On Automatic Categorization of Open Source Software

Shinji Kawaguchi [†], Pankaj K. Garg [††]
Makoto Matsushita [†] and 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
Sunnyvale, California, 94807, USA
garg@zeesource.net

Abstract

The number of Open Source software systems is increasing at a rapid rate. For example, SourceForge currently has about fifty-five thousand software systems registered, twenty-two thousand of which were added in the past twelve months. With such a large number of software projects, locating the right project for a given purpose can obviously be quite challenging. We propose to use automatic software categorization to address this challenge. At present, we leave open the issue of the nature of the categorization, and explore several known approaches like code clones-based similarity metric, decision trees, and latent semantic analysis. The results from applying each of the approaches gives us some insights into the problem space, and sets some directions for further work.

1. Introduction

We report on experiments to automatically deduce categories of Open Source software systems. Categorization of Open Source software systems can be helpful in several ways:

- Several *similar* software can be grouped together in a category for ease of browsing. For example, SourceForge [13] 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, so they can avoid duplicate work and promote more reuse. This becomes specially useful in situations like Corporate Source [5, 6], where global groups in companies may not be aware of the relationship among their work [7].

In the past, such relationships have been determined manually. With the increase in the number of Open Source software, e.g., SourceForge now has over fifty-five thousand software systems registered and continues to grow, such manual determination is not enough.

Automatic categorization of software systems is a novel and intriguing challenge. Manual categorization generally requires deep understanding of not only the target software system, but also other software systems and their classification policy. Past work in software engineering (e.g., see [2, 12]), has focused on determining relations *among components* of a given software system. We, however, propose finding relationships *among many software systems*.

We have experimented with three approaches for automatic categorization of software systems:

1. We use a similarity measure based on code-clone detection [8, 14]. Replicated code portions existing at different location in the source code are called *code clones*. The ratio of the total lines of code clones to the total lines of software is defined as the *similarity* of two software systems.

   This similarity measure has previously been useful in characterizing the evolution history of software systems, i.e., tracing ancestors and descendants of a software versions. In this paper, we investigate the use of this measure for software categorization.

2. Generating decision trees from example classes is a common approach in supervised machine learning. In this approach, an example category set of software and their features is fed to a learning algorithm. From this

example category set, the rule learner determines rules (or a decision tree) that helps categorize any future software system.

3. Latent Semantic Analysis(LSA) is a method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text [9]. LSA has found a variety of uses ranging from understanding human cognition [9] to data mining [3]. Also, it is used for clustering components in a software system [10]. We apply LSA for determining categories of software systems.

The rest of the paper is structured as follows: in sections 2, 3, and 4, we describe the application and use of SMAT, C4.5, and LSA, respectively. We conclude with a discussion of current and future work in section 5.

2. Similarity Measure by SMAT

SMAT is a tool to measure a similarity between two large software systems based on correspondence of each source-code line of the systems [14]. To get the correspondence efficiently, a fast code-clone detection tool CCFinder [8] is employed to detect file pairs sharing common clones. The detail line-by-line correspondence is then computed using file difference detection tool *diff* [4].

Similarity is defined as the ratio between LOC (lines of code) in the correspondence and the total LOC of two software systems. Thus, by counting the number of lines in the correspondence, the similarity of two software systems can be obtained.

This similarity measure characterizes evolution history of software systems. By applying it to BSD unix operating systems, we automatically classified versions of released operating systems into FreeBSD, NetBSD, and OpenBSD.

We applied this measure to classify several software systems in SourceForge. Table 1 shows the resulting similarity between each pairs of systems.

As seen from the table, the similarity values are generally low even for software systems that belong to the same manually determined category. This shows that although systems in the same category provide similar features, they do not share much code. This obviously raises the question of why these developers have chosen to provide different implementations for similar features?

3. Decision Trees

Decision trees are a machine learning approach for automatic classification of a data set. The decisions trees are based on certain *features* of the data points. Initially, an example set of data points with known classification and features is fed to a *rule-learner*. The rule-learner uses the example set to develop a set of rules. These rules operate on the features of the data points to classify any future data.

