Automatic Categorization Tool for Open Software Repositories

Shinji Kawaguchi[†] Pankaj K. Garg[††] Makoto Matsushita[†] Katsuro Inoue[†]

[†]Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka
560-8531, Japan
{s-kawagt, matusita, inoue}@ist.osaka-u.ac.jp

[††]Zee Source
1684 Nightingale Avenue, Suite 201
Sunnyvale, California, 94807, USA
garg@zeesource.net

ABSTRACT

The world of Open Source software has demonstrated the remarkable appeal of *communal software development.* Large number of software projects can leverage, reuse, and coordinate their work through Internet and web-based technology. For example, SourceForge currently hosts about sixty thousand software systems. Similar strategies have been suggested for corporate software development, through notions like Corporate Source and Progressive Open Source [6, 7]

When used in a corporate setting, infrastructures for project information sharing present new opportunities. For example, one would like to know all projects that have something in common, so that the project groups can collaborate and share their work. With thousands of projects, manually locating related projects can be difficult. Hence, we propose to use automatic software categorization to find clusters of related software projects, using only the source code from projects. Our experiments with a small set of C programs demonstrates potential for automatic categorization of software systems without human aid.

1. INTRODUCTION

The rapid use of Internet and Web-based technology has given rise to a novel, global software archiving service, pioneered in the Open Source community through SourceForge [17]. More recently, several large corporations are realizing the benefits of such services for their own, proprietary software development. For example, Hewlett-Packard Company, IBM, Motorola, Nokia, and Xerox, are some of the corporations that are known to have deployed such archival service for their own internal corporate network.

For large software archives, categorizing their contents for browsing and searching is essential for effective utilization of the software archive. Automatic categorization would be helpful in several ways:

- Several *similar* software can be grouped together in a category for ease of browsing. For example, SourceForge [17] categorizes software according to their function (editors, databases, etc.), and also has the notion of *software foundries* for related software.
- Developers working on a software system may be informed about related software. Finding related software systems has following advantages.
  1. Developers can learn “best practices” and programming idioms from existing software systems. From related software systems, they can get strategies or hints for software evolution. They can even evolve their software systems based on related software systems, and not have to create it from scratch.
  2. Developers can leverage each other’s work and promote more reuse. This becomes specially useful in situations like Corporate Source [7], where global groups in companies may not be aware of the relationship among their work [9].

In the past, such relationships have been determined by hand. Manual categorization generally requires deep understanding of not only the target software system, but also other software systems and their classification policy. With the increase in the number of software systems, e.g., SourceForge now has over sixty thousand software systems registered and continues to evolve, such manual identification is not enough.

Automatic categorization of software systems is a novel and intriguing challenge on software archive evolution. Past work in software engineering (e.g., see [4, 16]), has focused on determining *intra-component relations* of one given software system. We, however, propose finding *inter-component relations* of many software systems.

In this paper, we propose software automatic categorization system based on Latent Semantic Analysis(LSA). LSA is a method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text [11]. LSA has found a variety of uses ranging from understanding human cognition [11] to data mining [5]. Also, it is used for clustering components in a software system [14] and recovering document-to-source links [15].

We apply LSA for determining categories of software systems. We implemented the proposed method, and report on experimental results.

2. RELATED WORKS

Maarek et al. applied free-text indexing approach for software classification [13, 8]. They retrieved information from Unix man pages and classified Unix tools.

While quality and granularity of Unix man pages are highly uniform, the amount and quality of documents differ with software systems. Some software may have complete documentation, while others may have no or few documentation for their implementations.

From the viewpoint of retrieving information from source code, some existing clustering methods cluster one software into some functional parts for program understanding. Such software clustering methods use Latent Semantic Analysis [14], Self-Organizing Map [2], file structure and file names [1] or structure of program like call graph [3, 12].

In our previous work [10], we have tried measurement methods of software similarities. The hope was that we will relate "similar" systems together, and determine orthogonal categories like "editors," "databases," and so forth. We applied LSA to software systems and found that software similarity values are reflected only by *most influential aspects* of software systems. For example, the similarity value between database software with GTK interface and editors using GTK is very high. Although this phenomenon is not what we had hoped for, as be report in the rest of the paper, it is not necessarily bad.

3. CATEGORIZATION METHOD

The result of our previous work cited above indicates that software systems have multiple 'functional aspects'. Functional aspects are, for example, "compiler", "editor", "database", "runs on Windows", "supporting regular expression," and so forth. Consider an editor on Windows. This editor has not only "editor" functional aspect, but also "runs on Windows" functional aspect.

