connectors**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**InstSpec**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**LLInst**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**MsgSent**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**MsgRecv**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**MsgSelf**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
1.48	**Diags**	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	1.48	1.48	1.48	1.48
															Nota Final por Classe			**93.33**	**99.51**	**99.51**	**93.33**
															Nota Final do Aluno			**96.42**			

Fonte: Elaborado pelo autor

Tabela 6 – Nota calculada pra o Aluno C

Pi	Métricas (i)	Instrutor (X)				Aluno (Y)				f(Xi, Yi) = |Xi-Yi|				Di				nf(Di, Pi) = Pi - ((Di/100) * Pi)			
		Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido	Pessoa	Vendedor	Cliente	Pedido
20	**NumAttr**	3	2	1	3	null	5	4	3	null	3	3	0	100.00	100.00	100.00	0.00	0.00	0.00	0.00	20.00
20	**NumOps**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	20.00	20.00	20.00
1.48	**NumPubOps**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Setters**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Getters**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Nesting**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**IFImpl**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**NOC**	2	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
20.00	**NumDesc**	2	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	20.00	20.00	20.00
1.48	**NumAnc**	0	1	1	0	null	0	0	0	null	1	1	0	100.00	100.00	100.00	0.00	0.00	0.00	0.00	1.48
1.48	**DIT**	0	1	1	0	null	0	0	0	null	1	1	0	100.00	100.00	100.00	0.00	0.00	0.00	0.00	1.48
1.48	**CLD**	1	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**OpsInh**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**AttrInh**	0	3	3	0	null	0	0	0	null	3	3	0	100.00	100.00	100.00	0.00	0.00	0.00	0.00	1.48
1.48	**Dep_Out**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Dep_In**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**NumAssEl_ssc**	0	1	1	2	null	1	1	2	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**NumAssEl_sb**	0	1	1	2	null	1	1	2	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**NumAssEl_nsb**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**EC_Attr**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**IC_Attr**	0	0	0	1	null	0	0	1	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**EC_Par**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**IC_Par**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Connectors**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**InstSpec**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**LLInst**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**MsgSent**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**MsgRecv**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**MsgSelf**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
1.48	**Diags**	0	0	0	0	null	0	0	0	null	0	0	0	100.00	0.00	0.00	0.00	0.00	1.48	1.48	1.48
															Nota Final por Classe			0.00	75.56	75.56	100.00
															Nota Final do Aluno			62.78			

Fonte: Elaborado pelo autor

sobre o projeto e instruções para o teste da ferramenta, foi colocado diretamente no site o acesso ao formulário de pesquisa que será descrito na sequencia, e em relação ao teste da ferramenta foi disponibilizado os modelos XMI de entradas do Instrutor e Alunos A, B e C, os mesmos utilizados neste artigo para explicação do processo de cálculo e um video explicativo sobre como proceder o teste, caso o participante da pesquisa desejar ele poderia criar seus próprios diagramas de classe e exportá-los em XMI para submeter a avaliação no ModelGuru.

Perfil dos participantes. O perfil dos participantes da pesquisa pode ser acompanhado na tabela 7, onde é informado características e opinião dos particionantes. A pequisa foi realizada no entre Maio e Junho de 2022 e obteve um total de 14 participantes, sendo 10 alunos (71,4%) e 4 professores (28,6%). A idade dos participantes ficou acima dos 21 anos com maior parte dos participantes entre 26 e 40 anos (64,3%). Metade dos participantes (50%) possuem o grau de Mestrado, seguido por Graduação (28,6%), Doutorado (14,3%) e 1 participante com a Graduação em andamento. A maioria dos participantes (64%) trabalham como desenvolvedores de software ou Analista de sistemas, 28% são professores. Sobre sua relação com a modelagem de software e o uso de diagramas de classe da UML 14,3% dizem utilizar com uma boa frequência, 21,4% dizem utilizar as vezes, 35,7% raramente utilizam diagramas de classe e 28,5% utilizaram apenas na época do curso ou faculdade, nenhum participante disse nunca ter utilizado diagramas de classe da UML.

