A Similarity-Based Software Recommendation Method Reflecting User Requirements

Se In Baek, Yang-Eui Song, and Yong Kyu Lee

Department of Computer Science and Engineering, Dongguk University, Seoul, Korea

ljfis

Abstract

Existing software recommendation methods consider only the usage frequencies of software as far as we know. In this study, we propose a software recommendation method reflecting user requirements based on both the Boolean model and vector space model. A function matrix and function vector are made from the functional specification of each software type and stored in the database. First, it creates a requirement vector from a user's functional requirements of the desired software. Second, it makes a list of software with the same functions wanted using the function matrix based on the Boolean model. Third, the cosine similarities are calculated between the requirement vector and function vectors of the software in the list based on the vector space model. Finally, a software recommendation list is generated in descending order of similarity. Based on the experiment results, appropriate software well suited for user requirements can be recommended. This is because we searched for software that satisfies each user's requirements by using the cosine similarity function of information retrieval and recommended it according to the ranking. In the future, performance can be improved by utilizing statistical search techniques.

Keywords: Software recommendation, User requirements, Boolean model, Cosine similarity, Vector space model, Recommendation system

1. Introduction

In recent years, the scale of software has increased and developed. Owing to the advancement of various types of software development, coding education is currently being taught in elementary schools. As coding education expands, it is becoming important to select the right software according to the development environment. Software recommendations provide software usage and popularity rankings considering only the usage frequency of the software as far as we know [1–4]. These methods have the limitation that user requirements are not considered for a software recommendation. Furthermore, not having expert knowledge limits users in terms of selecting the appropriate software.

This paper proposes a method for recommending software by considering user requirements based on both the Boolean model and vector space model. To recommend software that reflects user requirements, a function matrix and function vector are generated from the functional specification of each software and stored in the database. The function matrix and function vector are extracted based on software functional specifications. In addition, the function

Received: Jul. 30, 2020
 Revised : Sep. 15, 2020
 Accepted: Sep. 15, 2020

Correspondence to: Yong Kyu Lee
 (ykleee@dongguk.edu)
 ©The Korean Institute of Intelligent Systems

©This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

vector is generated based on an analysis of the data evaluated by users. Furthermore, the functions required by a user and a vector are generated. For the user's functional requirements, the criteria users want can be set up through the addition of an importance condition according to software functions. When the user's requirements are entered, four phases are processed as follows. First, a requirement vector is generated from a user's functional requirements for the desired software. Second, a list of software is generated, which has the same wanted functions using the function matrix based on the Boolean model. Third, the similarities between the requirement vector and function vectors are calculated based on the generated software list. Similarities are calculated using the cosine similarity method of the vector space model. Finally, a software recommendation list is generated in descending order of similarity. It is expected that search performance will be improved by using the Boolean and vector space models. Based on the calculated similarity, an appropriate software recommendation list is generated for the user. Using the proposed method, software that reflects user requirements and the usage frequency thereof can be recommended.

The remainder of this paper is structured as follows. Section 2 describes previous related research, and Section 3 describes the proposed similarity-based software recommendation method that reflects user requirements. Section 4 discusses the test and evaluation results of the proposed method, and finally, Section 5 provides concluding remarks.

2. Related Research

Alongside the advancement in information communication, various recommendation systems are being developed [5, 6]. Conventional recommendation systems analyze user preferences or patterns and recommend appropriate items for users. Accordingly, to implement a personalized recommendation system, studies have been conducted on various recommendation methods such as information filtering methods and correlation analyses [7–9].

Of the various recommendation methods available, collaborative filtering methods are mainly used. Collaborative filtering methods can be delineated as user- and item-based methods. User-based collaborative filtering methods select neighboring users with a similar propensity as the customer by using the preference information input by the user. The methods then recommend the items commonly preferred by the selected neighbors. Item-based collaborative filtering methods select items

that have similar user evaluation scores as the neighbor items, and then predict the preference of the recommendation target customer for a certain item based on the evaluated scores of these items [10]. This calculation of similarity between users or items is the most important stage in collaborative filtering. Typical similarity calculation methods include the Boolean model, Euclidean distance, Jaccard distance, and cosine similarity in information retrieval [11–13].

