MUDABlue: An Automatic Categorization System for Open Source Repositories

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, Suite 201
Sunnyvale, California, 94807, USA
garg@zeesource.net

Abstract

Open Source communities typically use a software repository to archive various software projects with their source code, mailing list discussions, documentation, bug reports, and so forth. For example, SourceForge currently hosts over seventy thousand Open Source software systems. Because of the size of the rich information content, such repositories offer numerous opportunities for sharing information among projects. For example, one would like to know a set of projects that are related or similar to each other, so that the project groups can collaborate and share their work. With thousands of projects in typical repositories, however, manually locating related projects can be difficult. Hence, we propose MUDABlue, a tool that automatically categorizes software systems. MUDABlue has three major aspects: 1) it relies on no other information than the source code, 2) it determines category sets automatically, and 3) it allows a software system to be a member of multiple categories. MUDABlue has a web interface to visualize determined categories, which eases browsing a software repository. We show the effectiveness of MUDABlue's categorization capability by comparing its generated categories with that of some other existing research tools.

1 Introduction

The rapid use of Internet and Web-based technology has given rise to novel, global software archiving services, pioneered in the Open Source community through SourceForge [22]. 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 systems can be grouped together in a category for ease of browsing. For example, SourceForge categorizes software according to their primary 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 developing software.
    2. Developers can leverage each other's work and promote more reuse. This becomes especially useful in situations like Corporate Source [8], where global groups in companies may not be aware of the relationship among their work [11].

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 categorization policy. With the increase in the number of software systems, e.g., SourceForge now has over seventy 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. Past work in

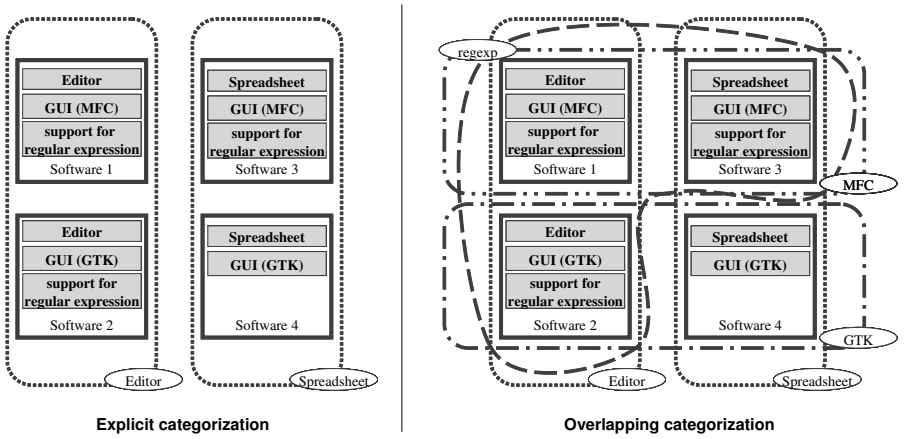

Figure 1. Categorizations overlapping disallowed and allowed

software engineering (e.g., see [6, 21]), has focused on determining *intra-component relations* of one given software system. We, however, propose finding *inter-project relations* of many software systems.

In this paper, we propose software automatic categorization system, MUDABlue. It is 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 [14]. LSA has found a variety of uses ranging from understanding human cognition [13] to data mining [7]. In software engineering field, it is used for clustering components in a software system [18], and recovering document-to-source links [19].

We apply LSA for automatic categorization of software systems. To apply LSA, we consider a software system as a document and an identifier as a word. We use LSA for retrieving categories, and then determining what software systems belong to those categories. We also implement the method and graphical user interface of retrieved categories. We define Unifiable Cluster Map method to draw a large number of categories and software systems. It is based on Cluster Map [9]. Using the MUDABlue interface, a user can browse a software repository more naturally.

2 Software Categorization

Our categorization process consists of two steps: First step is constructing concepts that define categories for subjects. Second step is categorizing items, i.e., assigning items to categories.