The software systems can be categorized with functional aspects on a nonexclusive basis. If a categorization is mutually exclusive, the categorization may capture only a few functional aspects.

We focus on identifiers (variable name, function name and so on) included in source code to retrieve a functional aspect. For example, "`gtk_window`" identifier represents some window, and source codes near the identifier would contain GUI operation.

As stated above, identifiers may represent a part of functions implemented in the program. If relationship between identifiers are found, they would represent one functional aspect. To determine relationships between identifiers, we use Latent Semantic Analysis(LSA), an information retrieval method explained below.

3.1 Latent Semantic Analysis (LSA)

Latent Semantic Analysis, LSA, is a practical method for the characterization of word meaning. LSA produces measures of word-word, and passage-passage relations which are well correlated with semantic similarity [11]. The method creates a vector description of documents. This representation is used for comparing and indexing documents, and various similarity measures can be defined.

Consider the six simple documents in Figure 1. In LSA, these documents are represented by a matrix shown in Table 1. Each column means a document and each row represents a word which may appear in the documents. Cell entries show the occurrence of the word in the document.

c1: Human machine interface for ABC computer applications
c2: A survey of user opinion of computer system response time
c3: Relation of user perceived response time to error measurement
m1: The generation of random, binary, ordered trees
m2: Graph minors IV: Widths of trees and well-quasi-ordering
m3: Graph minors: A survey

Figure 1: Example Input Documents

	c1	c2	c3	m1	m2	m3
computer	1	1	0	0	0	0
user	0	1	1	0	0	0
response	0	1	1	0	0	0
time	0	1	1	0	0	0
survey	0	1	0	0	0	1
trees	0	0	0	1	1	0
graph	0	0	0	0	1	1
minors	0	0	0	0	1	1

Table 1: An Example of LSA Matrix

Each row vector of this matrix indicates the characteristics of the word through the whole documents occurrences. This row vector can be used to determine the similarity of two words. A simple similarity definition used here is $cosine$ of two vectors.

In LSA, single value decomposition (SVD) is applied to the matrix. SVD is a form of factor analysis and acts as a method for reducing the dimensionality of the matrices. Why does LSA apply such translation? This is because a simple term-by-document matrix does not capture relationship among terms. Two documents show high similarity only when the documents have some same words; however, there are many synonyms. Thus similar documents do not always share completely same words. They may contain many synonyms. Using SVD, LSA can retrieve such undirectional relationship among documents. For more details, please refer [11].

3.2 Overview of Classification Method

Our method consists on 7 parts, explained in brief below:

1. Extract identifiers.

   First, we extract all identifiers from source code of software systems. We don't use reserved words in programming language and words in comments. Reserved words are meaningless from the viewpoint of function. Comments are abstract description, but amount and quality of comments in each software systems vary widely. Thus we cut out reserved words and comments.

2. Create identifier-by-software matrix.

   We create an identifier-by-software matrix, similar to the word-by-document matrix of Table 1.

3. Remove meaningless identifiers.

   Before performing LSA, we remove identifiers that appear in only one software system, or in more than half of software systems. Identifiers appearing in only one software are not meaningful in LSA. And, identifiers appearing in more than half of software systems are probably a general term and have no affect on categorization.

4. Perform LSA.

   We perform LSA for the identifier-by-software matrix without meaningless identifiers.

5. Compute $cosine$ of each identifiers and perform cluster analysis.

   From the matrix of LSA result, we compute $cosine$ values of each identifiers. Thereafter, we apply cluster analysis using calculated similarities. Cluster analysis is statistical analysis method that cluster individuals into clusters based on similarity among individuals.

6. Make software clusters from identifier clusters.

   From each identifier clusters, we retrieve software systems that contain one or more identifiers in the cluster, and make them a corresponding software cluster.

7. Make titles of software clusters.

   We obtain software clusters by previous steps, however, each software cluster needs description that explains what software systems are included. As titles, we use the ten highest score identifiers in the clusters.

4. EXPERIMENTS

As an implementation of our method, we created a prototype system. We experimented categorization of software systems using the prototype. The overall goals of our experiment were: Does our prototype categorize proper by target systems compared with existing manual categorization? Can our prototype categorize by the libraries use in the system?

4.1 Experiment Process

We collect sample data from SourceForge. We selected 41 C programs in five categories from SourceForge. The list of categories and software systems are in Table 2. Then we ran our categorization tool on the 41 programs.

4.2 Result

Table 3 shows a part of the categorization result by our method. Each row represents one cluster.

We got 40 clusters in total. The target systems in 18 clusters fall in the same categorization as SourceForge categorization. There are 8 clusters in which all software systems depend on same library or have same architecture. In top 20 clusters, 17 clusters fall in the same categorization in SourceForge or same library.

