Code Search based on CVS Comments: A Preliminary Evaluation

Annie Chen, Yun Ki Lee, Andrew Y. Yao, Amir Michail
School of Computer Science and Engineering
University of New South Wales
{anniec,s2251001,andrewy,amichail}@cse.unsw.edu.au

Abstract

We have built a tool, CVSSearch [1], that searches for fragments of source code by using CVS comments. (CVS is a version control system that is widely used in the open source community [3].) Our search tool takes advantage of the fact that a CVS comment typically describes the lines of code involved in the commit and this description will typically hold for many future versions. This paper provides a preliminary evaluation of this technique by 74 students at the University of New South Wales. Among our findings, CVS comments do provide a valuable source of information for code search that complements — but does not replace — tools that simply search the source code itself (e.g., grep).

1 Introduction

Search tools for source code are important in software maintenance activities [7]. However, if the code is poorly commented, then using a standard search tool, such as grep [5], is problematic. For example, it may be obvious from using the application that it has cut and paste functionality, though it may not be at all obvious how to use grep to find lines that implement this functionality. Since we can no longer depend on matching words in comments, we must use the search tool to match the code itself — which is difficult if we have never seen the code before.

If the code is well commented, one might expect standard search tools to work well. After all, comments are intended to not only state the purpose of the various pieces of code, but also to explain how that code works. In other words, they provide information at various levels of abstraction. Returning to our example, one would expect that doing a grep on "cut|paste" would match these words in comments and thus return those sections of code responsible for implementing cut and paste functionality.

However, naively searching through comments is problematic for various reasons. As Maarek et al. note, relating comments to the portion of code they concern is a very difficult task: "Although comments usually describe the containing routine or the one just below, in general it is impossible to automatically determine what part of the code is covered" [6, p. 802].

We have developed a general purpose search tool, CVSSearch [1], that also leverages natural language documentation — namely, CVS comments. CVS is an open source version control system that is widely used in the open source community [3]. Indeed, almost any large open source project makes use of CVS, particularly if multiple developers are involved.

CVS comments provide a particularly good source of documentation. While open source code may not always be well-commented, large open source applications almost always have very good CVS comments, particularly when many developers are involved.

Our CVS-based search tool takes advantage of the fact that: (1) a CVS comment typically describes the lines of code involved in the commit; and (2) that this description will typically hold for many future versions. In other words, CVSSearch allows one to better search the most recent version of the code by looking at previous versions to better understand the current version.

Our approach addresses several of the problems discussed with finding useful functional information for code: (1) by searching through natural language instead of (possibly uncommented) code, we avoid the problems of trying to extract useful functional information from free-form code with ad hoc naming conventions; and (2) by using CVS comments, we automatically have a very precise mapping of the commit comment and the lines of code that it refers to.

This paper provides a preliminary evaluation of this technique by 74 students at the University of New South Wales. As far as we know, our tool, CVSSearch, is the only one of its kind. Consequently, the survey in this paper is also the first one to evaluate CVS-based search tools in relation to conventional content-based ones such as grep. The survey not only yields insights into how well CVSSearch does in relation to grep, but also provides some guidance for building better CVS-based search tools in the future.

The remainder of the paper is organized as follows. Section 2 explains how we associate CVS comments with lines in the most recent version of the code. Section 3 describes our tool, CVSSearch. Section 4 presents the survey, its results, and an analysis thereof. Section 5 discusses related work. Section 6 summarizes the paper, concluding with future work.

2 Technique

To search for lines of code by their CVS comments, we produce a mapping between the comments and the lines of code to which they refer. Here we are only interested in the lines of code found in the newest version of each file. Observe that a line may be involved in multiple commits in which case it would have multiple CVS comments associated with it.

2.1 Algorithm Overview[1]

Consider a file f at version i which is then modified and committed into the CVS repository yielding version $i+1$. Moreover, suppose the user entered a comment C which is associated with the triple $(f, i, i+1)$.

By performing a diff[2] between versions i and $i+1$ of f, we can determine lines that are modified or inserted in version $i+1$; we associate comment C with all such lines. (Figure 1 shows such a diff, visually, between two successive versions of a file where $i = 48$; modified or inserted lines in version 49 are shaded.)