C4.5 [11] is a commonly available Open Source tool that can be used for classifying a set of data points. We used C4.5 on 41 software systems from SourceForge. For features, we used 3-gram representation of filenames used in the software source code. This is similar in principle to the approach used by Anquetil and Lethbridge [1] for their approach for clustering of software components. Instead of deriving concepts, however, we directly used 3-grams. The 41 software resulted in 6977 features (3-gram representation of filenames). The resulting decision tree is shown in Figure 1.

```
Read 41 cases (6977 attributes) from 3gram.data

Decision Tree:

tyx = t: xterm (2.0)
tyx = f:
|   _fu = t: database (6.0)
|   _fu = f:
|   |   mpe = t: videoconversion (3.0)
|   |   mpe = f:
|   |   |   alo = t: editor (4.0)
|   |   |   alo = f:
|   |   |   |   ops = t: database (2.0/1.0)
|   |   |   |   ops = f:
|   |   |   |   |   win = t: compilers (6.0)
|   |   |   |   |   win = f:
|   |   |   |   |   |   tin = t: compilers (2.0)
|   |   |   |   |   |   tin = f:
|   |   |   |   |   |   |   Lib = t: compilers (2.0)
|   |   |   |   |   |   |   Lib = f: boardgame (14.0/1.0)

Evaluation on training data (41 items):

        Before Pruning           After Pruning
       ----------------   ---------------------------
       Size      Errors   Size      Errors   Estimate

         17    2( 4.9%)     17    2( 4.9%)    (29.4%)   <<
```

Figure 1. Decision tree based on application of C4.5 to 41 software

As shown in Figure 1, the results of this classification are encouraging. The training data set recorded an error of less than 5%. The only drawback of this approach is that it requires up-front a set of example categories. We would like to discover categories or classifications that we don't even know exist! The latent semantic analysis approach presented in the next section provides such a capability.

4. 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 [9]. The method creates a vector description of documents. This representation is used for comparing and indexing documents, and various similarity measures can be defined using it.

LSA is based on a single value decomposition (SVD) of

	D1	D2	D3	D4	D5	E1	E2	E3	E4	X1	X2
D1: centrallix	1	0	0	0	0	0	0	0	0	0	0
D2: gtm_V43001A	0	1	0	0.00003	0.00012	0	0	0	0	0	0
D3: leap-1.2.6	0	0	1	0.00345	0.00003	0	0	0	0	0	0
D4: mysql-3.23.49	0	0.00003	0.00345	1	0.0111	0	0	0	0	0	0
D5: postgresql-7.2.1	0	0.00012	0.00003	0.0111	1	0	0	0	0	0	0
E1: gedit-1.120.0	0	0	0	0	0	1	0	0.00176	0	0	0
E2: gmas-1.1.0	0	0	0	0	0	0	1	0	0	0	0
E3: gnotepad+-1.3.3	0	0	0	0	0	0.00176	0	1	0.10	0	0
E4: peacock-0.4	0	0	0	0	0	0	0	0.10	1	0	0
X1: R6.3	0	0	0	0	0	0	0	0	0	1	0.64
X2: R6.4	0	0	0	0	0	0	0	0	0	0.64	1

Table 1. Similarity between Systems in SourceForge by Smat

a matrix derived from a word set of target documents. SVD is a form of factor analysis and acts as a method for reducing the dimensionality of the matrices.

Consider a simple matrix shown in Table 2. 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.

This matrix is the input of LSA, and SDV has been applied to this matrix. The result we obtain is the two-dimensionally reconstructed matrix shown in Table 3.

	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 2. An Example of Matrix for the Input of LSA

Each column vector of this matrix indicates the characteristics of the document through the word occurrences, which are not only directly visible ones but also indirectly related ones. This vertical vector can be used to determine the similarity of two documents. A simple similarity definition used here is $cosine$ of two vectors [9].

	c1	c2	c3	m1	m2	m3
computer	0.12	0.76	0.53	-0.02	-0.02	0.10
user	0.18	1.11	0.78	-0.04	-0.10	0.09
response	0.18	1.11	0.78	-0.04	-0.10	0.09
time	0.18	1.11	0.78	-0.04	-0.10	0.09
survey	0.11	0.75	0.45	0.10	0.46	0.55
trees	-0.02	-0.02	-0.11	0.16	0.64	0.59
graph	0.00	0.08	-0.09	0.24	0.99	0.93
minors	0.00	0.08	-0.09	0.24	0.99	0.93

Table 3. Resulting Two Dimensionally Reconstructed Matrix

4.1. Applying LSA to Classification of Software Systems

We applied LSA to classification of software systems. One key factor of this application is selection of words. We might be able to get the input word list for LSA from the documentation associated with target software systems. Also, we could obtain meaningful word lists from comments embedded into the source code.