For example, cluster 3, 8 or 9 are clustered since the software systems in those clusters use the same library. Cluster 3 contains software systems using YACC(Yet Another Compiler-Compiler). Cluster 8 and 9 contains software systems using GTK. In the same manner, we can get the following clusters.

Cluster 22 Software systems using regular expression pattern matching library.

Cluster 25 Software systems implementing JNI(Java Native Interface).

Cluster 30 Software systems using getopt, a function which parses a command line argument.

Cluster 32 Software systems implementing Python/C.

Cluster 35 Software systems using YACC(Yet Another Compiler-Compiler).

On the other hand, there are 12 software systems that are not classified into any categories.

4.3 Discussion

Comparing with existing categorization, many clusters of our result follow existing categorization however, our result does not cover whole existing categorization. This is because our result has software systems that are not classified any categories. Our method categorizes based on appearance frequency of identifiers. This makes software systems that have few tokens tend to be not classified any categories.

We verify that our method can retrieve new categorization that are not considered by existing categorization. Such categorizations are: (1) by libraries (GTK, yacc, etc.), and (2) by depending architecture (JNI, Python/C, etc.). Our method does not need any human knowledge. Thus, if a new library appears, our method can follow such change automatically.

About cluster titles, there are some unidentifiable titles like cluster 1; however, cluster 4 and 6 have clear-cut titles “AVI” and “board, ply”. The clusters with software systems using same library tend to have clear-cut titles.

5. CONCLUSION

In this paper, we have proposed automatic categorization method for many software systems. Our method finds categorization clusters and classifies software systems based on the clusters. We have shown this method can classify without any knowledge about target software systems.

For future work, we will seek how to determine parameters and retrieving intuitive cluster titles. Furthermore, we will add large-scale experimentation. To do this, we need to improve system performance and scalability.

6. REFERENCES

[1] N. Anquetil and T. Lethbridge. Extracting concepts from file names; a new file clustering criterion. In *International Conference on Software Engineering,(ICSE’98)*, pages 84–93, Apr 1998.

[2] A. Chan and T. Spracklen. Discoverying common features in software code using self-organising maps. In *International Symposium on Computational Intelligence (ISCI’2000)*, Kosice, Slovakia, August 2000.

Category	Software
boardgame	Sjeng-10.0, bingo-cards, btechmux-1.4.3, cinag-1.1.4, faile_1_4_4, gbatnav-1.0.4, gchch-1.2.1, ics-Drone, libgmonopd-0.3.0, netships-1.3.1, nettoe-1.1.0, nngs-1.1.14, ttt-0.10.0
compilers	clisp-2.30, csl-4.3.0, freewrapsrc53, gbdk, gprolog-1.2.3, gsoap2, jcom223, nasm-0.98.35, pfe-0.32.56, sdcc
database	centrallix, emdros-1.1.4, firebird-1.0.0.796, gtm_V43001A, leap-1.2.6, mysql-3.23.49, postgresql-7.2.1
editor	gedit-1.120.0, gmas-1.1.0, gnotepad+-1.3.3, molasses-1.1.0, peacock-0.4
videoconversion	dv2jpg-1.1, libcu30-1.0, mjpgTools, mpegsplit-1.1.1
xterm	R6.3, R6.4

Table 2: The list of sample software systems

[3] K. Chen and V. Rajlich. Case study of feature location using dependency graph. In *8th International Workshop on Program Comprehension (IWPC'00)*, pages 231–239, Limerick, Ireland, June 2000.

[4] S. C. Choi and W. Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, 7(1):66–71, Jan 1990.

[5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[6] J. Dinkelacker and P. Garg. Corporate Source: Applying Open Source concepts to a corporate environment (Position Paper). In *Proceedings of the 1st ICSE workshop on Open Source software engineering*, Toronto, Canada, 2001.

[7] J. Dinkelacker, P. Garg, D. Nelson, and R. Miller. Progressive Open Source. In *Proceedings of the International Conference on Software Engineering*, Orlando, Florida, 2002.

[8] W. B. Frakes and T. Pole. An empirical study of representation methods for reusable software components. *IEEE Transactions on Software Engineering*, 20(8):617–630, 1994.

[9] J. Herbsleb and A. Mockus. An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions. Software Engineering*, 2003.

[10] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue. Automatic categorization algorithm for evolvable software archive. In *2003 International Workshop on Principles of Software Evolution(IWPSE 2003)*, Sep 2003.

