The Role of the Version Control Information in Code Comprehension

Tibor Brunner
Department of Programming Languages and Compilers
Eötvös Loránd University
Budapest, Hungary
bruntib@caesar.elte.hu

Zoltán Porkoláb
Department of Programming Languages and Compilers
Eötvös Loránd University
Budapest, Hungary
gsd@caesar.elte.hu

***Abstract*—Most software comprehension frameworks use the source code as the main resource for information retrieval. Advanced code comprehension process, however, requires the utilization of the full knowledge portfolio of the software system. In this paper we investigate how the version control information of the project can be utilized for extending our apprehension of large legacy systems providing a better understanding of the software under examination. We show that some of the hidden structural connections between the elements of the program can be revealed most easily by the development history of the system. A prototype implementation of the method using git version control information has been implemented as an open source extension of the CodeCompass software comprehension framework.**

***Index Terms*—code comprehension, version control, git, software technology**

I. Introduction

It is a well known fact, that the largest cost factor of the software products for their whole lifetime is the maintenance cost. One of the reasons is that prior to any maintenance activity – new feature development, bug fixing, etc. – programmers first have to locate the place where the change applies, have to understand the actual code, and have to explore the connections to other parts of the software to decide *how* to interact in order to avoid regression. All these activities require an adequate understanding of the code in question and its certain environment – something is currently impossible to automate, therefore the developers should pay their expensive time to carry out these actions.

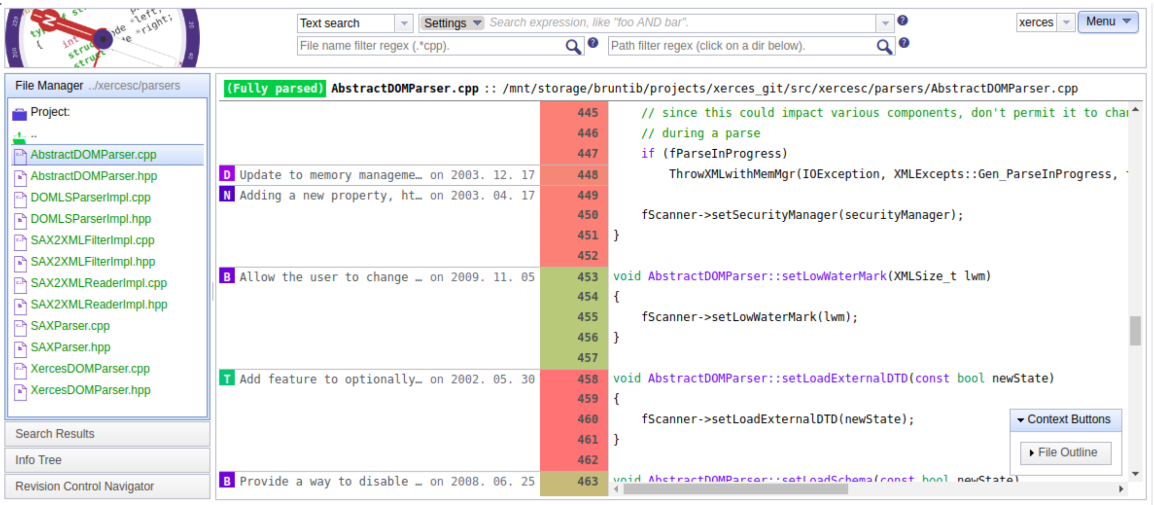
Therefore, it is not a surprise that code comprehension is a key factor of modern software development, exhaustively researched by both the industry and academy. Various scientific and industrial papers published on the topic in conferences, e.g. in the series of International Conference of Program Comprehension, and in the Intellectual outputs No. O1 and O2 of the Erasmus+ Key Action 2 (Strategic partnership for higher education) project No.2017-1-SK01-KA203-035402: "Focusing Education on Composability, Comprehensibility and Correctness of Working Software" [1], [2] among others.

Most of the comprehension approaches are based on the source code. That is a logical approach, as the actual software might already diverged from the original specification and the documentation might also be over dated. Therefore, typical comprehension tools analyse the source code, support fast navigation, feature location and reveal the internal structure of the software. However, not all the internal connections within the system can be detected by analysing the source. Virtual function calls on polymorphic objects, pointers, references, closures are among the program constructs where static analysis has limitations.