The Boolean model is a simple search model based on set theory and Boolean algebra, which is very intuitive and accurate. In the Boolean model, the index term is represented by its existence in the document, so the term-document frequency matrix has only a binary value. Query q consists of an index and three Boolean operators: AND, OR, and NOT [12]. The Euclidean distance calculates a similarity by calculating the distance between the vectors consisting of n points (real numbers) in an n dimensional Euclidean space. The distance is multiplied in each dimension, and after obtaining the sum of the values, the positive square root is calculated [13]. A Jaccard similarity shows the similarity between two groups, and is a proportion of the intersection size in the union size of two groups. A Jaccard distance is the opposite concept of a Jaccard similarity. This value is obtained by subtracting the proportion of the intersection size in the union size of sets X and Y from 1 [13]. The cosine similarity method measures how similar two vectors are by using the cosine value of the angle between the two vectors in the inner product space. The cosine similarity is calculated using Eq. (1). The same propensity is indicated if the resulting value is 1, and a different propensity is indicated if the value is zero [14].

$$\text{Cosine similarity} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}. \quad (1)$$

Typical recommendation systems include those used by GroupLens Research [15], Amazon [16], and Netflix [17]. However, there is a lack of studies on software recommendation systems for regular users and developers. Conventional software recommendation systems provide recommendations by considering the usage frequency and popularity of the software as far as we know [1–4].

In software recommendation methods according to user requirements [18], a software vector is generated using the functional specification of the software. In this study, a software function vector was generated by additionally analyzing information of the software evaluated by the user. It was improved in that the user's evaluation opinion was reflected.

Other studies related to software recommendation include a data-mining-based recommendation system for software evolution tasks [19], a requirements alternative flow detection method using use case patterns in a use case scenario [20], and a developer recommendation technique based on a topic model and social network [21]. Another study related to recommendations considering user characteristics includes the integration of user profiles and real-time context information reflecting time-based changes for the recommendation system [22].

These software recommendation methods have the limitation that user requirements cannot be reflected. Furthermore, if regular users who lack development skills have no prior knowledge, their ability to select the appropriate software is limited. Therefore, this study aims to propose a software recommendation method that reflects user requirements using the Boolean model and vector space model.

3. Software Recommendation Method

This study proposes a software recommendation method that reflects user requirements. The proposed method consists of a vector generation process and software retrieval process. Figure 1 shows the flow of software recommendation.

The proposed method consists of a vector generation process and software retrieval process. The vector generation process consists of the function matrix and function vector generation stage, user requirement functions, and a requirement vector generation stage. The software retrieval process consists of a software list generation phase using the Boolean model, and a similarity calculation phase based on the software list. The software recommendation list is generated based on the calculated similarity. The software list is generated using the Boolean model including the mandatory functions of user requirements. Then, the similarity between the requirement vector and function vectors are calculated based on the generated software list. The software recommendation list is generated based on the similarity. We expect that this will improve the search speed.

3.1 Generation of the Function Matrix and Function Vectors

To recommend software that reflects user requirements, the function matrix and function vectors are generated. To this end, the function matrix is defined based on software functional specifications, and the function vectors are assigned based on data collected from the web. The software information can be obtained by referencing data such as a dictionary or official

Figure 1. Flow of software recommendation.

homepage specifying the software information or conditions. Furthermore, it can be used by referring to the results of analyzing the data evaluating the software from users. Software can have various functions, and such functions can be added or modified according to user requirements. The generated function vectors are then stored in the database.

Table 1 shows the structuring of functions used as an example of software functions.

The software functions include the category, supported operating system, hardware specifications, programming language, other functions, usage level, and so on.

The categories include the Programming Language, Database Management System, and Web Design Programming according

Table 1. Software functions

Value	Function group	Functions
Boolean	Classification	Programming language, Database management system, Web design program, Editor program
Boolean & vector	Operating system	Linux, Windows, FreeBSD, Solaris
	Programming language	C, C++, Java, PHP
	Functions	Function1, Function2, Function3, Function4
Vector	Number of functions	Degree value
	Scalability	
	Security level	

to the software use. The operating systems are base development environments for the use of software, and can be classified as Linux, Windows, and other systems. The programming languages are classified as C, C++, Java, and other development languages. The number of functions, scalability, and so on are functions that can be quantified. The functions can be classified and defined based on the software category.

