Dependency Rank

Método de priorização de requisitos baseado nas relações de dependência identificadas por PLN

MarlonAndre PeronGeneroso[†]
Departamento de Informática
Universidade Federal do Paraná
Curitiba Paraná Brasil
marlonbsi@gmail.com

AndreyRicardo Pimentel
Departamento de Informática
Universidade Federal do Paraná
Curitiba Paraná Brasil
andreyrpimentel@gmail.com

ABSTRACT

Currently, the most commonly used software requirements prioritization techniques are highly dependent on human effort to achieve them [1]. Facing this problem, this paper presents an automated prioritization method based on dependency relationships between features.

The proposed method sought to reduce the amount of effort employed by automating part of this task in an attempt to provide greater agility and reliability to the process. Thus, we used the project requirements documentation as a basis for extracting these relationships.

A prototype using natural language processing tools was developed, its application aimed to recognize candidate classes contained in software requirements specification documents, written as user stories, thereby enabling the identification of existing links between the features. After this analysis, a suggested ranking, which employs as main criterion the prioritization of the requirements with the largest number of dependencies, is generated.

The method was tested in an experiment and its validation was assisted by professionals. The results showed that the strategy implemented to identify candidate classes reached an F1 score of 0.857. This index helped the prototype to classify up to 70% of the requirements within equivalent intervals to those obtained by human judgment, having as main challenge for future developments the increase of the subjectivity load of the method.

X CBSOFT'19, September, 2019, Salvador, Bahia Brasil.

KEYWORDS

Software requirements prioritization, Requirements interdependency, Natural Language Processing, User Stories.

1. Introdução

Em projetos de software, os desenvolvedores frequentemente enfrentam o desafio de priorizar requisitos. Isso ocorre em circunstâncias em que um projeto, desenvolvido de maneira incremental, possui um grande conjunto de requisitos a serem considerados, ou ainda em situações de orçamento insuficiente para atender a todos os requisitos levantados [2]. Em ambos os casos, não é simples a tarefa de escolher quais devem ser incluídos na próxima versão ou quais eventualmente não serão contemplados no projeto.

Métodos de priorização de requisitos de software visam equilibrar o benefício de um determinado requisito para o sistema com seu respectivo custo de desenvolvimento, tendo como meta buscar a inclusão dos recursos mais essenciais para a versão [2] (ou produto), gerando um incremento ou produto final que satisfaça as necessidades e expectativas do cliente.

Durante a fase de engenharia de requisitos é normal elaborar especificações usando linguagem natural (LN), pois tal adoção as torna claras, possibilitando a todos os envolvidos uma compreensão escrita do problema [3], esclarecendo objetivos e possíveis redundâncias no conjunto. Essa compreensão permite unir interesses de clientes e desenvolvedores para a formalização de um contrato.

Embora as especificações feitas em LN sejam de fácil compreensão para humanos, o uso de formatos não estruturados, como as especificações em histórias de usuários ou casos de uso, representa um desafio para a identificação automatizada de termos relevantes sobre o problema analisado [4]. Esse tipo de desafio fomentou o desenvolvimento de ferramentas de Processamento de Linguagem Natural (PLN), que podem ser notadas em nossa rotina diária com frequência cada vez maior. Pode-se citar como exemplo de utilização dessas ferramentas os mecanismos de busca na internet, tradutores e outros sistemas disponíveis [5].

Nos últimos anos, várias técnicas ou métodos de priorização de requisitos foram estudados e testados [1], essas técnicas visam facilitar a tarefa de priorização no contexto de projetos de software. Em recente estudo sobre o estado da arte foram encontradas 49 técnicas diferentes entre 73 estudos avaliados [1]. Mesmo diante de tal variedade, nenhuma técnica/método listado automatizava o processo de priorização (ou parte dele) a partir das informações contidas em especificações de requisitos em LN.

