science, engineering and health studies

Research Article

Science, Engineering and Health Studies
https://lio1.tci-thaijo.org/index.php/sehs
ISSN (Online): 2630-0087

Automatic recommendation of developers for open-source software tasks using knowledge graph embedding

*Pisol Ruenin, Morakot Choetkiertikul, Akara Supratak and Suppawong Tuarob**

Faculty of Information and Communication Technology, Mahidol University, Nakhon Pathom 73170, Thailand

***Corresponding author**:
Suppawong Tuarob
suppawong.tua@mahidol.edu

Received: *10 July 2022*
Revised: *18 October 2022*
Accepted: *8 November 2022*
Published: *6 December 2022*

Citation:
Ruenin, P., Choetkiertikul, M., Supratak, A., and Tuarob, S. (2022). Automatic recommendation of developers for open-source software tasks using knowledge graph embedding. Science, Engineering and Health Studies, 16, 22020006.

ABSTRACT

For software development to succeed, qualified developers with the necessary abilities are required to provide a high-performance solution. Since people have a wide range of skills, considering a wide range of developers to include in a team is an integral part of the selection process. This problem becomes more aggravating in online open-source software settings, where developers from around the globe become viable candidates. This paper proposed a method for recommending developers for a specific software task using knowledge graph embedding. The knowledge graph using data from Moodle, an open-source software project housed in the JIRA platform, was crafted. The constructed knowledge graph represented the relationship among software development factors, such as skills, developers' collaboration, task dependencies, task locality, and task creation dates. The link prediction protocol was used to recommend a list of developer candidates. The comparison of techniques with the existing developer recommendation algorithms showed that the developed approach outperformed those state-of-the-art recommendation baselines. The experiment results are encouraging and shed light on the possibility of extending the proposed algorithm to recommend software team members for various other roles, such as reviewers, testers, and integrators.

Keywords: developer recommendation; knowledge graph embedding; graph representation learning

1. INTRODUCTION

Due to growing communities of collaborative software development, numerous open-source and proprietary software projects are available for all-around users. The stakeholders involved in those projects consider multiple factors before releasing quality products. Those projects contain both success and failure, which are caused by various reasons. The consequences directly affect users, who use the particular software, especially in a startup company that usually fails at the early stage (Giardino et al., 2014).

Successful software projects are found to have the right combination of scope, time, cost, team compatibility, and skills. The exploratory survey by Agarwal and Rathod (2006) showed that the main success criteria include the software project's scope to understand the developed software's functionality and limitation. In addition, the analysis of open-source software in Sourceforge (Singh, 2010) significantly impacts a successful project because of the collaboration network between developers. Therefore, appropriate developers for a specific project are essential for the software product in the management, development, and evaluation aspects. However, manually selecting developers is challenging, especially in online open-source software settings, because the project leader must evaluate many developer candidates from a large pool of software practitioners worldwide.

Due to the general selection of candidates being a challenging problem, developer recommendations have been proposed to solve the selection of the proper person for a particular software project. They consider several aspects of developers and software development information and use different techniques affecting the development of progress procedure. For instance, DevRec is a recommendation system that recommends the developer for open-source software projects (Zhang et al. 2017). RECAST is a software team recommendation algorithms (Tuarob et al., 2021) that use machine learning techniques to extract meaningful team features to recommend developers with a high chance of success (Liu et al., 2014). However, the past research did not directly use the network relation between developers and tasks that could potentially significantly impact a project's success. Moreover, the knowledge graph is a novel relationship representation technique, and it is adapted to various domains, including the recommendation system.

This work proposed the developer recommendation framework in online open-source software development, which applied the knowledge graph to represent the relations between defined users and task entities. As a case study, the proposed method was evaluated on the Moodle project due to its reputation and the completeness of the data. We evaluated the developer recommendation from Moodle's tasks and formulated the problem for the knowledge graph structure with the link prediction technique.

2. MATERIALS AND METHODS

2.1 Dataset

The Moodle dataset containing the data pertaining to software development activities in Moodle's projects and tasks was used. Moodle is a well-known open-source platform in education that has been continuously updated and maintained. It uses JIRA for tracking its collaborative activities. The tracking system contained task information as well as the developers, who solve the task. Figure 1 illustrates the detail of a real-world task from Moodle, generally containing various information to describe the task, such as the users who work on the task and the essential descriptions of items that need to be done.

