Priorização de Atividades de Desenvolvimento de Software com uma Rede Neural Artificial

Maicon Junior Silveira
Universidade Tecnológica Federal do Paraná - UTFPR
mai_conjs@hotmail.com

Andre Luiz Marasca
Universidade Tecnológica Federal do Paraná - UTFPR
eng.andremarasca@gmail.com

Marlon Marcon
Universidade Tecnológica Federal do Paraná - UTFPR
marlonmarcon@utfpr.eudu.br

André Roberto Ortoncelli
Universidade Tecnológica Federal do Paraná - UTFPR
ortoncelli@utfpr.edu.br

ABSTRACT

Prioritization of development activities in software companies is commonly performed arbitrarily and subjectively, based solely on the experience of the company's employees. Aiming to contribute to the automation of this process, this paper presents a method of automatic prioritization of software development activities, using an algorithm that combines the use of a numerical regression-based Artificial Neural Network (RNA) with an ordering method. To validate the accuracy of the proposed method, a case study was conducted, with real data from a software development company. The experiments allowed to evaluate the accuracy of the proposed method qualitatively and quantitatively, generating satisfactory results, which indicate the feasibility of implementing the proposed method in the target company of the case study.

KEYWORDS

Machine Learning, Neural Network, Numerical Regression, Software Development Process

1 INTRODUÇÃO

Um Processo de Desenvolvimento de Software (PDS) consiste em um conjunto de atividades organizadas e realizadas de modo a definir, implementar, testar e manter um determinado software. Destaca-se que o PDS complexo, altamente sensível à interação humana e ao trabalho em equipe [7], [33].

Comumente as atividades desenvolvidas em um PDS não são executadas de forma automatizada (principalmente em pequenas e médias empresas), podendo gerar disparidade entre o processo especificado formalmente e o processo utilizado por uma empresa, tornando imprevisível a qualidade do processo e dos produtos produzidos [13].

Apesar das vantagens que a automatização pode trazer ao PDS, diferentes desafios podem ser encontrados, devido ao fato de o desenvolvimento de software ser uma atividade orientada a projetos e não possuir estágios típicos de produção, tais como, atividades ou interações repetíveis [13].

Na literatura, diferentes trabalhos apresentam métodos relacionados a priorização de atividades que compõe um PDS, tais como a priorização de casos de teste [9, 18], a priorização de Atividades de Desenvolvimento de Software (PADS) [29] e a Priorização de Requisitos de Software (PRS) [3, 5, 8, 12, 14, 16, 17, 23, 31].

Apesar de poucos métodos relacionados a PADS existirem na literatura, existe um número considerável de técnicas que podem ser aplicados para PRS (que são conceitos relacionados). Cabe destacar que nenhuma das abordagens propostas para PRS é considerada o "melhor", pois resultados "melhores" de priorização dependem da situação, sendo centrados no estudo e não universais [11].

Pode-se destacar também que grande parte dos métodos de PRS existentes não foram amplamente adotados, além de pesquisas apontarem que a maioria dos métodos sofrem com problemas de usabilidade e escalabilidade [10, 27], o que justifica a pesquisa e desenvolvimento e validação de novos métodos de priorização.

Técnicas de aprendizagem de máquina possibilitam que, a partir de exemplos previamente rotulados, se possa realizar inferência para valores não conhecidos. No processo de priorização de atividades, proposto neste trabalho, uma rede neural artificial foi utilizada para inferir a ordem de prioridade entre duas atividades, associada a um algoritmo de ordenação de dados.

A solução automática de PADS apresentada neste trabalho foi desenvolvida para atender a uma demanda de uma empresa de desenvolvimento de software de médio porte, localizada na região sudoeste do estado do Paraná. Tal empresa, em seu processo de desenvolvimento, divide os requisitos que devem ser implementados em atividades de desenvolvimento de software, considerando que cada requisito pode ser implementado em uma ou mais atividades, porém comumente os requisitos são desenvolvidos em apenas uma atividade. Dessa forma, o que diferencia os conceitos de requisitos e atividades é o fato de que um requisito pode ser implementado com o uso de uma ou mais atividades.

Nesse contexto esse trabalho apresenta duas contribuições principais: i) um método para priorização automática de atividades de desenvolvimento software com o uso de uma RNA baseada em regressão e em um método de ordenação; e ii) um estudo de caso conduzido para mensurar a acurácia do método, com atividades reais de uma empresa de desenvolvimento de software.

Visando apresentar tais contribuições, o restante desse trabalho encontra-se organizado da seguinte forma. A Seção 2 apresenta uma revisão bibliográfica relacionada a métodos automatizados relacionados a Engenharia de Requisitos. Na Seção 3 são apresentados detalhes sobre o método de priorização. A Seção 4 apresenta a metodologia utilizada para condução do estudo de caso e também os resultados obtidos. Por fim, a Seção 5 apresenta a conclusão do trabalho e informações sobre trabalhos que se pretende realizar.

2 TRABALHOS RELACIONADOS

Poucos estudos diretamente relacionados a priorização de atividades são apresentados na literatura, com destaque para [29] que apresenta uma técnica baseada em árvore de decisão.

Em [29], foi criada uma árvore de decisão com base no conhecimento de especialistas da empresa. A árvore de decisão recebe como entrada um vetor de características referente a uma determinada atividade e retorna como resultado da classificação, um nível de prioridade para a atividade (existindo 16 níveis de prioridade possíveis). Dessa forma o algoritmo apresenta problemas para priorizar um grande conjunto de atividades, já que muitas atividades recebem o mesmo nível de prioridade. As limitações desse estudo, motivaram o desenvolvimento do método que é apresentado neste trabalho.

Apesar da quantidade limitada de trabalhos referentes a PADS, uma quantidade considerável de trabalhos está relacionada a priorização de requisitos [3, 5, 8, 12, 14, 16, 17, 23, 31]. Como citado anteriormente os conceitos são relacionados. Nesse contexto, a Subseção 2.1 irá apresentar detalhes sobre métodos de PRS existentes na literatura.