Code comprehension may not be restricted to existing code bases. Important architectural information can be gained from the build system, like relations between libraries, binaries and source files [3], [4]. Even more interesting structural connections can be revealed from the history of the project development: e.g. which files were added or changed at the same time, how these changes related to certain commit messages and which lines added/removed/changed frequently simultaneously.

CodeCompass is an open source code comprehension framework [5] developed by Ericsson and Eötvös Loránd University, Budapest, to help the code comprehension process of large legacy systems. The tool based on the LLVM/Clang compiler infrastructure [6], [7], and has been designed to be extremely scalable, seamlessly working with many million lines of code. Fast search options help locating the requested feature by text search. Once the feature has been located, precise information on language elements for variables, inheritance and aggregation relationships of types, and call points of functions are provided by the LLVM/Clang infrastructure. Easy navigation possibilities and a wide range of visualizations extend far more than the usual class and function call diagrams help the user the more complete comprehension. To make the comprehension more extensive, CodeCompass utilises the full portfolio of information available including build commands but also utilizes version control information, if available; git commit and branching history, blame view are also visualized.

In this paper we investigate the role of the version control information for code comprehension purposes. In Section II we overview the main categories of the existing comprehension software using a specific tool as an example. Section III describes the architecture of CodeCompass. In Section IV we show CodeCompass support for the version control information to reveal hidden connections between otherwise unrelated code segments. Our paper concludes in Section V.


Text search
Settings
Search expression, like "foo AND bar".
File name filter regex (.*cpp).
Path filter regex (click on a dir below).
xerces
Menu
File Manager ../xercesc/parsers
Project:
..
Search Results
Info Tree
Revision Control Navigator
Context Buttons
File Outline

AbstractDOMParser.cpp
AbstractDOMParser.hpp
DOMLSParserImpl.cpp
DOMLSParserImpl.hpp
SAX2XMLFilterImpl.cpp
SAX2XMLFilterImpl.hpp
SAX2XMLReaderImpl.cpp
SAX2XMLReaderImpl.hpp
SAXParser.cpp
SAXParser.hpp
XercesDOMParser.cpp
XercesDOMParser.hpp

(Fully parsed) **AbstractDOMParser.cpp** :: /mnt/storage/bruntib/projects/xerces_git/src/xercesc/parsers/AbstractDOMParser.cpp

```
                                                     // since this could impact various components, don't permit it to char
                                                     // during a parse
                                                     if (fParseInProgress)
D Update to memory manageme… on 2003. 12. 17             ThrowXMLwithMemMgr(IOException, XMLExcepts::Gen_ParseInProgress, 
N Adding a new property, ht… on 2003. 04. 17
                                                     fScanner->setSecurityManager(securityManager);
                                                 }

B Allow the user to change … on 2009. 11. 05     void AbstractDOMParser::setLowWaterMark(XMLSize_t lwm)
                                                 {
                                                     fScanner->setLowWaterMark(lwm);
                                                 }

T Add feature to optionally… on 2002. 05. 30     void AbstractDOMParser::setLoadExternalDTD(const bool newState)
                                                 {
                                                     fScanner->setLoadExternalDTD(newState);
                                                 }

B Provide a way to disable … on 2008. 06. 25     void AbstractDOMParser::setLoadSchema(const bool newState)
```