Existing researches on automatic software categorization [17, 25] have focused on only second step, categorizing items. They do not care, however, about first step, determining categories. Their methods require manual determination of category sets. They are inadequate considering that the definition of category sets have major implications for the categorization results, and difficulties of constructing category sets increases in proportion to the number of subjects.

Categorization made by hand is typically based on usage of software systems. We hypothesize, however, that it is also useful to categorize based on the software's inner structure. For example, there are categorization based on its architecture (GUI application, CUI based filter, or server application) and categorization based on required libraries (MFC, GTK or regexp library). To reflect such inner structures in categorization, we need detailed knowledge of many libraries and architectures. Thus, automating such categorization is important.

Additionally, the existing researches categorize each software system into one category. Therefore it is hard to accommodate a variety of categorization criteria into a single category hierarchy. In Figure 1, software 1 and 2 have functionality for editing (inserting text, overwriting, deleting, copying, etc.) and are categorized as 'editor'. Furthermore, software 1 and 3 are implemented with MFC library. Then thus they could be categorized into a same category.

It is hard to represent such characteristics for categorization which does not allow overlapping. In our previous work [12], we have tried to measure similarities between software systems. 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, despite differences in their usage. The results indicate that software systems have multiple 'functional aspects.' Thus, catego-

rization which allows overlapping is more suitable for software categorization.

On the basis of stated above, we list the required characteristics for an ideal automatic software categorization:

1. Don't need pre-defined categorizations.

   Category sets should be generated automatically. This is because a software system has several attributes, such as usage, architecture and depended library. Making category sets reflecting all these attributes is hard to determine *a priori*.

2. Allow a software system to be a member of multiple categories.

   Software systems can be categorized with functional aspects on a non-exclusive basis. If a categorization is mutually exclusive, the categorization may capture only a few functional aspects.

3. Rely on source code only.

   Some software systems have various artifacts like design documents, build script, and so on. Although these artifacts have highly abstracted information compared to source code, the amount and quality of documents are quite different with software systems. In contrast, source code is usually available for all projects, or at least the ones worth categorizing. Thus, we should use nothing but source code. Depending on only source code enlarges the applicability of the method.

3 MUDABlue Method

In this paper, we propose automatic software categorization method implementing the three characteristics stated above. At first, we introduce LSA used by MUDABlue method. Then, we show key idea of how to retrieve categories automatically and explain our automatic categorization method.

3.1 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 [14]. 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 Table 1. In LSA, these documents are represented by a matrix shown in Table 2. Each column represents a document and each row represents a word which appears 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

Table 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 2. 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 undirected relationship among documents. For more details, please see [14].

3.2 Retrieving Category

To retrieve categories from various software systems, we focus on identifiers (variable names, function names, and so on). Such identifiers are usually labeled according to their contents and deeply relate to the behavior of source code. For example, identifier
"`gtk_window`" represents some window, and source code near "`gtk_window`" will contain some GUI operation on the window.

As stated above, identifiers may represent a part of functions implemented in the program. Thus we assume that semantically related identifiers retrieved from many identifiers represent one "concept." "`gtk_window`", "`gtk_main`" and "`gpointer`" seem to indicate that such

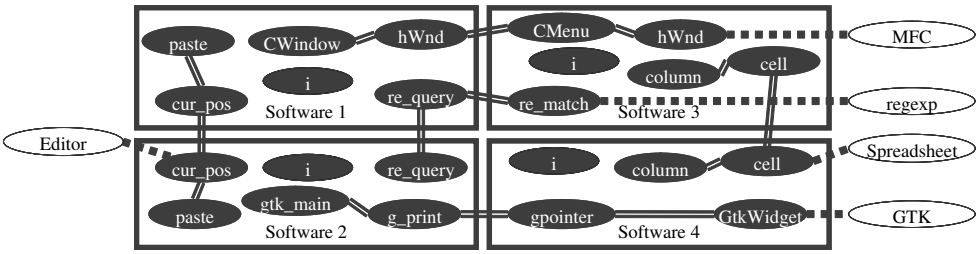

Figure 2. Retrieve categories from source code using relationships among identifiers

identifiers would be contained in GTK related software system. We define such a derived concept as a "category."