2.1 Priorização de Requisitos de Software (PRS)

Para que se possa priorizar requisitos, é necessário inicialmente definir os critérios de priorização, que podem envolver a importância do requisito, os riscos envolvidos, os custos, o tempo de desenvolvimento e a volatilidade dos requisitos [1].

Uma vez definidos os critérios que serão utilizados para priorizar requisitos, a atividade de priorização comumente ainda é uma atividade custosa, já que requer tempo para sua aplicação, disponibilidade dos *stakeholders* envolvidos e análise da dependência entre requisitos [6], além de contar com diferentes desafios para equalizar os requisitos específicos do cliente, do mercado, além dos problemas com a comunicação dos *stakeholders* e desenvolvedores [4], contando ainda com desafios específicos em tipos de empresas diferentes [28].

Diferentes métodos já foram explorados na literatura visando solucionar os problemas e desafios existentes relacionados a priorização de requisitos, entre eles podem-se destacar, as técnicas baseadas em um Processo Hierárquico Analítico [12], Custo-Valor [3], Votação Cumulativa [8], Votação Cumulativa Hierárquica [14], jogos de planejamento [16], aprendizagem de máquina [23], algoritmos genéticos [31] e também abordagens colaborativas [5].

O método baseado em um Processo Hierárquico Analítico consiste em uma abordagem genérica utilizada para tomada de decisão que tem sido adotada para priorização de requisitos. Tal método consiste em realizar comparações de pares de requisitos hierarquicamente classificados, permitindo atribuir notas em uma escala de um a nove, em relação a igualdade da prioridade entre os requisitos. Se existirem n requisitos será necessário fazer $n(n-1)/2$ analises [12]. Sendo um dos mais famosos métodos para priorização mais citados [25] e também um dos mais promissores [17].

A abordagem Custo-Valor é uma técnica bastante simples [3], na qual, os requisitos são priorizados com base em seu custo e valor relativo. Porém apesar de ser simples e requerer pouco tempo para ser utilizado, esse método considera apenas aspectos relacionados custo e valor, ignorando aspectos tal como risco e volatilidade, além de não suportar customização [12].

O método baseado em Votação Cumulativa também conhecido como teste dos 100 dólares, também é um método popular para priorização de requisitos. Nesse método os participantes recebem unidades fictícias de dinheiro, horas ou quaisquer outras unidades, distribuindo essas unidades pelos requisitos de acordo com sua prioridade. Um dos principais problemas desse método é a dificuldade em relação a escalabilidade [8].

Visando superar o problema apresentado pela Votação Cumulativa, foi criado o método de Votação Cumulativa a Hierárquica, que é uma mistura do método de Votação Cumulativa com o Processo Hierárquico Analítico [14].

Já o jogo do planejamento é utilizado comumente em conjunto com a metodologia ágil *eXtreme Programming* (XP). Com o jogo do planejamento, uma vez que os requisitos são elicitados e documentados em *story cards*, eles podem ser elicitados pelos clientes em três pilhas diferentes que representam três níveis de prioridade: i) aquelas sem as quais o sistema não funciona; ii) aqueles que são menos essenciais, mas fornecem valor comercial significativo; e iii) aqueles que seriam desejáveis [16].

Uma abordagem baseada em aprendizagem de máquina foi proposta em [23], combinando informações sobre as preferências das partes interessadas com as aproximações de ordenação de requisitos computados. Tal como o método apresentado nesse trabalho, o método proposto em [23] também trabalha com comparação de pares de atividades, porém não apresenta detalhes dos atributos utilizados pelo método de aprendizagem de máquina.

Também visando ranquear requisitos pela sua prioridade, no algoritmo genético proposto em [31], cada indivíduo na população que está sendo evoluído representa uma priorização alternativa dos requisitos, resolvendo problema de priorização através da minimização da função de *fitness*.

Já a abordagem colaborativa apresentada em [5], consiste em um processo colaborativo de priorização de requisitos, que explora os elementos de gamificação para motivar os *stakeholders* envolvidos, de modo que eles contribuam com o processo de tomada de decisão em relação a priorização de requisitos.

Essa seção apresentou alguns dos principais métodos utilizados para priorização de requisitos de software. Na literatura podem ser encontrados mais trabalhos relacionados, analisados em diferentes perspectivas por meio de mapeamentos e revisões de literatura [1], [6], [24], [26] e [30].

Além dos métodos desenvolvidos para priorização de requisitos, cabe destacar que na literatura diferentes abordagens baseadas em aprendizagem de máquina foram desenvolvidos e aplicados para a priorização de atividades e artefatos em diferentes contextos, tais como, a priorização na automação de redes de distribuição de energia [19], na área médica para priorização de pacientes [20], para a priorização de estudantes com risco de não se graduarem [2], entre outras aplicações.

3 MÉTODO PROPOSTO

No método de priorização proposto nesse trabalho, utiliza-se a associação entre um algoritmo de ordenação e um comparador de atividades. O algoritmo de ordenação recebe um conjunto de atividades não priorizadas e por meio da comparação de pares delas, ordena as atividades, produzindo um conjunto ordenado de atividades em relação a sua prioridade.

Visto que as atividades possuem diversas características que influenciam em suas prioridades, a comparação se torna difícil. Tal dificuldade motivou a utilização de um método baseados em

Table 1: Parâmetros utilizados no treinamento da RNA

Propriedade	Valor
Função de ativação	RELU
Otimizador	ADAM
Learning rate	0,001 (adaptativa)
Momentum	0,9
Número de épocas	1000

aprendizagem de máquina, que realiza tal comparação com base em exemplos anotados por especialistas. A Figura 1 apresenta como é a dinâmica de interação entre o algoritmo de ordenação e a RNA, no método proposto. O processo apresentado na Figura 1 modelado com a notação BPMN [32].

