Versioned Lexical Search

Senior Thesis

Michael Haufe

mlhaufe@uwm.edu

October 2014

Advisor: Dr. Ethan Munson

munson@uwm.edu

Submitted in partial fulfillment of the requirements

for the degree of Bachelor of Science, Computer Science

to the

Faculty of the College

of Engineering and Applied Science

University of Wisconsin-Milwaukee

Milwaukee, Wisconsin

Abstract

Two important tools utilized by a programmer include a Search tool capable of locating relevant information across a codebase, and a Version Control System (VCS) to manage the changes to that codebase over time. The Search tool is of even greater utility if it is aware of the syntax of the underlying codebase, referred to as a Lexical Search. While VCSs and Search tools are prevalent (to a lesser extent Lexical Search), the use and existence of a unified tool remains exceptional.

The goal of this thesis is to create a Versioned Lexical Search system in order to provide a user the capability to perform a syntactically aware search over a codebase and across time through an IDE and VCS. This will enable the ability to not only find a relevant syntactic form in the codebase, but to also see how that syntactic form changed over time as the application evolved and display it in a user-friendly manner. For example: one could search for a class named: "Foo", and issue a query to locate the earliest instance of that class in the repository and track its evolution through branches and merges.

The implementation described in this thesis replaces an earlier, standalone prototype implemented by Alex Garcia <adgarcia@uwm.edu> and Michael Haufe <mlhaufe@uwm.edu> and described in the Capstone Project "*Versioned Lexical Search Interface*"[1]

Content

1. List of Figures

2. Acknowledgments

- Dr. Ethan Munson, who years ago took a chance on a non-traditional student (and freshman) who wandered into his office one day looking to pick up any research project available. He has shown me, and continues to show that one's reach is not beyond their grasp if they are willing to exercise the work and dedication needed to accomplish a goal. I have come to also appreciate his unique approach to the field. Through actions and words, he has taught me that one shouldn't overlook nor take for granted the social interactions which are an integral part of the academic field.
- Dr. John Boyland, who has taught a significant number of my courses at the university. His courses and teaching methods have been inspirational for myself and many others. The challenge and depth of the material presented, while without a doubt enlightening, were made much more effective and valuable to me due to the passion he brought along with it. Even after a long work day, or after burning the midnight oil on some other project, and even in times of burn-out, I could always rely on being re-energized by the infectious energy he brought to the classroom.

3. Introduction

During the Software Development Process, two important tools utilized by Software Engineers include a Version Control System (VCS) such as Subversion and an IDE such as Eclipse in which development and code viewing are performed. In addition to their base functionality, these particular tools also often provide a means of performing search over the program source files they operate on. These search capabilities range from literal textual match systems, as can be seen in the majority of text editors, to a full grep-like search system or even a lexical search system which is aware of the syntax of the underlying text as can be seen in IDEs such as Eclipse for the Java language. This latter search system though is a rare occurrence in IDEs and VCSs. This is partly due to to the effort involved in creating such a system properly for a given set of languages, and also partly due to the desire of the implementers of these systems to provide general purpose tools to work with any textual file instead of a more limited set of programming language specific ones.

The benefits of being able to search across a codebase have long been recognized. Beyond the common case of manually entering a query to obtain results, search is also leveraged implicitly in more advanced cases such as refactoring subsets of code and locating references to a particular syntactic form. This latter case requires a search system beyond the simple textual match system. Instead, complex regular expressions or lexically aware search systems are utilized. A lexical search system is especially powerful and accurate over its or regex based alternatives as it can provide more accurate search results and allow one to issue more complex queries due to being aware of the underlying syntax of the language. The drawback to having a lexical search though is that it requires parsing of the source files before it can be used and is also slower than its simpler textual or regular expression based counterparts.

Despite the significant progress which has been made in both VCS and IDE features, one which has not been forthcoming is one which enables a programmer the ability to perform a lexical search across the history of a source code repository from an IDE. This capability would enable one to not only locate a desired syntactic construct, but would also enable them to follow trends in over time as the codebase evolved. Some example trends that could be followed include, but are not limited to: code size, cyclomatic complexity, and design patterns.

Having available such high level information can enable intriguing questions to be asked about a codebase which can then be evaluated. An example of a question that might be asked is: "Are all instances of switch statements in our project eventually refactored into a different form?". By following the evolutionary trend of these syntactic constructs, an answer could be found. If the answer is yes, then a decision could be made by a Software Engineering team, to disallow the use of switch statements in future development and thereby increasing productivity by preventing the refactoring effort that has had a tendency to plague the project.

What follows is a description of an application which enables the ability to execute lexical queries from an IDE, but unlike the lexical query functionality which may already be found in the IDE, this query also operates over the VCS associated with the given codebase, thus providing historical results for a given query. This search system is also loosely coupled from a given IDE or VCS and also from a given client's machine, allowing each to evolve separately without impacting one another.

4. Approach

The Versioned Lexical Search system will be realized by leveraging and integrating a number of existing and popular tools in order to minimize development time as well as reduce potential migration taxes towards adoption. More concretely, the company Semantics Designs[2] has developed a Lexical Search Tool which supports a number of programming languages. This tool will be integrated into Subversion[3] (VCS) to enable the desired Versioned Lexical Search capability. Subversion will also be extended to automatically generate and/or update the necessary Lexical Search database after every commit from a client. A RESTful[5] url will also be exposed on the same server to enable remote execution of desired queries and to update the Lexical Search database as needed from multiple clients. Additionally, the Eclipse[4] IDE will be extended in order to provide a user friendly interface to execute queries and display results inline.