Fig. 1: Git change view

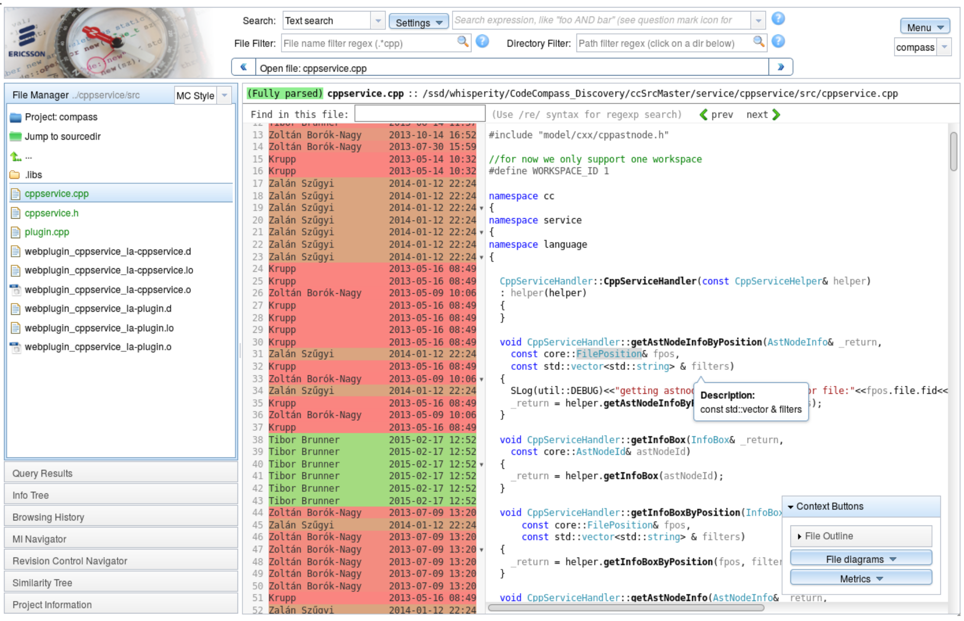
II. Related work

Code comprehension became a hot research topic recently, with dedicated user communities, proprietary and open source tools. On the software market there are several tools which aim some kind of source code comprehension. Some of them uses static analysis, others examine also the dynamic behavior of the parsed program. These tools can be divided into different archetypes based on their architectures and their main principles. On the one hand are tools having server-client architecture. Generally these tools parse the project and store all necessary information in a database. The (usually web-based) clients are served from the database. These tools can be integrated into the workflow as nightly CI runs. This way the developers can always browse and analyze the whole, large, legacy codebase. Also there are client-heavy applications where smaller part of the code base is parsed. This is the use case for IDE editors where the frequent modification of the source requires quick update of the database about analyzed results. In this section we present some tools used in industrial environment from each categories.

Woboq Code Browser [8] is a web-based code browser for C and C++. This tool has extensive features which aim for fast browsing of a software project. The user can quickly find the files and named entities by a search field which provides code completion for easy usability. The navigation in the code base is enabled through a web page consisting of static HTML files. These files are generated during a parsing process. The advantage of this approach is that the web client will be fast since no “on the fly” computation is needed on the server side while browsing. Hovering the mouse on a specific function, class, variable, macro, etc. can show the properties of that element. For example, in case of functions one can see its signature, place of its definition and place of usages. For classes one can check the size of its objects, the class layout and offset of its members and the inheritance diagram. For variables one can inspect their type and locations where they are written or read.

OpenGrok [9] is a fast source code search and cross reference engine. Opposed to Woboq, this tool doesn’t perform deep language analysis, therefore it is not able to provide semantic information about the particular entities. Instead, it uses *Ctags* [10] for parsing the source code only textually, and to determine the type of the specific elements. Simple syntactic analysis enables the distinguishing of function, variable or class names, etc. The search among these is highly optimized, and therefore very fast even on large code bases. The search can be accomplished via compound expressions, containing even wildcards, furthermore, results can be restricted to sub-directories. In addition to text search there is opportunity to find symbols or definitions separately. The lack of semantic analysis allows Ctags to support several (41) programming languages. Also an advantage of this approach is that it is possible to incrementally update the index database. OpenGrok also gives opportunity to gather information from version control systems like Mercurial, SVN, Git, etc. OpenGrok has the ability to search not only in the content of source files but in their history as well. Since most of these version control systems (VCS) provide search functionalities in the project history (including commit messages and source files), Open-Grok can forward these queries to the given VCS. However, there are no extra visualizations in order to display the “blame view” so the developer could understand what other relevant changes happened in other files in the same commit. The branches of the history are invisible too. CodeCompass intends to support these use-cases.

Understand [11] is not only a code browsing tool, but a also a complete IDE. Its great advantage is that the source code can be edited and the changes of the analysis can be seen immediately. Besides code browsing functions already mentioned for previous tools, Understand provides a lot of metrics and reports. Some of these are the lines of code (total/average/maximum globally or per class), number of coupled/base/derived classes, lack of cohesion [12], McCabe


Search:
Text search
Settings
Search expression, like "foo AND bar" (see question mark icon for
Menu
File Filter:
File name filter regex (.*cpp)
Directory Filter:
Path filter regex (click on a dir below)
compass
Open file: cppservice.cpp
File Manager ../cppservice/src
MC Style
Project: compass
Jump to sourcedir
...
.libs
cppservice.cpp
cppservice.h
plugin.cpp
webplugin_cppservice_la-cppservice.d
webplugin_cppservice_la-cppservice.lo
webplugin_cppservice_la-cppservice.o
webplugin_cppservice_la-plugin.d
webplugin_cppservice_la-plugin.lo
webplugin_cppservice_la-plugin.o
Query Results
Info Tree
Browsing History
MI Navigator
Revision Control Navigator
Similarity Tree
Project Information
Find in this file:
(Use /re/ syntax for regexp search)
prev
next
Description:
const std::vector & filters
Context Buttons
File Outline
File diagrams
Metrics