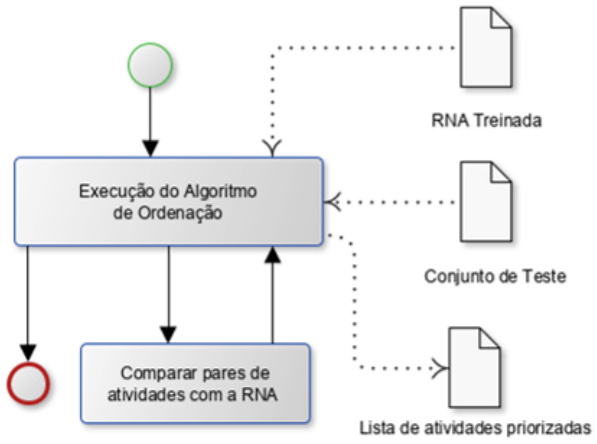

Figure 1: Dinâmica do método proposto para priorização (ordenação) das atividades

A presente proposta é independente de algoritmo de ordenação, porém, no escopo do testes, utilizou-se o Tim Sort [15]. A cada nova atividade adicionada a lista, uma reordenação das atividades (ou inserção ordenada) é exigida, para que as atividades estejam sempre priorizadas de maneira correta.

Essa Seção apresenta detalhes do método de priorização proposto. A Subseção 3.1 descreve o treinamento da RNA utilizada. A Subseção 3.2 apresenta informações sobre os conjuntos de atividades utilizadas. Por fim, a Subseção 3.3 descreve como o algoritmo de ordenação é utilizado para priorização das atividades.

3.1 Treinamento da Rede Neural

O processo de treinamento da RNA, é executado somente uma vez e utiliza pares de atividades, com prioridade conhecida. A RNA utilizada, foi do tipo Multi Layer Perceptron, com duas camadas escondidas, com 50 e 30 neurônios respectivamente. O processo de treinamento é apresentado na Figura 2 (assim como o processo apresentado na Figura 2 esse processo também foi modelado com a notação BPMN) e os parâmetros utilizados no treinamento constam na Tabela 1.

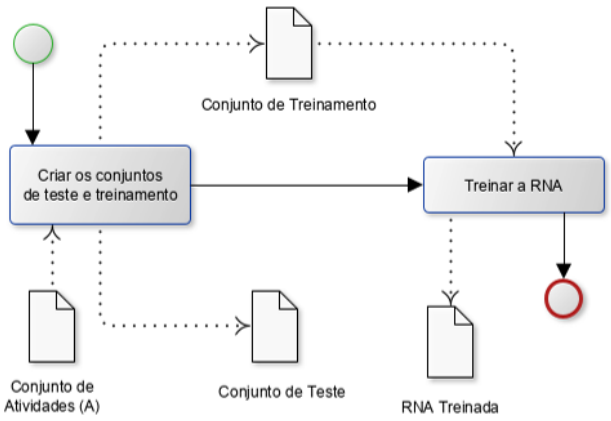

Figure 2: Processo de treinamento da RNA

3.2 Conjuntos de atividades

Essa Subseção apresenta a estrutura do conjunto de dados utilizado pelo método de priorização. O processo de priorização se inicia com um conjunto de atividades $A = [a_1, .., a_n]$, que é composto por n atividades.

Cada atividade $a_x \in A$ é composta por um vetor de 7 atributos, denotado por $a_x = [at_1, at_2, at_3, at_4, at_5, at_6, at_7]$. Pode-se denotar um atributo at_z da atividade a_x, utilizando a seguinte notação $a_x[at_z]$.

As informações armazenadas em cada um dos 7 atributos do vetor de atributos de uma atividade a_x são detalhadas a seguir:

- at_1: representa o **tipo de trabalho** que será realizado na atividade. Os tipos de trabalho possíveis são: i) BUG; ii) Customização paga; iii) Customização não paga; e iv) Melhoria. Cada um dos possíveis tipos de trabalho é detalhado a seguir:
  - BUG: refere-se a atividades referentes a correção de falhas dos softwares;
  - Customizações: Corresponde a novas funcionalidades que possuam muitas particularidades para um determinado cliente, tornando o recurso útil apenas a ele. Cada solicitação de customização é avaliada por uma equipe da empresa, que pode classificar a atividade como uma customização paga (que deve ser paga pela empresa solicitante) ou uma customização não paga (que será desenvolvida gratuitamente); e
  - Melhoria: Corresponde ao desenvolvimento de novas funcionalidades nos softwares que geram benefícios a todos os clientes que o utilizarem.
- at_2: denota a **origem da atividade**, que pode ser externa ou interna. A origem de uma atividade é considerada externa se ela foi registrada para algum dos clientes. Se a demanda pelo desenvolvimento de uma atividade partiu de uma investigação (demanda identificada) feita por um dos colaboradores da própria empresa, a origem dessa atividade é classificada como interna.
- at_3: é relacionada a **categoria do cliente**, ou seja, ao grupo em que o cliente pertence. Clientes podem fazer parte do

grupo *premium* (que geram um retorno financeiro fundamental para a empresa de desenvolvimento de software) ou do grupo normal (todo o cliente que não faz parte do grupo *premium*).

- at_4: esse atributo identifica se o cliente está ou não em **processo de implantação**. Estar em processo de implantação significa que o cliente adquiriu a licença de utilização do software recentemente e precisa de uma atenção especial, com o objetivo de que seu primeiro contato com a empresa não seja insatisfatório.
  Caso o cliente esteja em processo de implantação, as solicitações dele devem ser atendidas o mais rápido possível, pois o objetivo é tornar o cliente autossuficiente na utilização dos softwares, diminuindo assim custos relacionados ao processo de adequação do cliente.
- at_5: representa a **satisfação do cliente**, calculada através das notas de satisfação dadas mensalmente pelos usuários dos softwares. Os clientes são notificados mensalmente pelo próprio software para que atribuam uma nota entre 0 e 5, com relação a sua satisfação com a qualidade dos softwares e demais serviços fornecidos.
  Nesse contexto, o valor de at_5 é a média trimestral de todas as notas dadas por um usuário no último trimestre. Tal informação é utilizada para dar maior prioridade as atividades demandas por clientes com uma média de satisfação menor.
- at_6: esta informação está relacionada ao **tempo de espera** de uma atividade, ou seja, o tempo que uma atividade está parada sem nenhuma interação (mensurado em horas úteis por semana).
  Tal informação é utilizada com o objetivo de que o algoritmo atribua uma maior prioridade para atividades que estejam a muito tempo sem uma interação (uma atividade que esteja a muito tempo na lista de atividades à serem desenvolvidas), visando garantir que os clientes não fiquem insatisfeitos pela demora excessiva de uma entrega.
- at_7: define a **prioridade** de uma atividade. Dado um conjunto A com n elementos, o atributo prioridade de cada atividade será um número y (com um valor entre 1 e n) , que indica a prioridade de cada atividade, sendo que quanto mais próximo de 1 é o valor de y, maior é a prioridade da atividade.