Este trabalho investiga a possibilidade de priorizar corretamente requisitos de software, identificando automaticamente as relações de dependências entre eles com a aplicação de ferramentas de PLN, integradas a uma técnica de priorização por ranking [6]. Para validação dos resultados, o ranking obtido pelo protótipo do método automatizado é comparado a um conjunto de avaliações de profissionais da área no mesmo problema.

O restante desse artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados que auxiliaram na concepção do método proposto; a Seção 3 demonstra a arquitetura do método Dependecy Rank; a Seção 4 apresenta o experimento realizado para avaliar a proximidade alcançada pelo método em comparação às avaliações humanas. Por fim, a Seção 5 apresenta as conclusões e trabalhos futuros identificados neste estudo.

2. Trabalhos Relacionados

Os principais trabalhos relacionados a este estudo foram encontrados por meio de mapeamento sistemático de literatura, conforme [7]. O mapeamento investigou dentre as publicações na área de priorização de requisitos de software, quais utilizavam as dependências entre os requisitos como parâmetro explícito de priorização em suas técnicas/métodos. Os 12 artigos retornados como resultados acessíveis estão disponíveis para consulta no endereço https://bit.ly/2Hr0ixs.

O estudo apresentado por [6] baseia-se em preferências e dependências para definição de prioridades. O método desenvolvido pretendeu utilizar esses parâmetros para melhorar a qualidade da priorização de acordo com as preferências dos stakeholders de um projeto, assim como diminuir a carga de trabalho humano e a necessidade de envolvimento de engenheiros de requisitos muito experientes. De acordo com o autor, o DRank obteve melhorias em relação ao consumo de tempo ao ser comparado com outras técnicas já disponíveis.

Por sua vez, o método apresentado em [8] analisa requisitos funcionais para o modelo incremental de desenvolvimento de software. Esse método manipula informações sobre os relacionamentos de dependência entre os requisitos de um sistema, fornecidas por stakeholders, para preencher uma matriz contendo as restrições de implementação, adaptadas das definições de dependência apresentadas em [9].

A extração de informações em especificações feitas em LN foi investigada complementarmente em buscas por strings nas principais bibliotecas digitais. Nesse contexto, a ferramenta proposta por [10], que propõe automatizar a geração de diagramas de classes da UML a partir de descrições de problemas ou especificações de requisitos em LN, foi a escolhida para direcionar a identificação de termos do Dependency Rank.

Diante do exposto, este trabalho propõe um método de priorização de requisitos com classificação por ranking similar ao utilizado em [6]. Sua meta é gerar uma sugestão de ranking de requisitos intuitivo, que possa ser facilmente alterado, entendido e reproduzido manualmente por desenvolvedores, de modo a simplificar a participação de profissionais no experimento e a comparação dos resultados. A principal contribuição do ranqueamento proposto neste trabalho está na automatização do processo que pré-classifica os requisitos a partir de suas dependências, sem necessitar de dados fornecidos por usuários nessa fase.

As relações de interdependência são mapeadas considerando os tipos Constraint/Requiring, que aponta restrições de implementação [9], e, Structural/Similar_to, que aponta semelhança entre as funcionalidades [11]. Para isso, textos que contém às descrições de requisitos de um determinado problema são processados por analisadores léxicos e sintáticos do framework Apache OpenNLP em busca de classes candidatas, similarmente a [10].

3. Método Dependency Rank

Esta seção demonstra a arquitetura do Dependency Rank exemplificando sua aplicação com uma história de usuário hipotética. Uma visão geral é exibida na Figura 1.

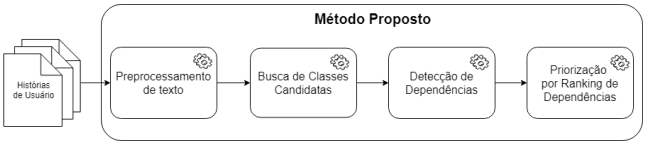

Figura 1: **Modelo geral do método proposto.**

Requisitos documentados em formato de histórias de usuário, descritos individualmente em arquivos tipo texto simples no idioma inglês compõem o conjunto de entradas, sendo que a indicação desses arquivos é a única etapa que demanda intervenção humana. O padrão de histórias de usuário foi definido com base em [12], de modo a possibilitar a identificação das strings delimitadoras e a busca dos termos de interesse.