Then, we define that if a software system has identifiers contained in a category, that software belongs to the category (Figure 2). In this way, if we can retrieve "editor" category which contains identifiers "`cut, copy, paste`" and "MFC" category which contains identifiers "`CWindow, CMenu`." Software 1 in Figure 2 belongs to both "editor" category and "MFC" category.

To determine relationships between identifiers, we use LSA. LSA calculates the similarity values between words. LSA is purely statistical method and can be applied without any human knowledge. At the same time, the similarity values of LSA reflect highly the semantic relationship among words. Using LSA, we can get right relationships, even if there are many synonyms and homonyms.

3.3 Overview of MUDABlue Method

As stated in Section 3.2, MUDABlue retrieves highly related identifiers, and considers them as a category. In this section, we show how to retrieve categories concretely. Figure 3 shows a dataflow of MUDABlue method. MUDABlue method consists of 7 parts, each explained below:

1. Extract identifiers.

   First, we extract all identifiers from source codes of software systems. From identifiers, we exclude reserved because they have no relation with software features. We also cut out comments. The reason is that amount and quality of comments in each software system vary widely, and many software systems have copyright notice or license terms in comments.

2. Create identifier-by-software matrix.

   Considering a software system as a document and an identifier as a word, we create an identifier-by-software matrix, similar to the word-by-document matrix of Table 2.

3. Remove useless 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 system 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. Apply LSA.

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

5. Retrieve categories using similarities between identifiers.

   From the matrix of LSA result, we compute similarities between all pairs of identifiers. As stated above, we use $cosine$ criterion for the similarity of each identifiers. Thereafter, we apply cluster analysis using calculated similarities. The cluster analysis is a statistical analysis method that classifies individuals into clusters based on the similarity among individuals. We call a cluster of identifiers an "identifier cluster." We consider each identifier cluster as a category.

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 software cluster's titles.

   Although we obtain software clusters by previous steps, each software cluster needs description that explains what software systems are included. To generate titles, we sum all identifier-vectors comprised in the identifier cluster. We use ten identifiers that have highest values in the summation vector.

4 MUDABlue System

We implemented the MUDABlue method described in Section 3.3. To facilitate utilization of categories, we developed a graphical interface for browsing software repository.

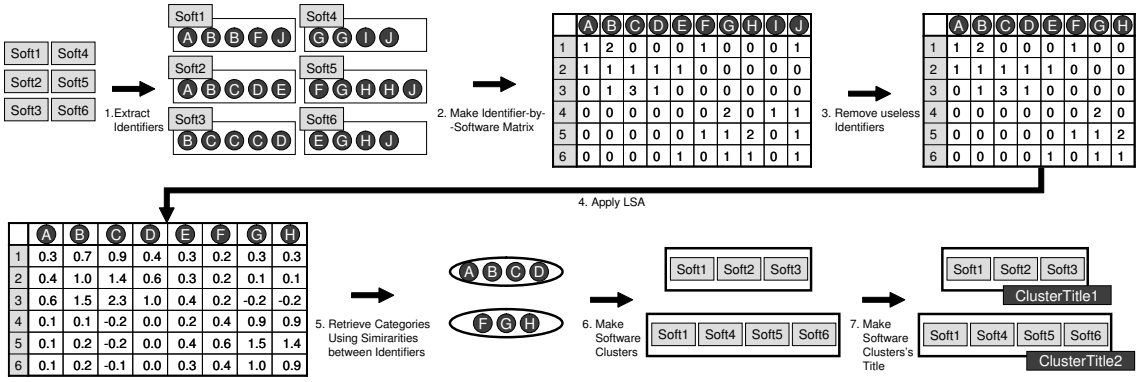

Figure 3. Algorithm of Retrieving Categories

Using our GUI, one can grasp a good overall picture of a software repository.