The Moodle dataset used in this research contained 88,655 tasks from several software projects, and the development dates commenced from 05/09/2002 to 22/05/2019. The projects consisted of various roles, including developers, integrators, reviewers, and testers. Specifically, there were 450 developers, who corresponded to the requirements in different tasks. The historical task data comprised the details describing the task characteristics, such as task dependency, task components, task type, and users working it. To project these pieces of task information into a knowledge graph's perspective, we extracted entities and relations from the Moodle tasks and used them to construct the knowledge graph. Figure 2 illustrates example of a real-world development task in knowledge graph constructed from the Moodle dataset.

The complete knowledge graph merged multiple sub-knowledge graphs, including user role, user expertise, user collaboration, task dependency, task locality, and task creation date graphs. Table 1 presents the knowledge graph statistics generated from the Moodle dataset. Specific sub-knowledge graphs independently had their entities and relations. The user collaboration graph had the smallest size, with 734 user entities, and was fewer than the total number of developers, with 16 developers working independently. However, the task creation graph was the most enormous and densest knowledge graph, representing the temporal relation of task creation. It contained every task entity connecting to the temporal (month-year) entity sequence. Eventually, the complete knowledge graph had distinct entities that added all relations from sub-knowledge graphs. In addition, the software development issue dataset also contained other roles, such as tester, integrator, and reviewer.

The graph density is the ratio between the number of entities and relations. It can measure the number of edges that can add to the graph. The graph density can be calculated from the following equation, where m denotes the number of relations, and n denotes the number of entities.

$$GraphDensity = \frac{m}{n(n-1)} \quad (1)$$

2.2 Methodology

The developer recommendation framework was introduced for software development. The software development data was represented in a knowledge graph structure, which exposed the various relation and entity types. The constructed knowledge graph was interpreted from the embedding model before the developer candidates were listed according to the score from the scoring function into the top-K ranking for particular tasks. The high level of the proposed framework is illustrated in Figure 3.

2.2.1 Proposed collaborative software development knowledge graph

A knowledge graph is regularly used to represent the relationship between data. It is defined as several entities and relationships that can be interpreted as insightful knowledge. Since selecting a suitable developer is relevant to various factors from historical development and characteristics, such as developer collaboration, technical skills, and experiences, we transformed the Moodle dataset into various knowledge graph features. The factors were constructed into individual knowledge graphs and combined by unionizing entities and relations, of which the end entities were unique. The following items indicated the proposed knowledge graph types, relationships, and entities:

1. User role graph: The user role graph $G_{ur} = (E_u, E_t, R_r)$ is a directed graph where $e_u \in E_u$ represents a user (developer candidate), $e_t \in E_t$ represents a task entity, and $r_r \in R_r$ represents a role relation. This knowledge graph represents a developer as an entity connecting to a task entity to determine that he/she is a developer. The user role graph is also treated as the base graph since it contains the role relations that could be used to determine suitable developers for a given task.
2. User expertise graph: The user expertise graph $G_{ue} = (E_u, E_t, E_s, R_{us}, R_{ts})$ is a directed graph, where the user entity $e_u \in E_u$, task entity $e_t \in E_t$, skill entity $e_s \in E_s$. $r_{us} \in R_{us}$ is the relation of user expertise and $r_{ts} \in R_{ts}$ is the relation of a required task skill. The technical skills are represented by the system components of a developer's

previous tasks. The user and task entities connect to the technical skills entities that represent the current user skills and required skills for the tasks.

3. User collaboration graph: The user collaboration graph $G_{uc} = (E_u, R_c)$ is an undirected graph, where $e_u \in E_u$ is a user entity, and $r_c \in R_c$ is the relation of collaboration between two users. The software tracking system of Moodle consists of several past tasks and historical data of user work collaboration. Thus, the user entities mutually connect to represent an association between developers.

4. Task dependency graph: The task dependency graph $G_{td} = (E_t, R_d)$ is directed graph, where task entity $e_t \in E_t$ and dependency relation $r_d \in R_d$. The software tracking system has several dependencies between tasks to indicate their relatedness, such as duplication, clone, block, etc. Therefore, the task entities connect to other dependent tasks, and the relations are dependency types.