Modelo: As a <role>, I want <goal/desire> so that <benefit>.

Exemplo: As a seller, I want to list products so that I can choose items to sell.

As histórias de usuário submetidas iniciam o processo pela etapa de preprocessamento de texto.

Nessa etapa, os arquivos que contém os requisitos são recebidos, populando um array no qual cada posição representa uma história de usuário. A sentença, após identificada, passa pelo processo de tokenização, com o auxílio das classes Tokenizer, TokenizerME e TokenizerModel do pacote opennlp.tools.tokenize do framework, criando um novo array, contendo os tokens reconhecidos. Para o exemplo citado, o array gerado deve possuir 17 posições, sendo: 0: "As", 1: "a", 2: "seller", 3: ",", 4: "I", 5: "want", 6: "to", 7: "list", 8: "products", 9: "so", 10: "that", 11: "I", 12: "can", 13: "choose", 14: "items", 15: "to", 16: "sell".

O array de tokens gerado na etapa anterior é fornecido como entrada do processo de marcação de parte do discurso, o POS Tagging, auxiliado nesse método pelas classes POSModel e POSTaggerME, do pacote opennlp.tools.postag do framework. Aapós a execução, são gerados outros dois arrays: tags e probabilidades.

O array de tags recebe uma etiqueta com a descrição do papel do token da mesma posição na sentença. Por sua vez, o array de probabilidades vincula a probabilidade da atribuição da tag estar correta para o mesmo token. O valor armazenado auxilia o método a encontrar o verbo principal da história de usuário e as classes candidatas envolvidas, conforme exposto adiante.

Para o exemplo, a sequência de tags esperada, considerando a lista de tags do Apache OpenNLP apresentada em [13], é: 0: "IN" (preposition or subordinating conjunction), 1: "DT" (determiner), 2: "NN" (noun, singular or mass), 3: ",", 4: "PRP" (personal pronoun), 5: "VBP" (verb, non-3rd person singular present), 6: "TO" (to), 7: "VB" (verb, base form), 8: "NNS" (noun, plural), 9: "RB" (adverb), 10: "IN" (preposition or subordinating conjunction), 11: "PRP" (personal pronoun), 12: "MD" (modal), 13: "VBP" (verb, non-3rd person singular present), 14: "NNS" (noun, plural), 15: "TO" (to), 16: "VB" (verb, base form).

Após a identificação de tags, são definidos os lemas, com o auxílio da classe opennlp.tools.lemmatizer.DictionaryLemmatizer, de forma a transformar os tokens identificados nas histórias de usuário em termos mais genéricos, evitando algumas ambiguidades. Para isso, os três arrays obtidos anteriormente (tokens, tags e probabilidades) devem ser analisados pelas funções implementadas. Esse processamento acarreta como saída um novo array contendo os lemas de cada história de usuário, em substituição ao de tokens. Sendo assim, tem-se como resultado esperado para o exemplo: 0: "As", 1: "a", 2: "seller", 3: ",", 4: "I", 5: "want", 6: "to", 7: "list", **8: "product"**, 9: "so", 10: "that", 11: "I", 12: "can", 13: "choose", **14: "item"**, 15: "to", 16: "sell".

Ao comparar os arrays de tokens e de lemas, constata-se que as posições 8 e 14 foram alteradas. Isso indica que o token "products" corresponde ao lema "product", enquanto o token "itens" corresponde ao lema "item". As substituições efetuadas evitam que palavras com a mesma raiz, que apresentam formas de escrita distintas, sejam interpretadas como termos diferentes. Logo, as posições do array de lemas que correspondem a substantivos no array de tags, ou seja, que contém "NN" em sua string, são classificados como classes candidatas do problema.