[11] T. K. Landauer and S. T. Dumais. Latent Semantic Analysis and the Measurement of Knowledge. In *Educational Testing Service Conference on Natural Language Processing Techniques and Technology in Assessment and Education*, princeton, 1994.

[12] G. A. D. Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. D. Carlini. Comprehending web applications by a clustering based approach. In *Proc. of 10th International Workshop on Program Comprehension(IWPC'02)*, pages 261–270, Paris, France, June 2002.

[13] Y. S. Maarek, D. M. Berry, and G. E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions of Software Engineering*, 17(8):800–813, 1991.

[14] J. I. Maletic and A. Marcus. Using latent semantic analysis to identify similarities in source code to support program understanding. In *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00)*, pages 46–53, November 2000.

[15] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering(ICSE2003)*, pages 125–135, Portland, OR, May 2003.

[16] R. Schwanke. An intelligent tool for re-engineering software modularity. In *Proc. of 13th International Conference on Software Engineering*, pages 83–92, Austin, Texas, USA, May 1991.

[17] SOURCEFORGE.net. `http://sourceforge.net`.

No.	Title of cluster	Software	The number of tokens
1	AOP, emitcode, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT, pic14_emitcode, iCode, etype	compilers/gbdk, compilers/sdcc	8597
2	CASE_IGNORE, CASE_GROUND_STATE, screen, CASE_PRINT, CASE_BYP_STATE, Widget, TScreen, CASE_IGNORE_STATE, CASE_PLT_VEC, CASE_PT_POINT	xterm/R6.3, xterm/R6.4	2160
3	YY_BREAK, yyvsp, yyval, DATA, yy_current_buffer, tuple, yy_current_state, yy_c_buf_p, yy_cp, uint32	compilers/gbdk, database/mysql-3.23.49, database/postgresql-7.2.1	223
4	AVI, cinfo, OUTLONG, avi_t, AVI_errno, hdrl_data, OUT4CC, nhb, ERR_EXIT, str2ulong	videoconversion/dv2jpg-1.1, videoconversion/libcu30-1.0, videoconversion/mjpgTools	177
5	domainname, msgid1, binding, msgid2, domainbinding, pexp, __builtin_expect, transmem_list, codeset, codesetp	boardgame/gbatnav-1.0.4, boardgame/gchch-1.2.1	165
6	board, num_moves, ply, pawn_file, npiece, pawns, moves, white_to_move, move_s, promoted	boardgame/Sjeng-10.0, boardgame/cinag-1.1.4, boardgame/faile_1_4_4	154
7	xdrs, blob, DB, UCHAR, XDR, mutex, key_length, logp, page_no, bdb	database/firebird-1.0.0.796, database/mysql-3.23.49	118
8	domainname, N_, binding, gchar, GtkWidget, PARAMS, codeset, gpointer, loaded_l10nfile, argz	boardgame/gbatnav-1.0.4, boardgame/gchch-1.2.1, editor/gnotepad+-1.3.3, editor/peacock-0.4	118
9	GtkWidget, gchar, gpointer, gint, widget, gtk_widget_show, N_, g_free, dialog, g_return_if_fail	boardgame/gbatnav-1.0.4, editor/gedit-1.120.0, editor/gmas-1.1.0, editor/gnotepad+-1.3.3, editor/peacock-0.4	104
10	AOP, emitcode, esp, IC_RESULT, IC_LEFT, obstack, aop, mov, aopGet, IC_RIGHT	compilers/clisp-2.30, compilers/gbdk, compilers/sdcc	100
11	tuple, uint32, plan, int32, lsn, elm, rec, interp, TCL_ERROR, finfo	database/mysql-3.23.49, database/postgresql-7.2.1	79
12	xdrs, blob, DB, UCHAR, XDR, mutex, key_length, logp, page_no, bdb	database/firebird-1.0.0.796, database/mysql-3.23.49	73
13	UCHAR, relation, stmt, trigger, yyvsp, yyval, t_data, plan, dbname, USHORT	database/firebird-1.0.0.796, database/postgresql-7.2.1	68
14	fout, interp, TCL_ERROR, typ, YY_RULE_SETUP, List, DATA, Tcl_Interp, id, YY_BREAK	compilers/freewrapsrc53, compilers/gbdk, compilers/gsoap2, database/postgresql-7.2.1	50
15	GtkWidget, gchar, gpointer, dlg, gint, g_free, gtk_widget_show, gtk, GList, GTK_BOX	editor/gedit-1.120.0, editor/gmas-1.1.0, editor/gnotepad+-1.3.3	46