(Fully parsed) **cppservice.cpp** :: /ssd/whisperity/CodeCompass_Discovery/ccSrcMaster/service/cppservice/src/cppservice.cpp

```
Zoltán Borók-Nagy    2013-10-14 16:52  #include "model/cxx/cppastnode.h"
Zoltán Borók-Nagy    2013-07-30 15:59
Krupp                2013-05-14 10:32  //for now we only support one workspace
Krupp                2013-05-14 10:32  #define WORKSPACE_ID 1
Zalán Szűgyi         2014-01-12 22:24
Zalán Szűgyi         2014-01-12 22:24  namespace cc
Zalán Szűgyi         2014-01-12 22:24  {
Zalán Szűgyi         2014-01-12 22:24  namespace service
Zalán Szűgyi         2014-01-12 22:24  {
Zalán Szűgyi         2014-01-12 22:24  namespace language
Zalán Szűgyi         2014-01-12 22:24  {
Krupp                2013-05-16 08:49
Krupp                2013-05-16 08:49    CppServiceHandler::CppServiceHandler(const CppServiceHelper& helper)
Zoltán Borók-Nagy    2013-05-09 10:06    : helper(helper)
Krupp                2013-05-16 08:49    {
Krupp                2013-05-16 08:49    }
Krupp                2013-05-16 08:49
Krupp                2013-05-16 08:49    void CppServiceHandler::getAstNodeInfoByPosition(AstNodeInfo& _return,
Zalán Szűgyi         2014-01-12 22:24      const core::FilePosition& fpos,
Krupp                2013-05-16 08:49      const std::vector<std::string> & filters)
Zoltán Borók-Nagy    2013-05-09 10:06    {
Zalán Szűgyi         2014-01-12 22:24      SLog(util::DEBUG)<<"getting astnod                    or file:"<<fpos.file.fid<<
Krupp                2013-05-16 08:49      _return = helper.getAstNodeInfoByP                    s);
Zoltán Borók-Nagy    2013-05-09 10:06    }
Krupp                2013-05-16 08:49
Tibor Brunner        2015-02-17 12:52    void CppServiceHandler::getInfoBox(InfoBox& _return,
Tibor Brunner        2015-02-17 12:52      const core::AstNodeId& astNodeId)
Tibor Brunner        2015-02-17 12:52    {
Tibor Brunner        2015-02-17 12:52      _return = helper.getInfoBox(astNodeId);
Tibor Brunner        2015-02-17 12:52    }
Tibor Brunner        2015-02-17 12:52
Zoltán Borók-Nagy    2013-07-09 13:20    void CppServiceHandler::getInfoBoxByPosition(InfoBox
Zalán Szűgyi         2014-01-12 22:24        const core::FilePosition& fpos,
Zoltán Borók-Nagy    2013-07-09 13:20        const std::vector<std::string> & filters)
Zoltán Borók-Nagy    2013-07-09 13:20    {
Zoltán Borók-Nagy    2013-07-09 13:20      _return = helper.getInfoBoxByPosition(fpos, filter
Zoltán Borók-Nagy    2013-07-09 13:20    }
Zoltán Borók-Nagy    2013-07-09 13:20
Krupp                2013-05-16 08:49    void CppServiceHandler::getAstNodeInfo(AstNodeInfo& _return,
Zalán Szűgyi         2014-01-12 22:24
```

Fig. 2: Git blame view

complexity [13] and many others. *Treemap* is a common representation method for all metrics. It is a nested rectangular view where nesting represents the hierarchy of elements, and the color and size dimensions represent the metric chosen by the user. For large code bases, the inspection of the architecture is necessary. Visual representation is one of the most helpful way of displaying such structures. Understand can show dependency diagrams based on various relations such as function call hierarchy, class inheritance, file dependency, file inclusion/import. The users can also create their custom diagram type via the API provided by the tool.

CodeSurfer [14] is similar to Understand in the sense that it is also a thick client, static analysis application. Its target is understanding C/C++ or x86 machine code projects. CodeSurfer accomplishes deep language analysis which provides detailed information about the software behavior. For example, it implements pointer analysis to check which pointers may point to a given variable, lists the statements which depend on a selected statement by impact analysis, and uses dataflow analysis to pinpoint where a variable was assigned its value, etc.