A identificação de classes candidatas viabiliza a detecção de dependências entre os requisitos que, para o Dependency Rank, é estabelecida ao encontrar uma classe candidata que é manipulada por 2 ou mais requisitos distintos. Nessas situações, um link de dependência é criado entre eles, caracterizando uma relação do tipo Structural/Similar_to [11], ou Constraint/Requiring [9], sem especificação de precedência, ou seja, os requisitos envolvidos apontam um para o outro.

Finalmente, o cálculo de prioridades é realizado ao receber o conjunto de histórias de usuário com suas respectivas estruturas de dados que contém as dependências identificadas na etapa anterior. As prioridades sugeridas são calculadas tendo em vista a quantidade de dependências de cada uma, ou seja, para o método, quanto mais links um requisito possui, maior será a prioridade sugerida para ele.

Um protótipo que utiliza o framework OpenNLP e linguagem de programação Java foi desenvolvido, em versão desktop, exclusivamente para a realização dos experimentos. Para propiciar a correta execução das 4 etapas citadas, foram implementados 4 métodos (funções) de busca de termos sobre o array de lemas.

Inicialmente uma rotina explora o intervalo entre […, I, want, to, …] e […, so, that, …] do array de lemas. O lema sinalizado como verbo, isto é, que contém "VB" na tag associada, com maior probabilidade é retornado, sendo considerado o verbo principal do requisito. Para o exemplo, seria retornado "list".

O segundo método percorre o array de lemas entre as strings [As, a, …] e […, I, want, to, ...] verificando quais itens do intervalo são substantivos, ou seja, cuja posição associada no array de tags contém "NN". O item de maior probabilidade vinculada é concatenado com substantivos vizinhos, quando for o caso, sendo retornado como resultado desse processamento. Dessa forma, possibilita-se que substantivos compostos possam ser identificados sem que o(s) lema(s) com menor probabilidade vinculada seja(m) ignorado(s), sendo possível identificar dependências tanto pelos termos simples quanto pelos termos concatenados. Para o exemplo citado no início da seção, o retorno é "seller".

O terceiro método explora o intervalo entre as posições de […, I, want, to, …] e […, so, that, …] em busca do substantivo de maior probabilidade, concatenando-o se necessário. O termo final

é retornado, sendo considerado a classe candidata do campo goal. "Product" é o retorno esperado no exemplo.

Por fim, o quarto método busca após as posições de […, so, that, …] o substantivo de maior probabilidade que possa caracterizar uma classe candidata no campo benefit. Assim como nos demais casos, esse termo pode ser concatenado com as posições vizinhas do array de lemas caso sejam substantivos. Sendo assim, o retorno esperado no exemplo é "item".

Todos os lemas reconhecidos nos campos goal/desire e benefit são considerados classes candidatas do problema em análise e, com exceção das classes do campo role, que simbolizam os atores do requisito, todos geram dependências. Aos termos concatenados, foi implementada uma regra que considera tanto o termo completo quanto suas partes como classes do problema, possibilitando a interpretação das partes como conceitos independentes que possuem relação com o contexto de aplicação do software.

Como exemplo, considerando o requisito "As a waiter I want to create a food order so that the kitchen can prepare it", temos que, seu processamento identifica no campo goal três classes: Food, Order e Food_order. Pode-se observar que, para um sistema de restaurante, os termos food e order podem representar objetos independentes, validando a hipótese citada.

4. Experimento e Resultados

Um sistema com 10 requisitos especificados em histórias de usuário foi submetido à análise de 12 profissionais da área de tecnologia da informação com conhecimento acadêmico e/ou prático em priorização de requisitos de software. O método Dependency Rank ou o protótipo gerado não foram apresentados aos participantes antes da realização do experimento.

No experimento, os profissionais realizaram tarefas semelhantes às do método proposto, identificando as classes candidatas envolvidas nas histórias de usuário e julgando as prioridades entre os requisitos. Os dados das avaliações individuais foram informados em um formulário em formato web. O experimento foi realizado propondo atender dois objetivos: (1) Comparar o conjunto de classes candidatas reconhecidas pelo método com as classes candidatas citadas pelos profissionais; (2) comparar o ranking de priorização sugerido pelo método com o extraído pelas avaliações dos profissionais.