The former approach would work if the documentation of each software systems are rich enough. Many Open Source software systems, however, do not have sufficient documentation, especially when a project is in its early phases. Also the granularity of description would be unstable over documentats. Some documents focus only on usage of the systems, while others only describe the details of implementation.

Using words from program comments is more closely related to the implementation of the target software systems, and the granularity would be more stable. In many Open Source software systems, however, comments in the source code contains a lot of sentences for their license policy and system evolution, which are very important for the developers of Open Source software, but which may not be relevant for classification purpose and would have to be removed. Identifying the license policy and evolution history automatically would not be so easy.

For the work reported in this paper, we deleted all comments and used only identifiers (variable, constants, and function names) found in the source code as words. A software system is composed of many source code files, and each source code file is made up of a sequence of tokens in a particular programming language. The tokens can be categorized into two sets: keywords specified by the programming language, and identifiers given by the developers. The keywords generally are common over many software systems, hence we use identifiers for the input of LSA.

The following process overviews the process of classification using LSA.

1. Collect source code files of software systems.
2. Remove comments and extract tokens. Keywords in the programming language are discarded and only identifiers are obtained.
3. Count the occurrence of each identifier and create the word matrix for the input of LSA.
4. Remove meaningless words (identifiers) from the matrix. Unique words appearing only in a software system are removed. Also, common words appearing in more than half of those systems are removed. These removed words would not contribute the classification.
5. Perform LSA.
6. Compute $cosine$ of each pair of the vertical vectors of the resulting matrix of LSA, and obtain the similarity value of two software systems.
7. Perform clustering using the similarity values.

4.2. Experiments

We collected the source code files of 41 software systems from SourceForge. They are classified into 6 groups (board game, compiler, database, editor, video conversion, and xterm) at SourceForge. This classification has been made by the developers decision.

The total number of different kinds of identifiers extracted from these 41 software systems are 164,102, and meaningless words mentioned above are removed from them. The remaining are 22,048 different identifiers. So a 41 x 22,048 matrix is the input of LSA.

The resulting similarity between the software systems is presented in Figure 2.

As you can see from Figure 2, the software systems in the manual classification groups of editor, video conversion, and xterm showed very high similarity each other. On the other hand, systems in board game, compiler, and database did not show high similarity. This is because in board game, compiler, or database, there are little common concept which characterize overall systems. Editor, video conversion, and xterm contain a lot of characterizing system call names and variable names among systems.

There are two systems in board game which show high similarity with editors. This is because it share the same GUI framework.

5. Conclusion

We have reported some preliminary work on *automatic categorization* of Open Source software systems. Such categorization can be useful for reification of a multitude of relationships among Open Source systems. To understand the nature of the problem and its parameters, we have reported on several experiments to applied well-known approaches to classification. Wherever possible, we have build-upon the results of previous work in the area of determining relationships among components of a software system. We applied a code-clone based similarity metric, a decision tree based approach, and a latent semantic analysis approach. In each of the cases, we have limited success with the parameters that we chose. Hence, further work is required to understand the appropriate automated techniques that can be applied for this purpose. We are actively pursuing this research direction.

Acknowledgments

References

[1] N. Anquetil and T. C. Lethbridge. Recovering Software Architecture from the Names of Source Files. *Journal of Software Maintenance: Research and Practice*, 11:201–221, 1999.

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

[3] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[4] Diffutls. `http://www.gnu.org/software/diffutils`.

[5] 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.

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

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

[8] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. "CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code". *IEEE Transactions. Software Engineering*, 28(7):654–670, 2002.

[9] 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.

[10] 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.

[11] Ross Quinlan. `http://www.cse.unsw.edu.au/~quinlan`.