The function vector values of the software are calculated based on the data collected from the web and statistical data. Eq. (2) is used to calculate the value of the function vector value, v_s .

$$v_s = (b * w_b) + \left(\frac{f}{gf} * w_f \right). \quad (2)$$

Here, b indicates the existence/non-existence of the corresponding function (Boolean value), f is the usage frequency of the corresponding function, and gf is the sum of the usage frequencies of the function group the corresponding function belongs to. In addition, w_b is the weight of the Boolean, and w_f is the weight of the usage frequency. Each has a value of between zero and 1 inclusively. Moreover, the sum of w_b and w_f is 1. As default values, w_b is set to 1, and w_f is set to zero, although each weight can be changed according to the user's choice. By setting the weights, the usage frequency can be considered. However, if the absolute value is the same as the classification, OS, w_b is set to 1, and w_f is set to zero. These functions are used in the Boolean model. The value calculated through Eq. (2) is represented as a function vector.

The function indicating degree such as number of functions, scalability, security level, and processing speed is normalized to a value between zero and 1. Normalization is calculated using Eq. (3).

$$v_n = \frac{v_f - \min(n)}{\max(n) - \min(n)}. \quad (3)$$

Here, v_n is the normalized degree of the function. n is a set of values $\{v_1, v_2, \dots, v_n\}$ for each function of the software, and v_f is the degree value of the function.

3.2 Generation of the Requirement Vector

To reflect user requirements, the functions and vector of user requirements are generated. The functions of user requirements are defined, and their values assigned according to the criteria of the user requirements. The functions of user requirements are calculated based on the importance conditions selected by the user. First, after assigning the mandatory/optionality function condition, the importance condition for the remaining functions is selected. For example, suppose that a user selects software functions a, b, c, d , and e , and assigns a and c as mandatory conditions. The vector values can then be given to the software functions selected by the user, as shown in Table 2. The functions a, c , and b, d , and e are assigned values between 0 and 1, respectively.

In this paper, the importance conditions refer to the importance ratio and priority, and the user can select any desired condition. The user requirement vector values are calculated according to the respective importance conditions, and herein, the sum of the respective user requirement vectors is 1. Next,

Table 2. Vector value according to mandatory/optionality condition

Function	Mandatory/Optionality condition	Vector value
a	Mandatory	1
b	Optional	0-1
c	Mandatory	1
d	Optional	0-1
e	Optional	0-1

with respect to the importance conditions, explanations are provided for the processes used to calculate the requirement vector values according to the assignment of the importance ratio value, calculating the requirement vector values according to the assignment of priority.

When a user assigns an importance ratio value as an importance condition for a software function, the requirement vector value, v_i , is calculated using Eq. (4). The calculated value indicates the value according to the importance ratio of the software function set up by the user. The user-assigned ratio value of the software function is the proportion of the ratio value of the corresponding function in the total sum, and the sum of all function values is 1.

$$v_i = \frac{I_k}{\sum_{k=1}^n I_k}. \quad (4)$$

Here, I_k is the ratio value for the corresponding function value, and n is the number of functions selected by the user. The weight is calculated by calculating the sum of all ratios. The value calculated through Eq. (4) is represented as a user vector. By setting the importance ratios for the software functions selected by the user, appropriate software for user requirements can be recommended.

When a user assigns a priority to an importance condition for a software function, the requirement vector value, v_p , is calculated using Eq. (5). The calculated value refers to the value obtained according to the priority of the software function set up by the user. The sum of all vector values is 1.

$$v_p = \frac{P_k^{-1}}{\sum_{k=1}^n P_k^{-1}}. \quad (5)$$

Here, P_k is a value assigned for the priority of the corresponding function value, and n is the number of functions selected by the user. To calculate the weight according to the priority, the priority value of each function is processed as a reciprocal, and the total sum of the values is calculated and used. The value calculated through Eq. (5) is represented as a requirement vector. By setting the priorities for the software functions selected by the user, appropriate software can be recommended for the user requirement.

The requirement vector value, v_r , according to the user requirement is defined as Eq. (6) by assigning the importance ratio and priority, among other factors. Depending on the situation, other conditions can be added beyond these two, and according to such conditions, the ratios are calculated. The total

Table 3. Software-function matrix

Function group	Function	SW-A	SW-B	SW-C	SW-D	SW-E
Operating system	Windows	1	1	1	1	0
	Linux	1	1	0	1	0
	FreeBSD	0	1	1	0	1
	Solaris	0	0	1	0	0

value of the ratios is set to 1.

$$v_r = \begin{cases} \text{Importation ratio,} & v_i, \\ \text{Priority,} & v_p. \end{cases} \quad (6)$$

3.3 Generation of Software List Using the Boolean Model

We used the Boolean model based on functions with Boolean values in the user requirements functions set. We then generated a software list using the Boolean model. To generate the software list, we used the generated software-function matrix.