5. Task locality graph: The task locality graph $G_{tl} = (E_t, E_p, R_{in})$ comprises the tuples of task entity $e_t \in E_t$, the project entity $e_p \in E_p$, and $r_{in} \in R_{in}$, denoting the relation of tasks that locate in the project. The tasks in software development are usually determined to develop technologies or frameworks by the domain of a project. The task entities connect to the local relation to the project entities.

6. Task creation date graph: The task creation date graph $G_{tcd} = (E_t, E_d, R_c)$, where the task entity $e_t \in E_t$, the date entity $e_d \in E_d$, and $r_c \in R_c$ denotes the task creation relation. The task creation date indicates the month and year entities chronologically connected and represents the time series of dates. Date entities link to task entities to denote the creation date. The time series of tasks could potentially allow the model to focus on more recent tasks and improve the selection of a suitable developer.

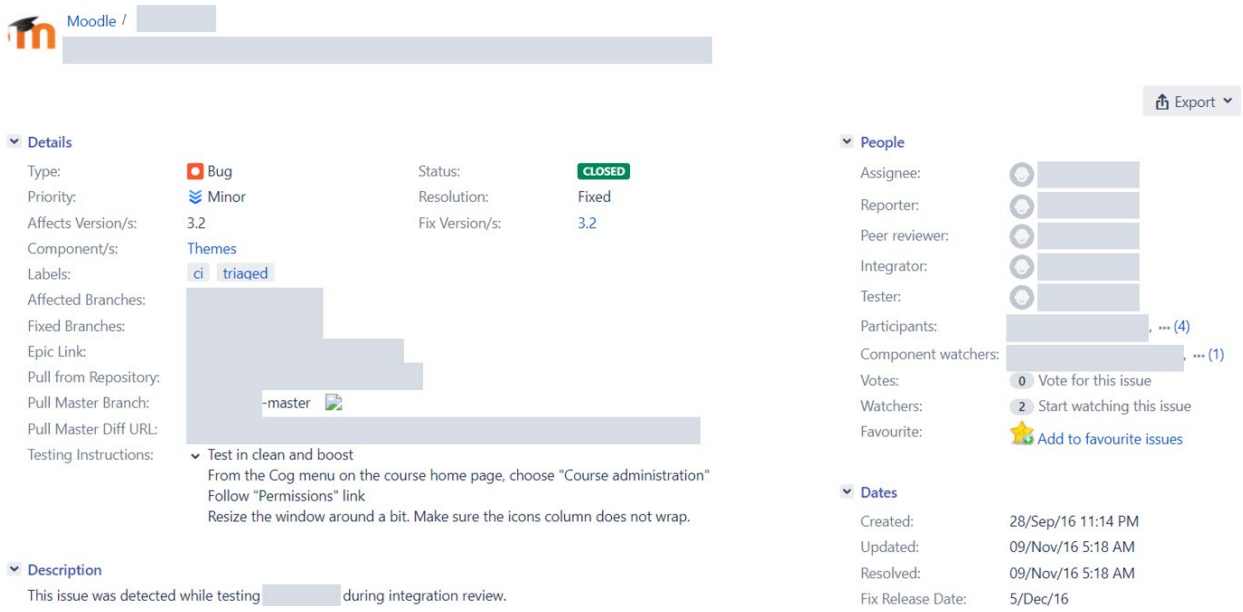

Figure 1. Example of a task description from Moodle

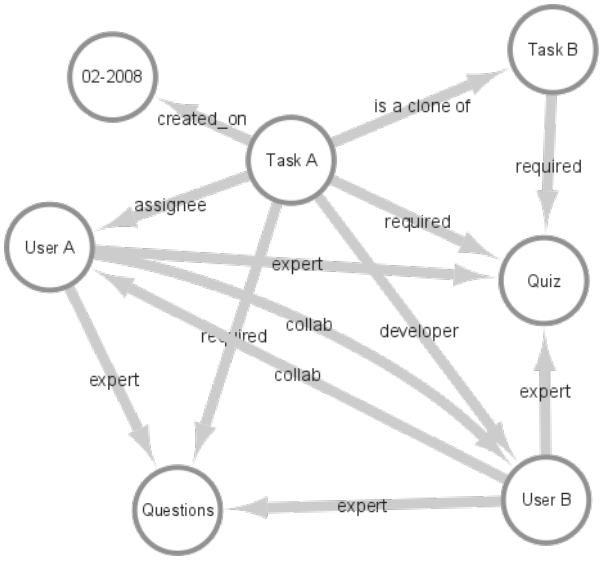

Figure 2. Example of a real-world software development task in knowledge graph representation