5. Implementation

5.1 Overview

The implementation of this Versioned Lexical Search system depends on the synthesis of four major components/subsystems:

1. A system of generating and maintaining a Lexical Search database
2. A public API for executing queries, initiating a database build, and returning structured results to a client
3. Extension of a VCS to initiate updates and rebuilds of the Lexical Search database following the creation of a new revision in the VCS
4. An integrated plugin into an IDE (Eclipse) in order to enable user friendly interaction with the server's exposed public APIs

A high level visualization of this implementation can be seen below:

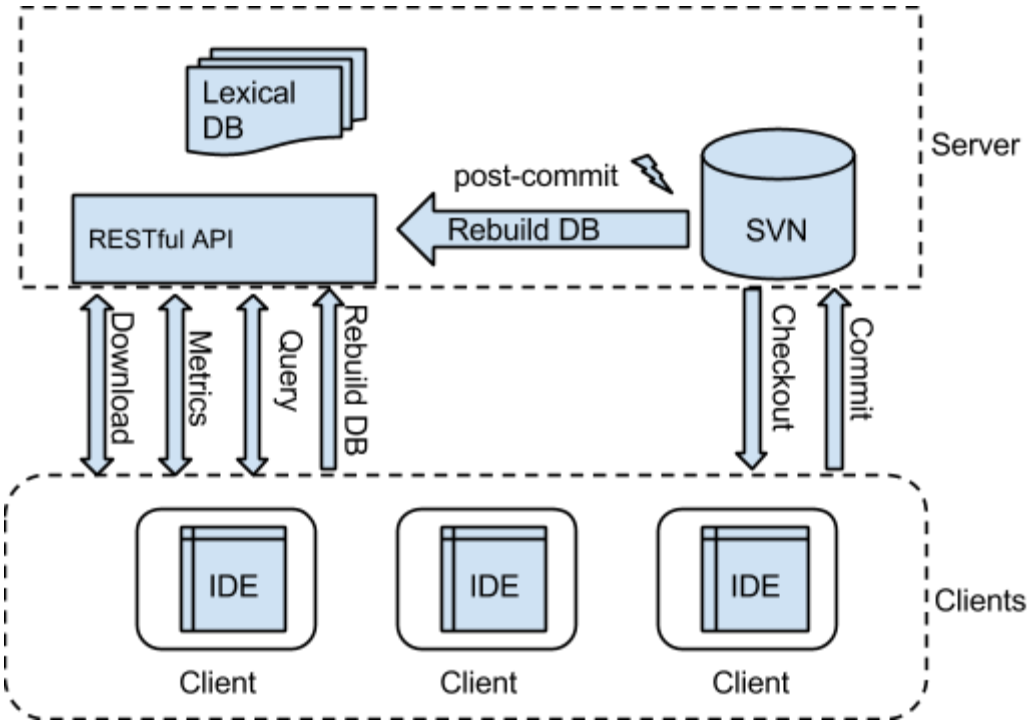


Figure 1 Client Server Interaction

5.2 Generating Lexemes and Search Database

Before a Lexical Query can be executed across the SVN repository, a search database must first be generated from the existing source files for the language. The repository for a given project can be significant in size though with revision counts in the millions. (For example, the Apache Batik project has ~1.6 million revisions). Single revisions can also be significant in size. Since it is not known before hand what ranges a client may wish to query nor how many clients will be performing queries, performing the generation of this search database dynamically when a request is made would be unwise. Instead a batch mode will be used to perform the desired processing for each revision in the desired range.

The processing of source files into a search database is accomplished by using two of Semantic Designs' tools: a Lexeme Extractor and a Search Engine Indexer. The Lexeme Extractor generates a file for each source file describing the lexemes contained therein. These lexeme files are then processed by the Search Engine Indexer into a single Search Engine index database with code metrics information. It is this database against which lexically aware queries are made.

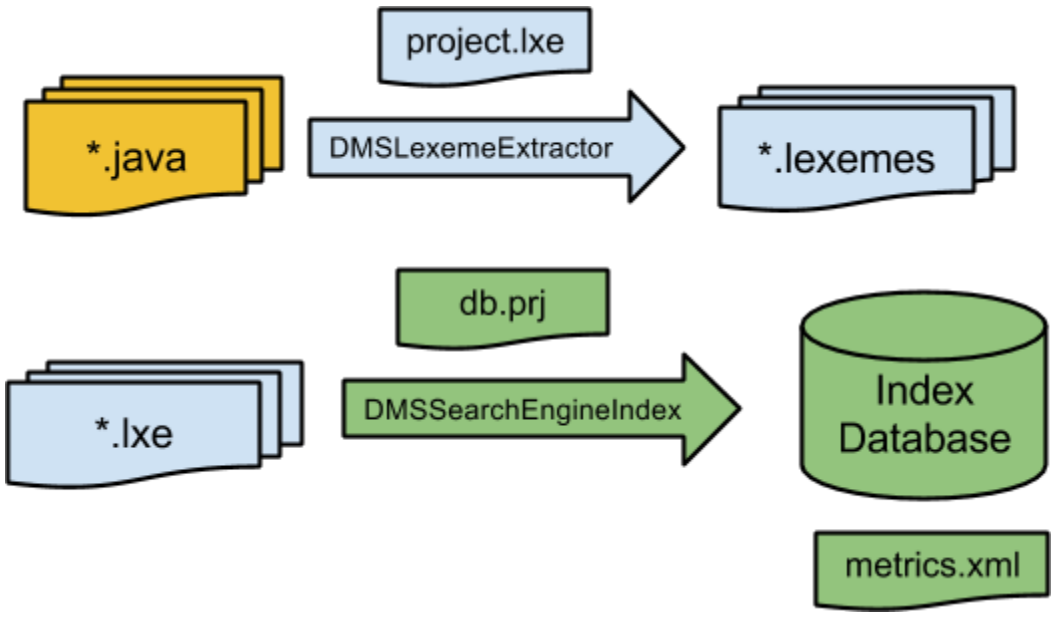


Figure 2 Default Generation of Index Database

The Semantic Designs' search tool was never designed to work across revisions of a codebase, nor to integrate with a VCS. To overcome this shortcoming and achieve the desired effect, the source files from each

revision of the project will be obtained from the VCS and combined into one massive directory structure which retains the original path and revision history in the VCS. As far as the Lexeme Extractor, Indexer, and Search Tool are concerned, this would simply be a single project as it is ignorant of any higher level semantic relations of a directory structure beyond the lexemes of the provided source files. A high level view of this process can be seen in the figure below:

