ConceptCloud: A Tagcloud Browser for Software Archives

Gillian J. Greene and Bernd Fischer
Computer Science Division, University of Stellenbosch, South Africa
ggreene@cs.sun.ac.za, bfischer@cs.sun.ac.za

ABSTRACT

ConceptCloud is an interactive browser for SVN and Git repositories. Its main novelty is the combination of an intuitive tag cloud interface with an underlying concept lattice that provides a formal structure for navigation. This combination allows users to explore repositories serendipitously, without predefined search goals and along different navigation paths. ConceptCloud can derive different lattice types for a repository and supports concurrent navigation in multiple linked tag clouds that can each be individually customized, which allows multi-faceted repository explorations.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*abstracting methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering, selection process*

Keywords

Browsing, Formal concept analysis, Tag clouds, Software repositories

1. OVERVIEW

Software archives such as SVN or Git repositories contain a wealth of implicit information. The ConceptCloud browser makes this information accessible to users. It allows them to explore a repository's meta-data and answer a variety of questions, e.g., "How have the active developers changed over time?", "Which topics has this developer been working on?", or "Which methods are often changed together?". Once users find an interesting aspect ConceptCloud allows them to drill down and investigate this further.

ConceptCloud uses a novel combination of informal tag clouds, which provide an intuitive user interface, and formal concept lattices [13, 6], which serve as underlying navigation structure. More specifically, it constructs a formal context from the meta-data that it extracts from the repository, and incrementally computes a concept lattice from this. It supports different types of formal contexts (i.e., revision-, file-and change-based), which enables different types of repository analyses. Users navigate through the underlying concept lattice by selecting and deselecting tags in the clouds; each navigation step updates the displayed cloud. ConceptCloud derives the tag clouds from the concepts corresponding to the selected tags, more precisely, from the attributes of all objects in these concepts' extents, together with the objects themselves. Each tag's weight is given by the number of objects that exhibit the respective attribute. Tags are colored according to the category of information that they represent (e.g. commit message, filename etc.).

Figure 1 shows ConceptCloud's main interface with a tag cloud derived for the JUnit project. The window shows on the left the list of selected tags ("merge" and "request", also shown in red in the cloud, c.f 1(i)) and the implied tags ("pull" and "from", c.f 1(ii)). The tag clouds can be customized in several ways. The slider on the left changes the minimum and maximum font sizes used in the clouds (c.f 1(iii)). The buttons in the top left corner control which tags are shown (c.f 1(iv)). For example, the view can be restricted to certain categories (here directory and weekday tags are not shown), to a given number of tags (here 200), or to tags with a minimum number of occurrences. By default it shows a single tag cloud to give a unified view; however, users can create multiple views that can be customized independently; Figure 3 shows the main interface with three additional views that display only file names, dates, and authors, respectively. The views are linked, i.e., a tag selection or de-selection in one view also updates all others, providing a multidimensional view similar to Crossfilter [1]. Users can also create views with "sticky" tags that cannot be de-selected (cf. Figure 2, sticky tags shown in red at the top of the panels).

ConceptCloud is implemented as a web-based tool (available at `www.conceptcloud.org`) but it can also be run locally. It can process local and remote Git and SVN repositories.

2. TOOL APPLICATIONS

ConceptCloud is not restricted to a fixed set of repository analyses. Instead, it provides a uniform framework that allows users i.e, developers and project managers, to analyze and display a repository's meta-data in different ways. In the following we sketch a few indicative applications.

Identifying Active Developers. We consider developers as active in a given time period if they make at least one commit in that period. With a revision-based context, ConceptCloud can then easily be used to find the most active developers, because their tags will be displayed biggest. It is also easy to break this down into different time periods, at different granularities (year, month or day). For example, to create a view of developers by year (as shown for the JUnit project in Figure 2), we only need to right click the tags of the desired years to create additional views where the respective years are sticky, and customize these views to show only authors