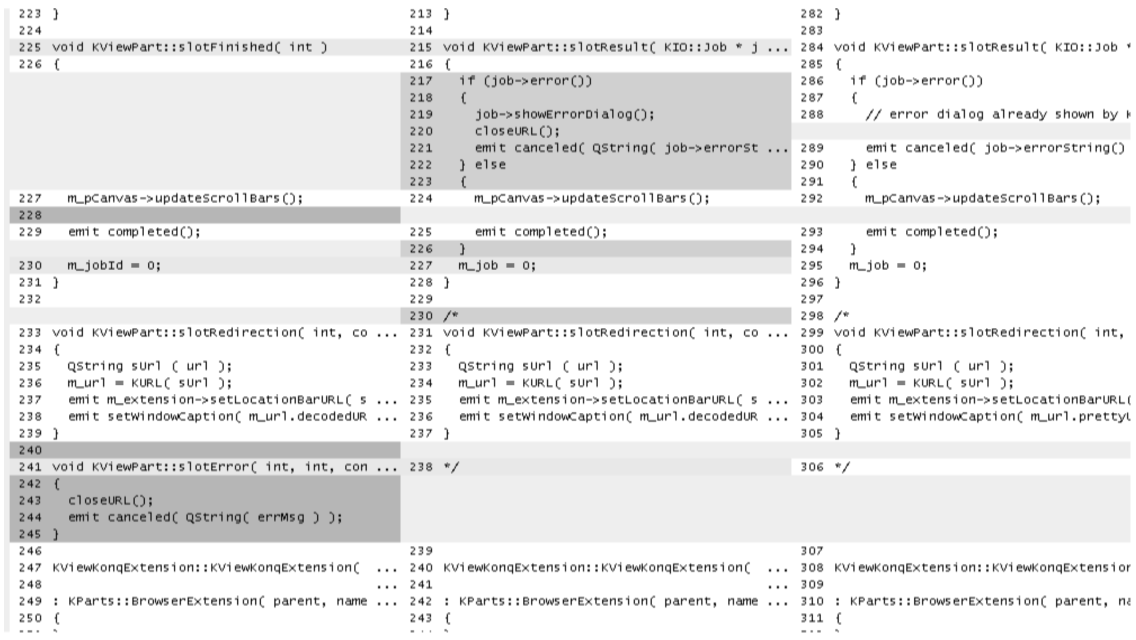
However, given we are interested in searching the most recent version of each file, we need a *propagation phase* during which the comments associated with version $i+1$ of f are "propagated" to the corresponding lines in the most recent version of f, say $j \geq i+1$. This is done by performing diffs on successive versions of f to track the movement of these lines across versions (even in the presence of changes to the lines themselves) until we reach version j. (Figure 1 shows the final outcome of this propagation phase in the third file which has version $j = 68$. Observe how the lines are matched up across versions 49 and 68 even though the line numbers have changed due to deletions/additions of preceding lines in the file over time.)

2.2 Database Storage and Querying

We have chosen the MG (Managing Gigabytes) [8] system for our database because it provides fast text retrieval on large text based database. For each line

[1]Details can be found in our previous paper [1].

[2]We actually use a somewhat more advanced version of GNU diff that detects "similar" lines [1].



```
}                                             }                                             }

void KViewPart::slotFinished( int )           void KViewPart::slotResult( KIO::Job * j ...  void KViewPart::slotResult( KIO::Job *
{                                             {                                             {
                                                if (job->error())                             if (job->error())
                                                {                                             {
                                                  job->showErrorDialog();                       // error dialog already shown by K
                                                  closeURL();
                                                  emit canceled( QString( job->errorSt ...      emit canceled( job->errorString()
                                                } else                                        } else
                                                {                                             {
  m_pCanvas->updateScrollBars();                  m_pCanvas->updateScrollBars();                m_pCanvas->updateScrollBars();

  emit completed();                               emit completed();                             emit completed();
                                                }                                             }
  m_jobId = 0;                                  m_job = 0;                                    m_job = 0;
}                                             }                                             }

                                              /*                                            /*
void KViewPart::slotRedirection( int, co ...  void KViewPart::slotRedirection( int, co ...  void KViewPart::slotRedirection( int,
{                                             {                                             {
  QString sUrl ( url );                         QString sUrl ( url );                         QString sUrl ( url );
  m_url = KURL( sUrl );                         m_url = KURL( sUrl );                         m_url = KURL( sUrl );
  emit m_extension->setLocationBarURL( s ...    emit m_extension->setLocationBarURL( s ...    emit m_extension->setLocationBarURL(
  emit setWindowCaption( m_url.decodedUR ...    emit setWindowCaption( m_url.decodedUR ...    emit setWindowCaption( m_url.prettyU
}                                             }                                             }

void KViewPart::slotError( int, int, con ...  */                                            */
{
  closeURL();
  emit canceled( QString( errMsg ) );
}

KViewKonqExtension::KViewKonqExtension(  ...  KViewKonqExtension::KViewKonqExtension(  ...  KViewKonqExtension::KViewKonqExtensio
                                         ...                                           ...
: KParts::BrowserExtension( parent, name ...  : KParts::BrowserExtension( parent, name ...  : KParts::BrowserExtension( parent, na
{                                             {                                             {
```

Figure 1: Three versions of kview_view.cc are shown: (1) v. 48 before commit on left; (2) v. 49 after commit in middle; and (3) v. 68 (most recent) on right.

of a file we store into MG its associated CVS comments all merged together. When given query words, MG returns all lines whose CVS comments contain at least one of those words. Moreover, the results are ranked, using cosine similarity [8, p. 185], so that lines returned first tend to contain the most query words — and with multiple occurrences of them.

