A KNOWLEDGE SHARING SYSTEM FOR SOFTWARE DEVELOPERS

Takayuki Shibata, Kazuyuki Nakamura, Takanobu Sato
Graduate Department of Information Technologies and Project Management, University of Aizu
Aizu-Wakamatsu City, Fukushima, 965-8580, Japan

Rentaro Yoshioka
Graduate Department of Computer and Information Systems, University of Aizu
Aizu-Wakamatsu City, Fukushima, 965-8580, Japan

Keywords: Knowledge sharing, Program reuse, Software development, Knowledge acquisition.

Abstract: Knowledge sharing is a key factor for increasing productivity of programmers and also in maintaining the quality of programs in companies. However, programmers tend to resort to outside resources for solving their problems. This paper proposes a system to facilitate active sharing of program related knowledge among a group of programmers in a company. The system introduces a flexible unit to define the target knowledge, defines a set of function tags to describe its functionality from a programming point of view, and a set of project tags to describe its environmental aspects. We illustrate the rigid structure and classification of the tags and how this approach can decrease the work load of programmers in registering and retrieving knowledge along with a few examples. In addition, a simple evaluation tests have been performed with an experimental implementation of the proposed system.

1 INTRODUCTION

When a programmer comes across problems during coding, where does he/she turn for help? Prying through documents, looking up the index of a favorite book, or running a search on the Internet, there are often many sources to choose from. There is a study that reports that, among professional programmers, the most preferred source to seek advice is (1) the Internet, (2) books, and (3) colleagues/supervisors (Kurata, 2003). Although this order might vary depending on whom you ask, definitely these are everyone's favorites. To a company engaged in software development, it is of great interest whether their programmers solve their problems efficiently and with sufficient quality. For instance, books can be considered as a relatively reliable source, considering the process it goes through before getting printed. On the other hand, books on more recent or evolving technologies can take some time until they get printed and must be properly replaced with old versions to be fully useful. As for efficiency, books are well organized and are fast when you are looking for something that is listed in the index but you might need to spend time to skim some chapters if it is not. Asking a colleague or boss, or any other knowledgeable person, about a coding problem is often a simple solution. It is possible to immediately receive advice but the quality of the answer could depend on the time your colleague can spare. Especially in companies, the fact that one person spends time to help another could be considered as undermining the efficiency of the organization as a whole. Compared to the two sources of knowledge we have just considered, the Internet has features that complement their drawbacks. The amount of resource available on the Internet is vast and covers a wide range of information which is usually updated at a much higher speed than books. Obviously, searching the Internet does not influence any other colleague's time so it is a promising candidate as an ideal source for increasing the productivity of programmers. Still, it is quite far from being perfect.

Shibata T., Nakamura K., Sato T. and Yoshioka R. (2010).
A KNOWLEDGE SHARING SYSTEM FOR SOFTWARE DEVELOPERS.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 499-503
DOI: 10.5220/0003005704990503

General search engines, such as Google and Yahoo, provide full-text search of resources available on the Internet. The problem with using general search engines is that its efficiency is unpredictable. Often, it takes much more time than expected to find anything useful and requires sufficient knowledge and experience to come up with the right keywords that will produce the expected search results. Moreover, although there is great amount of resources to search from, those with sufficient quality are much less. There are search engines devoted to specific fields of interest. For example, DBCLS is a search engine specialized in life science (Codase, 2010). Also there are web sites dedicated to code sharing, called code search engines (CSE) sites, such as, Google Code Search (Google, 2010), Koders, Codase (Koders, 2010). CSE sites allow users to post programs along with tags that represent the type of programming language, additional user specified tags (keywords) and text and are customized for sharing programs. However, other than the tag that represents language types, they basically perform a text search on the tags, text, and program and suffer from the same efficiency/quality problem of general search engines. There is a type of CSEs that collect programming knowledge in the form of code snippets (DZone, 2010) (Snipplr, 2010). These sites allow users to share small parts of programs and to retrieve them as templates for coding. However, even these sites do not provide any additional means for supporting a more efficient search.

We propose a system in which knowledge related to program development can be shared efficiently among a group of developers. The aim of the system is to increase the productivity of program development by providing access to a shared repository of knowledge useful for developers. This repository stores knowledge in units of logical fragments. A logical fragment can represent varying sizes of knowledge, from a few lines of code to a few files of programs. The knowledge is not limited to program code and can represent configuration files, documents, etc. A logical fragment is annotated by a set of tags that describe its function, called the function tag, and a tag describing its environmental attributes, called the project tag.

The paper is organized as follows: the next section presents the background and direction of our approach. The following section describes the specific method used to register and retrieve knowledge in our system. The final section provides summary and presents future work.