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

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

[14] Tetsuo Yamamoto, Makoto Matsusita, Toshihiro Kamiya, and Katsuro Inoue. "Measuring Similarity of Large Software Systems Based on Source Code Correspondence". Technical Report of Dept. of ICS, IIP-03-03-02, 2002.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
1	boardgame/Sjeng-10.0	1	0.52	0.07	0.61	0.73	0.23	0.43	0.61	0.2	0.65	0.31	0.21	0.58	0.12	0.49	0.62	0.47	0.13	0.03	0.38	0.14	0.44	0.47	0.34	0.68	0.37	0.5	0.69	0.54	0.6	0.05	0.02	0.05	0.06	0.02	0.16	0.61	0.13	0.6	−0.01	−0.01
2	boardgame/bingo-cards	0.52	1	0.03	0.69	0.57	0.49	0.58	0.53	0	0.42	0.25	0.02	0.56	0.09	0.58	0.59	0.18	0.05	0.03	0.03	0.11	0.25	0.17	0.19	0.64	0.1	0.02	0.62	0.13	0.71	0.38	0.35	0.37	0.41	0.36	0.56	0.79	0.57	0.59	0.13	0.13
3	boardgame/btechmux-1.4.3	0.07	0.03	1	0.02	0.06	0.02	0.04	0.11	0.55	0.06	0.02	0.03	0.61	0.01	0.02	0.04	0.02	0.03	0.01	0.01	0.01	0.04	0.02	0.02	0.03	0.01	0.03	0.05	0.04	0.02	0.01	0	0	0	0	0.01	0.03	0	0.02	0	0
4	boardgame/cinag-1.1.4	0.61	0.69	0.02	1	0.86	0.19	0.34	0.59	0.04	0.3	0.06	0.02	0.55	0.04	0.3	0.33	0.1	0.01	0.12	−0.06	0.01	0.22	0.09	0.03	0.96	0.04	0.37	0.74	0.09	0.95	0.05	0.01	0.03	0.09	0.03	0.06	0.51	0.07	0.36	0.01	0.01
5	boardgame/faile_1_4_4	0.73	0.57	0.06	0.86	1	0.18	0.35	0.61	0.17	0.39	0.29	0.21	0.66	0.15	0.57	0.3	0.13	0.14	−0.01	−0.05	0.24	0.51	0.13	0.06	0.9	0.08	0.67	0.7	0.13	0.76	0.02	−0.02	0.02	0.04	0	0.17	0.56	0.15	0.38	0.01	0.01
6	boardgame/gbatnav-1.0.4	0.23	0.49	0.02	0.19	0.18	1	0.93	0.28	0.1	0.18	0.33	0.13	0.2	0.22	0.15	0.11	0.09	0.09	0.04	0.03	0.03	0.17	0.09	0.04	0.14	0.16	0.06	0.27	0.09	0.2	0.95	0.95	0.96	0.96	0.95	0.06	0.17	0.03	0.17	0.02	0.02
7	boardgame/gchch-1.2.1	0.43	0.58	0.04	0.34	0.35	0.93	1	0.43	0.21	0.39	0.39	0.24	0.34	0.44	0.22	0.2	0.15	0.13	0.04	0.07	0.04	0.26	0.14	0.18	0.27	0.24	0.09	0.53	0.16	0.38	0.8	0.78	0.82	0.8	0.79	0.07	0.26	0.04	0.29	0	0
8	boardgame/icsDrone	0.61	0.53	0.11	0.59	0.61	0.28	0.43	1	0.24	0.37	0.27	0.26	0.72	0.12	0.42	0.41	0.16	0.72	0.04	0.13	0.06	0.61	0.15	0.02	0.58	0.26	0.17	0.85	0.29	0.68	0.08	0.05	0.09	0.11	0.06	0.15	0.42	0.09	0.44	0.03	0.03
9	boardgame/libgmonopd-0.3.0	0.2	0	0.55	0.04	0.17	0.1	0.21	0.24	1	0.39	0.75	0.84	0.37	0.06	−0.04	0.01	0.01	0.06	0.09	0.01	0.03	0.12	0.01	0.03	0	0.04	0.02	0.22	0.07	0.08	−0.04	0.01	0.06	0.01	0.03	0.1	−0.03	0.01	0.01	0.01	0.01