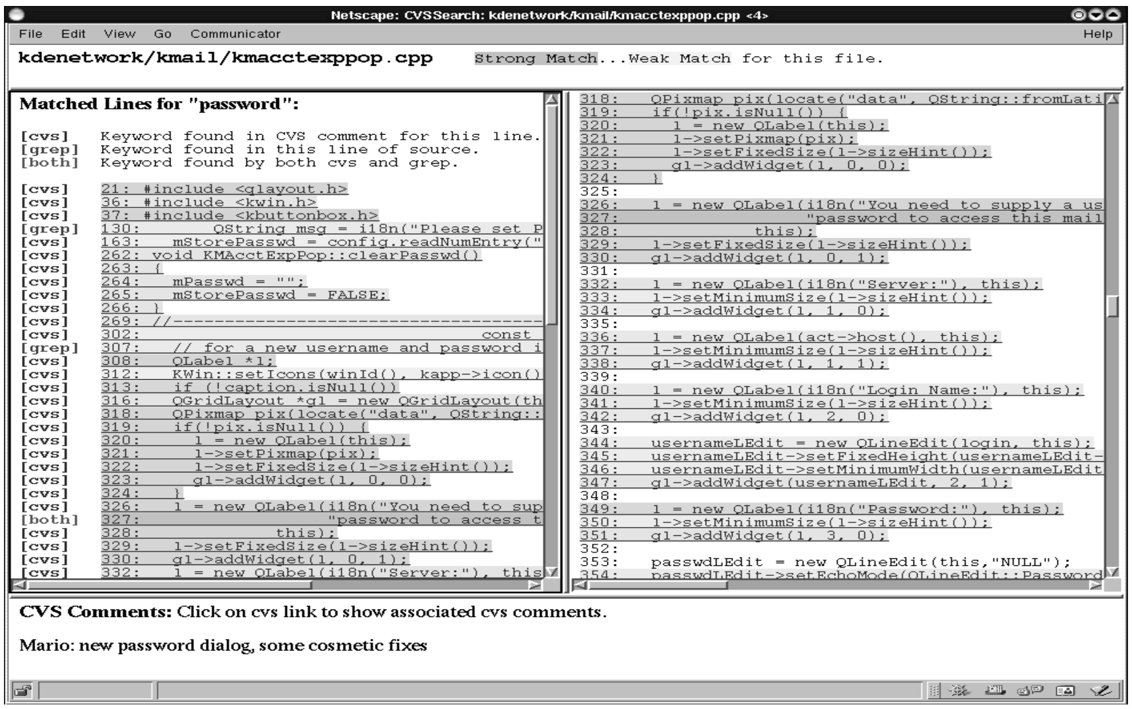
3 The Tool

We have built a web-based demonstration tool, CVSSearch, that combines matching based on CVS comments with conventional content-based matching using grep.[3] Users can enter keywords to search for and select an application to search in. When a user queries the database, we combine the results given by MG with matches returned by grep. For example, if the user searches for the words "cut paste", then we also perform a grep query "cut|paste" to look for lines of code that contain one of those words.

Our tool actually returns a ranked list of files first, where the score of a file is computed as $\sum_{i>j} \frac{S_i * S_j}{(i-j)^2}$, with the summation over all lines that matched the query from that file and where S_k indicates the score of line k — either given by MG for CVS-based matches or equal to half the maximum score returned by MG for grep matches. This formula rewards those files that tend to have many matches close together where the matches themselves have high scores S_k. A ranked list of files is then displayed, showing the number of lines with CVS comment matches and the number of lines with grep matches for each file.

A user can then click on a file to examine its line matches. (See Figure 2 where the user has clicked on a file returned by the query "password" in an email client.) Matches are shown on the left with a tag to display the type of match, i.e. CVS, grep or both. Moreover the matched lines are highlighted according to how strong the match is. Darker highlight

[3]The demonstration tool is available at http://www.cse.unsw.edu.au/~amichail/cvssearch.


Netscape: CVSSearch: kdenetwork/kmail/kmacctexppop.cpp <4>
File Edit View Go Communicator Help

kdenetwork/kmail/kmacctexppop.cpp   Strong Match...Weak Match for this file.