MUDABlue system consists of two parts: (1) database, and (2) user interface. Database part retrieves categories from source code contained in a software repository, and stores them into the database. User interface part enables browsing the software repository using categories stored in the database.

In this section, we give an explanation of Unifiable Cluster Map (UCM). UCM is developed to display relationships among categories and software systems graphically. Then we describe each part of MUDABlue system.

4.1 Unifiable Cluster Map

To draw categories retrieved by MUDABlue method, two requirements must be fulfilled. It must draw multiple memberships and have high scalability. Considering such requirements, we define a Unifiable Cluster Map method to draw categories and software systems based on Cluster Map [9].

In Cluster Map, a category is represented as a node (right side of Figure 5). A category is represented as a black node with a light-gray label. A charcoal gray square represents a cluster of elements. Elements belonging to the same categories are grouped into one cluster. The numeric characters inside the squares shows what elements are contained in the clusters. Edges between a category node and a cluster circle indicate elements inside the circle belonging to the category. In the right side of Figure 5, the item 1 and 2 belong to the category A and B. Cluster Map method draws multiple memberships in smart way. However, it does not have enough scalability.

To adapt for dozens of categories, we define UCM. In UCM, you can refine categories by unifying categories interactively. For instance, UCM draws some categories located higher position in the category hierarchy at first. If the user has some interest in a category, the user can expand it and get more detailed categories. If a particular category is not what the user wants, the user can collapse it.

Figure 4 shows how to unify categories. We would unify the category B and C. If the category B and C are unified, the cluster 1, 2 and 3 relate same categories. Thus, cluster 1, 2 and 3 are unified. Generally, if two categories are unified, we also unify all clusters which relate same categories. Repeating such unification, a highly abstracted figure is obtained like Figure 5.

4.2 Database

Database part retrieves categories using the method described in Section 3.3, and stores them into a database. Database part requires name of software systems and their source codes. It assumes that there is a directory for each software system, and source code is placed under the directory. All these directories are placed under a directory specified in a configuration file.

4.3 User Interface

Our system has a web-based interface (Figure 6). This interface consists of four parts. The top part is a keyword input box. The middle of the page is Unifiable Cluster Map view (UCM view). The left side of bottom is category hierarchy view, and at the right side of bottom is category/software list view.

Keyword input box provides the ability to search categories and software systems with given keyword.

UCM view provides the whole picture of a software repository. It draws categories, software systems, and their relationships. In UCM view, categories are arranged in a hierarchy by their similarities. UCM provides the brief map of categories and aids the exploration of details for interested users.

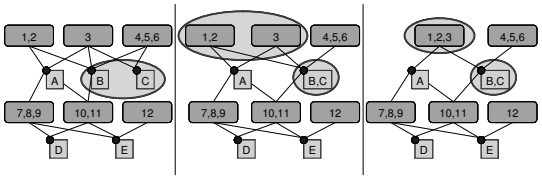

Figure 4. Unification of categories

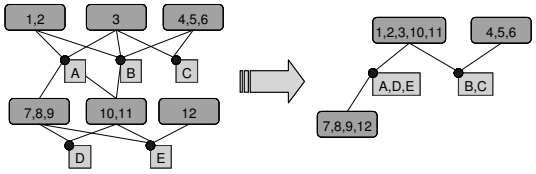

Figure 5. Collapsed categories

Category hierarchy view is another hierarchical view of categories. It presents category tree in direct way. If a user already has a desired category, he or she can access the category directly using this view.

Category/software list view shows the result of keyword search or selected category/software in the UCM view or category hierarchy view. The list of category shows selected categories and their software systems and the list of software systems shows selected software systems. In both pages, the category name and software name are links for detailed information pages.