Development tools The aforementioned tools are mainly designed for code comprehension. Another application area of static analysis is writing the code itself. This is a very different way of working in many aspects, which requires a slightly different tool set. Maybe the most widespread IDEs are *NetBeans* [15] and *Eclipse* [16] primarily for Java projects, and *QtCreator* [17] mainly for C++ projects. The recent open source tools tend to be pluginable so their functions can easily be extended according to special needs and domain specific tasks. The greatest benefit of these tools is the ability of incremental parsing, which means the real-time re-analysis of small deviations in the source code. The *Visual Studio* [18] IDE has a rich interface for code comprehension features, like go to definitions and all references among others.

III. The CodeCompass architecture

CodeCompass provides a read-only, searchable and navigable snapshot of the source code, rendered in both textual and graphical formats. CodeCompass is built with a traditional server-client architecture as depicted in Figure 3. The server application provides a Thrift [19] interface to clients over HTTP transport. The primary client that comes pre-packaged with the tool is a web browser based single-page HTML application written in HTML and JavaScript.

Since the interface is specified in the Thrift interface definition language, additional client applications (such as a command line client or an IDE plugin) can be easily written in more than 15 other languages supported by Thrift (including C/C++, Java, Python etc.). An experimental Eclipse plugin is already implemented.

A parsed snapshot of the source code is called a *workspace*. A workspace is physically stored as a relational database instance and additional files created during the *parsing process*. The parsing process consists of running different *parser plugins* on the source code. The most important parser plugins are:

A **search parser** iterates recursively over all files in the source folder and uses Apache Lucene [20] to collect all words from the source code. These words are stored in a search index, with their exact location (file and position).

The **C/C++ parser** iterates over a JSON compilation database containing build actions, using the LLVM/Clang parser [7] and stores the position and type information of specific AST nodes in the database. This database will be used by the *C/C++ language service* to answer Thrift calls regarding C/C++ source code.

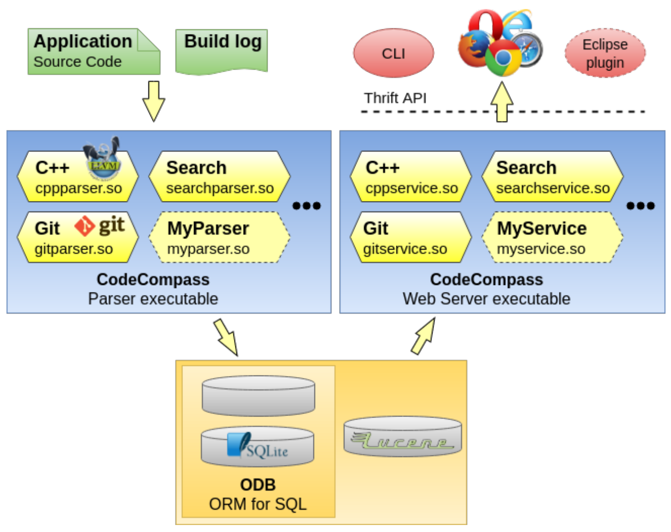


Fig. 3: CodeCompass architecture

The **Java parser** iterates over the same JSON compilation database containing build actions, using the Eclipse JDT parser [21] and stores the position and type information of specific AST nodes in the database. This database will be used by the *Java language service* to answer Thrift calls regarding Java source code.

Among other parser plugins the **Git parser** reads the version control information from the source tree (in the `.git` directories) and stores it into the project database.

CodeCompass has an extensible architecture, so new parser plugins can be written easily in C/C++ language. Parser plugins can be added to the system as shared objects.

On the **webserver**, Thrift calls are served by so-called *service plugins*. A service plugin implements one or more Thrift services and serves client requests based on information stored in a workspace. A Thrift service is a (remotely callable) collection of method and type definitions. All Thrift services have one implementation with the exception of the *language service*, which is implemented for C/C++, Java and Python. The *language service* is distinct in the sense that it provides the basic code navigation functionality for the languages it is implemented for. To put it simply, if this interface is implemented for a language, the user will be able to click and query information about symbols in the source code view of a file written in the given language.