Matched Lines for "password":

```
[cvs]    Keyword found in CVS comment for this line.
[grep]   Keyword found in this line of source.
[both]   Keyword found by both cvs and grep.

[cvs]    21: #include <qlayout.h>
[cvs]    36: #include <kwin.h>
[cvs]    37: #include <kbuttonbox.h>
[grep]   130:        QString msg = i18n("Please set P
[cvs]    163:   mStorePasswd = config.readNumEntry("
[cvs]    262: void KMAcctExpPop::clearPasswd()
[cvs]    263: {
[cvs]    264:   mPasswd = "";
[cvs]    265:   mStorePasswd = FALSE;
[cvs]    266: }
[cvs]    269: //-----------------------------------
[cvs]    302:                                   const
[grep]   307:   // for a new username and password i
[cvs]    308:   QLabel *l;
[cvs]    312:   KWin::setIcons(winId(), kapp->icon()
[cvs]    313:   if (!caption.isNull())
[cvs]    316:   QGridLayout *gl = new QGridLayout(th
[cvs]    318:   QPixmap pix(locate("data", QString::
[cvs]    319:   if(!pix.isNull()) {
[cvs]    320:     l = new QLabel(this);
[cvs]    321:     l->setPixmap(pix);
[cvs]    322:     l->setFixedSize(l->sizeHint());
[cvs]    323:    gl->addWidget(l, 0, 0);
[cvs]    324:   }
[cvs]    326:   l = new QLabel(i18n("You need to sup
[both]   327:                        "password to access t
[cvs]    328:                 this);
[cvs]    329:   l->setFixedSize(l->sizeHint());
[cvs]    330:   gl->addWidget(l, 0, 1);
[cvs]    332:   l = new QLabel(i18n("Server:"), this
```

```
318:   QPixmap pix(locate("data", QString::fromLati
319:   if(!pix.isNull()) {
320:     l = new QLabel(this);
321:     l->setPixmap(pix);
322:     l->setFixedSize(l->sizeHint());
323:    gl->addWidget(l, 0, 0);
324:   }
325:
326:   l = new QLabel(i18n("You need to supply a us
327:                       "password to access this mail
328:                 this);
329:   l->setFixedSize(l->sizeHint());
330:   gl->addWidget(l, 0, 1);
331:
332:   l = new QLabel(i18n("Server:"), this);
333:   l->setMinimumSize(l->sizeHint());
334:   gl->addWidget(l, 1, 0);
335:
336:   l = new QLabel(act->host(), this);
337:   l->setMinimumSize(l->sizeHint());
338:   gl->addWidget(l, 1, 1);
339:
340:   l = new QLabel(i18n("Login Name:"), this);
341:   l->setMinimumSize(l->sizeHint());
342:   gl->addWidget(l, 2, 0);
343:
344:   usernameLEdit = new QLineEdit(login, this);
345:   usernameLEdit->setFixedHeight(usernameLEdit-
346:   usernameLEdit->setMinimumWidth(usernameLEdit
347:   gl->addWidget(usernameLEdit, 2, 1);
348:
349:   l = new QLabel(i18n("Password:"), this);
350:   l->setMinimumSize(l->sizeHint());
351:   gl->addWidget(l, 3, 0);
352:
353:   passwdLEdit = new QLineEdit(this,"NULL");
354:   passwdLEdit->setEchoMode(QLineEdit::Password
```

CVS Comments: Click on cvs link to show associated cvs comments.

Mario: new password dialog, some cosmetic fixes

Figure 2: Search results for "password" query on an email client application.

denotes stronger matches and weaker highlight denotes weaker matches. (Observe that many of the CVS matches are relevant to the "password" query although that word doesn't appear in those lines.)

On the right, the tool shows the source code for that file. A user can click on any of the matched lines on the left to bring the source code on the right to that particular line so the user can examine that line in its context. The tool also displays the associated CVS comments for the selected line in the bottom frame, so user can see why that line was matched. (In Figure 2, the user has clicked on line 318, a CVS match, with the associated CVS comment shown in the bottom frame.)

4 Survey

In this section, we describe our CVSSearch survey and analyze its results. The survey was advertised to students taking computer science courses at the University of New South Wales. A small monetary incentive was used to encourage students to participate. In total, 74 took part in the survey. Of the these participants, 4% were from first year, 22% were from second year, 49% were from third year, and 18% were from fourth year, leaving about 8% whose year was unknown or have graduated. Their majors included: Computer Science (31%), Software Engineering (30%), and Computer Engineering (23%).

4.1 Survey Part 1

The survey was given in two parts. The first part, shown in Table 1, does not involve the CVSSearch tool at all. Rather, it is an attempt to capture typical search habits. Such information can help us understand the mind set of developers as well as to guide the design of search tools in general.