Um conjunto de atividades A é utilizado para criar outros dois conjuntos A_{teste} e $A_{treinamento}$. A RNA utilizada foi treinada com um conjunto de atividades $A_{treinamento}$. Após o treinamento da RNA, o conjunto A_{teste} foi utilizado para testar a acurácia do método de priorização.

Para criar os conjuntos $A_{treinamento}$ e A_{teste}, inicialmente se extrai do conjunto A (composto por n atividades) um subconjunto auxiliar A_{aux}, composto por m atividades (tal que $m < n$). O conjunto A_{aux} será utilizado como base para criar o conjunto $A_{treinamento}$.

O conjunto A_{teste} é composto por toda a atividade a_x contida no conjunto A ($a_x \in A$) mas que não faça parte do conjunto A_{aux} ($a_x \notin A_{aux}$). Sendo o conjunto A composto por n atividades e conjunto A_{aux} composto por m atividades, o conjunto A_{teste} contará com $n - m$ atividades.

Já o conjunto $A_{treinamento}$ será criado a partir do conjunto A_{aux}. Cada combinação possível entre pares de atividades do conjunto (a_x, a_y) $in A_{aux}$ é adicionado ao conjunto $A_{treinamento}$. Portanto, considerando que o conjunto A_{aux} possui m atividades, o conjunto $A_{treinamento}$ irá possuir m^2 pares de atividades.

Cada par de atividades $(a_x, a_y) \in A_{treinamento}$ representa um vetor de 13 atributos, composto pelos seis primeiros atributos das atividades a_x e a_y e também um atributo $priori_{xy}$, que identifica qual atividade possui a maior prioridade, tendo seu valor definido com base no valor dos atributos $a_x[at_7]$ e $a_y[at_7]$, tal como é apresentado na Equação 1.

$$priori_{x,y} = \begin{cases} -1, & \text{se } a_x[at_7] > a_y[at_7] \\ 0, & \text{se } a_x[at_7] = a_y[at_7] \\ 1, & \text{se } a_x[at_7] < a_y[at_7] \end{cases} \quad (1)$$

O processo de criação do conjunto $A_{treinamento}$, com base no conjunto A_{aux} é ilustrado na Figura 3.

3.3 Algoritmo de ordenação

O algoritmo de ordenação utilizado neste trabalho foi o Tim Sort [15]. Um algoritmo de ordenação consiste em um método que recebe como parâmetro um conjunto de elementos e coloca tais elementos em uma determinada ordem, sendo que as mais utilizadas são as ordens numérica e a lexicográfica. Nesse trabalho as atividades foram ordenadas em relação a sua prioridade.

O algoritmo Tim Sort é um método híbrido de ordenação que combina características dos algoritmos Merge Sort e Insertion Sort, que no caso médio executa $n.log_2(n)$ comparações, porém comumente executa um número menor de comparações por explorar características positivas de dois métodos de ordenação.

Neste trabalho o algoritmo ordenação recebeu como parâmetro um conjunto de atividades de desenvolvimento de software (A_{teste}) e uma função de priorização, para definir a ordem relativa das atividades (em relação a sua prioridade).

A função de priorização utilizada para definir a prioridade entre pares de atividades, consiste em uma RNA baseada em regressão numérica (descrita na Subseção 3.1), que recebe como entrada um par de atividades $a_1, a_2 \in A_{teste}$ e retorna um valor r entre -1 e 1, tal que, se $r = 0$ as duas atividades a_1 e a_2 têm a mesma prioridade, se $-1 < r < 0$ a atividade a_1 tem a maior prioridade e se $0 < r < 1$ a atividade a_2 tem a maior prioridade. O que diferencia o resultado da RNA em relação a Equação 1, é o fato de que com a Equação 1 são atribuídos apenas valores inteiros, porém como a RNA é baseada em regressão logística, ela pode retornar valores fracionários.

Destaca-se que o uso de um conjunto de valores entre -1 e 1, já foi explorado em outros trabalhos existentes na literatura, também relacionados a priorização com uso algoritmos de aprendizagem de máquina [23], [21], de modo similar ao utilizado no método de priorização apresentado nessa Seção.

Como o método de ordenação Tim Sort, ordena um conjunto comparando pares de elementos, tal método consegue priorizar um conjunto de atividades, comparando pares de atividades com a RNA, retornando assim, uma lista de atividades em ordem da maior para a menor prioridade.

Destaca-se que o método de ordenação Tim Sort foi selecionado para ser utilizado neste trabalho devido a sua eficiência, porém

A_{aux}

atividade	at_1	at_2	at_3	at_4	at_5	at_6	at_7
at_1	BUG	Interna	Premium	Sim	2	10	1
at_2	BUG	Externa	Normal	Sim	2,42	5	2
at_3	Melhoria	Interna	Normal	Não	3,82	1	3

