Software Component Recommendation Using Collaborative Filtering

Makoto Ichii[†] Yasuhiro Hayase[†] Reishi Yokomori[‡] Tetsuo Yamamoto[*] Katsuro Inoue[†]

[†]Osaka University
{m-itii,y-hayase,inoue}@ist.osaka-u.ac.jp

[‡]Nanzan University
yokomori@it.nanzan-u.ac.jp

[*]Ritsumeikan University
tetsuo@cs.ritsumei.ac.jp

Abstract

Software component retrieval systems are widely used to retrieve reusable software components. This paper proposes recommendation system integrated into software component retrieval system based on collaborative filtering. Our system uses browsing history to recommend relevant components to users. We also conducted a case study using programming tasks and found that our system enables users to efficiently retrieve reusable components.

1 Introduction

A software component retrieval system (retrieval system), or a source code search engine, is widely accepted as a tool for finding reusable software components or helpful code examples. For example, Google code search [1] and Krugle[2] are publicly available through WWW. There are academically developed system such as SPARS-J [6] and Sourcerer [1]. Hummel et al., reports the growth of the database of publicly-available code search engines [5], on which needs for easily-accessible component are reflected.

However, developers, especially who are not familiar with search engines, often have trouble to retrieve reusable components with search engines. One reason for this is constructing "good" queries requires experience to the search engines. For finding components suitable for his/her requirement from large component repository, refinement of search queries based on try-and-error is necessary. In other case, a component found by a retrieval system requires further components that are directly or indirectly related to the component. In addition, code examples are required to reuse components.

In this paper, we propose software component recommendation approach based on collaborative filtering (CF) technique. CF is a technique to recommend items to a target user by feeding back reputation of other users who have similar preferences/interests with the target user. Various e-commerce systems use CF to recommend items to customers.

We have developed component recommendation system integrated into SPARS-J. Our system automatically recommends potentially-beneficial components for users based on their browsing history. Our approach assumes that the developers who have similar browsing history require similar components. Our approach has potential to help users finding a set of reusable components including dependency and example code as they are guided by experienced antecessors without labored try-and-error.

We have conducted a case study in order to evaluate how our system can help developers; eight participants have tried coding tasks with/without our system.

This paper is constructed as follows: Section 2 describes about the recommendation systems as the background. We explain our recommendation method in Section 3 and its implementation in Section 4. Section 5 describes about the case study. We conclude this paper in Section 6.

2 Recommendation System

Various works have tried to support developers who have trouble to understanding a software system based on the *content-based* approach, i.e., analysis of the software system itself. Find-Concept [8] helps developers to finding concerns, or implementation which they are interested in, based on a hybrid technique of structural program analysis and natural language processing.

Meanwhile, *collaborative filtering* (CF) is a technique to recommend items to a user that matches to preferences of the user from vast amount of items by sharing reputations to items among users. [4]. GroupLens [7] is an automated collaborative filtering system that recommends articles of Usenet. A user of GroupLens *votes ratings* of articles they read in five-point scale. Then GroupLens recommends articles that are highly-rated by users whose ratings are similar to the target user.

CF can enhance content-based filtering tools: providing filtering of items whose content is hard to analyze; recommendations based on quality and taste; and serendipitous

[1] http://www.google.com/codesearch
[2] http://www.krugle.org

978-1-4244-3740-5/09/$25.00

recommendations [4]. DeLine et al. proposes a system that helps program comprehension by filtering and recommending the resources in the project of a developer based on navigation data shared among the project team [3]. We believe that CF approach can improve retrieval systems by achieving filtering that is based on implicit relevance between components and personalized to target user, i.e. suitable for requirements and restrictions of the user's task. In addition, a user may get suggestion of some "better" components than ones that a user considers to reuse.

3 A method for component recommendation

This section introduces a method for recommending software components using CF on search history. The method extends the GroupLens algorithm, which uses correlation between users, for component recommendation. The method consists of following steps.

Gather Ratings for Components: Store browsing history of each user as ratings for components on a database.

Compute Correlation: Obtain correlations between users based on ratings in the database.

Compute Recommendation Score: Obtain scores of all components using ratings made by a user and its correlation to other users.

Recommend Components: Present components for the user based on the scores of the components.

Details of these steps are described below.

3.1 Gather Ratings for Components

User ratings for components are essential for CF. Therefore, recommendation system has to gather ratings voted by users. However, explicit ratings for components are undesirable since effort for the inputting is not negligible. Our method uses *implicit rating* [4] for avoiding the effort. More specifically, a component is rated 1 (i.e. good) by a user *iff* the user browses the source code of the component. Our approach allows users to *cancel* their ratings, i.e., users can improve the recommendation by deleting their browsing history.

CF is based on user correlation and assumes that preferences of users are stable. This assumption is problematic for search systems because user preference varies according to change of search purpose.

When using a search system, rating preference is stable while search purpose is same. Therefore, we employed a search purpose as a *user* in context of CF, and then used similarity between search purposes for component recommendation. Accordingly, the system has to detect changes of search purpose and split browsing log at the changes. But detecting the change from user action is difficult generally. We treat an end of using a search system as a change of search purpose, and ratings from beginning to end of use (i.e. session) as *user* rating. A user and a browsing session will not be distinguished hereinafter except for necessary contexts.