Language service: This service provides query methods for symbols, for source files and globally for the workspace. It can return the symbol for a given source position, can return the corresponding node type and the corresponding source code fragment and documentation, the references to the symbol, diagrams – in GraphViz Dot format – or all information about the node in a tree format. For files, the service can return file level diagrams (in GraphViz Dot format), references to the file from other files, extended information about the file in a tree format, syntax highlighting data. This extended information includes, among others, all type and function definition within the file, list of files this file is referred to and referred from. The service can also return all defined symbols in a workspace (such as namespaces, types, functions), so symbol catalogs can be implemented.

Search Service: This service provides 4 different type of queries: search in *text* for words, search among *symbol definitions* only, search among *file names* and suggest search phrases, based on a search *phrase fragment*. Text, definition and file search can be filtered by file name and containing directory. Definition search can be filtered for a certain language (C/C++, Java, JavaScript, Python, etc.) and symbol type (function definition, type definition, variable declaration, etc.) Developers can add additional service plugins delivered as shared objects in run-time.

The **Web-based user interface** is organized into a static *top area*, extensible *accordion modules* on the left and also extensible *center modules* on middle-right – see Figure 4.

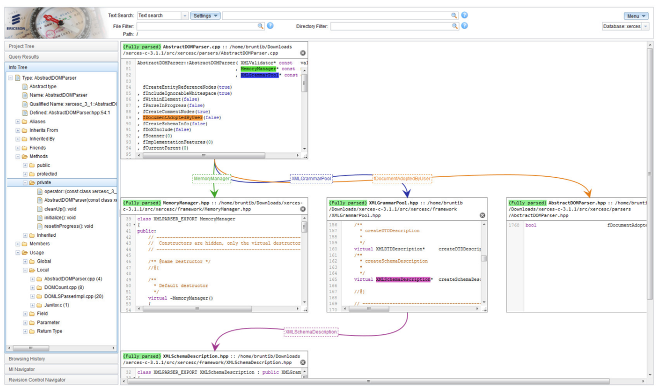


Fig. 4: CodeCompass user interface

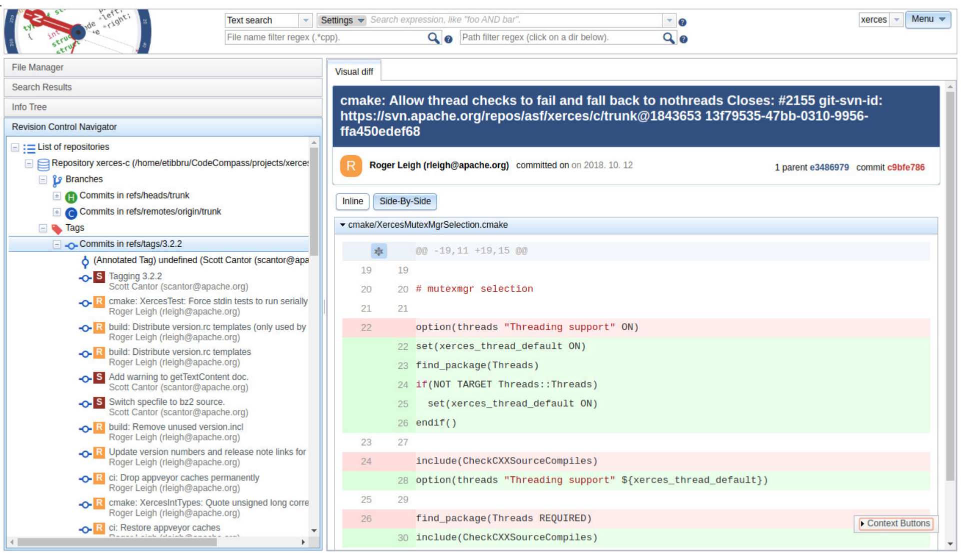
The source code and different visualizations are shown in the center, while navigation trees and lists, such as file tree, search results, list of static analysis (CodeChecker) bugs, browsing history, code metrics and version control navigation is shown on the left. New center modules and accordion panels can be added by developers.

The top area shows the search toolbar, the currently opened file, the workspace selector, simple navigation history (breadcrumbs) and a generic menu for user guides.

IV. Version Control Support in CodeCompass

CodeCompass supports the code comprehension via various visualizations to present the different views of git information related to the project. As a starting point, one can initiate the **blame view** on any source code. Git blame view shows line-by-line the last changes (commits) to a given source file as seen on Figure 2. The background color of the committer also holds information: the commit that happened recently are colored lighter green, while older changes are darker red. This view is excellent to review why certain lines were added to a source file.