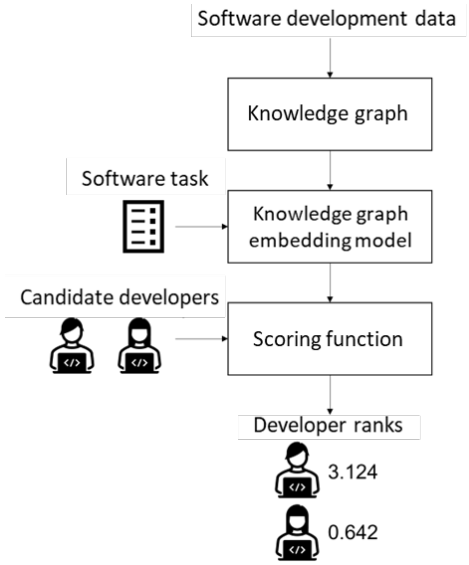

Figure 3. Proposed method for developer recommendation

Table 1. Knowledge graph statistics from the Moodle dataset

Statistics	Value
#User role nodes	71,928
#User role edges	162,191
#User expertise nodes	31,299
#User expertise edges	28,943
#User collaboration nodes	734
#User collaboration edges	13,660
#Task dependency nodes	31,299
#Task dependency edges	28,943
#Task locality nodes	88,673
#Task locality edges	88,665
#Date creation nodes	88,871
#Date creation edges	88,870
#Total nodes	90,666
#Total edges	505,759
Density	6.15×10^{-5}

2.2.2 Knowledge graph embedding

A knowledge graph is a combination of a mixture of entities and relations. Ordinarily, the representative of the knowledge graph is a set of triples. It consists of the subject, predicate, and object with the (sub, pred, obj) notation. The types of entities and relations are specific to the problem domain and application.

Knowledge graph embedding is a technique to semantically project the knowledge graph components, including entities and relations, onto a low-dimensional vector space. Different embedding models have different scoring functions, depending on their architectures and underlying computational principles. The scoring function guides the embedding model to optimize each triple's vector to minimize the loss. The following knowledge graph embedding models were evaluated in this work, where e_s denotes a subject entity vector, e_o denotes an object entity vector, and r_p denotes a predicate relation vector.

1. TransE proposed the L_1 or L_2 norm between embedded subject entities using the difference between the embedded predicate relation and embedded object entity (Bordes et al., 2013).

$$S_{TransE} = -\|e_s + r_p + e_o\|_n \tag{2}$$

2. DistMult proposed the trilinear dot product (Yang et al., 2015).

$$S_{DistMult} = \langle r_p, e_s, e_o \rangle \tag{3}$$

3. ComplEx proposed the extension of DistMult by using the trilinear Hermitian dot product (Trouillon et al., 2016).

$$S_{ComplEx} = Re\left(\langle r_p, e_s, \overline{e_o} \rangle\right) \tag{4}$$

4. HolE proposed the circular correlation re-defined from the ComplEx where n is the dimension of the complex embedding (Nickel et al., 2016).

$$S_{HolE} = \frac{2}{n} S_{ComlEx} \tag{5}$$

5. ConvKB proposed convolution layers with the dot product where *concat* is the concatenate operator, g is the non-linear activation function, $*$ is the linear convolution operator, Ω is a set of filters, and W is a weight vector (Nguyen et al., 2017).

$$S_{ConvKB} = concat\left(g\left(\left[e_s, r_p, e_o\right] * \Omega\right)\right) \cdot W \tag{6}$$

Since each triple score represents its plausibility, which is essential for performing the missing link prediction protocol, therefore, we can rank the top-K triple combinations of each task by considering the score that represents the chance of a candidate developer being chosen for the task if he/she were to be selected by the assignee. Note that since different knowledge graph embedding algorithms have different scoring functions, the score range of each function is also distinct and should not be directly compared across different embedding algorithms.