O ranking de prioridades informado pelos participantes foi definido subjetivamente de acordo com seus próprios critérios de avaliação, não foram indicadas quaisquer técnicas ou estipulados parâmetros para direcionar essa tarefa. Portanto, a comparação entre os dois rankings visa compreender o impacto das relações de dependências no julgamento subjetivo dos profissionais, bem como, se esse critério pode ser usado como determinante para priorizar os requisitos de software avaliados.

A ordem de priorização sugerida pelo Dependency Rank foi obtida a partir do processamento do conjunto de requisitos de entrada pelo protótipo que implementa o método proposto.

4.1 Elementos de Entrada

O conjunto composto por dez histórias de usuário analisados pelos profissionais, escritas no idioma inglês, é apresentado a seguir:

- R1. As a waiter I want to create a food order so that the kitchen can prepare it.
- R2. As a waiter I want to create a drink order so that the bar can prepare it.
- R3. As a waiter I want to open the table bill so that I can add orders.
- R4. As a waiter I want to change an order so that I can fix a wrong order.
- R5. As a waiter I want to cancel an order so that I can warn the kitchen or the bar.
- R6. As a waiter I want to close the table bill so that the bill can be calculated.
- R7. As a waiter I want to set the table so that I can open an empty bill.
- R8. As a waiter I want to print the bill so that I can deliver it to the customer.
- R9. As a manager I want to view the list of orders so that I can distribute tasks.
- R10. As a manager I want to manage the stock so that I can enable items in the menu.

Visando atender o padrão de entrada definido, cada um dos requisitos listados foi escrito em um arquivo do tipo texto simples em codificação Unicode UTF-8.

4.2. Identificação de Classes Candidatas

Essa seção exibe os resultados referentes às classes candidatas encontradas pelo método e pelos profissionais no experimento.

Após a execução do protótipo, o método encontrou 16 classes candidatas nas histórias de usuário do conjunto submetido, sendo: Waiter, Manager, Food, Order, Food_order, Kitchen, Drink, Drink_order, Bar, Table, Bill, Table_bill, Customer, Task, Stock, Menu.

A lista de classes candidatas identificadas pelos profissionais foi informada por meio de um formulário web. Todas as classes reconhecidas pelos participantes foram consideradas para posterior comparação com a lista obtida pelo método. Sendo assim, o conjunto de classes dos profissionais foi composto pelos seguintes termos: order (9), bill (5), stock (5), table bill (5), table

(4), customer (3), drink (3), drink order (3), food (3), food order (3), kitchen (3), bar (2), menu (2), menu item (2), product (2), waiter (2), manager (1), payment (1), stock item (1). O valor entre parênteses indica a quantidade de profissionais a reconhecer a classe.

Para comparar com as relações de classes reconhecidas pelo método e pelos profissionais, foram utilizadas métricas para modelos de classificação[1] buscando o número de verdadeiros positivos (TP), o número de falsos positivos (FP) e o número de falsos negativos (FN). Após a obtenção dos referidos valores, foram calculadas as métricas de classificação Precisão (P), Recall (R) e F1 Score (F1).

Nesse cenário foram encontrados 15 verdadeiros positivos no experimento, já que das 16 classes candidatas reconhecidas pelo método apenas uma (task) não consta na lista dos profissionais, sendo assim, o termo task foi o único falso positivo do experimento. Além disso foram encontrados 4 falsos negativos: product, payment, stock_item e menu_item.

Fundamentado nesses dados, chegou-se aos seguintes valores de P, R e F1:

- P = 15 / (15 + 1) = 0,938
- R = 15 / (15 + 4) = **0,789**
- F1 = 2 * 0,938 * 0,789 / (0,938 + 0,789) = **0,857**.