Foram feitas também perguntas especificas para alunos e professores da areá da computação com o objetivo de segmentar a visão de ambos os lados no processo avaliativo de diagramas. Aos alunos quando perguntados sobre a metodologia atual de avaliação se ela é eficaz responderam que Concordam Parcialmente 60%, sobre a qualidade do feedback no retorno das avaliações foram questionados que á impacto na qualidade da aprendizagem e 40% dizem concordar totalmente e 30% parcialmente, já sobre o tempo de retorno do feedback 70% dos participantes

Tabela 7 – Resultados de Perfil dos participantes

Pergunta	Resposta	#	%
Qual a sua idade?	Até 15 Anos	0	0,0%
	De 16 á 20 Anos	0	0,0%
	De 21 á 25 Anos	2	14,3%
	De 26 á 30 Anos	4	28,6%
	De 31 á 40 Anos	5	35,7%
	De 41 á 50 Anos	2	14,3%
	Mais de 50 Anos	1	7,1%
Qual o seu maior grau de escolaridade?	Fundamental	0	0,0%
	Médio	0	0,0%
	Técnico	0	0,0%
	Superior (Graduação em Andamento)	1	7,1%
	Superior (Graduação)	4	28,6%
	MBA/especialização	0	0,0%
	Mestrado	7	50,0%
	Doutorado	2	14,3%
Qual sua ocupação atual?	Professor(a)	4	28,6%
	Programador(a)/Desenvolvedor(a) de Software	7	50,0%
	Analista de Sistemas	2	14,0%
	Estagiário(a)	0	0,0%
	Suporte	0	0,0%
	Não estou exercendo nenhuma profissão no momento	0	0,0%
	Arquiteto	1	7,1%
Qual sua relação com a modelagem de software e o uso de diagramas da UML como o diagrama de classe?	Utilizo diariamente	0	0,0%
	Utilizo com uma boa frequência	2	14,3%
	Utilizo as vezes	3	21,4%
	Raramente utilizo	5	35,7%
	Usei apenas na época do curso/faculdade	4	28,5%
	Não utilizo diagramas da UML	0	0,0%
Em qual categoria você se enquadra?	Professor ou Ex-professor de Computação	4	28,6%
	Aluno ou Ex-aluno de Computação	10	71,4%
	Outro	0	0,0%
(Aluno) A metodologia atual usada para avaliar os diagramas na modelagem de software é eficaz	Concordo Totalmente	0	0,0%
	Concordo Parcialmente	6	60,0%
	Neutro	2	20,0%
	Discordo Parcialmente	2	20,0%
	Discordo Totalmente	0	0,0%
(Aluno) O modelo atual de feedback (retorno) da avaliação de diagramas é eficiente e ajuda você no entendimento sobre onde estão suas dificuldades	Concordo Totalmente	3	30,0%
	Concordo Parcialmente	4	40,0%
	Neutro	1	10,0%
	Discordo Parcialmente	2	20,0%
	Discordo Totalmente	0	0,0%
(Aluno) O tempo elevado de retorno no feedback pelo professor na avaliação de diagramas impacta na sua qualidade de aprendizagem	Concordo Totalmente	7	70,0%
	Concordo Parcialmente	2	20,0%
	Neutro	0	0,0%
	Discordo Parcialmente	0	0,0%
	Discordo Totalmente	1	10,0%
(Professor) A forma atual de avaliação utilizada onde o professor deve avaliar modelo a modelo de cada um dos alunos é eficiente e garante a correção uniforme entre todos os diagramas	Concordo Totalmente	0	0,0%
	Concordo Parcialmente	1	25,0%
	Neutro	1	25,0%
	Discordo Parcialmente	2	50,0%
	Discordo Totalmente	0	0,0%
(Professor) Sobre o feedback (retorno) da avaliação, no modelo tradicional de correção de diagramas é garantido que a resposta devolvida ao aluno o ajuda no processo de aprendizagem	Concordo Totalmente	1	25,0%
	Concordo Parcialmente	2	50,0%
	Neutro	1	25,0%
	Discordo Parcialmente	0	0,0%
	Discordo Totalmente	0	0,0%
(Professor) Através da forma convencional de correção de diagramas é possivel identificar com exatidão em quais aspectos da modelagem (ex: Complexidade, Acoplamento, Herança...) o aluno obteve maior dificuldade?	Sim	3	75,0%
	Não	1	25,0%