All components in MUDABlue interface collaborate closely with each other. If a category was selected in UCM, the category is selected in category hierarchy view and category/software list view, too, and vice versa.

In this section, we present explanation about category hierarchy view, and how to construct hierarchy among categories at first. Then we explain UCM view.

4.3.1 Category hierarchy view

In category hierarchy view, the automatically determined categories are reconstructed into hierarchy and similar categories are arranged close to each other. In this way, the user can grasp the outline of categories more easily.

To determine category hierarchies, we define quantitative similarity between categories. In this paper, we use **odds ratio** with some modification. The odds ratio is a general measure for relationship between two categories. Say there is population S and its subsets $A, B \subset S$, our odds ratio $or'(A, B)$ is defined as below:

$$or'(A,B) = \begin{cases} ad/bc & \text{if } bc \neq 0 \\ ad|S|^2 & \text{otherwise} \end{cases}$$

where

$$a = |\bar{A} \cap \bar{B}|, b = |A \cap \bar{B}|, c = |\bar{A} \cap B|, d = |A \cap B|.$$

We apply cluster analysis to categories using $or'(A, B)$. Then we make the dendrogram of categories used for the category hierarchy view.

4.3.2 Unifiable cluster map view

Category hierarchy view enables viewing categories along with their relationships with each other. This view, however, does not present any information about the contained software systems. To grasp the overview of a software repository, such a view is insufficient.

In Figure 6, the middle of the interface is UCM part. A category is represented as a black node with a light-gray label. A group of software systems is represented as a charcoal gray square. An edge between a category node and a software group square depicts the fact that the software systems belong to the pointed category. The retrieved categories are unified using UCM method described in Section 4.1. The user can expand and collapse the categories interactively. The user can double-click the category node and expand it to determine its children categories. If the user turns the mouse wheel up, the selected category is collapsed again.

We use TouchGraph [24] library to implement UCM view. TouchGraph is graph drawing library. It dynamically arranges nodes by their edge connection, and allows the user to move nodes around the screen.

5 Experimentation

We have conducted some experiments with the MUDABlue software categorization system. The overall goals of our experiment were: Does our prototype properly categorize by target systems compared with existing manual categorization? Can our prototype categorize by the libraries used by the subject software systems?

5.1 Experiment Process

We collected sample data from SourceForge, selecting 41 C programs in five categories from SourceForge. The list of categories and software systems are shown in Table 3. Then we ran MUDABlue for the 41 programs.

Next, we evaluate the result using *precision* and *recall*. For this work, we define precision and recall from the view point of "Compared to an *ideal* categorization, how specific and exhaustive is the categorization determined by MUDABlue?" Let S be a set of all software systems contained in a repository. Precision and recall are defined as:

$$\text{precision} = \frac{\sum_{s \in S} \text{precision}_{\text{soft}}(s)}{|S|}$$