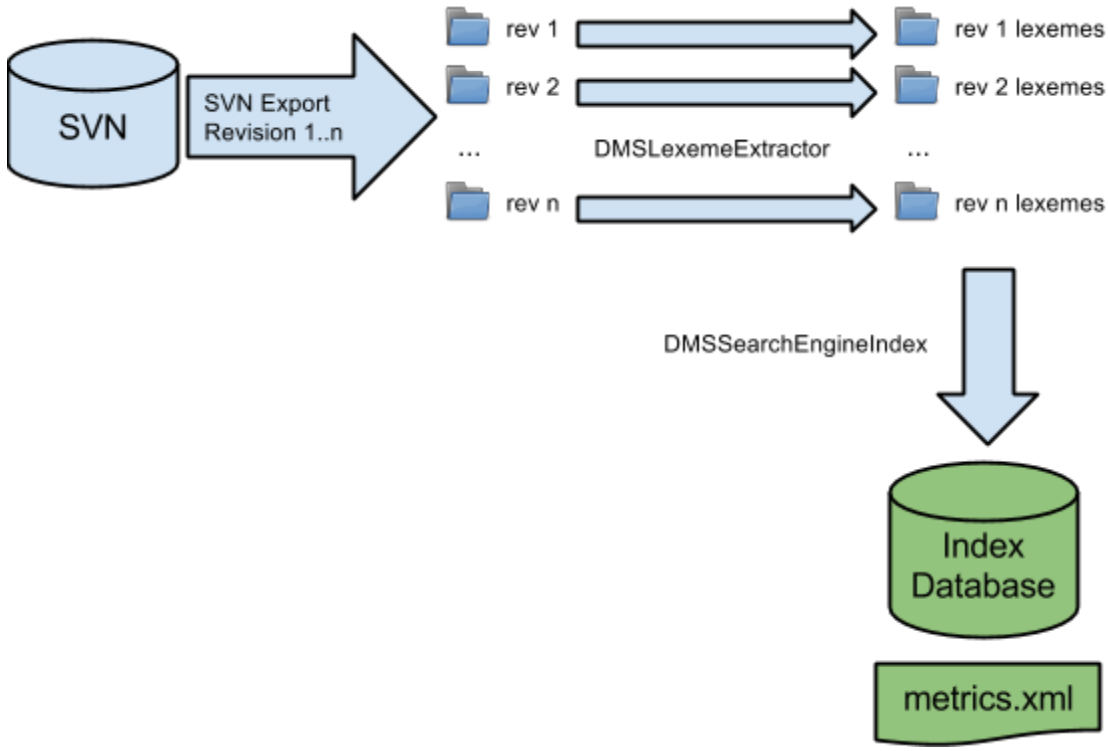


Figure 3 Modified Generation of Index Database

5.3 RESTful Interface

Having the ability to execute search queries and initiate a rebuild remotely from a variety of endpoints is desirable for scalability and utility. The main features necessary to utilize the Versioned Lexical Search database are exposed with three URIs: a method to execute a (re)build of the Search Database based on a given revision range, a method to execute lexical queries across a revision range, and a method to obtain code metrics for a given source file across a revision range. There is also a utility URI to download a file from a specific revision.

5.3.1 (Re)building Search Database

Initiating a rebuild of the the search database to include a particular range can be accomplished by executing an HTTP POST request to the Versioned Lexical Search server with the necessary parameters:

```
POST http://vls-server
    cmd=rebuild
    repo=<REPO>
    start=<START>
    end=<END>
```

where the parameter "cmd" is the "rebuild" command; "repo" is the name of the repository (ie. "batik"); and the parameters "start" and "end" are the starting and ending revisions in which to perform the rebuild.

Rebuilds are incremental and do not recreate the lexemes of existing source files nor export revisions from the VCS that have already been exported. Only the Search Index Database is recreated based on the lexemes newly made available from the union of the old and new revisions exported and processed from the VCS.

Response:

The response from the server is simply a 204 status code when it completes (Successfully processed, no content to return). This can potentially be modified to return a log of the process, or other desired information of the rebuild process. Given that the execution of a rebuild across a large range of revisions can take a significant amount of time to complete, a connection timeout is probable. This command is best used for smaller revision ranges. This command is idempotent.

5.3.2 Lexical Search Queries

In order to execute a query against the Lexical Search database remotely, an HTTP GET request to the server is performed with the necessary parameters:

```
GET http://vls-server
    cmd=query
    repo=<REPO>
    start=<START>
    end=<END>
    query=<QUERY>
```

where the parameter "cmd" is the "query" command; "repo" is the name of the repository (ie. "batik"); the parameters "start" and "end" are the starting and ending revisions in which to perform the query; and "query" is the search string is the target pattern to locate. The format for legal queries can be located in the Appendix.