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2661676

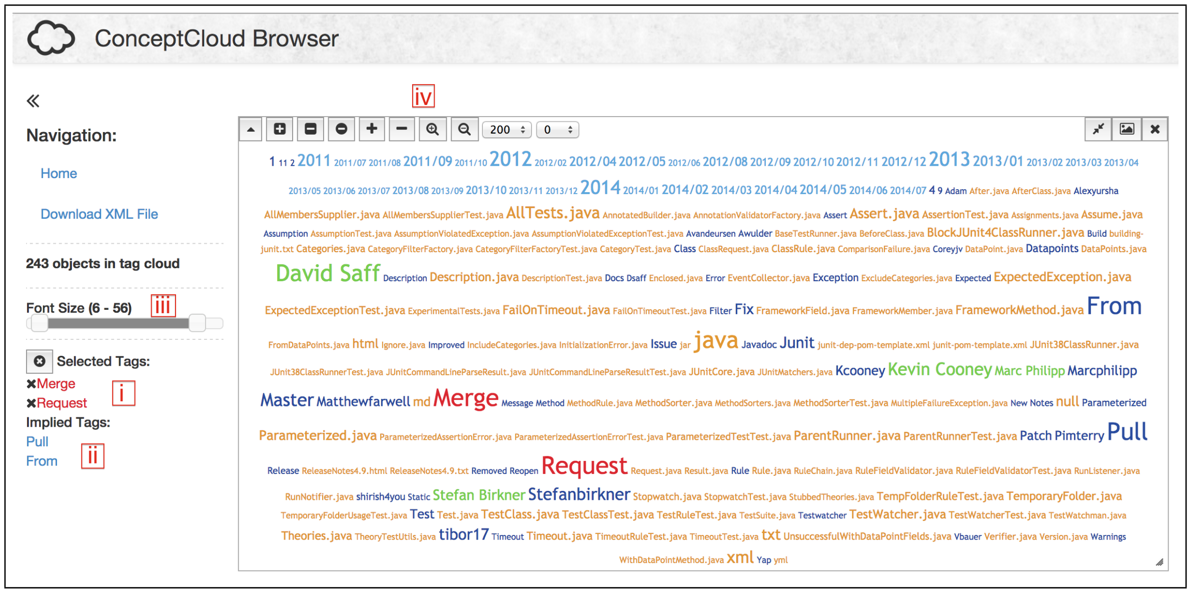
ConceptCloud Browser
«
Navigation:
Home
Download XML File
243 objects in tag cloud
Font Size (6 - 56)
iii
Selected Tags:
×Merge
×Request
i
Implied Tags:
Pull
From
ii
iv
200
0
1 11 2 2011 2011/07 2011/08 2011/09 2011/10 2012 2012/02 2012/04 2012/05 2012/06 2012/08 2012/09 2012/10 2012/11 2012/12 2013 2013/01 2013/02 2013/03 2013/04
2013/05 2013/06 2013/07 2013/08 2013/09 2013/10 2013/11 2013/12 2014 2014/01 2014/02 2014/03 2014/04 2014/05 2014/06 2014/07 4 9 Adam After.java AfterClass.java Alexyursha
AllMembersSupplier.java AllMembersSupplierTest.java AllTests.java AnnotatedBuilder.java AnnotationValidatorFactory.java Assert Assert.java AssertionTest.java Assignments.java Assume.java
Assumption AssumptionTest.java AssumptionViolatedException.java AssumptionViolatedExceptionTest.java Avandeursen Awulder BaseTestRunner.java BeforeClass.java BlockJUnit4ClassRunner.java Build building-junit.txt Categories.java CategoryFilterFactory.java CategoryFilterFactoryTest.java CategoryTest.java Class ClassRequest.java ClassRule.java ComparisonFailure.java Coreyjv DataPoint.java Datapoints DataPoints.java
David Saff Description Description.java DescriptionTest.java Docs Dsaff Enclosed.java Error EventCollector.java Exception ExcludeCategories.java Expected ExpectedException.java
ExpectedExceptionTest.java ExperimentalTests.java FailOnTimeout.java FailOnTimeoutTest.java Filter Fix FrameworkField.java FrameworkMember.java FrameworkMethod.java From
FromDataPoints.java html Ignore.java Improved IncludeCategories.java InitializationError.java Issue jar java Javadoc Junit junit-dep-pom-template.xml junit-pom-template.xml JUnit38ClassRunner.java
JUnit38ClassRunnerTest.java JUnitCommandLineParseResult.java JUnitCommandLineParseResultTest.java JUnitCore.java JUnitMatchers.java Kcooney Kevin Cooney Marc Philipp Marcphilipp
Master Matthewfarwell md Merge Message Method MethodRule.java MethodSorter.java MethodSorters.java MethodSorterTest.java MultipleFailureException.java New Notes null Parameterized
Parameterized.java ParameterizedAssertionError.java ParameterizedAssertionErrorTest.java ParameterizedTestTest.java ParentRunner.java ParentRunnerTest.java Patch Pimterry Pull
Release ReleaseNotes4.9.html ReleaseNotes4.9.txt Removed Reopen Request Request.java Result.java Rule Rule.java RuleChain.java RuleFieldValidator.java RuleFieldValidatorTest.java RunListener.java
RunNotifier.java shirish4you Static Stefan Birkner Stefanbirkner Stopwatch.java StopwatchTest.java StubbedTheories.java TempFolderRuleTest.java TemporaryFolder.java
TemporaryFolderUsageTest.java Test Test.java TestClass.java TestClassTest.java TestRuleTest.java TestSuite.java Testwatcher TestWatcher.java TestWatcherTest.java TestWatchman.java
Theories.java TheoryTestUtils.java tibor17 Timeout Timeout.java TimeoutRuleTest.java TimeoutTest.java txt UnsuccessfulWithDataPointFields.java Vbauer Verifier.java Version.java Warnings
WithDataPointMethod.java xml Yap yml