$A_{treinamento}$

Atividade a_x						**Atividade a_y**						$priori_{x,y}$
$a_x[at_1]$	$a_x[at_2]$	$a_x[at_3]$	$a_x[at_4]$	$a_x[at_5]$	$a_x[at_6]$	$a_y[at_1]$	$a_y[at_2]$	$a_y[at_3]$	$a_y[at_4]$	$a_y[at_5]$	$a_y[at_6]$	
BUG	Interna	Premium	Sim	2	10	BUG	Interna	Premium	Sim	2	10	0
BUG	Interna	Premium	Sim	2	10	BUG	Externa	Normal	Não	4,42	5	1
BUG	Interna	Premium	Sim	2	10	Melhoria	Interna	Normal	Não	3,82	1	1
BUG	Externa	Normal	Não	4,42	5	BUG	Interna	Premium	Sim	2	10	-1
BUG	Externa	Normal	Não	4,42	5	BUG	Externa	Normal	Não	4,42	5	0
BUG	Externa	Normal	Não	4,42	5	Melhoria	Interna	Normal	Não	3,82	1	1
Melhoria	Interna	Normal	Não	3,82	1	BUG	Interna	Premium	Sim	2	10	-1
Melhoria	Interna	Normal	Não	3,82	1	BUG	Externa	Normal	Não	4,42	5	-1
Melhoria	Interna	Normal	Não	3,48	1	Melhoria	Interna	Normal	Não	3,82	1	0

Figure 3: Exemplo da criação do conjunto $A_{treinamento}$

pode ser substituído por qualquer método de ordenação, que realize a ordenação por meio da comparação entre pares de elementos.

4 ESTUDO DE CASO

Para que fosse possível mensurar a acurácia do método automático de PADS apresentado nesse trabalho, o método foi implementado na linguagem de programação Python com uso da biblioteca de aprendizagem de máquina Scikit-learn [22]. Após implementado o método foi avaliado por meio de um estudo de caso com atividades reais da empresa parceira do projeto, que forneceu informações sobre as suas atividades de desenvolvimento, que foram utilizadas para o desenvolvimento do *dataset* utilizado nos experimentos.

Essa seção têm o objetivo de descrever detalhes sobre o experimento realizado. A Subseção 4.1 apresenta as métricas utilizadas para avaliar quantitativamente os resultados do experimento. O *dataset* utilizado é apresentado na Subseção 4.2. Os resultados experimentais obtidos são apresentados na Subseção 4.3. Por fim, as limitações do estudo são apresentadas na Subseção 4.4.

4.1 Métricas de avaliação

Para avaliar de forma quantitativa os resultados obtidos pela execução do método proposto, foram utilizadas o Índice de Jaccard (Equação 2) e a Similaridade do cosseno (Equação 3). Ambas medidas retornam valores de similaridade entre 0 e 1, tal que quanto mais próximo de 1 for o resultado, maior é a similaridade.

Considerando dois conjuntos B e C, as Equações 2 e 3 apresentam respectivamente como são calculados o Índice de Jaccard e a Similaridade de cosseno.

O Índice de Jaccard, também conhecido como *Intersection over Union*, mede a similaridade entre dois conjuntos de dados levando em consideração a razão entre o tamanhos do conjunto de intersecção (ou seja, dados corretamente classificados) e de união.

$$J(B, C) = \frac{|B \cap C|}{|B \cup C|} \tag{2}$$

A similaridade do cosseno, avalia o nível de coesão entre dois conjuntos a partir da distância angular entre eles no espaço. Essa métrica é particularmente importante para o contexto deste trabalho, pois permite avaliar se uma priorização, mesmo quando equivocada ao final do processo, ficará próxima da ordem ideal esperada.

$$cos_{sim} = cos(\theta) = \frac{B \cdot C}{\|B\| \cdot \|C\|} \tag{3}$$

4.2 *Dataset*

A empresa parceira do projeto forneceu para o desenvolvimento da pesquisa um *dataset* com informações sobre as atividades realizadas entre os anos de 2017 e 2018. Destaca-se que informações sobre os clientes da empresa e também sobre a descrição das atividades foram omitidas devido a uma restrição imposta pela empresa.

A partir do dados fornecidos foi possível extrair os seis primeiros atributos de cada atividade adicionada ao conjunto A, porém não foi possível extrair a prioridade das atividades utilizando apenas os dados disponibilizados, pois a empresa utilizava apenas uma escala de 1 a 3 para identificar a prioridade de cada atividade. Tal escala de três níveis se mostrou insuficiente para obter resultados satisfatórios com o método de priorização proposto.

Em virtude dessa limitação, para definir a prioridade das atividades, foi necessário realizar um processo de priorização manual das atividades. A priorização manual das atividades foi feita pelos funcionarios da empresas responsáveis pela priorização das atividades. Devido ao custo e tempo demandados para a rotulação manual das atividades não foi possível avaliar a acurácia do método em relação a todas as atividades já realizadas pela empresa. Para tal

Table 2: Número atividades priorizadas com erro nos experimentos

Conjunto de teste	Número de atividades				
	25	35	50	75	100
$A1_{teste}$	2	2	2	8	11
$A2_{teste}$	4	12	18	38	56
$A3_{teste}$	8	10	18	30	34
$A4_{teste}$	6	12	20	34	46
$A5_{teste}$	8	10	18	34	46
Média	5,6	9,2	15,2	28,8	38,6

avalição, foram utilizados seis conjuntos de atividades compostos por 100 atividades distintas (totalizando 600 atividades).

Dos seis conjuntos de atividades rotulados manualmente por especialistas da empresa, um deles foi utilizado para criar o conjunto ($A_{treinamento}$) da RNA e os outros cinco conjuntos foram utilizados para criar cinco conjuntos de teste (A_{teste}), que foram utilizados para testar o método de priorização.

O conjunto $A_{treinamento}$ foi composto por atividades referentes ao mês de janeiro de 2018, já os demais conjuntos utilizados como teste, foram criados com atividades selecionadas de modo aleatório. Os cinco conjuntos de teste utilizados foram intitulados como: $A1_{teste}$, $A2_{teste}$, $A3_{teste}$, $A4_{teste}$ e $A5_{teste}$. Ressalta-se que o conjunto de treinamento, apesar de conter apenas 100 atividades, é composto por todas as possíveis combinações de pares de atividades, com as respectivas prioridades entre si, o que totaliza um conjunto de 10000 amostras.

Cada conjunto de 100 atividades gerou além do próprio, outros quatro subconjuntos de atividades, com 25, 35, 50, 75 atividades cada. Tais conjuntos foram construídos a partir de seleção aleatória das atividades. Destaca-se que o subconjunto de 35 atividades é composto pelas 25 atividades do conjunto de 25 atividades e mais 10 atividades selecionadas aleatoriamente. Da mesma forma a atividade que faz parte de um subconjunto maior está contida no subconjunto com menos atividades.