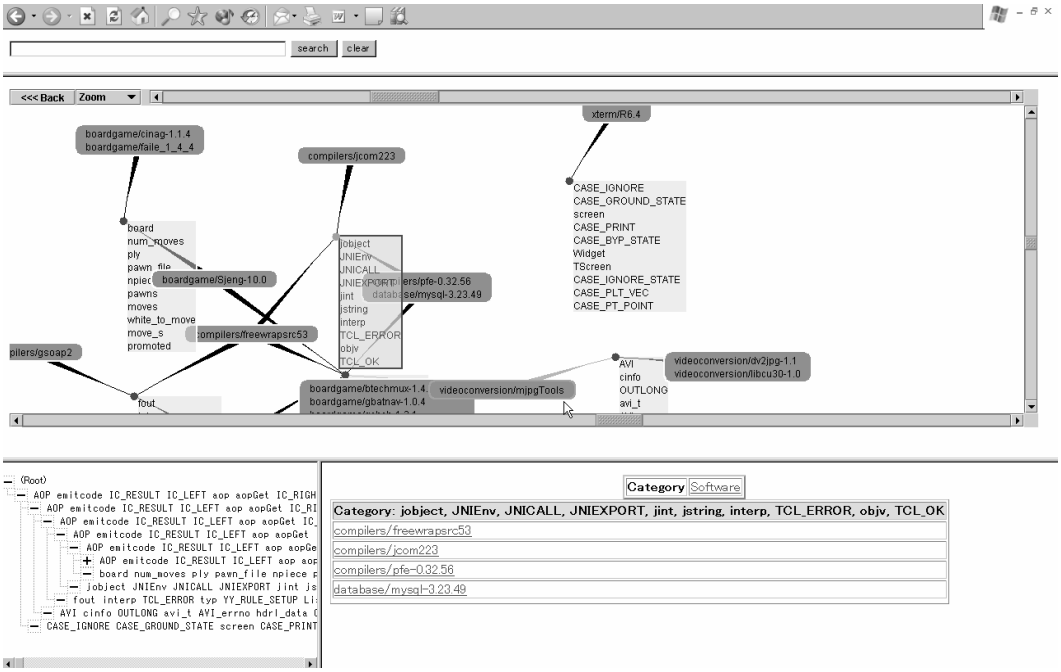

Figure 6. MUDABlue Interface

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, icsDrone, 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 3. The list of sample software systems

$$\begin{aligned}
\mathrm{precision}_{\mathrm{soft}}(s) &= \frac{|C_{\mathrm{MUDABlue}}(s) \cap C_{\mathrm{Ideal}}(s)|}{|C_{\mathrm{MUDABlue}}(s)|} \\
\mathrm{recall} &= \frac{\sum_{s \in S} \mathrm{recall}_{\mathrm{soft}}(s)}{|S|} \\
\mathrm{recall}_{\mathrm{soft}}(s) &= \frac{|C_{\mathrm{MUDABlue}}(s) \cap C_{\mathrm{Ideal}}(s)|}{|C_{\mathrm{Ideal}}(s)|},
\end{aligned}$$

where $C_{\mathrm{MUDABlue}}(s)$ is a set of categories containing software s, generated by MUDABlue, $C_{\mathrm{Ideal}}(s)$ is a set of categories containing software s, determined manually by the experimenters. In both of the criteria, larger value is preferable.

Besides precision and recall, we also employ F-Value to be an integrated measure for performance evaluation. F-value is harmonic average of precision and recall, denoted as $\frac{2pr}{p+r}$ where p:precision, r:recall.

5.2 Result

Table 4 shows excerpt of categories generated by MUDABlue. A row represents one category. Each column represents title of the category, software systems belong to the category and the number of identifiers constructing the category. We got 40 categories in this experimentation. 18 categories are the same as defined manually in SourceForge, and 8 categories are new categories based on depending libraries and architectures. All 8 new categories are No.3, 35 (YACC category), No.8, 9 (GTK category), No.22 (regexp category), No.25 (JNI category), No.30 (getopt category) and No.32 (Python/C category).

In Table 4, categories No.1, 2, 4, 5, 6, 7 and 10 are the same category as SourceForge, and categories No.3, 8 and 9 are new categories. Category No.3 contains software systems using YACC, and category No.8 and 9 contain software systems using GTK library.

No.	Title of cluster	Software	# 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
⋮			
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 4. MUDABlue Result (excerpt)

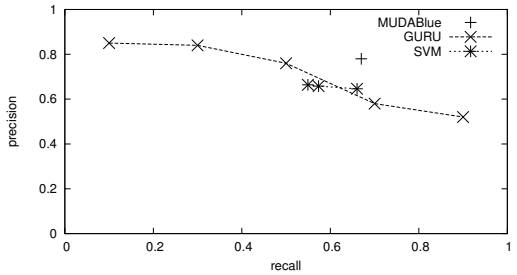

Figure 7. Precision-recall graph

Figure 7 shows the result of recall and precision. We also shows the result of GURU [17] and SVM based method proposed by Ugurel et al. [25] (SVM stands for Support Vector Machine). They proposed automatic software categorization methods using information retrieval (IR) technique similar to MUDABlue. GURU applied free-text indexing approach for software categorization. They retrieved information from Unix man pages and categorized Unix tools. Ugurel et al. applied SVM to documents of software systems to categorize software systems based on their usage.