Table 3 shows an example of the software-function matrix. If the operating system in the user's requirement is "Windows AND Linux", we take the vectors for Windows and Linux, and then do a bitwise AND. $11110 \text{ AND } 11010 = 11000$ results are SW-A, SW-B, and SW-D. The Boolean model can be used for any query in the form of a Boolean expression of functions, that is, in which functions are combined with the operators AND, OR, and NOT. We generated the software list by using the Boolean model, and pass passed the software list on to the next step.

3.4 Similarity Calculation between the Requirement Vector and Function Vectors

To recommend software that reflects user requirements, the similarity between the user's requirement function and software function is calculated. To calculate the similarity, a cosine similarity equation of the vector space model is used. With the cosine similarity calculation method, the similarity is measured using the cosine value between the requirement vector and function vectors. The cosine value has a value of between zero and 1 inclusively, and when it approaches 1, the user requirement and software function are determined as similar. The cosine similarity between the requirement vector and function vectors is calculated using Eq. (7).

$$\text{Sim}(\vec{s}, \vec{u}) = \frac{\sum_{i=1}^n s_i u_i}{\sqrt{\sum_{i=1}^n s_i^2} \sqrt{\sum_{i=1}^n u_i^2}}. \quad (7)$$

Here, s is the function vector value, and u is the requirement vector value. The similarity between the requirement vector and function vectors are calculated, and the software recommendation list produced by sorting the similarity values in descending order.

4. Experiment

To verify the software recommendation method proposed in this study, a test was conducted using an example recommending a database management system (DBMS).

4.1 Functional Specifications of Software

The types and functions of DBMS were defined by referencing functional specifications provided by DB-Engines [4] as of July 2020. Table 4 shows some of the DBMS functional specifications.

4.2 Recommendation Result

Based on Table 4, a user can select the requirement conditions. Table 5 shows some of the requirements selected by user A.

The value column is chosen for each software function, and the importance ratio is selected as an importance condition.

Table 6 shows the requirements selected by user B. The value column is chosen for each software function, and the priority is selected as an importance condition.

Using Eqs. (2) and (3), the function vector values are calculated. The requirement vector values for users A and B are calculated using Eq. (6) according to the selection of the importance conditions. According to user A's requirements, the importance ratio is set and the requirement vector value calculated using Eq. (4). According to user B's requirements, the priority is set, and Eq. (5) is used.

According to the requirements of users A and B, the functions of "Primary database model" and "Server operating systems" are Boolean values. Therefore, the DBMS list including the required functions is generated.

As a result, DBMS-A, B, C, and D were generated. Based on the generated DBMS list, the similarity is calculated using Eq. (7).

The DBMS recommendation list according to the requirements of user A is shown in Table 7, whereas the list according to the requirements of user B is shown in Table 8.

Table 4. Functional specifications of DBMS

Software	Primary database model	Server operating systems	Data scheme	XML support	Supported programming languages
DBMS-A	Relational DBMS	AIX, HP-UX, Linux, OSX, Solaris, Windows, z/OS	Yes	Yes	C, C#, C++, Clojure, Cobol, Delphi, Eiffel, Erlang, Fortran, Groovy, Java, JavaScript, Lisp, ...
DBMS-B	Relational DBMS	FreeBSD, Linux, OSX, Solaris, Windows	Yes	Yes	Ada, C, C#, C++, D, Delphi, Eiffel, Erlang, Haskell, Java, JavaScript, Objective-C, OCaml, Perl, PHP, Python, ...
DBMS-C	Relational DBMS	Linux, Windows	Yes	Yes	C++, Delphi, Go, Java, JavaScript, PHP, Python, R, Ruby, VisualBasic, ...
DBMS-D	Relational DBMS	Linux, OSX, Solaris, Windows	Schema-free	-	Actionscriptinfo, C, C#, C++, Clojureinfo, ColdFusioninfo, Dinfo, Dartinfo, Delphiinfo, Erlang, Java, JavaScript, Lispinfo, ...
DBMS-E	Key-value store	BSD, Linux, OSX, Windows	Schema-free	No	C, C#, C++, Clojure, Crystal, D, Dart, Elixir, Erlang, Fancy, Go, Haskell, Haxe, Lisp, Java, JavaScript, Lua, Perl, PHP, ...
DBMS-F	Wide column store	BSD, Linux, OSX, Windows	Schema-free	-	C#, C++, Clojure, Go, Erlang, Java, Haskell, Perl, PHP, Python, Ruby, Scala, ...