4.3. **Comparação entre os rankings**

O ranking obtido pelo método é apresentado na Tabela 1. Os requisitos encontram-se ordenados por posição de prioridade, seguidos da quantidade de links de dependência encontrados após análise automatizada. Como não foram definidos critérios adicionais de avaliação, as situações de empate foram mantidas, considerando o intervalo do empate.

Tabela 1: **Ranking obtido pelo método.**

Posição	Req.	Dependências
1	R3	8: R1, R2, R4, R5, R6, R7, R8, R9
2 - 6	R1	5: R2, R3, R4, R5, R9
2 - 6	R2	5: R1, R3, R4, R5, R9
2 - 6	R4	5: R1, R2, R3, R5, R9
2 - 6	R5	5: R1, R2, R3, R4, R9
2 - 6	R9	5: R1, R2, R3, R4, R5
7 - 9	R6	3: R3, R7, R8
7 - 9	R7	3: R3, R6, R8
7 - 9	R8	3: R3, R6, R7
10	R10	0: -

As posições ou intervalos do ranking gerado pelo método foram comparadas com as informações de prioridade fornecidas pelos profissionais. Os julgamentos dos 12 participantes compõem a Tabela 2, em que cada linha corresponde ao julgamento de um profissional, enquanto as colunas indicam as respectivas posições atribuídas.

Tabela 2: **Julgamento dos profissionais.**

Prof.	1	2	3	4	5	6	7	8	9	10
P1	R10	R9	R3	R4	R2	R1	R5	R6	R8	R7
P2	R1	R2	R3	R8	R4	R5	R6	R7	R9	R10
P3	R10	R9	R1	R2	R3	R7	R4	R5	R6	R8
P4	R10	R3	R1	R2	R7	R4	R5	R6	R8	R9
P5	R10	R9	R7	R3	R2	R1	R4	R5	R6	R8
P6	R7	R3	R1	R2	R5	R6	R4	R8	R10	R9
P7	R10	R7	R3	R1	R2	R4	R5	R9	R6	R8
P8	R3	R10	R1	R2	R6	R4	R5	R8	R9	R7
P9	R7	R3	R1	R2	R6	R9	R10	R8	R5	R4
P10	R3	R4	R1	R5	R2	R6	R7	R8	R9	R10
P11	R10	R7	R3	R1	R2	R9	R4	R5	R6	R8
P12	R7	R3	R1	R2	R4	R9	R5	R6	R8	R10

Como pode ser observado, existe uma grande variação entre as prioridades atribuídas. Isso se deve ao fato de que a essas avaliações contaram com critérios pessoais e subjetivos, muitas vezes não implementáveis computacionalmente. A referida variação pode ser melhor observada na Tabela 2 Tendo como exemplo R7, vemos que 5 profissionais o julgaram como sendo de prioridade muito alta (1ª ou 2ª posição), 2 como média/alta (entre 3ª e 5ª), 3 como média/baixa (entre 6ª e 8ª) e outros 2 como prioridade baixa (9ª ou 10ª posições), o que evidencia a diferença nas análises pessoais.

Logo, para comparar a proximidade atingida pelo Dependency Rank, foi considerada a quantidade de avaliações que cada requisito obteve dos profissionais no intervalo sugerido após processamento.

1 LEAL, R. S. (https://bit.ly/2keKqUt)

Analisando os dois ranking, pode-se observar que 6 dos 10 requisitos receberam dos profissionais, pelo menos metade das avaliações no mesmo intervalo sugerido pelo Dependency Rank, sendo:

- R1 (entre 2 e 6): 11 avaliações no intervalo
- R2 (entre 2 e 6): todas as avaliações no intervalo
- R4 (entre 2 e 6): 7 avaliações no intervalo
- R6 (entre 7 e 9): 8 avaliações no intervalo
- R8 (entre 7 e 9): 7 avaliações no intervalo
- R9 (entre 2 e 6): 6 avaliações no intervalo

Além disso, R3, classificado na primeira posição pelo método, estaria entre os 3 requisitos de maior prioridade para esse problema para 10 profissionais, sendo que para 2 deles, ocuparia exatamente a primeira posição.