Response:

```
Content-Type: text/csv

<LINE>,<REVISION>,<PATH>,<MATCH>
...
```

```
<LINE>          The line number where the match was found
<REVISION>      The revision number of the match
<PATH>          The file path in the repo. of the match.
                Example: "trunk/src/jpcsp/JpcspMainGUI.java"
<MATCH>         The surrounding line in the source file of the match
                Example (for the query 'static'):
                    "public static void main(String args[]){"
```

5.3.3 Code Metric Queries

During the generation of the Lexical Search database, code metrics are gathered as well. These metrics include: cyclomatic complexity, totals for lines of code, blank lines, and comment lines. By executing a GET request to the following URL, this information can be obtained for a given file across a range of revisions:

```
GET http://vls-server
    cmd=metrics
    repo=<REPO>
    start=<START>
    end=<END>
    path=<PATH>
```

Response:

```
    Content-Type: text/csv


<REV>,<PATH>,<TOTAL_LINES>,<CODE_LINES>,<COMMENT_LINES>,<BLANK_LINES>,<COMMENTS>,<CYCLO>
...
```

```
<REV>              The revision of the file
<PATH>             The path of the file
                   Ex: "trunk/src/jpcsp/JpcspMainGUI.java"
<TOTAL_LINES>      The total number of lines present in the file
<CODE_LINES>       The number of code lines
<COMMENT_LINES>    The number of comment lines
<BLANK_LINES>      The number of blank lines
<COMMENTS>         The comment count, in contrast to how many lines
                   these comments consume in the file
<CYCLO>            The cyclomatic complexity of the file
```

5.3.4 Downloading Files

To avoid requiring a client to have SVN command line tools installed or having to manually navigate through SVN's web interface to view a file, there is a URL exposed that enables the download of a file from the repository for a specific revision:

```
    GET http://vls-server
        cmd=download
        repo=<REPO>
        rev=<REV>
        path=<PATH>
```

where the parameter "cmd" is the "download" command; "repo" is the name of the repository (ie. "jpcsp"); the parameter "rev" is the revision number of the file, and "path" is the qualified path of the desired file (ie. "trunk/src/jpcsp/SomeFile.java")

Response:

```
    Content-Type: text/plain
```

The result of the GET query is the desired file.

5.4 SVN Extension

SVN provides a means of extending functionality through a feature called "hooks". A hook is a program which is executed by SVN following a predefined event such as the commitment of a new revision to the repository. This specific event "post-commit" is the one which will be leveraged to ensure that the Lexical Search database remain synced with the codebase in the repository.

With the availability of a url to perform the necessary update and generation of Lexical Search databases for a given repository, the extension of SVN becomes a trivial matter. When the "post-commit" hook is executed by SVN, the hook executes an HTTP POST request to the service url with current repository name and revision number. Ex:

```
POST http://vls-server
    cmd=rebuild
    repo=jpcsp
    start=173
    end=173
```

While the assumption is that the SVN server is on the same machine as the Lexical Database, this is not required. The local domain in the example could be replaced by an external one at the penalty of efficiency. When the SVN server and the Lexical Database are indeed on the same machine, and a local url is given (i.e. 127.0.0.1), this is generally resolved by the machine in a way that avoids utilizing the network hardware.

5.5 Eclipse Plugin

While the exposed URLs provide an endpoint agnostic means with which to execute commands, they are not user friendly. Extending an IDE (Eclipse in this case), with a means to execute the commands behind the scenes and provide graphical views of the responses can aid usability of the server and provide a clearer understanding of a query result in context.

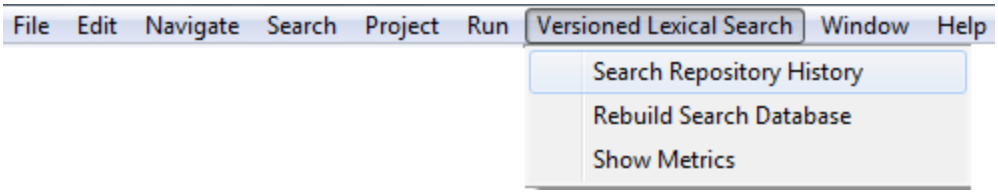


Figure 4 VLS System Menu

The first addition to the IDE is a new System Menu item which contains a list of commands supported by the plugin. Each of these menu items will yield a prompt which will perform queries against a given Versioned Lexical Search server and provide graphical results where applicable.