4.3 Resultados Experimentais

Cinco experimentos foram realizados, cada um com um dos cinco conjuntos de teste criados. Em cada experimento, ordenaram-se utilizando o método proposto, os cinco conjuntos de atividades.

Após a execução dos experimentos, para avaliar o método de priorização proposto foram realizadas dois tipos de analises: i) uma análise quantitativa para mensurar a acurácia do método; e ii) uma análise qualitativa com base na opinião de especialistas da empresa.

A análise quantitativa utilizou as métricas descritas na Subseção 4.1. Além dessas métricas, também foi utilizado o número de erros de priorização identificados após a priorização das atividades.

Para verificar a eficiência do algoritmo de priorização, foram contabilizados quantos elementos foram posicionados de forma errada na lista ordenada (número de erros de priorização), em relação a lista de referência. Os resultados são apresentados na Tabela 2.

Além de contabilizar os erros, a partir das listas de prioridade, foram calculados os índices de similaridade de Jaccard e Similaridade do Cosseno entre as listas resultantes e as de referência. Tais

Table 3: Índice de Jaccard para os conjuntos de teste e de referência

Conjunto de teste	Número de atividades				
	25	35	50	75	100
$A1_{teste}$	0,920	0,943	0,960	0,893	0,890
$A2_{teste}$	0,840	0,657	0,640	0,493	0,440
$A3_{teste}$	0,680	0,714	0,640	0,600	0,660
$A4_{teste}$	0,760	0,657	0,600	0,547	0,540
$A5_{teste}$	0,680	0,714	0,640	0,547	0,540
Média	0,7760	0,7371	0,6960	0,6160	0,6140

Table 4: Resultados da métrica de similaridade do cosseno para os testes realizados

Conjunto de teste	Número de atividades				
	25	35	50	75	100
$A1_{teste}$	0,9993	0,9997	0,9999	0,9999	0,9999
$A2_{teste}$	0,9991	0,9975	0,9969	0,9978	0,9978
$A3_{teste}$	0,9935	0,9959	0,9913	0,9965	0,9974
$A4_{teste}$	0,9980	0,9975	0,9985	0,9992	0,9987
$A5_{teste}$	0,9987	0,9993	0,9994	0,9993	0,9990
Média	0,9977	0,9980	0,9972	0,9986	0,9986

índices de similari- dade são apresentados respectivamente nas Tabelas 3 e 4.

A partir da análise das Tabela 2 e 3, quando se avalia somente o número de erros de posicionamento identificados, aparentemente a priorização das atividades não apresenta resultados satisfatórios, sendo que a taxa de erros varia de 22 (para conjuntos de 25 atividades) a 38% (para conjuntos 100 atividades), porém um erro de ordenação não pode ser visto como um erro absoluto. Apesar de ocorrer um erro na decisão da prioridade, é importante que atividades com prioridade maior fiquem na maior parte dos casos a frente das demais.

Os resultados obtidos a partir da similaridade do cosseno, contrapõem os resultados da Tabela 2, demonstrando que, apesar da ocorrência de erros de priorização, no geral a taxa de similaridade entre as listas de referência e as priorizadas pelo algoritmo são muito altas ($> 0,99$). Os resultados, apresentados na Tabela 4, demonstram que apesar de ocorrerem erros de decisão, as listas de priorização resultantes não são muito distantes das de referência.

Apesar da priorização ser importante, ela apresenta um guia de quais atividades devem ser tratadas antes das outras. Entretanto no caso de equipes de desenvolvimento relativamente grande, uma priorização "perfeita não terá impacto específico no tempo de solução destas atividades. Por exemplo em uma equipe de desenvolvimento que possui dez programadores, se considerar que as atividades possuem tempo semelhante para solução, estas serão removidas da lista em blocos de dez, e, para este caso um erro de posicionamento de uma atividade de até dez posições poderia ser aceitável. O tamanho deste bloco pode ser variável de acordo com o tamanho da equipe

Table 5: Resultados da métrica de Jaccard para cada conjunto de testes, levando em conta a tolerância de erros na priorização das atividades

	Número de atividades				
% tolerância	25	35	50	75	100
0,00%	0,776	0,737	0,696	0,616	0,614
5,00%	0,952	0,931	0,920	0,931	0,924
10,00%	0,984	0,966	0,944	0,973	0,972
15,00%	0,984	1,000	0,984	0,995	0,996
20,00%	1,000	1,000	0,992	1,000	1,000
25,00%	1,000	1,000	1,000	1,000	1,000

por exemplo e com isso uma certa taxa de tolerância ao erro pode ser aceitável na lista resultante.

Buscando realizar uma análise mais aprofundada desta relação, foram coletados dados do Índice de Jaccard considerando uma tolerância aceitável com base no tamanho da lista de atividades. A Tabela 5 e a Figura 4 apresentam os resultados médios desta avaliação, considerando uma tolerância de 0% a 25% na distância entre uma atividade posicionada erroneamente na lista resultante e sua respectiva posição de referência.

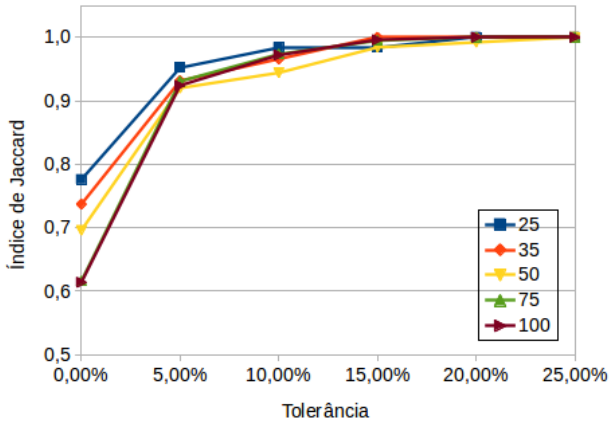

Figure 4: Resultados do Índice de Jaccard considerando a taxa de tolerância na distância entre os erros de priorização.