Question #1 asks whether they typically use regular expressions or exact matches when searching

	Survey Part 1		
1.	I normally search through source code		
	(a) using regular expressions (e.g., “drag.*event”)	36	(49%)
	(b) using exact matches (e.g., “drag_and_drop_event”)	38	(51%)
2.	When searching through code that I am familar with, I normally		
	(a) search for identifier names (e.g., classes/functions, etc.)	64	(86%)
	(b) search for words in code comments	10	(14%)
3.	When searching through code that I am *not* familiar with, I normally		
	(a) search for identifier names (e.g., classes/functions, etc.)	35	(48%)
	(b) search for words in code comments	38	(52%)
4.	When looking at code that I am *not* familiar with, I typically		
	(a) look at the code comments to try to understand the code	7	(9%)
	(b) look at the code (and not the comments) itself to try to understand what it does	4	(5%)
	(c) look at both the code and its comments to try to understand it	63	(85%)
5.	I usually search through code		
	(a) that I am familiar with (e.g., have seen before, wrote, etc.)	56	(76%)
	(b) that I am *not* faimilar with (e.g., have not seen before)	18	(24%)
6.	How often do you need to search code that you are *not* familiar with?		
	(a) rarely	24	(32%)
	(b) sometimes	31	(42%)
	(c) frequently	19	(26%)
7.	I normally		
	(a) have difficulty searching for items in code that I have seen before	7	(10%)
	(b) have little difficulty searching for items in code that I have seen before	66	(90%)
8.	I normally		
	(a) have difficulty searching for items in code that I have *not* seen before	65	(89%)
	(b) have little difficulty searching for items in code that I have *not* seen before	8	(11%)
9.	The items I search for tend to be		
	(a) physically close together	28	(38%)
	(b) spread across several places	46	(62%)
10.	When I am searching code I am familiar with, I can		
	(a) usually tell if my search yields lines that I wanted to find	66	(89%)
	(b) have difficulty knowing if my search yields lines that I wanted to find	8	(11%)
11.	When I am searching code that I am *not* familiar with, I can		
	(a) usually tell if my search yields lines that I wanted to find	14	(19%)
	(b) have difficulty knowing if my search yields lines that I wanted to find	60	(81%)
12.	If I had a tool that allowed me to more easily search code that I am not familiar with		
	(a) I would be more likely to use other people’s code	56	(76%)
	(b) I would not be more likely to use other people’s code	18	(24%)
13.	If I had a tool that allowed me to more easily search code that I am not familiar with,		
	(a) I would be more likely to understand other people’s code	57	(77%)
	(b) I would not be more likely to understand other people’s code	17	(23%)

Table 1: Survey part 1.

through source code. Surprisingly, almost exactly half use each type of search. We observe that regular expressions are usually used to match identifiers in the code rather than words in the comment. Thus, this result gives an indication that many developers search for identifiers directly rather than going through comments. Indeed, this is supported by Question #2, where 86% of students say that whenever they search through code they are familiar with, they typically search for identifier names (e.g., classes/functions) directly. However, this percentage goes down to 48% with unfamiliar code as shown in Question #3; in this case, it appears that developers search more for comments than they usually would in their own code. Indeed, Question #4 shows that 85% of developers typically look at both the code and comments in unfamiliar code to try to understand what it does. CVSSearch facilitates this task since it returns code-based matches (using grep) as well as CVS comment matches.

Nonetheless, question #5 shows that only 24% of the students usually search unfamiliar code — where we would expect CVSSearch to excel — which is not surprising given that they are undergraduates. We would expect this number to be higher in industry and the open source community. Question #6 gives a more precise breakdown: 32% rarely search through unfamiliar code; 42% sometimes do such searches; while 26% frequently search through foreign code. This means that 68% of students would be interested in a tool that facilities search in unfamiliar code. Again, this number is probably higher in industry and the open source community.

Questions #7 and #8 confirm some common sense intuition: 90% of students have little difficulty searching through familiar code while 89% have difficulty with unfamiliar code. Thus tool support to facilitate this latter task is essential. Moreover, given that developers would be unfamiliar with identifiers in foreign code, it is important to enhance searching for comment words as much as possible — as we do with CVSSearch using CVS comments.

We also wanted to see whether developers typically search for code that is physically close together or spread out across several places. Question #9 shows that 62% search for code that is spread out; developers are more interested in finding aspects than they are in finding functions or classes. CVSSearch searches on a line-by-line basis, so it can return aspects than span parts of various functions/classes.

The following questions are more subtle. They concern the ability of developers to determine whether the lines returned by a search tool really do correspond to the code that they were after. From Question #10, we see that 89% of students say that it is easy to recognize relevant lines in familiar code. However, this number goes down to 19% in unfamiliar code as shown in Question #11. CVSSearch actually uses the CVS comments themselves to explain the lines matched by a search so users have a better idea whether the matched lines are relevant or not (and indeed, why they matched in the first place).