5.5.1 Executing a Search Query

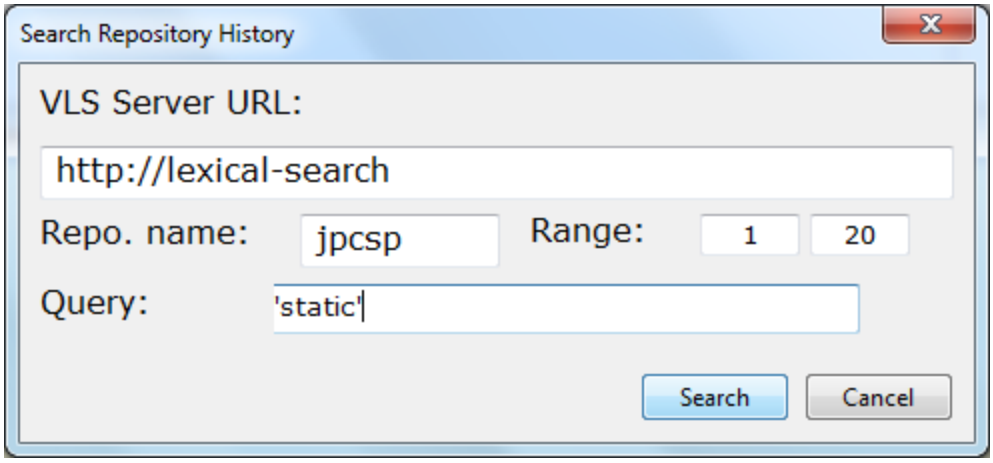
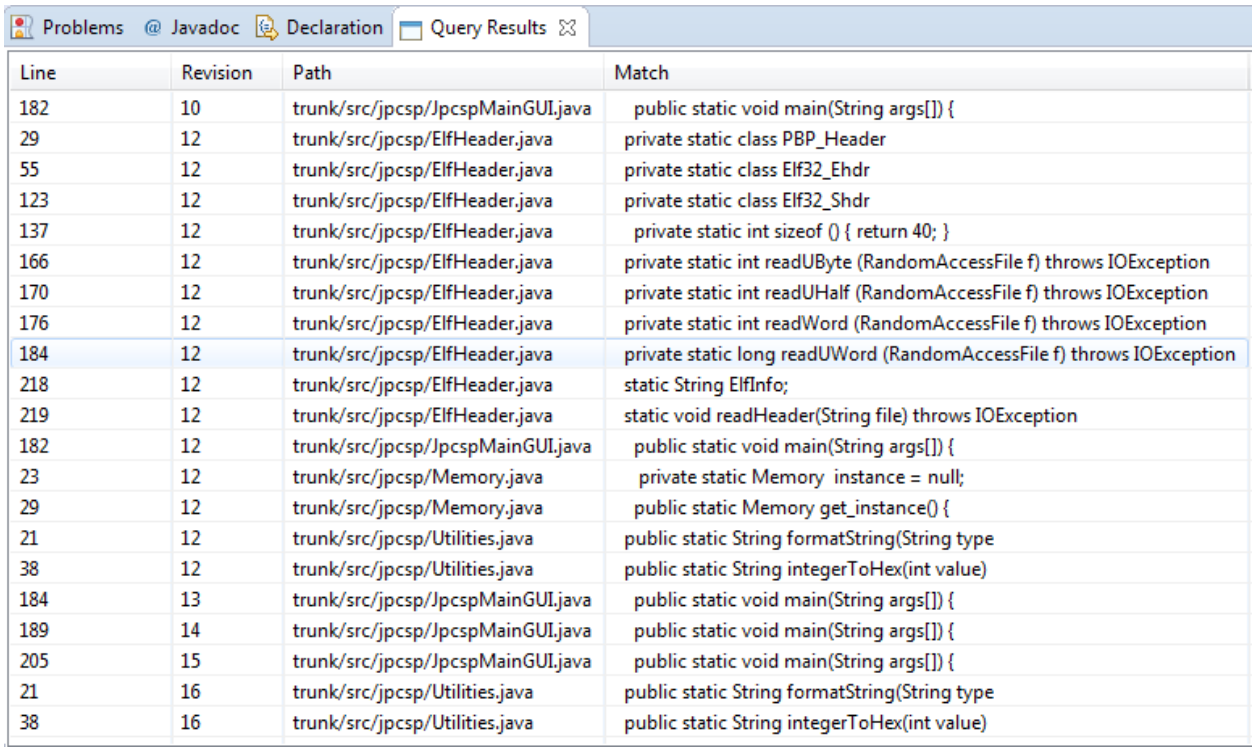


Figure 5 Search Repository Prompt

The first menu option provides a form with which to execute lexical search queries against a Versioned Lexical Search server. Upon executing the search, the query url (described in section 4.3.2) is generated in the proper format then visited. The response CSV is then parsed and displayed in tabular form.

5.5.2 Viewing Search Results



Line	Revision	Path	Match
182	10	trunk/src/jpcsp/JpcspMainGUI.java	public static void main(String args[]) {
29	12	trunk/src/jpcsp/ElfHeader.java	private static class PBP_Header
55	12	trunk/src/jpcsp/ElfHeader.java	private static class Elf32_Ehdr
123	12	trunk/src/jpcsp/ElfHeader.java	private static class Elf32_Shdr
137	12	trunk/src/jpcsp/ElfHeader.java	private static int sizeof () { return 40; }
166	12	trunk/src/jpcsp/ElfHeader.java	private static int readUByte (RandomAccessFile f) throws IOException
170	12	trunk/src/jpcsp/ElfHeader.java	private static int readUHalf (RandomAccessFile f) throws IOException
176	12	trunk/src/jpcsp/ElfHeader.java	private static int readWord (RandomAccessFile f) throws IOException
184	12	trunk/src/jpcsp/ElfHeader.java	private static long readUWord (RandomAccessFile f) throws IOException
218	12	trunk/src/jpcsp/ElfHeader.java	static String ElfInfo;
219	12	trunk/src/jpcsp/ElfHeader.java	static void readHeader(String file) throws IOException
182	12	trunk/src/jpcsp/JpcspMainGUI.java	public static void main(String args[]) {
23	12	trunk/src/jpcsp/Memory.java	private static Memory instance = null;
29	12	trunk/src/jpcsp/Memory.java	public static Memory get_instance() {
21	12	trunk/src/jpcsp/Utilities.java	public static String formatString(String type
38	12	trunk/src/jpcsp/Utilities.java	public static String integerToHex(int value)
184	13	trunk/src/jpcsp/JpcspMainGUI.java	public static void main(String args[]) {
189	14	trunk/src/jpcsp/JpcspMainGUI.java	public static void main(String args[]) {
205	15	trunk/src/jpcsp/JpcspMainGUI.java	public static void main(String args[]) {
21	16	trunk/src/jpcsp/Utilities.java	public static String formatString(String type
38	16	trunk/src/jpcsp/Utilities.java	public static String integerToHex(int value)

Figure 6 Query Result View

The results of the query executed from the prompt described in the previous section are displayed in tabular form. The columns match the given columns in the parsed CSV provided from the VLS server:

"Line": The line in the source file where the match occurs
"Revision": The revision of the source code file in the repository.
"Path": The qualified path of the file in the repository.
"Match": A preview of the line in which the match was discovered. Used as a contextual view to aid in finding a desired match.