Table 5. User A's functional requirements

No	Function	Value	Importance condition
1	Primary database model	Relational DBMS	Mandatory
2	Server operating systems	Windows	Mandatory
3	Data scheme	Yes	50
4	XML support	Yes	80
5	APIs and other access methods	ADO.NET	70
6	Supported programming languages	PHP, Python	60
7	Triggers	Yes	40
8	Foreign keys	Yes	60
9	In-memory capabilities	Yes	30

Table 6. User B's functional requirements

No	Function	Value	Importance condition
1	Primary database model	Relational DBMS	Mandatory
2	Server operating systems	Linux	Mandatory
3	Data scheme	Yes	4
4	XML support	Yes	3
5	Supported programming languages	Python	1
6	In-memory capabilities	Yes	2

Table 7. DBMS recommendation list according to user A's requirements

Rank	DBMS software	Similarity
1	DBMS-C	0.9756
2	DBMS-B	0.9372
3	DBMS-A	0.8522
4	DBMS-D	0.6513

Table 8. DBMS recommendation list according to user B's requirements

Rank	DBMS software	Similarity
1	DBMS-D	0.8911
2	DBMS-B	0.8897
3	DBMS-A	0.8730
4	DBMS-C	0.8673

Figure 2 shows a comparison of the similarity results for user A and user B.

As a result of examining the software recommendation lists according to each user requirement, it was confirmed that appropriate software was recommended with respect to each requirement. As a further experiment, the proposed method was

Figure 2. Comparison of the similarity results for user A and user B.

compared with other comparison methods. Figure 3 shows the results.

The proposed method is a result of calculating the similarity by reflecting the importance of the user's requirements and weighting the characteristic values of the software. Comparative method A is a result of calculating the similarity without reflecting the weight on the characteristic value of the software in the proposed method. Comparative method B is a result of calculating the similarity without reflecting the importance of

Figure 3. Comparison of the similarity results among different methods according to user requirement: (a) user A and (b) user B.

the user's requirements. Comparative method C is a result of calculating the similarity without reflecting any of them.

As a result of comparing the four methods, it was confirmed that the similarity results were different. The proposed method can yield more appropriate recommendation results by reflecting user's requirements and the software characteristics. While existing software recommendations consider the frequency of use, in this study, it was possible to recommend more suitable software to the user by reflecting that user's requirements.

5. Conclusion

In this paper, we proposed a similarity-based software recommendation method that reflects user requirements. We generated a function matrix and function vectors from software functional specifications. Furthermore, we generated a requirement vector from the software function conditions selected by the user. Thereafter, we generated a software list using the function matrix based on the Boolean model. Next, we calculated cosine

similarity between the requirement vector and function vectors of the software list based on the vector space model. Finally, we generated a software recommendation list in descending order of similarity.

To evaluate the effectiveness of the proposed method, we tested it using DBMS functional specifications. In the results of the software recommendation according to user requirements, an appropriate DBMS was recommended with respect to such requirements. By assigning different importance conditions and evaluation weights for the software functions, users can set their requirements in various ways. By reflecting the user requirements and recommending the appropriate software, users are provided with assistance. We also believe that search performance was improved by using Boolean retrieval and cosine similarity.

In the future, we expect that performance can be improved by utilizing statistical search techniques. An additional study will be carried out on software recommendation by considering the quality of the software as well as the software functions and user's level.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

Acknowledgements

This research was supported by the Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (Ministry of Trade Industry and Energy) (No. N0001884, HRD program for Embedded Software). This is an expanded version of the paper entitled "A Software Recommendation Method Considering User Requirements" presented at the 20th International Symposium on Advanced Intelligent Systems (ISIS 2019), which was held in Jeju Island, Korea, December 4-7, 2019.

References

- [1] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80-86, 2010. <https://doi.org/10.1109/MS.2009.161>
- [2] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogeve, and S. Ofek-Koifman, "Personalized recom-