Clicking to the committers name of the blame view CodeCompass brings us to the **commit information**. This contains


Text search
Settings
Search expression, like "foo AND bar".
File name filter regex (.*cpp).
Path filter regex (click on a dir below).
xerces
Menu
File Manager
Search Results
Info Tree
Revision Control Navigator
Visual diff
Inline
Side-By-Side
Context Buttons

List of repositories
Repository xerces-c (/home/etibbru/CodeCompass/projects/xerces
Branches
Commits in refs/heads/trunk
Commits in refs/remotes/origin/trunk
Tags
Commits in refs/tags/3.2.2
(Annotated Tag) undefined (Scott Cantor (scantor@apa
Tagging 3.2.2
Scott Cantor (scantor@apache.org)
cmake: XercesTest: Force stdin tests to run serially
Roger Leigh (rleigh@apache.org)
build: Distribute version.rc templates (only used by
Roger Leigh (rleigh@apache.org)
build: Distribute version.rc templates
Roger Leigh (rleigh@apache.org)
Add warning to getTextContent doc.
Scott Cantor (scantor@apache.org)
Switch specfile to bz2 source.
Scott Cantor (scantor@apache.org)
build: Remove unused version.incl
Roger Leigh (rleigh@apache.org)
Update version numbers and release note links for
Roger Leigh (rleigh@apache.org)
ci: Drop appveyor caches permanently
Roger Leigh (rleigh@apache.org)
cmake: XercesIntTypes: Quote unsigned long corre
Roger Leigh (rleigh@apache.org)
ci: Restore appveyor caches

cmake: Allow thread checks to fail and fall back to nothreads Closes: #2155 git-svn-id: https://svn.apache.org/repos/asf/xerces/c/trunk@1843653 13f79535-47bb-0310-9956-ffa450edef68

Roger Leigh (rleigh@apache.org) committed on on 2018. 10. 12 1 parent e3486979 commit c9bfe786

cmake/XercesMutexMgrSelection.cmake

```
@@ -19,11 +19,15 @@

# mutexmgr selection

option(threads "Threading support" ON)
set(xerces_thread_default ON)
find_package(Threads)
if(NOT TARGET Threads::Threads)
  set(xerces_thread_default ON)
endif()

include(CheckCXXSourceCompiles)
option(threads "Threading support" ${xerces_thread_default})

find_package(Threads REQUIRED)
include(CheckCXXSourceCompiles)
```

Fig. 5: Git branch view

the exact date and the message of the comment. Here we still can inspect the committed code. We see this on Figure 1.

CodeCompass can also show Git commits in a filterable list ordered by the time of commit. This search facility can be used to list changes made by a person or to filter commits by relevant words in the commit message.

Many times when we are reading the source code and find a fragment which is interesting in some ways or seems suspicious, we would also like to find what other parts of the current file have been modified at the same time. These colors are thus visual aid for determining which modifications belong to the introduction of the same new feature or bugfix.

Usually it is a project level decision whether the explanatory comments about the reasons of a modification should be incorporated in the source code or should be described in the commit message. The advantage of writing these comments in the source code is that this way these comments will be version controlled and also inseparable from the code. However, some information belong to the commit message of the modification (e.g. the issue which this commit solves, some links to external pages or earlier commits, etc.). In CodeCompass we would like to present these information to the user too, immediately at the currently displayed file and line.

In case of a software project the evolution of the program may also carry useful information. The organization of the development process is also up to the project members. One of the most common structures is to maintain a *master* branch which always contains the latest version of the project. When a new feature is created or a bugfix is being introduced then these are developed on a separate, so called *feature branch*. This branch contains one or more commits which make up the change together. When the introduced feature is stable on the feature branch then the changes are merged to the master. The bigger a new feature is, the more suitable it is to separate it on consecutive commits.

CodeCompass implements **branch view** that presents which commits have been developed on a separate branch and thus belong together. On Figure 5 we see a typical branch view.

All of the visualizations are based on similar graphical appearance of existing revision control tools, to minimize the cognitive effort for the developers when using revision control related information in CodeCompass.

Such use of revision control information of large legacy projects can reveal hidden connections in large software systems and help the complete comprehension of these projects.

V. Conclusion

Code comprehension is an important research area to support better understanding of large industrial software systems to reduce otherwise high maintenance costs. As most of the maintenance activities, like bug fixing, feature extension or modification require the full understanding of the system, comprehension is an expensive process requiring expert knowledge both from the domain and the implementation side. Tools supporting comprehension are mainly based on the static analysis of the source code. The structure of the system: class relationships, function call chains, and similar features can be determined with proper accuracy in this way.