Opening Search Result

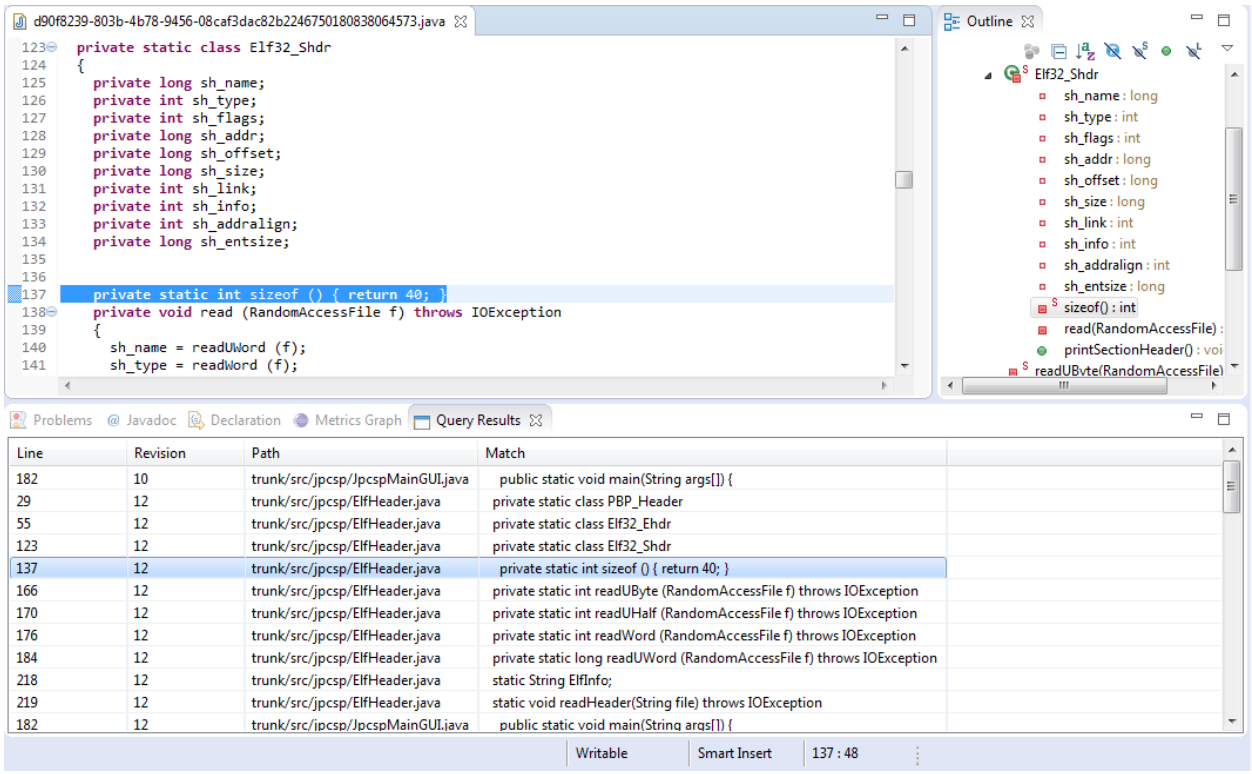


Figure 7 Opening Query Result

By double clicking on one of the search results in the tabular view, the specified file is downloaded from the VLS server (by executing a request against the URL specified in section 4.3.4) and opened in the IDE with the specified line highlighted.

5.5.3 Executing Code Metrics Query

Obtaining statistics on a repository across a specific range can be obtained from the IDE by using the "Show Metrics" prompt. Along with the VLS server url and a revision range, a qualified file path must be provided as well. The results of the query are then displayed graphically as a plot:

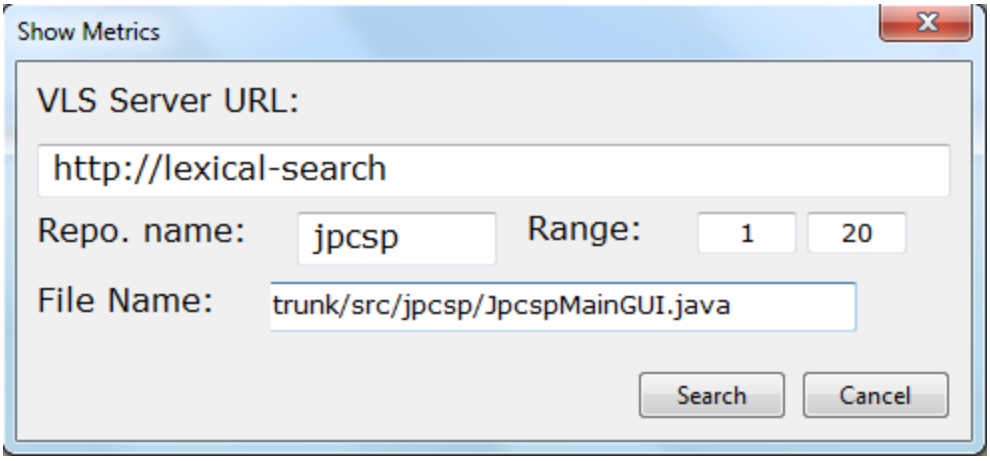


Figure 8 Show Metrics Prompt

5.5.3 Code Metrics Trend Lines

The resulting CSV of a metrics query is parsed and displayed in a graphical line plot across the specified revisions. The missing revision numbers from the graph occur when the file is unchanged in the Subversion Server (executing an "svn log" command on the revision range would exclude the file for the given revision).