O caso específico de R10 chama a atenção por ser o requisito com maior diferença ao comparar as avaliações profissionais com o ranking do método. Mesmo que esse requisito conste entre os de maior prioridade para a maioria dos profissionais, 3 entre 12 o julgaram como sendo o de menor prioridade, estando em concordância com o ranking sugerido.

5. Conclusões e trabalhos futuros

Este trabalho teve como objetivo desenvolver um método de priorização de requisitos de software, denominado Dependency Rank, capaz de identificar automaticamente as dependências entre o conjunto de requisitos de um software utilizando ferramentas de PLN.

O protótipo desenvolvido obteve precisão de 0,938, recall de 0,789 e F1 score de 0,857 na identificação de classes candidatas do experimento realizado, possibilitando a geração de um ranking sugerido que se aproximou das avaliações dos profissionais, já que para 6 requisitos o julgamento de prioridades esteve de acordo com, pelo menos a metade das avaliações, alcançando boa aproximação para outros 2 requisitos do conjunto.

Com base nisso, conclui-se que a adoção do critério de priorização de requisitos baseado em dependências identificadas por ferramentas de PLN, pode auxiliar desenvolvedores na tarefa de priorização. Ainda que não forneça resultados exatos, os experimentos mostram que é possível alcançar resultados aproximados.

Entende-se ainda que a utilização do método proposto em processos de software que contem com um grande conjunto de requisitos diminuiria a quantidade de tempo/esforço necessário para o julgamento dessas prioridades ou definição de incrementos, pois fornece uma perspectiva sobre as dependências entre eles.

Diante disso, identifica-se como possibilidades de trabalhos futuros, a implementação de rotinas que tenham como objetivo resolver ou, pelo menos, amenizar as limitações de subjetividade, complementando o método com recursos de aprendizagem de máquina; e utilização de verbos nos critérios de priorização, possibilitando criar uma estratégia para definir precedências entre os requisitos.

REFERÊNCIAS

[1] Achimugu, Philip, et al. "A systematic literature review of software requirements prioritization research." Information and software technology 56.6 (2014): 568-585.

[2] Wiegers, Karl. "First things first: prioritizing requirements." Software Development 7.9 (1999): 48-53.

[3] Pressman, Roger, and Bruce Maxim. Engenharia de Software-8ª Edição. McGraw Hill Brasil, 2016.

[4] Indurkhya, Nitin, and Fred J. Damerau, eds. Handbook of natural language processing. Vol. 2. CRC Press, 2010.

[5] Cambria, Erik, and Bebo White. "Jumping NLP curves: A review of natural language processing research." IEEE Computational intelligence magazine 9.2 (2014): 48-57.

[6] Shao, Fei, et al. "DRank: A semi-automated requirements prioritization method based on preferences and dependencies." Journal of Systems and Software 126 (2017): 141-156.

[7] Petersen, Kai, Sairam Vakkalanka, and Ludwik Kuzniarz. "Guidelines for conducting systematic mapping studies in software engineering: An update." Information and Software Technology 64 (2015): 1-18.

[8] Alzyoudi, Ruba, Khaled Almakadmeh, and Hutaf Natoureah. "A Probability Algorithm for Requirement Selection In Component-Based Software Development." Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication. ACM, 2015.

[9] Zhang, Zhang. Specifying Functional Requirements Dependency in the REWiki. MS thesis. 2013.

[10] Deshpande, Dhanraj and Joshi S. D. "Textual requirement analysis for UML diagram extraction by using NLP." International journal of computer applications, v. 50. 2012.

[11] Dahlstedt, Asa G., and Anne Persson. "Requirements interdependencies-moulding the state of research into a research agenda." Proceedings of Ninth International Workshop on Requirements Engineering: Foundation for Software Quality. 2003.

[12] Bourque, Pierre, and Richard E. Fairley. Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press, 2014.

[13] Schweinberger, Martin. Part-Of-Speech Tagging with R. < http://martinschweinberger.de/docs/articles/PosTagR.pdf>, 2016.