Figure 1: ConceptCloud main interface, Top 200 view of JUnit (revision-based context, directories and weekdays not shown)

tags. In the perspective revealed in Figure 2 we can see how the team evolved from a single developer in 2000, and how the combinations of different developers and the number of changes they are making of the project differ substantially from year to year.

Identifying Expertise. In a revision-based context, selecting tags from commit messages reveals all other information from commits that contain the selected words in their messages. This allows us to identify developers who are experts in an area, by virtue of frequently using certain words in their commit messages. For example, the main view on the left of Figure 3 shows the tag cloud from the iText project after selection of "findbugs" (c.f 3(i)); category-specific views, such as the file (c.f 3(ii)), date (c.f 3(iii))and author (c.f 3(iv)) views shown on the right, further elaborate different dimensions of the topic. Hence, we can easily see all developers working on the topic "findbugs" and all files that have been changed when this commit message was used.

Identifying Collaboration. If we assume that two developers collaborate by committing the same file in different revisions, then we can see the collaboration between one given developer and all others simply by opening the tag cloud from the file-based context, and selecting the developer; the tag sizes of the other developers then represent the number of files that both developers have worked on and thus the strength of the collaboration. If we select a group of developers, we see which other developers work with this core, and what files they collaborate on (see Figure 4). We can also select a file to identify which developers have collaborated on this file and what else they have collaborated on (see Figure 5). Note how this gives a different view on the same underlying meta-data.

Identifying Co-changed Methods. A change-based context from a repository that contains Java files will produce a tag cloud that also contains method signatures. Selecting one of the method signatures in the cloud will then give us a view of all commits in which that method has been changed. All the other large method tags in that cloud are the frequently co-changed methods. On selection of method "allowText(TextRenderInfo)" from the JUnit project we see a view that contains all developers that have changed that method, all files in which the method has been located and all other methods which have been changed at the same time.

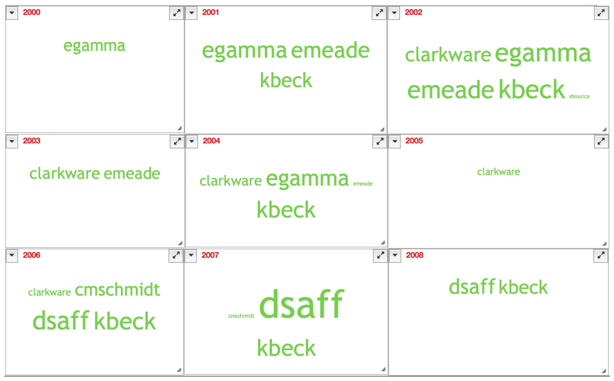

Figure 2: Author views of JUnit, restricted to different years.