We have to consider about same or similar components, which are copied for reusing, in a search target. Users of search system can arbitrarily choose one component from similar components. However, this choice affects harmful for CF because votes for the similar components are scattered for each components. We adopted a set of similar components as a basic unit of recommendation for alleviating this problem.

3.2 Compute Correlation

CF requires correlation between users for the purpose of speculating users similar to a user to whom the system recommends items.

GroupLens defined a correlation between two users as similarity of ratings for components that are voted *both* of the two users. However, this definition is inappropriate for our method. In our method, vote is done by browsing and ratings for browsed components are always 1. Therefore similarity between sessions is 1 if common component are browsed in the sessions, otherwise similarity is 0. Furthermore, sessions are generally short and contain few records of browsing, sessions rarely share browsed components. So we employed Breese's recommendation algorithm[2] which defines the correlation between two users as similarity of ratings for components that is voted *either or neither* of the two users. Rating for a browsed component is 1, and for not browsed component is 0.

Correlation $c(a, i)$ between user a and i is defined as follows. I_i means the components that is voted by user i, and I means all components that is registered in recommendation system.

$$v_{i,j} = \begin{cases} 1 & \text{if } j \in I_i \\ 0 & \text{if } j \notin I_i \end{cases}, \ \bar{v}_i = \frac{1}{|I|} \sum_{j \in I} v_{i,j}$$

$$c(a,i) = \frac{\sum_{j \in I} (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_{j \in I} (v_{a,j} - \bar{v}_a)^2 \sum_{j \in I} (v_{i,j} - \bar{v}_i)^2}}$$

3.3 Compute Recommendation Score

CF computes a recommendation score for each component based on ratings made by the users whose behavior is similar to a receiver of the recommendation. In particular, recommendation score is an estimation value for the rating that the receiver will vote. An estimation score is a weighted average of scores made by other users; the weight is a similarity for the receiver.

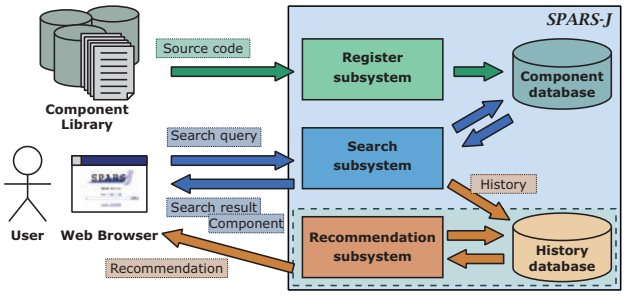

Figure 1. SPARS-J with recommendation subsystem

In our method, a set of the similar users are preliminary computed for speed of recommendation. The set consists of the users who satisfy the following conditions: 1) he/she shares browsed components with a receiver, 2) His/her correlation to the receiver is positive, and 3) he/she browsed two or more components. The weight for the averaging is square of the correlation according to [2].

Consider set of similar user $U = \{i | I_a \cap I_i \neq \phi \wedge c(a,i) > 0 \wedge |I_i| > 1\}$. Recommendation score of component k for user a is defined as follows:

$$p_{a,k} = \frac{\sum_{i \in U} c(a,i)^2 v_{i,k}}{\sum_{i \in U} |c(a,i)^2|}$$

3.4 Recommend Components

The system recommends components in a descending order of the scores. With the exception of recommendation, the components that the receiver has already seen are eliminated from recommendation, since these components are well-known for the receiver.

4 Implementation on SPARS-J

We implemented our recommendation method on SPARS-J, which is a retrieval system for Java components. Our method is built into SPARS-J as recommendation subsystem. Figure 1 is the architecture diagram of SPARS-J with the recommendation subsystem.

4.1 Record browsing history

Our system distinguishes a *session* using a session cookie, which is generated when a searcher accesses to SPARS-J using a web browser and is expired when he/she closes the browser window.

When he or she opens a source code of a component, a vote for the component is stored in the history database with the session identifier that is assigned to the session cookie.

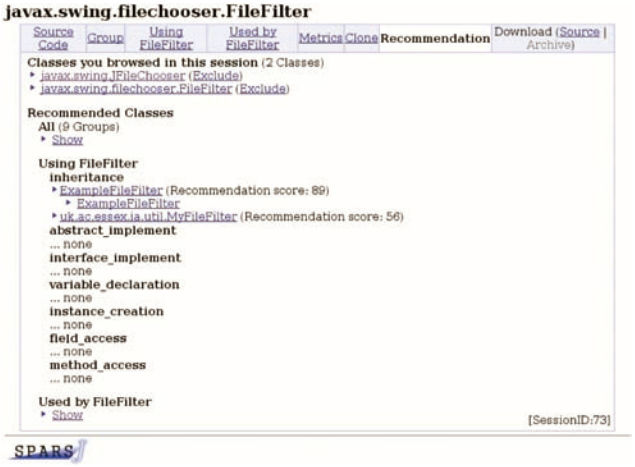

Figure 2. Recommendation page

4.2 Display recommended components

Recommendation page displays a list of recommended components whose recommendation score is higher than certain threshold.