Fonte: Elaborado pelo autor

Tabela 8 – Resultados do questionário TAM

Pergunta	Resposta	#	%
Achei o ModelGuru fácil de usar	Concordo Totalmente	12	85,7%
	Concordo Parcialmente	1	7,14%
	Neutro	1	7,14%
	Discordo Parcialmente	0	0,0%
	Discordo Totalmente	0	0,0%
Achei a interface do ModelGuru atrativa	Concordo Totalmente	6	42,8%
	Concordo Parcialmente	5	35,7%
	Neutro	3	21,4%
	Discordo Parcialmente	0	0,0%
	Discordo Totalmente	0	0,0%
Achei o ModelGuru fácil de aprender	Concordo Totalmente	11	78,5%
	Concordo Parcialmente	3	21,4%
	Neutro	0	0,0%
	Discordo Parcialmente	0	0,0%
	Discordo Totalmente	0	0,0%
O ModelGuru facilitaria o processo de aprendizado na modelagem de software	Concordo Totalmente	8	57,1%
	Concordo Parcialmente	3	21,4%
	Neutro	1	7,14%
	Discordo Parcialmente	2	14,2%
	Discordo Totalmente	0	0,0%
O ModelGuru ajudaria na avaliação baseada em atributos de qualidade	Concordo Totalmente	8	57,1%
	Concordo Parcialmente	5	35,7%
	Neutro	0	0,0%
	Discordo Parcialmente	1	7,14%
	Discordo Totalmente	0	0,0%
O feedback automático e baseado em métricas potencializa a aprendizagem	Concordo Totalmente	8	57,1%
	Concordo Parcialmente	2	14,2%
	Neutro	3	21,4%
	Discordo Parcialmente	1	7,14%
	Discordo Totalmente	0	0,0%
Utilizaria o ModelGuru como ferramenta para o processo de aprendizado na criação de diagramas	Concordo Totalmente	8	57,1%
	Concordo Parcialmente	3	21,4%
	Neutro	2	14,2%
	Discordo Parcialmente	1	7,14%
	Discordo Totalmente	0	0,0%

Fonte: Elaborado pelo autor

concordam totalmente que isso causa impacto no processo de aprendizagem.

Os professores foram perguntados se a forma tradicional de avaliação garante a correção uniforme e homogênea entre os diagramas e 25% concordaram parcialmente, 50% discordaram parcialmente e 25% ficaram neutros. Sobre o retorno de suas avaliações aos alunos foram questionados se a forma atual de feedback garante evolução no processo de aprendizado do aluno e 50% concordaram parcialmente. Ao final foi questionado ao professor se através da forma convencional de correção é possível identificar com exatidão os aspectos da modelagem como Complexidade, Acoplamento, Herança e Tamanho, e a maioria (75%) concordam que sim, a forma convencional permite esta percepção.

Questionário TAM. Na tabela 8 estão os resultados das perguntas relacionadas ao questionário TAM, as três primeiras questões são referentes a facilidade de uso, as três seguintes sobre a facilidade de uso e a ultima questão é sobre a intenção de uso da ferramenta.

Sobre as questões de facilidade de uso, 85,7% ou mais dos participantes acharam o ModelGuru fácil de usar, 42,8% acharam a interface atrativa e 35,7% concordaram parcialmente, sobre a facilidade de aprender no ModelGuru 78,5% ou mais dos participantes concordaram. Em relação a percepção de utilidade 78,5% dos participantes acham que o ModelGuru facilitaria

o processo de aprendizagem (57,1% concordam totalmente e 21,4% concordam parcialmente), 57,1% ou mais dos participantes concordam que o feedback automatizado e baseado nas métricas potencializa a aprendizagem. Em relação a intensão de uso 78,5% utilizariam a ferramenta (57,1% concordam totalmente e 21,4% concordam parcialmente).