However, software systems may contain hidden relationships between their components. Connections between configuration files and their application in the code, dynamic library usage and similar are hardly can be detected by static analysis. In the same time, those connections are likely reflected by the software development process, which can be retrieved from the version control information.

CodeCompass is an open source code comprehension framework which is intended to collect the whole information portfolio of the system under investigation. This includes not only the internal structure revealed from the source code, but also additional information, like the build and the git version control information.

The blame view shows the last committers line by line for the source lines visually expressing the "age" of the code. From here, the developer easily can navigate to the commit information, to check the commit message and the other files affected by the commit. One can also compare the code of different comments. Finally, we can inspect the development of the project by the traditional branch view. The applied visualizations inherit the graphical interface of the usual version control tools to make them familiar for the developers.

All these possibilities make the comprehension more complete thus in the further development process help to increase the code quality. The full implementation is available as open source at [22].

ACKNOWLEDGMENT

This work is supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

REFERENCES

[1] T. Brunner, "Codecompass: an extensible code comprehhension framework," Eötvös Loránd University, Faculty of Informatics, Budapest, Tech. Rep. IK-TR1, May 2018.

[2] C. Szabó. (2018) Programme of the winter school of project no.2017-1-sk01-ka203-035402: "focusing education on composability, comprehensibility and correctness of working software". TUKE Kosice. Accessed 02-July-2019. [Online]. Available: https://kpi.fei.tuke.sk/sites/www2.kpi.fei.tuke.sk/files/personal/programme\\\_of_the_first_intensive_programme_for_higher_education_learners_in\\\_the_frame_of_the_project.pdf

[3] R. Szalay, Z. Porkoláb, and D. Krupp, "Towards better symbol resolution for C/C++ programs: A cluster-based solution," in *IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2017, pp. 101–110.

[4] ——, "Measuring mangled name ambiguity in large c/c++ projects," in *Zoran Budimac (Ed), Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications. Belgade, Serbia 2017.*, 2017. [Online]. Available: http://ceur-ws.org/Vol-1938/paper-sza.pdf

[5] Z. Porkoláb, T. Brunner, D. Krupp, and M. Csordás, "Codecompass: An open software comprehension framework for industrial usage," in *Proceedings of the 26th Conference on Program Comprehension*, ser. ICPC '18. New York, NY, USA: ACM, 2018, pp. 361–369. [Online]. Available: http://doi.acm.org/10.1145/3196321.3197546

[6] The LLVM Compiler Infrastructure. https://llvm.org/. [Online]. Available: https://llvm.org/

[7] Clang: a C language family frontend for LLVM. https://clang.llvm.org/. [Online]. Available: https://clang.llvm.org/

[8] Code Browser by Woboq for C and C++. https://woboq.com/codebrowser.html. [Online]. Available: https://woboq.com/codebrowser.html

[9] OpenGrok. https://opengrok.github.io/OpenGrok. [Online]. Available: https://opengrok.github.io/OpenGrok

[10] Exuberant CTAGS. http://ctags.sourceforge.net. [Online]. Available: http://ctags.sourceforge.net

[11] SciTools: Understand. https://scitools.com. [Online]. Available: https://scitools.com

[12] B. Henderson-Sellers, *Object-oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[13] McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308–320, 1976.

[14] CodeSurfer. https://www.grammatech.com/products/codesurfer. [Online]. Available: https://www.grammatech.com/products/codesurfer

[15] Apache NetBeans. https://netbeans.org. [Online]. Available: https://netbeans.org

[16] Eclipse. https://www.eclipse.org/ide/. [Online]. Available: https://www.eclipse.org/ide/

[17] Qt Creator. https://www.qt.io/. [Online]. Available: https://www.qt.io/

[18] Visual Studio. https://visualstudio.microsoft.com. [Online]. Available: https://visualstudio.microsoft.com

[19] Apache Thrift. https://Thrift.apache.org. [Online]. Available: https://Thrift.apache.org

[20] Apache Lucene. https://lucene.apache.org/core/. [Online]. Available: https://lucene.apache.org/core/

[21] Eclipse Java development tools (JDT). http://www.eclipse.org/jdt. [Online]. Available: http://www.eclipse.org/jdt

[22] CodeCompass website. https://github.com/Ericsson/CodeCompass. [Online]. Available: https://github.com/Ericsson/CodeCompasss

The URLs were retrieved on 8 September 2019 if other date is not given.