In this graph, points located in the upper right area imply good result i.e. both precision and recall are high. This graph indicates that MUDABlue categorizes software with higher precision and recall, when compared to some existing systems. Actually, the F-Value of MUDABlue is 0.72 while the best F-Value of GURU is 0.6591 (where $\mathrm{precision} = 0.9$ and $\mathrm{recall} = 0.52$) and of SVM is 0.6531 (where $\mathrm{precision} = 0.66$ and $\mathrm{recall} = 0.65$)

6 Discussion

6.1 MUDABlue categorization method

In this paper, we have confirmed that MUDABlue can properly categorize based on not only usages but also architectures and depended libraries. For instance, MUDABlue retrieved GTK category, YACC category and JNI category. Considering MUDABlue needs no human knowledge about software systems, if a new library would appear, MUDABlue will follow the new library without any human operation.

While MUDABlue got higher score over other research tools for precision and recall, comparing such values directly does not capture the essence of MUDABlue's advantage over them. The reason is that MUDABlue is completely different in some regards, as described below.

MUDABlue generates categories by itself, and does not use a pre-defined category set. Additionally, MUDABlue allows a software system to belong to multiple categories. Such differences are desirable characteristics for software categorization. Thus, the result indicates that MUDABlue has better precision and recall, and provides some additional suitable characteristics.

Another major difference between MUDABlue and other research tools is subject of IR method. MUDABlue applies LSA to source codes. In contrast, the existing tools (SVM and GURU) apply IR method to software documents. As mentioned in Section 2, depending on documents is not reasonable in the Open Source context because of the disparity of their amount and quality.

About titles of a cluster, some titles are easy to under-

stand, and others are not. Categories No.4 and 6 have titles easy to understand, like “AVI” and “board, ply”. Generally, categories based on depending libraries or architectures tend to have comprehensible titles. In contrast, categories which are based on software usage tend to have obtuse titles like category No.1. This is because identifiers typically have names related to the domain, not the domain name itself.

6.2 MUDABlue interface

MUDABlue provides several ways to browse a software repository. Such hybrid methods are implementable and useful. Frakes et al. [10] empirically investigated what several presentation methods’ effectiveness. They state that “There were no significant differences in search effectiveness.” and “Even though the methods were not significantly different in terms of recall and precision, they found different items.” Hence, we believe that more than one method should be presented to the user, rather than any given single browsing interface.

To draw overlapping categories, we used Cluster Map method, derived from the concept of Venn diagrams, with some customizations. InfoCrystal [23] is another research derived from Venn diagrams. In InfoCrystal, a Venn diagram is exploded into disjoint subsets. Next, the subsets were represented by icons whose shapes reflect the number of criteria satisfied by their contents. Dividing to the subsets, InfoCrystal adapts to the growth of categories. InfoCrystal, however, draws an n-polygon as outer frame of the figure, thus if there were dozens of categories, the figure would be hard to read. Also, InfoCrystal has no way of collapsing categories.

Another way to draw overlapping categories is arranging all elements according to their belongings. Sakai et al. [20] and Allan et al. [1] propose such methods. These research works arrange elements belonging to same category closely and vice versa. In these ways, elements visualization is achieved reflecting relations in overlapping categories. They, however, treat categories as just constraints and do not visualize what elements belong to the categories.

7 Related Work

In this section, we will review researches about application of IR method for software engineering world. Researches about automatic software categorization and about UCM are mentioned at section 6.