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi apresentado o ModelGuru, uma abordagem baseada em métricas para realizar avaliação de diagrama de classes da UML. Foi apontado as dificuldades enfrentadas pelos instrutores de modelagem de software para realizar a avaliação de modelos. A ausência de uma forma homogênea e automatizada para realizar a avaliação torna inviável a elaboração de exercícios maiores, dificulta o *feedback* e compromete o processo de aprendizagem do aluno. A partir disto, foi identificada a oportunidade de pesquisa que foi explorada durante o trabalho. A proposta da abordagem foi sustentada através de um estudo de caso, uma pesquisa de aceitação de tecnologia (TAM) e o desenvolvimento de um protótipo. De acordo com este trabalho foi identificado um grande potencial na abordagem e foi coletado informações sobre pontos a serem melhorados que podem ser explorados em próximas versões. A partir dos resultados do questionário foi identificado que existe tanto utilidade quanto intenção de uso, o que evidenciou um bom potencial para a abordagem. Porém foi identificado que esta abordagem não consegue sozinha identificar variações de soluções corretas para o mesmo exercício, do ponto de vista matemático as métricas nos fornecem características de todo o modelo de forma eficaz, mas do ponto de vista da modelagem aspectos como a liberdade de modelagem precisam ser mais bem avaliadas.

Para os trabalhos futuros é possível: (1) adicionar novas camadas de avaliação sob a camada baseada em métricas, como em exemplo a avaliação semântica entre relacionamentos, atualmente a análise das métricas apenas detecta alterações mas ainda não é possível estabelecer na etapa de cálculo da nota a correspondência correta de desconto quando um modelo, por exemplo, possuir uma realização no lugar de uma generalização; (2) realizar a coleta dos dados e persistir em base de dados para permitir a observabilidade das informações, salvar estes dados em banco e utilizar uma plataforma como o Grafana possibilitaria uma análise de desempenho de turmas de alunos em tempo real e histórico, agregando valor a ferramenta; (3) estender o uso das métricas do SDMetrics para métricas de pacote e regras de *design*, foram utilizados apenas as 30 métricas de classe padrões do SDMetrics, em novos estudos podem ser adicionadas as demais métricas disponíveis com o propósito de um maior alcance na avaliação da modelagem; (4) realizar o estudo sobre aplicação da abordagem em um ambiente de empresas; (5) evoluir o protótipo ModelGuru e adicionar novas funcionalidades, em exemplo, a possibilidade de criação de lista de exercícios pelo instrutor.

Referências

ADAMS, J. C. **Computing is the Safe Stem Career Choice today**. 2014. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/180053-computing-is-the-safe-stem-career-choice-today/fulltext>.

BIAN, W.; ALAM, O.; KIENZLE, J. Automated grading of class diagrams. **Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2019**, p. 700–709, 2019.

CANAL, J.; FARIAS, K.; GONCALES, L. An algorithm for distance calculation between UML sequence diagrams. **IEEE Latin America Transactions**, v. 16, n. 4, p. 1200–1205, 2018. ISSN 15480992.

COFFEY, J. W. A method to evaluate differences between student UML class diagrams. **Journal of Computing Sciences in College**, p. 68–74, 2014.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying systematic reviews to diverse study types: An experience report. In: **First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)**. [S.l.: s.n.], 2007. p. 225–234.

FARIAS, K. et al. On the uml use in the brazilian industry: A state of the practice survey (s). In: **SEKE**. [S.l.: s.n.], 2018. p. 372–371.

FARIAS, K. et al. Toward an architecture for model composition techniques. **Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE**, v. 2015-January, p. 656–659, 2015. ISSN 23259086.

FARIAS, K.; SILVA, B. C. d. What's the grade of your diagram? towards a streamlined approach for grading uml diagrams. In: **Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings**. [S.l.: s.n.], 2020. p. 1–2.