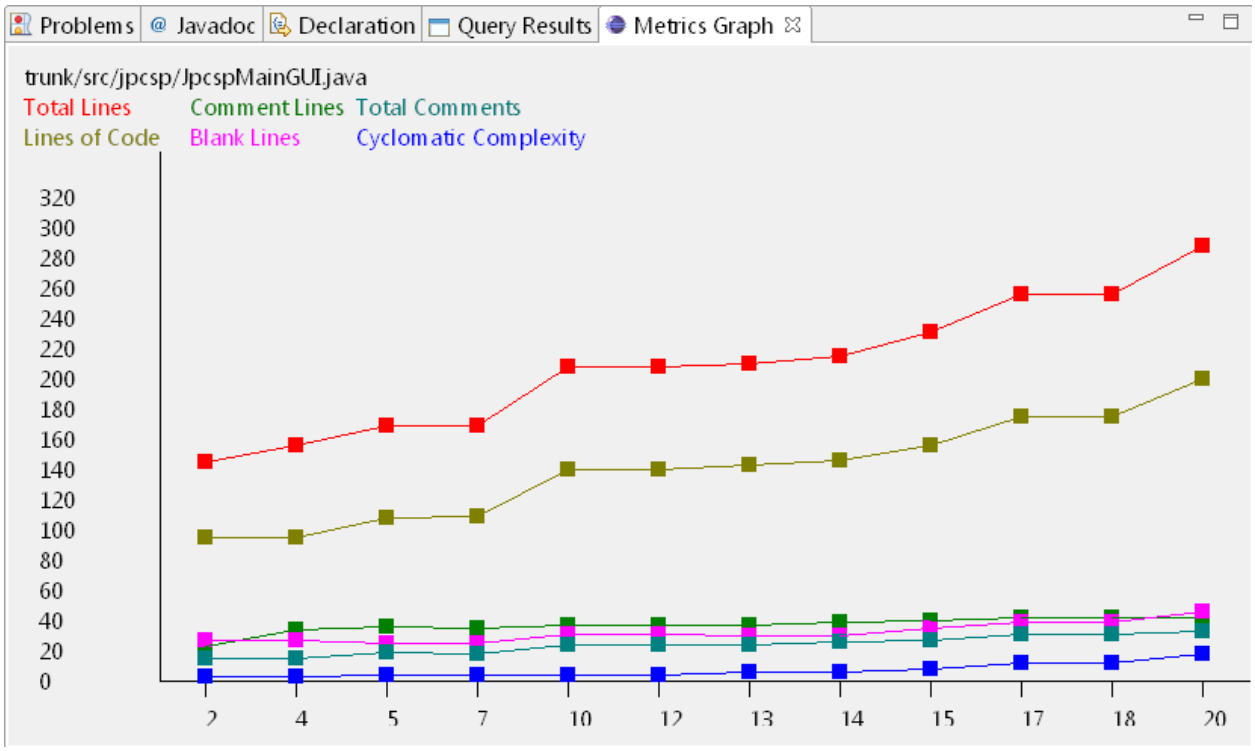


Figure 9 Metrics Graph View

Rebuilding Versioned Lexical Search Database

The SVN repository is already extended to automatically rebuild the VLS database when a commit occurs (Section 4.4). In the event that the database needs to be rebuilt for a given revision range, this functionality is also exposed in the IDE through the "Rebuild Search Database" menu option:

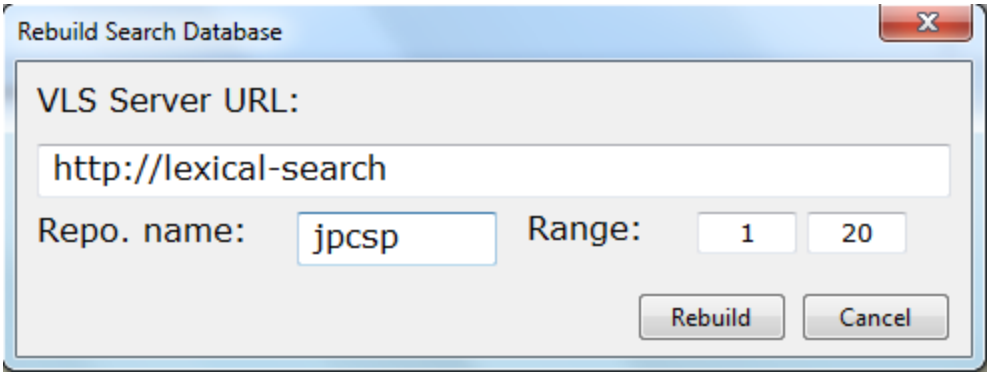


Figure 10 Rebuild Database Prompt

Executing a rebuild on a large revision range can potentially be significantly time consuming. As a result, smaller ranges should be used. If a complete rebuild needs to be done, this can potentially cause timeout issues with the IDE. The rebuild process will continue regardless of the client status.

6. Related Work

Three systems have been identified as being similar to the work accomplished here: LXR [6], OpenGrok [7], and FishEye[8].

The LXR Cross Referencer is a source code indexer and cross-referencer. It provides a web-based interface to navigate code by transforming source code files into hypertext documents with the identifiers of the program text into navigable links. A query system is provided, but is limited to regular expressions, identifiers, and file name queries. The LXR system does follow a similar approach to managing multiple revisions though from a VCS by leveraging a sub-directory structure that encodes the information. This system is also similar in that it uses an indexing and database creation phase before the system can be utilized. The LXR database only contains the identifiers and their locations though, and is not lexically away beyond this.

FishEye is a commercial revision control browser created by Atlassian, Inc. which provides the ability to browse and search the revision history of a number of repositories in a single web-based view. The query language the system utilizes for performing searches is a SQL-like DSL referred to as "EyeQL". Unlike the system described in this paper, FishEye is a read-only view into the underlying VCSs, also the query language provided by FishEye operates on the metadata of the repository files and not the content of the files themselves. For example: directory, revision, author, and date. The FishEye system can not perform lexically aware queries over the underlying source files.