2.2.3 Evaluation

To measure the developer recommendation performance and to allow a fair comparison with the baselines, we adopted the evaluation protocol and metrics used by RECAST (Tuarob et al., 2021), including mean reciprocal rank (MRR), mean rank (MR), Hit@K, mean average precision (MAP). These evaluation metrics are defined as follows:

1. MRR is an average of the correct recommended developers' multiplicative inverse ranks, where Q is the query set, and $rank_p$ is the rank of the first correctly recommended developer.

$$MRR = \frac{1}{|Q|}\sum_{p \in Q} \frac{1}{rank_p} \tag{7}$$

2. MR is an average of true recommended developers' ranks, where Q is the query set, and $rank_p$ is the rank of the correct recommended developer.

$$MR = \frac{1}{|Q|}\sum_{p \in Q} rank_p \tag{8}$$

3. Hit@K is a ratio between the number of correct recommended developers in top-K ranks and total recommended developers where K is the number of considered recommended developers, Q is the query set, and $H\left(d_{i_0}, d_{i_{1..k}}\right)$ is a function that returns 1 if task i's actual developer d_{i_0} is in recommended developers no more than K-th rank, otherwise returns 0.

$$Hit@K = \frac{1}{|Q|}\sum_{i=1}^{|Q|} H\left(d_{i_0}, d_{i_{1..k}}\right) \tag{9}$$

4. MAP is an average evaluation of recommendation precision and is computed at considered several top-K recommended developers where Q is the query set, K is the number of considered recommended developers, i.e., $K = 1,2,3,\ldots,10$, and $H(d_0, d_i)$ is a function that returns 1 if developer d_0 is equivalent to d_i, otherwise returns 0.

$$Precision@K = \frac{\sum_{i=1}^{K} H(d_0, d_i)}{K} \tag{10}$$

$$MAP = \frac{1}{|Q|}\sum_{j=1}^{|Q|} \frac{\sum_{k=1}^{10} Precision@K}{10} \tag{11}$$

3. RESULTS

3.1 Knowledge graph embedding evaluation

To choose the most suitable knowledge graph model for the software development data, we experimented with and compared different embedding models in terms of the ability to encode the semantics of the knowledge graph. Accordingly, we randomly dropped 20% of the user role triples as a test set, and the rest is the training set. The knowledge graph embedding encoded the entities and relation types into a vector of 200 dimensions. Then, the link prediction found the true entity using the scoring function to calculate the triples' scores from the test set and ranked all user entities according to the scores. We used MRR as the main evaluation criteria while also considering MR, Hit@10, and Hit@100.

The evaluation results from Table 2 were evident that TransE cannot capture the semantics of the knowledge graph, resulting in the worst performance in all metrics. Interestingly, HolE performs on par with DistMult, but the MR of HolE was quite high because it could not interpret some entities or relation types, consequently increasing the average ranks. However, ConvKB outperformed other models. Although ConvKB's MRR was equal to ComplEx's at 0.43, it overcame ComplEx in other metrics, including MR, Hit@10, and Hit@100. Thus, the ConvKB was chosen as the graph embedding model for the next evaluation on the developer recommendation task.

After the embedding model learns the semantics of the knowledge graph, the entities and relations would become a vector. According to the example embedded developer vectors from Figure 4, the developers with comparable characteristics were located close to each other when converted to embedded vectors. Here, the principal component analysis was used to reduce the dimensions for visualization. Furthermore, the embedding space was also generally close together to the same as other entity and relation types.

Table 2. Knowledge graph embedding evaluation result comparison between TransE, DistMult, ComplEx, HolE, and ConvKB on the Moodle dataset

Model	MRR	MR	Hit@10	Hit@100
TransE	0.00	787.05	0.00	0.01
DistMult	0.38	88.99	0.56	0.79
ComplEx	0.43	90.66	0.61	0.81
HolE	0.38	269.93	0.52	0.64
ConvKB	0.43	18.56	0.69	0.96

Note: MRR = mean reciprocal rank, MR = mean rank, and MAP = mean average precision

3.2 Developer recommendation evaluation

The developer recommendation applied the ConvKB, the most appropriate model for our dataset. The developer candidates comprised the users who used to be developers in the previous tasks. Besides, the developers were filtered by their activeness, where we considered the developers who had activities within three months before creating a task. The scoring function calculated all developer candidates that were combined with task entities and developer relations for compatibility with the link prediction protocol. The top-K triples with the highest scores would be listed to present the most suitable developers for a particular task.