2 BACKGROUND

This research is concerned with knowledge management within an organization to support software engineering. We are especially interested in increasing the productivity of programmers while increasing the quality of the code that they produce. Knowledge management is a means of solving business challenges and many methods have been practiced (Alavi and Leider, 2001). The first generation systems were focused on the documents created by users. The second generation systems focused on the people who possess knowledge. These systems were realized as web sites that provide a public place where questions on particular topics could be posted and people with the knowledge could answer. A particular group of web sites that support interaction and exchange of knowledge between people sharing common interests (Social Networking Sites) were also introduced. The first generation systems suffered from the high-cost of accumulating information and the second generation systems suffered the difficulty of evaluating the effects since effects were difficult to visualize. In more recent attempts, methods that try to evaluate the contents that each user possesses and to connect them have also emerged.

Existing CSEs can be classified into two types. In the first type, the system parses various files, which were registered as one set of files related to a project, and performs searches. In the second type, lines of code are registered individually as snippets. In both types, in addition to the code, additional information provided by users and the system are subject to search. For example, types of programming languages or licenses are used by some systems. Based on a finding that many programmers perform search related to API (application programming interface), troubleshooting, implementation, development tools, language syntax and semantics, a system to assist problem solving by automatically collecting and extracting significant information from web pages with sample codes, Java archive (JAR) files, Java documents (JavaDoc) pages has been proposed (Thummalapenta and Xie, 2007). There are also many approaches related to component-based software engineering that aim at solving the problem with programming efficiency and quality by reusing software as components (Heineman and Councill, 2001).

Compared to current CSEs, the system we propose introduces a greater structure to the format in which programs and related information are

registered. This structure allows the system to better understand and manipulate the knowledge when performing searches and presenting the results to users. The fundamental policy is to extract as much knowledge as possible from the person registering it. Instead of simply allowing the user to register a file, the system will ask the user to specify which lines are important and what they represent. This brings up another important point that our system does not use free-word tags but introduces a predefined set of tags. Tags are often sources of ambiguity and causes searches to return unrelated information. In some social networking sites (SNS), folksonomy is used to bring some kind of order to freely defined tags (Golder and Huberman, 2006). It is possible to create a hierarchical structure from these tags by creating clusters but the quality of the resulting classification depends on the number of tags and usually requires a huge set (Niwa, Doi, Honiden, 2006). We prefer to use a well organized set of tags, where each tag represents some range of meaning so that we do not need to prepare a tag for everything. The impreciseness can be supplemented by using multiple tags and intelligent user interfaces technologies. The main framework supporting our idea is described in the next section.

3 KNOWLEDGE REPRESENTATION METHOD

This system aims at managing two types of knowledge. The first type is the knowledge related to programming of software (in other words, coding). This is the practical knowledge required for developing software systems, such as, syntax of programming languages, logics of specific algorithms, usage of specific libraries, etc. These are the knowledge that becomes useful in developing programs. The second type is the knowledge related to project management. This is the knowledge related to managerial aspects, such as, system design documents, project balance sheets, schedule, etc. These are the knowledge that becomes useful in planning projects and creating similar documents in the future. Since there are many existing approaches in sharing the second type of knowledge, our main focus is on how to effectively share the first type of knowledge. Our approach to dealing with programming related knowledge is based on “logic fragments” and “tag-based retrieval”.

3.1 Logic Fragments

A “logic fragment” (LF) is a unit of knowledge representing some meaningful function within a program. Each LF consists of more than one file (usually a program) and a set of line numbers that specify a region of interest (RI). Figure 1 depicts the simplest form of LF. In the figure, the code to be shared is marked as an RI by the line numbers.

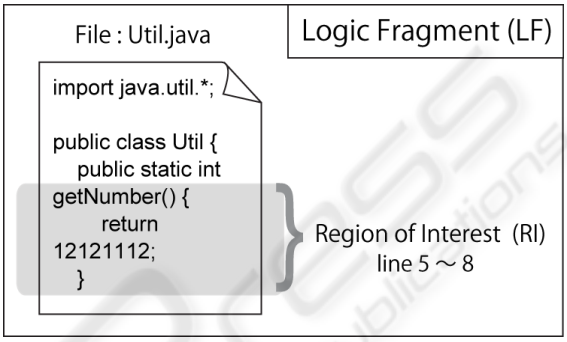

Figure 1: Logic Fragment and Region of Interest.

Then, the original program and the line numbers are stored as an LF. This structure, compared to cutting out and storing only the lines that are related to the target function, is more suitable in program reuse since other parts of the program provides additional information for understanding the RI. Let us consider an example. Suppose that a programmer wishes to share the following knowledge: “How to extract the month and year from a formatted string of characters”. The programmer will first open the program file that contains this code. Then the lines that actually correspond to that code should be specified (Supposing that this program contains code that performs other functions as well.). Multiple RIs can be used within an LF. As already mentioned, an LF can consist of multiple files so that functions that span across multiple files can be represented. An important rule of LF is that all LFs should be ready for compilation and execution. This is based on the thought that understanding of programs is better accomplished by actually running it. So, supportive files, such as additional programs and libraries, can be attached to an LF.