10	boardgame/netships-1.3.1	0.65	0.42	0.06	0.3	0.39	0.18	0.39	0.37	0.39	1	0.47	0.44	0.37	0.1	0.27	0.34	0.27	0.16	0.21	0.11	0	0.25	0.27	0.8	0.32	0.15	0.19	0.43	0.16	0.31	0.02	0.02	0.06	0.05	0.04	0.25	0.37	0.22	0.26	0.02	0.02
11	boardgame/nettoe-1.1.0	0.31	0.25	0.02	0.06	0.29	0.33	0.39	0.27	0.75	0.47	1	0.91	0.21	0.06	0.31	0.16	0.13	0.09	0.11	0.01	0.27	0.34	0.14	0.01	0.04	0.08	0.11	0.24	0.09	0.07	0.17	0.23	0.28	0.24	0.25	0.42	0.26	0.32	0.21	0.06	0.06
12	boardgame/nngs-1.1.14	0.21	0.02	0.03	0.02	0.21	0.13	0.24	0.26	0.84	0.44	0.91	1	0.13	0.11	0.05	0.03	0.02	0.13	0.01	0	0.02	0.15	0.02	0.01	−0.02	0.03	0.05	0.25	0.06	0.06	−0.05	0.01	0.07	0.01	0.04	0.22	0.04	0.13	0.05	0.00	0.00
13	boardgame/ttt-0.10.0	0.58	0.56	0.61	0.55	0.66	0.2	0.34	0.72	0.37	0.37	0.21	0.13	1	0.2	0.62	0.45	0.13	0.44	0.05	0.08	0.05	0.5	0.13	0.06	0.6	0.18	0.4	0.62	0.22	0.51	0.06	0.01	0.05	0.06	0.02	0.38	0.61	0.36	0.5	0.07	0.07
14	compilers/clisp-2.30	0.12	0.09	0.01	0.04	0.15	0.22	0.44	0.12	0.06	0.1	0.06	0.11	0.2	1	0.2	0.04	0.05	0.05	0.02	0.06	0.02	0.24	0.05	0.01	0.01	0.06	0.06	0.37	0.08	0.06	0.05	0	0.07	0	0.02	0.03	0.06	0.04	0.13	0	0
15	compilers/csl-4.3.0	0.49	0.58	0.02	0.3	0.57	0.15	0.22	0.42	−0.04	0.27	0.31	0.05	0.62	0.2	1	0.48	0.21	0.28	0.04	0.01	0.37	0.62	0.21	0.02	0.41	0.24	0.45	0.38	0.13	0.23	0.05	0.01	0.03	0.04	0.01	0.76	0.84	0.75	0.66	0	0
16	compilers/freewrapsrc53	0.62	0.59	0.04	0.33	0.3	0.11	0.2	0.41	0.01	0.34	0.16	0.03	0.45	0.04	0.48	1	0.16	0.06	0.06	0.76	0.02	0.24	0.16	0.15	0.34	0.1	0.05	0.38	0.81	0.38	0.02	0	0.01	0.03	0	0.54	0.69	0.51	0.7	0.07	0.07
17	compilers/gbdk	0.47	0.18	0.02	0.1	0.13	0.09	0.15	0.16	0.01	0.27	0.13	0.02	0.13	0.05	0.21	0.16	1	0.04	0.08	0	0.05	0.12	1	0.04	0.25	0.04	0.08	0.17	0.09	0.07	0.01	−0.01	0.01	0.01	0	0.02	0.38	0.07	0.08	0	0
18	compilers/gprolog-1.2.3	0.13	0.05	0.03	0.01	0.14	0.09	0.13	0.72	0.06	0.16	0.09	0.13	0.44	0.05	0.28	0.06	0.04	1	0.07	0	0.02	0.6	0.04	0.02	0.07	0.03	0.08	0.34	0.05	0.06	0.02	0	0.02	0.01	0	0.08	0.06	0.02	0.02	0.01	0.01
19	compilers/gsoap2	0.03	0.03	0.01	0.12	−0.01	0.04	0.04	0.04	0.09	0.21	0.11	0.01	0.05	0.02	0.04	0.06	0.08	0.07	1	0.01	0.07	0.25	0.08	0.01	0.1	0.04	0.04	−0.01	0.03	0.04	0	0.01	0.02	0.04	0	0.02	0.08	0.01	0.06	0	0