According to the evaluation results in Table 3, our approach surpassed the baselines, RECAST (Tuarob et al., 2021) and Liu et al. (2014), in terms of MRR, MR, Hit@10, Hit@100, and MAP. Specifically, our recommendations had 93% correct developers in the top 100 ranks of the tasks in the test set, but RECAST (Tuarob et al., 2021) and the method of Liu et al. (2014) yielded only 45% and 46%, respectively. Furthermore, on average, a correct developer was ranked among the top 10 recommendations. Our method's MRR and MAP also outperformed RECAST (Tuarob et al., 2021) by over 120%

Table 4 shows an example of recommended developers for a task in the Moodle dataset. The usernames were anonymized for privacy. They are uniquely represented as Uxxx, and the user numbers were sequentially enumerated. This example task was related to removing the personal data of inactive users. In this task, our approach could raise the proper developer's rank to first rank, while RECAST had the correct developer at the tenth rank, and Liu's method could not find the correct developer in the top-ten results at all.

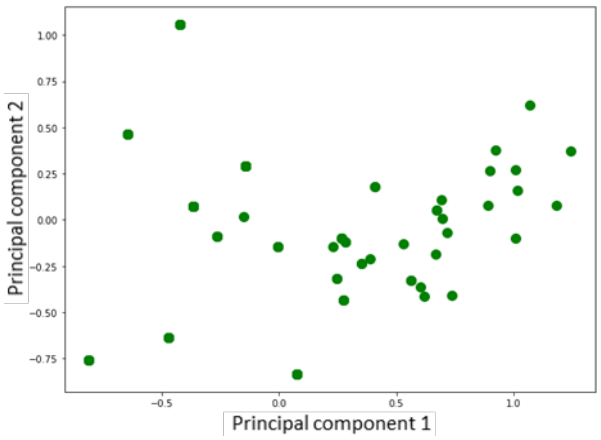

Figure 4. Example of randomly selected developers' embedded vectors projected onto the two-dimensional space using principal component analysis

Table 3. Developer recommendation evaluation results on the Moodle dataset

Metric	Liu	RECAST	Current
MRR	0.08	0.20	0.44
MR	76.53	62.78	23.15
Hit@10	0.12	0.41	0.59
Hit@100	0.45	0.46	0.93
MAP	0.07	0.20	0.43

Note: MRR = mean reciprocal rank, MR = mean rank, and MAP = mean average precision

Table 4. Example of top 10 recommended developers by Liu et al. (2014), RECAST (Tuarob et al., 2021), and the current approach

Assignee	U100		
Developer	U101		
Rank	**Liu**	**RECAST**	**Current**
1	U102	U112	U101
2	U103	U113	U113
3	U104	U114	U121
4	U105	U115	U122
5	U106	U116	U123
6	U107	U117	U117
7	U108	U118	U124
8	U109	U119	U125
9	U110	U120	U126
10	U111	U101	U127

4. DISCUSSION

The developer recommendation attempted to serve the most suitable developer for the task. Various recommendation techniques were used to solve the problem, emphasizing the different aspects of what considers an appropriate developer. Zhang et al. (2017) presented the developer recommendation, DevRec. Their algorithm considered two aspects, including development activity and knowledge-sharing activity. Social coding activities focus on Github's commits, forks, and watches of the specific repositories to represent developers' interests. Knowledge-sharing activities focus on tags in particular posts that the developers answered or asked questions on StackOverflow. Tuarob et al. (2021) proposed RECAST, a machine learning based software team recommendation algorithm that could be tweaked for individual role recommendation. They collected large-scale software development datasets from successful open-source software tasks, including Moodle, Apache, and Atlassian. Their approach considers role requirements, technical skills, and team compatibility. The machine learning techniques, such as Logistic Regression and Random Forest, were applied as scoring functions before ranking the top-K candidate teams. Although the RECAST's main objective is to recommend suitable software teams for a task, it also has a single-role recommendation protocol for developers, integrators, testers, and reviewers. They presented several evaluation metrics, including MRR, MR, precision, and MAP. Additionally, they formulated the single recommendation protocol on Liu's work which also proposed a team recommendation algorithm that applies machine learning to learn the weights of team quality and experience features from historical project information. In their evaluation, the results in developer recommendations are compared to Liu's method and a random baseline.