From the viewpoint of retrieving information from source code, some existing clustering methods cluster one software system into some functional parts for program understanding. Such software clustering methods use Latent Semantic Analysis [18], Self-Organizing Map [4], file structure and file names [2] or structure of program like call graph [5, 16]. They divide one software systems into some “component” (i.e. source file, function, and so forth).

Some research works apply IR method to retrieving reusable components from a software system. These method search components highly related with a given query. For example, CodeBroker [26] is one of the component retrieving systems for Java language. CodeBroker present methods highly related with a given query. To determine relationship between a method and a query, it applies LSA to a query and JavaDoc comment of methods.

Marcus et al. [19] proposed a method automatically recovering links between source codes and design documents. Their method applies LSA to comments of source codes and design documents. Then, they calculate similarity and recover the links.

Lucca et al. [15] apply SVM to bug tracking system. They use SVM to decide responsible person of new bug entry. They compare new bug entry and all submitted entries using SVM and suggest the person who is responsible for similar entries.

Brun [3] uses SVM and decision tree for software fault identification. At first, some test software systems are prepared. Half of them contain errors and the others are not. Next, some properties are retrieved from the test software systems and SVM and decision tree learner are trained with the properties. The learner is used to determine whether given software system contains error or not.

8 Conclusion

In this paper, we have proposed MUDABlue method, an automatic categorization method for a large collection of software systems. MUDABlue method does not only categorize software systems, but also determines categories from the software systems collection automatically. We have shown MUDABlue method can categorize without any knowledge about target software systems. Then, we implemented MUDABlue interface, a category-based software repository browsing system. MUDABlue enables browsing a repository categorized, where a software system can belong to multiple categories.

For future works, we will apply our method to various, more large-scale data sets. While we examine our method with 41 software systems, real software archives are larger. Increase of software systems would improve the result of LSA because LSA retrieves latent relation using statistical method. To experiment with a large-scale data set, improving scalability of MUDABlue will be essential. At the present time, most time-consuming process is cluster analysis of all identifiers. We need to use some approximate algorithm and reduce the computational effort.

Although MUDABlue automates categorization of a

software archive, it does not accept any human guidance. Combining our automated categorization and human guidance may result in a powerful classification.

References

[1] J. Allan, A. V. Leouski, and R. C. Swan. Interactive cluster visualization for information retrieval. Technical Report IR-116, Center for Intelligent Information Retrieval, University of Massachusetts, Amherst, 1997.

[2] 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.

[3] Y. Brun. Software fault identification via dynamic analysis and machine learning. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 16, 2003.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] C. Fluit, M. Sabou, and F. van Harmelen. Supporting user tasks through visualisation of light-weight ontologies. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003.

[10] 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.

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

[12] 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.

[13] 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.

[14] T. K. Landauer, P. W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[15] G. Lucca, M. D. Penta, and S. Gradara. An approach to classify software maintenance requests. In *International Conference on Software Maintenance (ICSM'02)*, Montreal, Quebec, Canada, Oct 2002.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] E. Sakai, K. Yamaguchi, and S. Kawai. Visualization model of hierarchical sets based on perception distance (p-distance) of graphical objects. In *Sixth International Conference on Computational Graphics and Visualization Techniques(COMPUGRAPHICS '97)*, pages 397–407, Dec 1997.

[21] 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.

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

[23] A. Spoerri. Infocrystal: a visual tool for information retrieval. In *Proceedings of the second international conference on Information and knowledge management*, pages 11–20, Washington, D.C., United States, Nov 1993.

[24] TouchGraph. `http://www.touchgraph.com/`.

[25] S. Ugurel, R. Krovetz, C. L. Giles, D. M. Pennock, E. J. Glover, and H. Zha. What's the code? automatic classification of source code archives. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 632–638, Edmonton, Alberta, Canada, Jul 2002.

[26] Y. Ye and G. Fischer. Supporting reuse by delivering task-relevant and personalized information. In *24th international conference on Software engineering(ICSE 2002)*, pages 513–523, Orlando, Florida, USA, May 2002.