FERNÁNDEZ-SÁEZ, A. M.; CHAUDRON, M. R.; GENERO, M. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. **Empirical Software Engineering**, Empirical Software Engineering, v. 23, n. 6, p. 3281–3345, 2018. ISSN 15737616.

HASKER, R. UMLGrader: an automated class diagram grader. **Journal of Computing Sciences in Colleges**, v. 27, n. 1, p. 47–54, 2011. ISSN 1937-4771.

HASKER, R. W.; ROWE, M. UMLint: Identifying defects in UML diagrams. **ASEE Annual Conference and Exposition, Conference Proceedings**, 2011. ISSN 21535965.

JOHNSON, R. et al. **Spring Framework Documentation**. 2022. Disponível em: <https://docs.spring.io/spring-framework/docs/current/reference/html/>.

JÚNIOR, E.; FARIAS, K. Modelgame: A quality model for gamified software modeling learning. In: **15th Brazilian Symposium on Software Components, Architectures, and Reuse**. [S.l.: s.n.], 2021. p. 100–109.

JÚNIOR, E.; FARIAS, K.; SILVA, B. A survey on the use of uml in the brazilian industry. In: **Brazilian Symposium on Software Engineering**. [S.l.: s.n.], 2021. p. 275–284.

KITCHENHAM, B.; BRERETON, P. A systematic review of systematic review process research in software engineering. **Information and Software Technology**, v. 55, n. 12, p. 2049–2075, 2013. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584913001560>.

MARANGUNIć, N.; GRANIć, A. Universal access in the information society international journal technology acceptance model: a literature review from 1986 to 2013. **Universal Access in the Information Society**, v. 14, p. 1–15, 02 2014.

MENZEN, J. P.; FARIAS, K.; BISCHOFF, V. Using biometric data in software engineering: a systematic mapping study. **Behaviour & Information Technology**, Taylor & Francis, v. 40, n. 9, p. 880–902, 2021.

MILLER, F. P.; VANDOME, A. F.; MCBREWSTER, J. **Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau?Levenshtein Distance, Spell Checker, Hamming Distance**. [S.l.]: Alpha Press, 2009. ISBN 6130216904.

RUBERT, M.; FARIAS, K. On the effects of continuous delivery on code quality: A case study in industry. **Computer Standards & Interfaces**, Elsevier, v. 81, p. 103588, 2022.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **Unified Modeling Language Reference Manual, The (2nd Edition)**. [S.l.]: Pearson Higher Education, 2004. ISBN 0321245628.

SINGER, N. **The hard part of Computer Science**. 2019. Disponível em: <https://www.nytimes.com/2019/01/24/technology/computer-science-courses-college.html>.

STIKKOLORUM, D. R. et al. Towards automated grading of UML class diagrams with machine learning. **CEUR Workshop Proceedings**, v. 2491, p. 1–13, 2019. ISSN 16130073.

SUZUKI, J.; YAMAMOTO, Y. Making uml models interoperable with uxf. In: BÉZIVIN, J.; MULLER, P.-A. (Ed.). **The Unified Modeling Language. «UML»'98: Beyond the Notation**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 78–91. ISBN 978-3-540-48480-6.

VESIN, B. et al. Web-based educational ecosystem for automatization of teaching process and assessment of students. **ACM International Conference Proceeding Series**, 2018.

WüST, J. **SDMetrics - The design quality metrics tool for UML models**. 2021. Disponível em: <https://www.sdmetrics.com/index.html>.

XMI®. The Object Management Group®, 2015. Disponível em: <https://www.omg.org/spec/XMI/>.

YI, T.; WU, F.; GAN, C. A Comparison of Metrics for UML Class Diagrams ACM SIGSOFT Software Engineering Notes. **Software Engineering Notes**, v. 29, n. 5, p. 1–6, 2004.

YOU, E. **Vue.js - The Progressive JavaScript Framework**. 2022. Disponível em: <https://vuejs.org/>.