The knowledge graph was also adopted in addition to software development. Ye et al. (2021) proposed a framework for drug-target interactions. They use the knowledge graph to learn a low-dimensional representation for each entity before combine with a neural factorization machine. This work consists of four main parts: discovering heterogeneous information, dimensional reduction with principal component analysis, information integration, and collaborative recommend-dation. Gong et al. (2021) presented a framework for safe medicine recommendations. They used knowledge graph embedding to compress the medical information to lower dimensional space. The link prediction considers the diagnoses and drug reactions.

5. CONCLUSION

The developer suggestion for specific software development tasks was proposed by applying the knowledge graph technology to the Moodle dataset. The entities and relations in the knowledge graph were encoded into a latent vector using a graph embedding algorithm, which was then used to generate the latent vectors for entities and relations. The candidate developers were selected based on the results of the scoring function, which was used to determine their suitability for the task. The evaluation showed that the proposed approach outperformed the state-of-the-art techniques in all aspects.

Due to the fact that a software team comprises a variety of roles that are not exclusively for developers, we intend to extend the proposed developer recommendation to include these roles in our future work. Furthermore, because software development typically involves a team comprising a diverse range of individuals and roles to produce a high-quality product, we would like to modify the knowledge graph to recommend an entire team of software practitioners for a given task. In addition, we wish to investigate other open-source software projects to better understand their collaborative behaviors to improve our recommendation algorithms for generalization.

ACKNOWLEDGMENT

This research project was partially supported by Faculty of Information and Communication Technology, Mahidol University, Thailand.

REFERENCES

Agarwal, N., and Rathod, U. (2006). Defining 'success' for software projects: An exploratory revelation. *International Journal of Project Management*, 24(4), 358-370.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the 27th Conference on Neural Information Processing Systems*, 1282. Nevada, USA.

Giardino, C., Wang, X., and Abrahamsson, P. (2014). Why early-stage software startups fail: A behavioral framework. In *Proceedings of Software Business. Towards Continuous Value Delivery*, pp. 27-41. Paphos, Republic of Cyprus.

Gong, F., Wang, M., Wang, H., Wang, S., and Liu, M. (2021). SMR: Medical knowledge graph embedding for safe medicine recommendation. *Big Data Research*, 23, 100174.

Liu, H., Qiao, M., Greenia, D., Akkiraju, R., Dill, S., Nakamura, T., Song, Y., and Nezhad, H. M. (2014). A machine

learning approach to combining individual strength and team features for team recommendation. In *Proceedings of the 13th International Conference on Machine Learning and Applications*, pp. 213-218. Michigan, USA.

Nguyen, D. Q., Nguyen, T. D., Nguyen, D. Q., and Phung, D. (2017). A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 327-333. Louisiana, USA.

Nickel, M., Rosasco, L., and Poggio, T. (2016). Holographic embeddings of knowledge graphs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp 1955-1961. Arizona, USA.

Singh, P. V. (2010). The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. *ACM Transactions on Software Engineering and Methodology*, 20(2), 6.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on Machine learning*, pp. 2071-2080. New York, USA.

Tuarob, S., Assavakamhaenghan, N., Tanaphantaruk, W., Suwanworaboon, P., Hassan, S. U., and Choetkiertikul, M. (2021). Automatic team recommendation for collaborative software development. *Empirical Software Engineering*, 26, 64.

Yang, B., Yih, W. T., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the 3rd International Conference on Learning Representations*, arXiv:1412.6575. California, USA.

Ye, Q., Hsieh, C. Y., Yang, Z., Kang, Y., Chen, J., Cao, D., He, S., and Hou, T. (2021). A unified drug-target interaction prediction framework based on knowledge graph and recommendation system. *Nature Communications*, 12, 6775.

Zhang, X., Wang, T., Yin, G., Yang, C., Yu, Y., and Wang, H. (2017). DevRec: A developer recommendation system for open source repositories. In *Proceedings of the 16th International Conference on Software Reuse*, pp. 3-11. Salvador, Brazil.