Questions #11 and #12 confirm some more common sense intuition: students believe that a tool that allows them to better search unfamiliar code would help them reuse (or salvage) existing code — with 76% saying so — as well as to understand it better — with 77% saying so. Indeed, much research as been done to facilitate software reuse by building component retrieval systems based on natural language documentation. CVSSearch is a general search tool — not restricted to component libraries — that uses the natural language documentation in CVS comments.

4.2 Survey Part 2

After filling out part 1 of the survey, students were then asked to complete part 2, which involves actually using the CVSSearch tool to perform queries. The first question involved evaluating queries of their choosing on certain applications. (See Table 2.) Most other questions were multiple choice. (See Table 3.) Students were also asked to list positive and negative aspects of CVSSearch as well as make recommendations for future improvement. Some of their written comments will be interspersed with our

KDE Application	Lines of Code	Average # Rev./File	DB Build (min.)	DB Space (KB)	Average # CVS Comments/Line
konqueror	24,253	38.3	23	2,911	1.5
korganizer	43,188	10.8	9	5,028	1.2
kmail	40,325	32.9	24	3,672	1.4
knode	33,721	14.7	8	2,237	1.3
kword	48,883	22.5	47	4,475	1.6

Figure 3: CVSSearch statistics for KDE applications. (Timing was done on a 700 Mhz Pentium III with 256 MB.)

analysis of the query evaluations and multiple choice questions below.

Evaluating Search Effectiveness Question #1 of part 2 asked them to perform at least 2 queries for each of the following KDE applications: konqueror (a web browser), kmail (an email client), knode (a newsreader), korganizer (a scheduling program), and kword (a word processor). (See Figure 3 for applications statistics.) Moreover, they were instructed to use reasonable queries given the application type. For each query submitted, students were to indicate whether CVSSearch alone or grep returned better results — or whether they were both about the same. This should be clear since CVSSearch clearly informs the user whether line matches were obtained from CVS, grep, or both. (See Figure 2.)

The students were not familiar with the source code of these KDE applications so our survey is testing how well CVSSearch compares with grep on unfamiliar code. In future surveys, we shall consider both familiar and unfamiliar code.

Table 2 shows the results of the query evaluations, broken by application type, as well by total. Of the 703 queries submitted across all applications by students, 40% were better handled through CVS comments, 32% were better handled by grep, and 28% were handled equally well by both approaches.

To see whether this result is statistically significant in showing whether CVSSearch is the better tool, we use the "sign test". Specifically, consider the null hypothesis H_0 that the probability of the CVSSearch tool performing better on a query is equal to the probability of grep being better — that is, 1/2 for each. We consider the alternative hypothesis H_a, that the probability of CVSSearch being better is greater than 1/2. Suppose that q out of n evaluations show CVSSearch to be better, then the probability that H_0 is true given the observed data is equal to $\sum_{i=q}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) (1/2)^n$; this is the so-called *p-value*. The lower this probability, the more likely that H_0 is incorrect, thus allowing us to more confidently accept the alternative hypothesis H_a. It is common practice to look for p-values of 0.05 or less to indicate strong rejection of H_0. One final technical point to mention is our handling of ties (i.e., "same" in Table 2). Rather than simply omitting tied cases, we allocate half of the ties to one tool and half to the other tool. Doing so yields more conservative p-values.

In Table 2 we show p-values for both "CVSSearch being better" and "grep being better". We see that the results from the 703 evaluations indicate that CVSSearch is actually more likely than grep to give better answers overall, with a p-value of 0.0143. This is strong indication that overall, CVSSearch performs better searches than grep using CVS comments alone (e.g., without looking at the code itself using grep). Moreover, if we look at the breakdown by application type, we see that there is strong indication that CVSSearch outperforms grep for kmail and kword with p-values of 0.0249 and 0.000497, re-

	CVSSearch	grep	same	Total	CVSSearch better	grep better
					p-value	p-value
konqueror	57 (40%)	41 (29%)	44 (31%)	142 (100%)	0.104	0.923
kmail	63 (46%)	38 (28%)	37 (27%)	138 (100%)	0.0249	0.984
knode	53 (38%)	48 (34%)	39 (28%)	140 (100%)	0.400	0.664
korganizer	41 (29%)	68 (48%)	32 (23%)	141 (100%)	0.991	0.0141
kword	70 (49%)	29 (20%)	43 (30%)	142 (100%)	0.000497	0.9997
Total	284 (40%)	224 (32%)	195 (28%)	703 (100%)	0.0143	0.988

Table 2: Survey part 2. Question #1.