20	compilers/jcom223	0.38	0.03	0.01	−0.06	−0.05	0.03	0.07	0.13	0.01	0.11	0.01	0	0.08	0.06	0.01	0.76	0	0	0.01	1	−0.01	0.12	0	0.1	−0.07	0.03	0	0.05	0.97	0.01	0.01	0.01	0.02	0	0	0.06	0.11	0	0.35	0	0
21	compilers/nasm-0.98.35	0.14	0.11	0.01	0.01	0.24	0.03	0.04	0.06	0.03	0	0.27	0.02	0.05	0.02	0.37	0.02	0.05	0.02	0.07	−0.01	1	0.7	0.06	0.01	−0.01	0.04	0.05	0.09	0.04	0.03	0.01	0	0.01	0.04	0.01	0.07	0.05	0.01	0.09	0	0
22	compilers/pfe-0.32.56	0.44	0.25	0.04	0.22	0.51	0.17	0.26	0.61	0.12	0.25	0.34	0.15	0.5	0.24	0.62	0.24	0.12	0.6	0.25	0.12	0.7	1	0.12	0.04	0.26	0.14	0.36	0.44	0.23	0.21	0.04	0.01	0.05	0.06	0.02	0.18	0.27	0.1	0.27	0.07	0.07
23	compilers/sdcc	0.47	0.17	0.02	0.09	0.13	0.09	0.14	0.15	0.01	0.27	0.14	0.02	0.13	0.05	0.21	0.16	1	0.04	0.08	0	0.06	0.12	1	0.04	0.24	0.04	0.08	0.16	0.09	0.05	0.01	0	0.01	0.01	0	0.02	0.38	0.08	0.08	0	0
24	database/centrallix	0.34	0.19	0.02	0.03	0.06	0.04	0.18	0.02	0.03	0.8	0.01	0.01	0.06	0.01	0.02	0.15	0.04	0.02	0.01	0.1	0.01	0.04	0.04	1	0.03	0.04	0.03	0.1	0.04	0.06	0.02	0	0.01	0.01	0.01	0.02	0.05	0.02	0.01	0	0
25	database/emdros-1.1.4	0.68	0.64	0.03	0.96	0.9	0.14	0.27	0.58	0	0.32	0.04	−0.02	0.6	0.01	0.41	0.34	0.25	0.07	0.1	−0.07	−0.01	0.26	0.24	0.03	1	0.03	0.53	0.67	0.1	0.86	0.01	−0.04	−0.01	0.04	−0.02	0.1	0.59	0.12	0.35	0	0
26	database/firebird-1.0.0.796	0.37	0.1	0.01	0.04	0.08	0.16	0.24	0.26	0.04	0.15	0.08	0.03	0.18	0.06	0.24	0.1	0.04	0.03	0.04	0.03	0.04	0.14	0.04	0.04	0.03	1	0.07	0.41	0.12	0.16	0.05	0.06	0.03	0.05	0.01	0.05	0.21	0.02	0.7	0.01	0.01
27	database/gtm_V43001A	0.5	0.02	0.03	0.37	0.67	0.06	0.09	0.17	0.02	0.19	0.11	0.05	0.4	0.06	0.45	0.05	0.08	0.08	0.04	0	0.05	0.36	0.08	0.03	0.53	0.07	1	0.11	0.12	0.13	0.03	0	0.02	0	0.01	0.03	0.29	0.03	0.14	0	0
28	database/leap-1.2.6	0.69	0.62	0.05	0.74	0.7	0.27	0.53	0.85	0.22	0.43	0.24	0.25	0.62	0.37	0.38	0.38	0.17	0.34	−0.01	0.05	0.09	0.44	0.16	0.1	0.67	0.41	0.11	1	0.21	0.85	0.02	−0.02	0.03	0.04	−0.01	0.09	0.45	0.06	0.56	0	0
29	database/mysql-3.23.49	0.54	0.13	0.04	0.09	0.13	0.09	0.16	0.29	0.07	0.16	0.09	0.06	0.22	0.08	0.13	0.81	0.09	0.05	0.03	0.97	0.04	0.23	0.09	0.04	0.1	0.12	0.12	0.21	1	0.15	0.02	0.02	0.03	0.03	0.01	0.09	0.24	0.02	0.46	0	0
30	database/postgresql-7.2.1	0.6	0.71	0.02	0.95	0.76	0.2	0.38	0.68	0.08	0.31	0.07	0.06	0.51	0.06	0.23	0.38	0.07	0.06	0.04	0.01	0.03	0.21	0.05	0.06	0.86	0.16	0.13	0.85	0.15	1	0.03	0	0.02	0.08	0.01	0.03	0.45	0.03	0.43	0	0