É possível verificar que admitindo somente 5% de tolerância, a priorização das atividades retorna um índice maior que 92 de similaridade entre as listas priorizadas e de referência. Outro ponto a se destacar, é que os erros de priorização, geralmente ocorrem em posições de menor prioridade. A partir da Figura 5 é possível verificar que para as listas com 100 posições, as atividades de maior prioridade (mais altas) não sofrem mudanças de priorização. Quando ocorrem, até a metade da lista, são em sua maioria erros de no máximo três posições.

Ressalta-se que a partir dessa análise, este problema pode ser mitigado no momento de reordenação das prioridades, seja por entrada de novas atividades, seja por decisão estratégica da empresa que adote tal priorização.

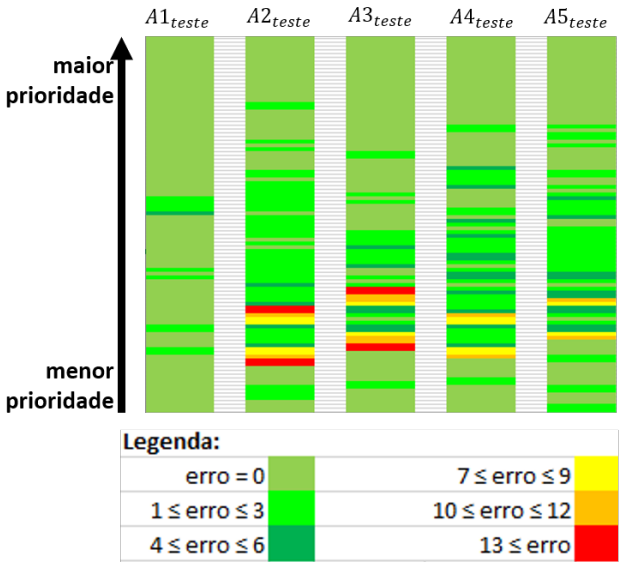

Figure 5: Representação gráfica do erro dos experimentos com conjuntos de 100 atividades

Além das análises quantitativas, uma análise qualitativa dos resultados obtidos com o algoritmo de priorização proposto foi feita, com base na opinião de especialistas da empresa parceira do projeto (profissionais responsáveis pela priorização das atividades). De acordo com a opinião desses especialistas, os resultados obtidos são compatíveis com a priorização que é feita atualmente na empresa, sendo viável utilizar o algoritmo de ordenação proposto.

4.4 Limitações do Estudo

Bons resultados foram obtidos com o estudo de caso apresentado, porém algumas limitações podem ser indicadas em relação ao trabalho. A primeira limitação a ser indicada é referente a impossibilidade de validar o método com todas as atividades já desenvolvidas pela empresa alvo do estudo de caso, devido a limitação em relação a informação de prioridade que não estava armazenada na base de dados, sendo uma atividade custosa rotular todas as atividades em relação a sua prioridade. Porém, mesmo assim, o método de priorização foi avaliado com um conjunto significativo de atividades.

Outra limitação se deu em relação ao estudo de caso envolver apenas uma empresa, o que aconteceu devido à dificuldade de se obter *datasets* relacionados as atividades reais de desenvolvimento.

Por fim, aponta-se como limitação o conjunto de atributos utilizados para treinar a RNA utilizada, diferentes atributos poderiam ter sido explorados, tais como custo e tempo previstos desenvolvimento, que poderiam proporcionar um resultado melhor de priorização. Um conjunto maior de informações não foi utilizado, pois tais dados não estavam armazenados na base de dados da empresa, porém cabe destacar que mesmo sem explorar o uso de um conjunto maior de informações, se obteve bons resultados de priorização.

5 CONCLUSÃO

Esse trabalho apresentou um método de priorização de atividades de desenvolvimento de software, implementado de acordo com as demandas apresentadas por uma empresa de desenvolvimento de software do sudoeste do Paraná. Um estudo de caso foi conduzido com registros reais de atividades da empresa, no qual o método de priorização proposto apresentou boa acurácia atendendo as expectativas da empresa.

Foi demonstrado que apesar da ocorrência de erros na priorização das atividades, os resultados obtidos encorajam a aplicação deste método na priorização de atividades de desenvolvimento de software. Não se pode afirmar que o método consegue atender as demandas de priorização de atividades de qualquer empresa de desenvolvimento de software, porém é uma abordagem promissora de priorização, já que não sofre com problemas de escalabilidade, tal como outros métodos de priorização existentes na literatura [10] e [27]. Além disso o método consegue priorizar rapidamente um conjunto de atividades, podendo ser executado a cada inserção de uma nova atividade na lista de espera para o desenvolvimento, mantendo assim a lista de atividades priorizada.

5.1 Trabalhos futuros

Em trabalhos futuros pretende-se utilizar o método de priorização produzido em cenários reais de desenvolvimento de software, adicionando o algoritmo ao sistema de gestão de projetos e atividades que é utilizado atualmente pela empresa parceira do projeto.

Além disso, também pretende-se desenvolver alternativas para superar as limitações do estudo de caso conduzido, explorando um conjunto maior de atributos e atividades e tentando também explorar *datasets* com informações sobre mais de uma empresa.

AGRADECIMENTOS

Agradecemos a empresa Visual Software que cedeu a base de dados com informações sobre as atividades de desenvolvimento que foram fundamentais para a execução dos experimentos e desenvolvimento da pesquisa.

REFERENCES

[1] P.; Achimugu, A. Selamat, R. Ibrahim, and M. N. Mahrin. 2014. A systematic literature review of software requirements prioritization research. *Information and software technology* 56, 6 (2014), 568–585.

[2] E. Aguiar, H. Lakkaraju, N. Bhanpuri, D. Miller, B. Yuhas, and K. L. Addison. 2015. Who, when, and why: a machine learning approach to prioritizing students at risk of not graduating high school on time. In *International Conference on Learning Analytics And Knowledge*. ACM, 93–102.

[3] T. Amelia and R. B. Mohamed. 2018. Review on Cost-Value Approach for Requirements Prioritization Techniques. In *International Conference on Information Technology, Computer, and Electrical Engineering*. IEEE, 310–314.