	Survey Part 2		
2.	Generally speaking, which tool results in more false positives?		
	(a) CVSSearch (without grep matches)	40	(55%)
	(b) Grep Search	17	(23%)
	(c) About the same	16	(22%)
3.	Generally speaking, which tool results in more false negatives?		
	(a) CVSSearch (without grep matches)	25	(34%)
	(b) Grep Search	17	(23%)
	(c) About the same	32	(43%)
4.	Generally speaking, do CVSSearch and Grep results complement each other?		
	(a) yes	56	(76%)
	(b) no	18	(24%)
5.	Generally speaking, did you find the CVS comments helpful in understanding code?		
	(a) yes	56	(76%)
	(b) no	18	(24%)
6.	Generally speaking, were you able to easily determine if grep matches were relevant?		
	(a) yes	50	(68%)
	(b) no	24	(32%)
7.	Generally speaking, where you able to easily determine if CVS matches were relevant?		
	(a) yes	6	(8%)
	(b) no	9	(12%)
	(c) yes but only after looking at the associated CVS comments	59	(80%)
8.	Are you familiar with CVS?		
	(a) yes	19	(26%)
	(b) no	55	(74%)
9.	Do you feel understanding the purpose of CVS is necessary to use CVSSearch well?		
	(a) yes	51	(69%)
	(b) no	23	(31%)
10.	Do you have a general understanding of how CVSSearch works?		
	(a) yes	58	78%
	(b) no	16	22%
11.	Did you read the CVSSearch paper?		
	(a) yes	36	(49%)
	(b) no	38	(51%)

Table 3: Survey part 2. Questions #2 – 11.

spectively. Moreover, there is some weaker evidence that CVSSearch is also better with konqueror and knode, with p-values of 0.104 and 0.400, respectively. However, CVSSearch clearly loses to grep on korganizer, where grep has a p-value of 0.0141. So, while all queries combined indicate that CVSSearch is better than grep overall, the results are less clear for particular applications. We suspect that a combination of factors lead to this variation: the quality of CVS comments used, the quality of identifier names, as well as the student's familiarity with the application type.

With respect to the advantages of searching CVS comments, one person said "The CVS results do increase the likelihood of finding a relevant result and in some cases can return much better results (than grep)." Another said "…when names used aren't the best or are abbreviated, then the user is dependent on comments. For these cases CVS(Search) was better than grep."

Another person addresses related words: "It searches a variety of related and relevant words, for example, when 'date' is searched, results including 'time' are also returned." This property follows naturally from our approach since many lines with words such as "time" also have associated CVS comments with "date". Another person writes "It (CVSSearch) helps locate parts of the code better than grep, especially on conceptual ideas, where they (are) most likely mentioned in the CVS comments."

Another writes "CVS comments (are) often an improvement over much of the open source code commenting I've seen…allows for longitudinal (time wise) searching, e.g., bug-fixes, recent areas of development." Other comments along this direction: "good for getting a perspective on what works and what's under development, and locating these points quickly" and "CVSSearch provides more comments behind why the code was written".

We now consider the remaining questions in part 2 of the survey, which are shown in Table 3. From Question #2, we see that 55% of students believe CVSSearch yields more false positives (e.g., matches not relevant to the query) while only 23% say grep yields more. False positives arise mainly from two sources: (1) large commits where multiple changes were performed so we do not have a precise mapping of which part of CVS the comment refers to which change in the code; and (2) inaccuracies in the way we propagate lines from the past to the present code. We discuss ways to alleviate these problems in Section 6.

From Question #3, we see that 34% believe CVSSearch has more false negative (e.g., matches that should have been shown but were not) while 23% say grep has more false negatives. False negatives arise mainly from two sources: (1) inadequate CVS comments; and (2) inaccuracies in the way we propagate lines from the past to the present code. Again, we address these problems in Section 6.

Concerning false positive and false negatives, one person said "Too many false positives, and also some clear matches in the code are not returned." Others felt that stemming was partly at fault for false positives: "…searching in the CVS comments is a good but stemming generalizes too much and we tend to get a lot of unwanted results".

Nonetheless, question #4 shows that 76% of students say that CVSSearch and grep complement each other. That is, one tool comes up with relevant matches that the other has missed and vice versa. The p-value for this statement is very small at 0.55×10^{-5}, and so we can be confident of its validity. While the fact that CVSSearch and grep complement each other shows that CVS comments are a valuable source of information for search, it also means that we should not only rely on them alone but must include the code itself in the search. One person writes "Keep the search engine along with the grep feature. They help each other, I think. If CVS fails, it really helps to have an alternative." Another said "Thorough, when used with grep at least."

Using CVS Comments to Understand Code Another important result comes from Question #5: 76%

of students said that CVS comments are helpful in understanding code. (Again with p-value 0.55×10^{-5}.) This observation has two implications: (1) the CVS comments can themselves be used to explain matches (as we do now using CVSSearch); and (2) CVS comments can be used to explain code whether or not a search was involved in retrieving that code. Regarding the former point, one person writes "CVS comments associated with the matched files were very helpful in not only understanding and determining relevance of code but also showing the role of the code in the entire program." Regarding the latter point, one person writes "CVS comments are extremely helpful in deciphering obscure bits of code."