16	UCHAR, relation, stmt, trigger, yyvsp, yyval, t_data, plan, dbname, USHORT	database/firebird-1.0.0.796, database/postgresql-7.2.1	43
17	AOP, emitcode, mfp, ic, uchar, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT	compilers/gbdk, compilers/sdcc, database/mysql-3.23.49	36
18	adr, FX, word, stm, ED, xt, REF, prop, term, FP	compilers/gprolog-1.2.3, compilers/pfe-0.32.56	35
19	AOP, emitcode, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT, pic14_emitcode, iCode, etype	compilers/gbdk, compilers/sdcc, database/firebird-1.0.0.796	31
20	dyn, FPRINTF, process_id, p_offset, ctl, rab, que, io_ptr, prior, PRINTF	database/firebird-1.0.0.796, database/gtm_V43001A_src_linux	29
21	dyn, FPRINTF, process_id, p_offset, ctl, rab, que, io_ptr, prior, PRINTF	database/firebird-1.0.0.796, database/gtm_V43001A_src_linux	27
22	regparse, dbp, mech, reginput, flagp, NOTHING, tuple, db, __P, regnode	boardgame/btechmux-1.4.3, database/leap-1.2.6, database/mysql-3.23.49	26
23	rectype, argp, rec, fileid, save_errno, data_len, qp, argpp, int4, dbp	database/gtm_V43001A_src_linux, database/mysql-3.23.49	26
24	AOP, emitcode, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT, pic14_emitcode, iCode, etype	compilers/gbdk, compilers/sdcc, videoconversion/mjpgTools	26
25	jobject, JNIEnv, JNICALL, JNIEXPORT, jint, jstring, interp, TCL_ERROR, objv, TCL_OK	compilers/freewrapsrc53, compilers/jcom223, compilers/pfe-0.32.56, database/mysql-3.23.49	24
26	entrypoint, USHORT, TEXT, yyvsp, raddr, R, UCHAR, yyval, blob, REQ	compilers/clisp-2.30, database/firebird-1.0.0.796	17
27	int32_t, dbp, cinfo, net, unpack, argp, sinfo, cur1, purpose, mysql	database/mysql-3.23.49, videoconversion/mjpgTools	17
28	AOP, emitcode, mfp, ic, uchar, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT	compilers/gbdk, compilers/sdcc, database/mysql-3.23.49	16
29	USHORT, UCHAR, blob, REQ, NULL_PTR, hIcon, SCHAR, interp, wndclass, bdb	compilers/freewrapsrc53, database/firebird-1.0.0.796	16
30	optind, nextchar, __P, optstring, last_nonopt, option_index, uchar, optarg, pfound, dbp	boardgame/ttt-0.10.0, compilers/clisp-2.30, database/mysql-3.23.49	15
31	int4, ctl, tn, rec, semid, blkno, ti, oprtype, save_errno, AH	database/gtm_V43001A_src_linux, database/postgresql-7.2.1	14
32	notify, mech, PyObject, fargs, Node, Name, pset, zone, tprintf, NOTHING	boardgame/btechmux-1.4.3, database/postgresql-7.2.1	11
33	interp, notify, dbp, tuple, mech, PyObject, uint32, plan, int32, buff	boardgame/btechmux-1.4.3, database/mysql-3.23.49, database/postgresql-7.2.1	10
34	adr, stm, AOP, emitcode, operands, ASSERT, IC_RESULT, pred, lg, REF	compilers/gprolog-1.2.3, compilers/sdcc	9
35	yyvsp, yyn, PARAMS, codeset, domainname, msgid1, binding, msgid2, yylsp, domainbinding	boardgame/gbatnav-1.0.4, boardgame/gchch-1.2.1, compilers/clisp-2.30	9
36	ERREXIT, picture, pool_id, USHORT, get_buffer, output_buf, cinfo, xxx, UCHAR, streams	database/firebird-1.0.0.796, videoconversion/mjpgTools	9
37	REF, dyn, USHORT, vec, path_name, clause, STATUS, E, UCHAR, CSB	compilers/gprolog-1.2.3, database/firebird-1.0.0.796	8
38	AOP, emitcode, pfile, ic, IC_RESULT, IC_LEFT, aop, aopGet, IC_RIGHT, pic14_emitcode	compilers/gbdk, compilers/sdcc, database/postgresql-7.2.1	7
39	ic, ply, npiece, score, AOP, pawn_file, uchar, bking_loc, wking_loc, emitcode	boardgame/Sjeng-10.0, compilers/gbdk	7
40	clause, cinfo, pred, ci, Group, Np, word, X, A, tmp4	compilers/gprolog-1.2.3, database/postgresql-7.2.1, videoconversion/mjpgTools	6

Table 3: 41