3. APPROACH

ConceptCloud uses concept lattices as an underlying navigation structure and tag clouds to present the information that is available in the lattice. We briefly sketch how we construct the concept lattices from the repository, how we derive a tag cloud from the concepts in the lattice, and how we drive the navigation in the concept lattice via the tag cloud.

Formal Concept Analysis. Formal concept analysis [13, 6] applies lattice-theoretic methods to investigate abstract relations between objects and their attributes. Such *contexts* can be imagined as cross tables where the rows are objects and the columns are attributes. Concepts are pairs of objects and attributes which are synonymous and so characterize each other. They are maximal rectangles (modulo permutation of rows and columns) in the context table. The objects (attributes) of a concept are called its *extent* (*intent*). Concepts are partially ordered by inclusion of extents such that a concept's extent includes the extent of all of its subconcepts; a similar order on intents follows by duality.

The basic theorem of formal concept analysis states that the structure induced by the concepts of a formal context and their ordering is always a complete lattice. Such *concept lattices* have strong mathematical properties and reveal hidden structural and hierarchi-

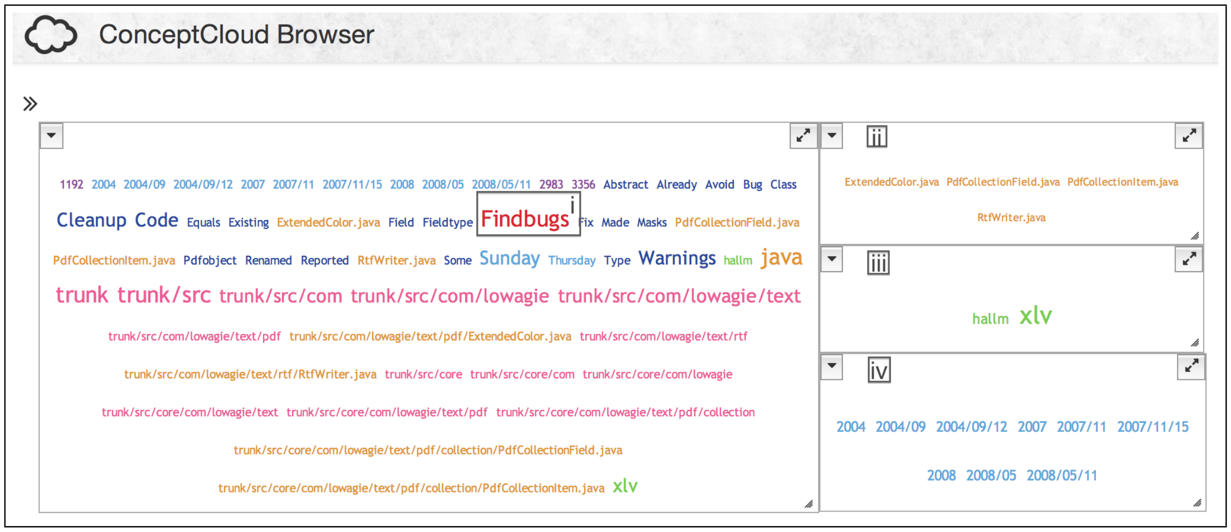
ConceptCloud Browser
1192 2004 2004/09 2004/09/12 2007 2007/11 2007/11/15 2008 2008/05 2008/05/11 2983 3356 Abstract Already Avoid Bug Class
Cleanup Code Equals Existing ExtendedColor.java Field Fieldtype Findbugs Fix Made Masks PdfCollectionField.java
PdfCollectionItem.java Pdfobject Renamed Reported RtfWriter.java Some Sunday Thursday Type Warnings hallm java
trunk trunk/src trunk/src/com trunk/src/com/lowagie trunk/src/com/lowagie/text
trunk/src/com/lowagie/text/pdf trunk/src/com/lowagie/text/pdf/ExtendedColor.java trunk/src/com/lowagie/text/rtf
trunk/src/com/lowagie/text/rtf/RtfWriter.java trunk/src/core trunk/src/core/com trunk/src/core/com/lowagie
trunk/src/core/com/lowagie/text trunk/src/core/com/lowagie/text/pdf trunk/src/core/com/lowagie/text/pdf/collection
trunk/src/core/com/lowagie/text/pdf/collection/PdfCollectionField.java
trunk/src/core/com/lowagie/text/pdf/collection/PdfCollectionItem.java xlv
ii
ExtendedColor.java PdfCollectionField.java PdfCollectionItem.java
RtfWriter.java
iii
hallm xlv
iv
2004 2004/09 2004/09/12 2007 2007/11 2007/11/15
2008 2008/05 2008/05/11

Figure 3: Main view, and category-specific file, date and author viewers after selection of "findbugs".