Identifying Relevant Matches Questions #6 and #7 explore the issues of whether matches returned by CVSSearch and grep could easily be inspected for relevance to the query, respectively. For grep, 68% of students said that determining relevance for matches returned was easy. For CVSSearch, only 8% said that determining relevance for matches returned was easy but 80% said that it was so if they also looked at the associated CVS comments for those line matches. This supports our belief that showing CVS comments for matched lines is critical in this type of tool.

Leveraging CVS Knowledge Questions #8 – 11 explore whether understanding the way CVSSearch works is important in using it effectively. Question #8 asks whether they are familiar with CVS. Only 26% said yes. (However, all participants were given a short explanation of what CVS is used for beforehand.) Question #9 asked them whether understanding the purpose of CVS is necessary to use CVSSearch effectively — of which 69% said yes. Indeed, one person wrote "...I feel that only people familiar with CVS can use it properly."

This result is somewhat surprising. When we first built CVSSearch, we saw it as just another search tool with a completely different implementation. Yet, students believe that understanding the implementation is really important for effective usage — since presumably the results of the search would be more predictable. Consequently, we need to re-examine our decision to "hide" CVS features (e.g., such as revision numbers, code branches, authors of changes, commit dates, etc.) from our tool.

A deeper analysis of the data shows evidence that people that are familiar with CVS tend to like grep better, but the p-value is not that low at 0.255. In contrast, those people not familiar with CVS tend to like CVSSearch better with a p-value of 0.00183. Again, this result is somewhat surprising and perhaps shows that CVS experts have certain preconceptions about CVS comments that got in the way.

Question #10 shows that 78% of the students have a general idea of how CVSSearch works while 49% have read our previous CVSSearch paper [1]. So, it appears they had some understanding of how the search was done.

5 Related Work

There is a myriad of search tools for code, some of which search through code directly [2, 5, 7] and others that search the natural-language documentation (such as comments or manual pages) associated with the code [4, 6].

In the first category, we find lexical tools such as grep [5]. Such tools are based on regular expressions, and while simple to use, have problems searching for certain constructs. For example, grep is not designed to search for patterns spanning multiple lines or two statements at the same level of nesting. Consequently, search tools have been developed that parse the source code [2, 7] to alleviate such problems.

As discussed in Section 1, tools of this type, whether lexical or syntactic, are quite difficult to use if the user is not familiar with the code at all. This is particularly an issue when the code is poorly commented since the query words would no longer likely match words in the natural-language code comments.

In the second category, we find tools that are based on natural-language documentation associated with the code [4, 6]. It is worthwhile noting that such tools are designed for retrieval of reusable components whereas CVSSearch is a general purpose search tool. Moreover, as far as we know, CVSSearch is the only search tool based on CVS comments —- and consequently, our evaluation of this type of tool is also a first.

6 Conclusions and Future Work

From our survey analysis in Section 4, we see that CVS comments do provide a valuable source of information that complements — but does not replace — content-based matching (e.g., using grep). Moreover, CVS comments are also good at explaining the lines matched, and indeed, can be used as an additional source of documentation for code irrespective of search. We were surprised that students felt that knowledge of CVS was important in using CVSSearch effectively. In retrospect, it is understandable that users would want to have some idea how their search tools work.

It is clear that false positives are a problem with our current approach. We plan to pursue both short and long term strategies to reduce false positives. In the short term, we plan to get the user more involved in the search process so that false positives are easily detected and ignored. For example, we will make the mapping between CVS comments and matched lines in a file more explicit, so the user can easily ignore matches with irrelevant comments.

In the long term, we plan to improve the CVS-based search itself to reduce false positives. We believe this can be done by either increasing the granularity of the matched items from lines to member functions, increasing the granularity of the propagation phase from lines to member functions (which have names so their movements are more easily tracked across versions and files), or both.

References

[1] Annie Chen, Eric Chou, Joshua Wong, Andrew Y. Yao, Qing Zhang, Shao Zhang, and Amir Michail. CVSSearch: Searching through source code using CVS comments. Submitted for publication. Available from http://www.cse.unsw.edu.au/˜amichail/cvssearch.

[2] P. Devanbu. GENOA — a customizable, language and front-end independent source code analyzer generator. In *Proceedings of the 14th International Conference on Software Engineering*, pages 307–317, 1992.

[3] K. F. Fogel. *Open Source Development with CVS*. Coriolis Inc., 2000.

[4] M. R. Girardi and B. Ibrahim. Using English to retrieve sofware. *The Journal of System and Software*, 30(3):249–270, 1995.

[5] B. W. Kernighan and R. Pike. *The Unix Programming Environment*. Prentice Hall, 1984.

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

[7] S. Paul and A. Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994.

[8] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.