31	editor/gedit-1.120.0	0.05	0.38	0.01	0.05	0.02	0.95	0.8	0.08	−0.04	0.02	0.17	−0.05	0.06	0.05	0.05	0.02	0.01	0.02	0	0.01	0.01	0.04	0.01	0.02	0.01	0.05	0.03	0.02	0.02	0.03	1	0.99	0.99	0.99	0.99	0.01	0.06	0.00	0.03	0	0
32	editor/gmas-1.1.0	0.02	0.35	0	0.01	−0.02	0.95	0.78	0.05	0.01	0.02	0.23	0.01	0.01	0	0.01	0	−0.01	0	0.01	0.01	0	0.01	0	0	−0.04	0.06	0	−0.02	0.02	0	0.99	1	0.99	0.99	1	0.02	0.04	0	0.02	0.01	0.01
33	editor/gnotepad+-1.3.3	0.05	0.37	0	0.03	0.02	0.96	0.82	0.09	0.06	0.06	0.28	0.07	0.05	0.07	0.03	0.01	0.01	0.02	0.02	0.02	0.01	0.05	0.01	0.01	−0.01	0.03	0.02	0.03	0.03	0.02	0.99	0.99	1	0.99	1	0.03	0.05	0	0.01	0.01	0.01
34	editor/molasses-1.1.0	0.06	0.41	0	0.09	0.04	0.96	0.8	0.11	0.01	0.05	0.24	0.01	0.06	0	0.04	0.03	0.01	0.01	0.04	0	0.04	0.06	0.01	0.01	0.04	0.05	0	0.04	0.03	0.08	0.99	0.99	0.99	1	1	0.03	0.08	0.01	0.04	0.01	0.01
35	editor/peacock-0.4	0.02	0.36	0	0.03	0	0.95	0.79	0.06	0.03	0.04	0.25	0.04	0.02	0.02	0.01	0	0	0	0	0	0.01	0.02	0	0.01	−0.02	0.01	0.01	−0.01	0.01	0.01	0.99	1	1	1	1	0.02	0.04	0	−0.01	0	0
36	videoconversion/dv2jpg-1.1	0.16	0.56	0.01	0.06	0.17	0.06	0.07	0.15	0.1	0.25	0.42	0.22	0.38	0.03	0.76	0.54	0.02	0.08	0.02	0.06	0.07	0.18	0.02	0.02	0.1	0.05	0.03	0.09	0.09	0.03	0.01	0.02	0.03	0.03	0.02	1	0.79	0.99	0.59	0.02	0.02
37	videoconversion/libcu30-1.0	0.61	0.79	0.03	0.51	0.56	0.17	0.26	0.42	−0.03	0.37	0.26	0.04	0.61	0.06	0.84	0.69	0.38	0.06	0.08	0.11	0.05	0.27	0.38	0.05	0.59	0.21	0.29	0.45	0.24	0.45	0.06	0.04	0.05	0.08	0.04	0.79	1	0.81	0.75	0.01	0.01
38	videoconversion/mjpgTools	0.13	0.57	0	0.07	0.15	0.03	0.04	0.09	0.01	0.22	0.32	0.13	0.36	0.04	0.75	0.51	0.07	0.02	0.01	0	0.01	0.1	0.08	0.02	0.12	0.02	0.03	0.06	0.02	0.03	0.00	0	0	0.01	0	0.99	0.81	1	0.56	0	0
39	videoconversion/mpegsplit-1.1	0.6	0.59	0.02	0.36	0.38	0.17	0.29	0.44	0.01	0.26	0.21	0.05	0.5	0.13	0.66	0.7	0.08	0.02	0.06	0.35	0.09	0.27	0.08	0.01	0.35	0.7	0.14	0.56	0.46	0.43	0.03	0.02	0.01	0.04	−0.01	0.59	0.75	0.56	1	0.02	0.02
40	xterm/R6.3	−0.01	0.13	0	0.01	0.01	0.02	0	0.03	0.01	0.02	0.06	0.00	0.07	0	0	0.07	0	0.01	0	0	0	0.07	0	0	0	0.01	0	0	0	0	0	0.01	0.01	0.01	0	0.02	0.01	0	0.02	1	1
41	xterm/R6.4	−0.01	0.13	0	0.01	0.01	0.02	0	0.03	0.01	0.02	0.06	0.00	0.07	0	0	0.07	0	0.01	0	0	0	0.07	0	0	0	0.01	0	0	0	0	0	0.01	0.01	0.01	0	0.02	0.01	0	0.02	1	1

Figure 2. Similarity of software systems by LSA