Figure 2 is a sample of recommendation page. The list contains hyper links to details pages for each recommended components. Recommendation score is also shown in the list; the user can refer the score for selecting components.

There are two way to receive recommendation: flat view and relation view. The flat view shows all recommended component in descending order of recommendation score. The relation view displays components classified by those relations to a certain component.

5 Case study

We conducted a case study to evaluate whether our recommendation method can help users to retrieve reusable components in coding tasks. Eight participants (A1 $\sim$ A8) tried four Java coding tasks (P1 $\sim$ P4) with/without the recommendation subsystem. The all participants have enough Java experiments to solve the tasks; they are members of a software engineering laboratory and they have used Java in their research.

In each task, they were asked to implement some feature into skeleton code using only SPARS-J and the documentation of Java API. The resources for running the programs are provided so that the participants can test their programs quickly when they finished to code. The subjects of the tasks are image file conversion (P1), file transfer using FTP (P2), GUI (P3) and database access (P4). We suppose that the number of reused components to finish the each task range from 2 to 5; and the lines of code range from 10 to 50.

Table 1. Result of the case study

	Search time (min)				Precision			
	P1	P2	P3	P4	P1	P2	P3	P4
A1	28	25	**14**	**3**	0.42	0.06	**1.00**	**0.67**
A2	47	19	**60**	**38**	0.53	0.38	**0.50**	**0.64**
A3	14	2	**3**	**7**	0.23	1.00	**1.00**	**0.83**
A4	50	28	**9**	**12**	0.26	0.15	**1.00**	**1.00**
G1 Ave.	34.8	18.5	**21.5**	**15.0**	0.36	0.18	**0.88**	**0.79**
A5	**4**	**2**	9	23	**1.00**	**1.00**	1.00	0.55
A6	**3**	**4**	18	45	**1.00**	**1.00**	1.00	0.63
A7	**28**	**5**	44	13	**0.78**	**0.40**	0.55	0.73
A8	**15**	**2**	26	23	**0.75**	**1.00**	0.37	0.73
G2 Ave.	**12.5**	**3.2**	24.2	26.0	**0.89**	**0.73**	0.73	0.66

We set up SPARS-J for the case study with the database comprising Java API and demos, open source software packages and samples retrieved from WWW. The database contains enough reusable components and their sample codes to accomplish the tasks. Total number of components is about 35,000. The case study begins with an empty history database

5.1 Procedure

Before the case study, every participant takes one-hour lecture about the system usage including a training task. We grouped the participants into two groups (G1 and G2) so that the groups have similar programming ability based on the memers' Java experiments and the time spent to accomplish the training task.

At first, the members of G1 (A1 $\sim$ A4) tries P1 and P2; and the members of G2 (A5 $\sim$ A8) tries P3 and P4 without recommendation. Then, G1 tries P3 and P4; and G2 tries P1 and P2 with recommendation.

5.2 Result and discussion

Table 1 presents the search time and the precision of the retrieved components. The search time indicates (*total time to accomplish the task*) - (*coding time*); and the precision is defined as $|$*the components used for implementation*$| \div |$*the components displayed in SPARS-J*$|$. The participant/task combinations using the recommendation are presented with the boldface. We can see that the recommendation successfully helps users by reducing searching time and improves precision.

The result shows that difference of P3 is smaller than the other tasks. This is likely because A5 and A6, who work on P3 without recommendation, have knowledge on the subject area of P3 enough to navigate themselves to the components available for the task.

We also found that A2 and A7 did not improve the efficiency with recommendation. Their common behavior is that they briefly read a lot of components at the beginning of their activity. As the result, the available components were recorded into the history without recognized as "available" and never recommended. An idea to support such users is changing the policy to gather the rating to ignore components that had been viewed only in a minute.

6 Conclusion

This paper proposed software component recommendation based on collaborative filtering (CF). We also implemented the recommendation system that is integrated into SPARS-J. The result of the case study shows that CF is effective to improve search efficiency of reusable components. Our future work is improving recommendation by sophisticating logic to gather ratings from browsing history.

Acknowledgements This work was supported in part by "Global COE (Centers of Excellence) Program" of the Ministry of Education, Culture, Sports, Science and Technology, Japan; and was conducted in part as a part of Stage Project, the Development of Next Generation IT Infrastructure, supported by Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: A search engine for open source code supporting structure-based search. In *Proc. OOPSLA '06*, pages 25–26, Oct. 2006.

[2] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. UAI '98*, pages 43–52, 1998.

[3] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *Proc. VL/HCC '05*, pages 241–248, 2005.

[4] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. SIGIR '99*, pages 230–237, 1999.

[5] O. Hummel and C. Atkinson. Using the web as a reuse repository. In *Proc. ICSR '06*, pages 217–230, July 2006.

[6] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, Mar. 2005.

[7] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proc. CSCW '94*, pages 175–186, 1994.

[8] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proc. AOSD '07*, pages 212–224, 2007.