3.2 Tag-based Retrieval

The knowledge of an LF is specified by two types of tags: function and project. A **function tag** is used to describe the functional meaning of that knowledge. A **project tag** is describes the features specific to the concrete instance of that knowledge.

Category	Description	Examples of subcategory
Basics	This category is involved with basics of programming languages such as syntax, data types, basic input and output, etc.	Data type, operator, input/output, function, exception handling
Data Structure	This category is for definitions and uses of data structures.	Array, enumeration, stack, queue, list, tree
Mathematics	This category relates to commonly-used calculations and calculations for specific data structures.	Statistics, analysis, geometry, cryptography, graph
String	This category is about string processing and attributes.	Attribute, search/select/replace, filter/conversion
Date/Time	This category treats functions of date and time such as acquisition of specific days and judgment of specific places and measurement of time.	Format, calendar, time zone, sleep
File	This category covers processes with files.	Operation, conversion, file type, directory
System	This category relates to functions and processes managed by systems.	Process management, memory management, file system
Graphics	This category is for graphical output different from consol output and relates to functions on windows, image processing, and so on.	Diagram, 3D graphics, window, button
Database	This category is about structures of databases and processes on databases.	Schema, table, security, operation
Network	This category treats processes using network protocols.	E-mail, DNS, HTTP, FTP, IP
Web	This category is for web components and processes on them such as buttons, layouts, update of views, pop-up, etc.	Output, button, layout, transition
User-Defined	This category is for specialized knowledge, not general one, defined by organizations.	Organizations will define here.

Figure 2: The top level items of function tags.

An LF represents knowledge of one or more program-related functions. Each LF is attached at least one **function tag**. Based on our analysis of past and current methods and systems, we have opted to predefine the possible types of functions rather than allowing users to use free-words. In defining the tags, a classification of functions was created by analyzing existing knowledge from books, programming languages, software applications, etc. The functions tags have a hierarchical tree structure of three levels. The top level items are shown in Figure 2. There are 12 categories in the first level. For example, the basic category contains functions that directly relate to features of programming languages, such as, variable definitions, declarations, operators, input/output functions, etc. The user-defined category is a special category in which functions related to specialized knowledge may be added. This allows groups of users (i.e. companies) to add custom tags that are specific to their field of activity. Since this is a newly proposed classification of functions, it is expected that users will require some time to understand its organization.

A **project tag** represents features related to the specific environment in which the program was intended for, such as, programming languages, operating systems, dependent libraries and frameworks, etc. Since these attributes are often defined by the requirements of development projects in companies, we have named them the project tags. The project tags are especially useful when retrieving knowledge registered by members of the same project.

4 CONCLUSIONS

A knowledge representation method for a system specialized in sharing knowledge related to program development has been presented. This system is aimed to be used by a group of developers (within in a company) by registering useful program code as reusable knowledge. In this system, fragments of program code that implement some function are registered. In the proposed method, a unit called Logic Fragment (LF) and Region of Interest (RI) are used to define the body of knowledge. Two types of tags, called function and project, are proposed to describe the content meaning of each LF. The function tag is used to describe the function it provides. The project tag is used to describe the environmental features related to the program, such as, programming language, dependant libraries, operating system, etc. to supplement the retrieval process. Users may retrieve knowledge from the system by specifying one or more logic tags. The

search results may be sorted / filtered with the project tags. Although the details could not be included in this paper due to space constraints, an experimental system was implemented and usability tests were performed. From these tests, we have obtained results that show that this system can be effective in organizing program-related knowledge and that it may contribute to increasing the efficiency of searches and maintaining quality of acquired knowledge.

REFERENCES

Alavi, M., Leidner, D., 2001. Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues, MIS Quarterly Vol.25 No.1, pp.107-136.

Codase homepage. Retrieved February 1st, 2010, from http://www.codase.com/.

Code Snippets homepage. Retrieved February 1st, 2010, from http://snipplr.com/.

DZone Snippets home page. Retrieved February 1st, 2010, from http://snippets.dzone.com/.

Golder, S.A., Huberman, B.A., 2006. The Structure of Collaborative Tagging. *Journal of Information Science*, 32, 198-208.pp. 101-127.

Google Code Search homepage. Retrieved February 1st, 2010, from http://www.google.com/codesearch.

Heineman, G.T., Councill, W.T., 2001. *Component-Based Software Engineering*, Addison-Wesley, New Jersey.

Koders homepage. Retrieved February 1st, 2010, http://www.koders.com/.

Kurata, Y., 2003. Questionnaire: Awareness of skills and career development of IT engineers. Retrieved at http://y-kurata.com/2002project.htm.

Niwa, S.,Doi, T., Honiden, S., 2006. Web Page Recommender System based on Folksonomy Mining, *Proc. 3rd International Conference on Information Technology: New Generations (ITNG'06)*, pp.388-393.

Thummalapenta, S., Xie, T., 2007. PARSEWeb: A Programmer Assistant for. Reusing Open Source Code on the Web, *Proc. 22nd International Conference on. Automated Software Engineering*, pp.204–213.