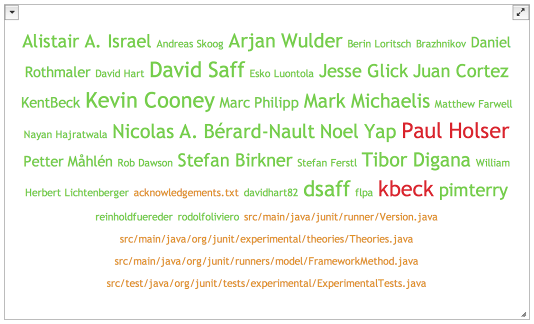
Alistair A. Israel Andreas Skoog Arjan Wulder Berin Loritsch Brazhnikov Daniel
Rothmaler David Hart David Saff Esko Luontola Jesse Glick Juan Cortez
KentBeck Kevin Cooney Marc Philipp Mark Michaelis Matthew Farwell
Nayan Hajratwala Nicolas A. Bérard-Nault Noel Yap Paul Holser
Petter Måhlén Rob Dawson Stefan Birkner Stefan Ferstl Tibor Digana William
Herbert Lichtenberger acknowledgements.txt davidhart82 dsaff flpa kbeck pimterry
reinholdfuereder rodolfoliviero src/main/java/junit/runner/Version.java
src/main/java/org/junit/experimental/theories/Theories.java
src/main/java/org/junit/runners/model/FrameworkMethod.java
src/test/java/org/junit/tests/experimental/ExperimentalTests.java

Figure 4: Author collaboration in JUnit.

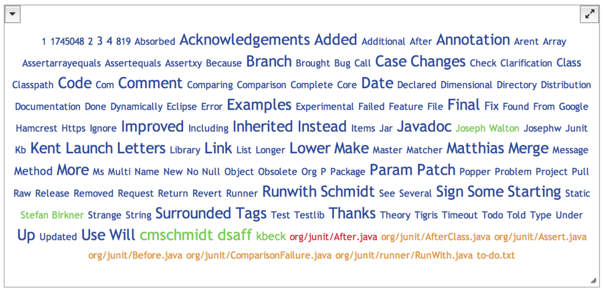
1 1745048 2 3 4 819 Absorbed Acknowledgements Added Additional After Annotation Arent Array
Assertarrayequals Assertequals Assertxy Because Branch Brought Bug Call Case Changes Check Clarification Class
Classpath Code Com Comment Comparing Comparison Complete Core Date Declared Dimensional Directory Distribution
Documentation Done Dynamically Eclipse Error Examples Experimental Failed Feature File Final Fix Found From Google
Hamcrest Https Ignore Improved Including Inherited Instead Items Jar Javadoc Joseph Walton Josephw Junit
Kb Kent Launch Letters Library Link List Longer Lower Make Master Matcher Matthias Merge Message
Method More Ms Multi Name New No Null Object Obsolete Org P Package Param Patch Popper Problem Project Pull
Raw Release Removed Request Return Revert Runner Runwith Schmidt See Several Sign Some Starting Static
Stefan Birkner Strange String Surrounded Tags Test Testlib Thanks Theory Tigris Timeout Todo Told Type Under
Up Updated Use Will cmschmidt dsaff kbeck org/junit/After.java org/junit/AfterClass.java org/junit/Assert.java
org/junit/Before.java org/junit/ComparisonFailure.java org/junit/runner/RunWith.java to-do.txt

Figure 5: Author collaboration on a specific file (After.java).

cal properties of the original relation. They can be computed automatically from any given relation between objects and attributes. The greatest lower bound or *meet* and least upper bound or *join* can also be expressed by the common attributes and objects.