The OpenGrok system is most similar to the work described in this paper and is an open source Source Code Search Engine which provides plugins for a number of IDEs and provides a web-based, read-only view into a number of VCSs. OpenGrok's lexical search functionality is the most advanced of the three systems described in this section; it is capable of locating symbols and definitions in a VCS and displaying the results. While development of OpenGrok began in late 2006, it is still considered a relatively young product (version 0.12 at the time of this writing) and lacks some of the more advanced query features available in the Semantic Designs search engine utilized by the project described in this paper. OpenGrok's search capabilities, while lexical, are limited to identifying definitions, symbols, path, and revision history. More complex information such as queries based on token classification, or range queries are not supported. The syntax, instead of being free-form and compositional, instead utilizes a syntax similar to what

is used in Google's search engine: "path:Makefile defs:target". There is also limited regular expression support. OpenGrok also presents historical results in a different manner than the Versioned Lexical Search system (VLS) in this paper. In VLS the approach followed is to identify a syntactic form and display the evolution of that form over time. In OpenGrok, the approach is to identify a result, and then navigate from that file in a means similar to LXR, namely to follow definition references through hypertext to other source files.

7. Conclusion

In this thesis a description of a system was provided which has the capability of performing lexically aware queries across a Version Control System from an IDE or through a RESTful API which yields meaningful results for a provided syntactic form as well as a means of obtaining metric data about a codebase throughout its evolution.

The querying functionality and related results were enabled by extending and integrating into a popular IDE (Eclipse) as a plugin, by adding a post-commit hook to a popular VCS (Subversion), exposing a set of RESTful service urls to provide an API, and leveraging a commercial language parser + search engine to build a lexically aware search database (Semantic Designs "Source Code Search Engine").

While there exists a number of similar tools in the same spirit as the system described in this paper, none of those tools evaluated were designed with the goal of following the evolution of a syntactic construct over time in a codebase. Instead, these tools emphasized the ability to locate a symbol or definition and then provide a means to navigate from that result through the rest of the codebase by following a graph of references generated from source file metadata and matching definition results.

8. Future Work

- **Performance**
  Regrettably, the means for the current implementation to interact on a server with the Semantic Designs tools and SVN is limited to a set of command line executables which complicate parallel processing as well as require the implementation to parse results utilizing regular expressions on a set of strings ("svn log", "DMSSearchEngine", etc). Having a set of APIs available from both systems which can return structured data would aid in increasing the performance of this system.
- **Broader Repository Support**
  SVN is not the only VCS which exposes "hook"-like functionality or a means to query and export file contents. Given that these are the only requirements in order to integrate a VCS with the creation of the necessary Lexical Search databases, integration with these alternative VCSs is a logical next step.
- **Broader IDE support**
  Given that the efforts of Lexical Search database creation and Version Control is managed on a remote server behind a set of URLs, there is no tight coupling to a client beyond a plugin for one or more desired development environments to enable a user-friendly view for executing queries and viewing results. The extension of the Eclipse IDE as a proof-of-concept can be seen as a first step do to the popularity of a variety of text-editors and other IDEs in use
- **Broader Language Support**
  The current implementation only supports the Java programming language. The Semantic Designs Search Engine can be utilized further to enable queries across dozens of other programming languages but this requires further effort in the command line parsers for each specific language.

9. Appendix

9.1 Query Syntax

The Semantic Designs command line tool for performing search queries (DMSSearchEngine) supports a variety of patterns. The exposed URL and IDE options exposed to the client will forward the textual query pattern to this tool and execute it as provided. Not all features of the search engine are exposed, but for the features which are indeed made available can be found in this section. This information is derived from the provided documentation that accompanies the installation of the Semantic Designs tools[2].

Reserved Tokens

Reserved words in the source text are discovered by singly quoting the desired token:

```
'static'
```

A sequence of these tokens may also be used. This sequence is represented by the juxtaposition of quoted token forms:

```
'public' 'void'
```

Token Classifications

Tokens which are not Reserved Words can be specified by classification identifiers using the form: <ID>=<VALUE>, where <ID> is a letter representing a token's classification:

```
I K O P S N F C
```

Which stand for: Identifier, Keyword, Operator, Punctuation, String, Number, Float and Comment, respectively. The <Value> is the value of that token. For example, to locate the Identifier named "ElfHeader", a query of the following form can be used:

```
I=ElfHeader
```

This form is composable and sequentiable with other query forms as well:

```
'public' 'void' I=ElfHeader
```

Wildcards

Identifier token forms can be specified with wildcard characters '*' in order to execute generic queries:

```
I=Elf*
```

Which finds all identifiers with the prefix 'Elf'

Nearby Queries

A sequence of token patterns do not have to be adjacent when creating a query. To find token forms separated by an irrelevant or unknown set of tokens in-between the two, the '...' form is used to ignore the intermediary tokens:

```
'for' '(' ... ')'
```

10. References

[1] Alex Garcia. *Versioned Lexical Search Interface*.
University of Wisconsin-Milwaukee, 2012

[2] "Source Code Search Engine" *Semantic Designs*
21 July 2014. Web. 21 Jul. 2014 <http://semdesigns.com/Products/SearchEngine>

[3] "Apache™ Subversion®" *Apache Subversion*
21 July 2014. Web. 21 Jul. 2014 <http://subversion.apache.org/>

[4] "Eclipse desktop & web IDE" *The Eclipse Foundation*
21 July 2014. Web. 21 Jul. 2014 <http://eclipse.org/ide/>

[5] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. p. 76-105

[6] "LXR Cross Referencer" *The LXR Project*
01 October 2014. Web. 01 Oct. 2014 <http://lxr.sourceforge.net>

[7] "{OpenGrok" *OpenGrok*
01 October 2014. Web. 01 Oct. 2014 <https://opengrok.github.io/OpenGrok/>

[8] "FishEye" *Atlassian FishEye*
01 October 2014. Web. 01 Oct. 2014 <https://www.atlassian.com/software/fisheye>

Changeset

---

1. Introduction Section added
2. Conclusion Section expanded
3. Font sizing corrected across document
4. Spelling corrections across document
5. Table of Contents updated
6. List of Figures Section updated
7. Added Related Work section
8. Added reference to REST and related works