[4] M. I. Babar, M. Ramzan, and S. A. K. Ghayyur. 2011. Challenges and future trends in software requirements prioritization. In *International Conference on Computer Networks and Information Technology*. IEEE, 319–324.

[5] P. Busetta, F. M. Kifetew, D. Munante, A. Perini, A. Siena, and A. Susi. 2017. Tool-supported collaborative requirements prioritisation. In *IEEE Annual Computer Software and Applications Conference*, Vol. 1. IEEE, 180–189.

[6] C. Cavalcanti, M. Lencastre, R. Fagundes, T. Santos, and D. Ferreira. 2018. Mechanisms to Support Requirements Prioritization: A Systematic Mapping Review. In *Workshp de Engenharia de Requisitos*.

[7] P. Clarke, R. V. O'Connor, and B. Leavy. 2016. A complexity theory viewpoint on the software development process and situational context. In *International Conference on Software and Systems Process*. ACM, 86–90.

[8] L. Dean and W. Don. 2003. Managing software requirements: A use case approach. In *Addison Wesley*.

[9] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia. 2018. A Test Case Prioritization Genetic Algorithm guided by the Hypervolume Indicator. *IEEE Transactions on Software Engineering* (2018).

[10] A. Ejnioui, C. Otero, and L. Otero. 2012. A simulation-based fuzzy multi-attribute decision making for prioritizing software requirements. In *Annual conference on Research in information technology*. ACM, 37–42.

[11] M. A. A. Elsood, H. A. Hefny, and E. s. Nasr. 2014. A goal-based technique for requirements prioritization. In *International Conference on Informatics and Systems*. IEEE, 310–314.

[12] N. Garg, M. Sadiq, and P. Agarwal. 2017. GOASREP: Goal oriented approach for software requirements elicitation and prioritization using analytic hierarchy process. In *International Conference on Frontiers in Intelligent Computing: Theory and Applications*. Springer, 281–287.

[13] G. Grambow, R. Oberhauser, and M. Reichert. 2011. Towards automatic process-aware coordination in collaborative software engineering. *Conference on Software and Data Technologies* (2011).

[14] B. B. Jawale, G. K. Patnaik, and A. T. Bhole. 2017. Requirement Prioritization Using Adaptive Fuzzy Hierarchical Cumulative Voting. In *IEEE International Advance Computing Conference*. IEEE, 95–102.

[15] Y. B. Jmaa, K. M. A. Ali, D. Duvivier, M. B. Jemaa, and R. B. Atitallah. 2017. An Efficient Hardware Implementation of TimSort and MergeSort Algorithms Using High Level Synthesis. In *International Conference on High Performance Computing & Simulation*. IEEE, 580–587.

[16] L. Karlsson, T. Thelin, B. Regnell, P. Berander, and C. Wohlin. 2007. Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques. *Empirical Software Engineering* 12, 1 (2007), 3–33.

[17] J. A. Khan, I. U. Rehman, Y. H. Khan, I. J. Khan, and S. Rashid. 2015. Comparison of requirement prioritization techniques to find best prioritization technique. *International Journal of Modern Education and Computer Science* 7, 11 (2015), 53.

[18] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng. 2018. Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology* 93 (2018), 74–93.

[19] A. S. Maykot, E. A. C. Aranha Neto, and N. de A. Oliva. 2019. Priorização da Automação de Redes de Distribuição Utilizando Algoritmos Genéticos com Foco no Self-Healing. In *Computer on The Beach*. 927–929.

[20] G. Mylavarapu and J. P. Thomas. 2017. A multi-task machine learning approach for comorbid patient prioritization. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 3877–3881.

[21] G. Paetzold and L. Specia. 2017. Lexical simplification with neural ranking. In *Conference of the European Chapter of the Association for Computational Linguistics*, Vol. 2. 34–40.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[23] A. Perini, A. Susi, and P. Avesani. 2013. A machine learning approach to software requirements prioritization. *IEEE Transactions on Software Engineering* 39, 4 (2013), 445–461.

[24] A. M. Pitangueira, R. S. P. Maciel, and M. Barros. 2015. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. *Journal of Systems and Software* 103 (2015), 267–280.

[25] M. S. Rahim, A. Z. M. E. Chowdhury, and S. Das. 2017. Rize: A proposed requirements prioritization technique for agile development. In *IEEE Region 10 Humanitarian Technology Conference*. IEEE, 634–637.

[26] N. Riegel and J. Doerr. 2015. A systematic literature review of requirements prioritization criteria. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 300–317.

[27] F. Sher, D. N. A. Jawawi, R. Mohamad, and M. I. Babar. 2014. Requirements prioritization techniques and different aspects for prioritization a systematic literature review protocol. In *Malaysian Software Engineering Conference*. IEEE, 31–36.

[28] G. Silva, W. Santos, and M. Lencastre. 2018. Priorização de Requisitos em Startups no Armazém da Criatividade (Porto Digital): Resultados Preliminares. In *Workshp de Engenharia de Requisitos*.

[29] M. J. Silveira, P. H. Bordignon, R. T. Pagno, M. Marcon, and A. R. Ortoncelli. 2017. Uma Técnica de Priorização de Atividades de Desenvolvimento de Software Baseada em Árvore de Decisão. In *Congresso de Ciência e Tecnologia da UTFPR Câmpus Dois Vizinhos*.

[30] R. Thakurta. 2017. Understanding requirement prioritization artifacts: a systematic mapping study. *Requirements engineering* 22, 4 (2017), 491–526.

[31] P. Tonella, A. Susi, and F. Palma. 2013. Interactive requirements prioritization using a genetic algorithm. *Information and software technology* 55, 1 (2013), 173–187.

[32] Stephen A White. 2004. Introduction to BPMN. *Ibm Cooperation* 2, 0 (2004), 0.

[33] M. Yilmaz, R. V. O'Connor, and P. Clarke. 2012. A systematic approach to the comparison of roles in the software development processes. In *Int. Conference on Software Process Improvement and Capability Determination*. Springer, 198–209.