- mendation of social software items based on social relations,” in *Proceedings of the 3rd ACM Conference on Recommender Systems*, New York, NY, 2009, pp. 53-60. <https://doi.org/10.1145/1639714.1639725>
- [3] C. K. Ke, S. F. Chang, C. Y. Jen, and J. C. Liao, “Software recommendation of application market by semantic network and multi-criteria decision analysis,” in *Multimedia and Ubiquitous Engineering*. Heidelberg, Germany: Springer, 2014, pp. 317-322. https://doi.org/10.1007/978-3-642-54900-7_45
- [4] DB-Engines ranking, Available <https://db-engines.com/en/ranking>
- [5] M. Das, G. De Francisci Morales, A. Gionis, and I. Weber, “Learning to question: leveraging user preferences for shopping advice,” in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, IL, 2013, pp. 203-211. <https://doi.org/10.1145/2487575.2487653>
- [6] J. Son, S. B. Kim, H. Kim, and S. Cho, “Review and analysis of recommender systems,” *Journal of Korean Institute of Industrial Engineers*, vol. 41, no. 2, pp. 185-208, 2015. <https://doi.org/10.7232/JKIE.2015.41.2.185>
- [7] J. B. Schafer, J. Konstan, and J. Riedl, “Recommender systems in e-commerce,” in *Proceedings of the 1st ACM Conference on Electronic Commerce*, Denver, CO, 1999, pp. 158-166. <https://doi.org/10.1145/336992.337035>
- [8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Analysis of recommendation algorithms for e-commerce,” in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, Minneapolis, MN, 2000 pp. 158-167. <https://doi.org/10.1145/352871.352887>
- [9] B. W. Jin, Y. S. Cho, and K. H. Ryu, “Personalized e-commerce recommendation system using RFM method and association rules,” *Journal of the Korea Society of Computer and Information*, vol. 15, no. 12, pp. 227-235, 2010. <https://doi.org/10.9708/jksci.2010.15.12.227>
- [10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, China, 2001, pp. 285-295. <https://doi.org/10.1145/371920.372071>
- [11] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, 3rd ed. Auckland, New Zealand: McGraw-Hill, 1987.
- [12] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. New York, NY: ACM Press, 1999.
- [13] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Data Sets*. Cambridge, UK: Cambridge University Press, 2014.
- [14] S. Tata and J. M. Patel, “Estimating the selectivity of TF-IDF based cosine similarity predicates,” *ACM SIGMOD Record*, vol. 36, no. 2, pp. 7-12, 2007. <https://doi.org/10.1145/1328854.1328855>
- [15] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “GroupLens: an open architecture for collaborative filtering of netnews,” in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, 1994, pp. 175-186. <https://doi.org/10.1145/192844.192905>
- [16] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76-80, 2003. <https://doi.org/10.1109/MIC.2003.1167344>
- [17] Netflix Prize, Available <http://netflixprize.com>
- [18] S. I. Baek, Y. E. Song, and Y. K. Lee, “A software recommendation method considering user requirements,” in *Proceedings of the International Symposium on Advanced Intelligent Systems (ISIS)*, Jeju, Korea, 2019, pp. 487-489.
- [19] S. A. Lee and S. W. Kang, “Data mining based recommendation system for software evolution,” *Communications of the Korean Institute of Information Scientists and Engineers*, vol. 31, no. 10, pp. 39-46, 2013.
- [20] Y. R. Kim, D. Y. Ko, S. Y. Park, and J. T. Kim, “Requirements alternative flow detection method using use case patterns at use case scenario,” *Journal of Korean Institute of Information Scientists and Engineers: Software and Applications*, vol. 40, no. 9, pp. 491-501, 2013.
- [21] G. Yang, T. Zhang, and B. Lee, “A developer recommendation technique based on topic model and social network,” *Journal of Korean Institute of Information Scientists and Engineers: Software and Applications*, vol. 41, no. 8, pp. 557-568, 2014.

- [22] S. I. Lee and S. Y. Lee, "Integration of user profiles and real-time context information reflecting time-based changes for the recommendation system," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 8, no. 4, pp. 270-275, 2008.

Se In Baek received her B.S. and M.S. degrees in computer science and engineering from Hanyang Woman's University and Dongguk University respectively. She is currently a Ph.D. candidate in the computer science and engineering department at Dongguk University, Seoul, Korea. Her research interests include database systems, information retrieval, and big data management.

E-mail : sein3051@dongguk.edu

Yang-Eui Song received his B.S., M.S. and Ph.D. degrees in computer science and engineering from Dongguk University. He is currently a professor in the Information Security Department at Dongguk University Computer Science Institute, Seoul, Korea. His research interests include information retrieval, big data analytics and security.

E-mail : redcroix@dongguk.edu

Yong Kyu Lee received his B.S., M.S. and Ph.D. degrees in computer science from Dongguk University, KAIST, and Syracuse University respectively. He is currently a professor in the Computer Science and Engineering Department at Dongguk University, Seoul, Korea. His research interests include database systems, information retrieval, big data management, sensor network, and web science.

E-mail : yklee@dongguk.edu