Contexts from Repositories. ConceptCloud can construct different types of formal contexts, which enables different types of repository analyses. In a *revision-based context* we interpret the revisions as objects and the meta-data as attributes: each revision is associated with its own meta-data (e.g., author, words from the log message, or changed files) as attribute. Since the revisions reflect the project's development over time, such contexts can give us a historical overview of the project. In a *file-based context* we interpret the files as objects but derive the attributes from the revisions' meta-data: each file receives all attributes from all revisions

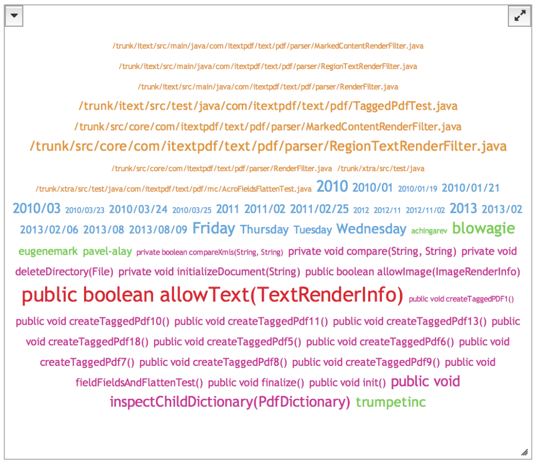
/trunk/itext/src/main/java/com/itextpdf/text/pdf/parser/MarkedContentRenderFilter.java
/trunk/itext/src/main/java/com/itextpdf/text/pdf/parser/RegionTextRenderFilter.java
/trunk/itext/src/main/java/com/itextpdf/text/pdf/parser/RenderFilter.java
/trunk/itext/src/test/java/com/itextpdf/text/pdf/TaggedPdfTest.java
/trunk/src/core/com/itextpdf/text/pdf/parser/MarkedContentRenderFilter.java
/trunk/src/core/com/itextpdf/text/pdf/parser/RegionTextRenderFilter.java
/trunk/src/core/com/itextpdf/text/pdf/parser/RenderFilter.java /trunk/xtra/src/test/java
/trunk/xtra/src/test/java/com/itextpdf/text/pdf/mc/AcroFieldsFlattenTest.java 2010 2010/01 2010/01/19 2010/01/21
2010/03 2010/03/23 2010/03/24 2010/03/25 2011 2011/02 2011/02/25 2012 2012/11 2012/11/02 2013 2013/02
2013/02/06 2013/08 2013/08/09 Friday Thursday Tuesday Wednesday achingarev blowagie
eugenemark pavel-alay private boolean compareXmls(String, String) private void compare(String, String) private void
deleteDirectory(File) private void initializeDocument(String) public boolean allowImage(ImageRenderInfo)
public boolean allowText(TextRenderInfo) public void createTaggedPDF1()
public void createTaggedPdf10() public void createTaggedPdf11() public void createTaggedPdf13() public
void createTaggedPdf18() public void createTaggedPdf5() public void createTaggedPdf6() public void
createTaggedPdf7() public void createTaggedPdf8() public void createTaggedPdf9() public void
fieldFieldsAndFlattenTest() public void finalize() public void init() public void
inspectChildDictionary(PdfDictionary) trumpetinc

Figure 6: Method selection with change-based context.

that involve the file. Such contexts are useful for collaboration- or localization based analysis. In a *change-based context* we use pairs of files and revisions as objects, and each revision's changes to the corresponding file as (additional) attributes. Such contexts allow us to analyze the different individual changes to files that the different revisions introduce.

Tag Clouds from Concepts. For the tag cloud interface we construct a tag cloud from a set of concepts. For this we collect all attributes of the defining concept of each object in the extent of the focus concept; we also add the object itself, to allow navigation via both attributes and objects. Note that each object is used only once, even if it occurs in the extent of several concepts.

Navigating Concept Lattices with Tag Clouds. Our navigation algorithm is refinement-based and the browser maintains a focus concept, from which it renders the tag cloud as described above; when the user selects (or deselects) a tag, the browser updates the focus and re-renders the tag cloud. When the user selects a tag, the browser updates the focus by computing the meet of that tag's defining concept and the old focus. After a tag deselection, the browser re-computes the focus as the meet of the defining concepts of the remaining selected tags.

Table 1: ConceptCloud results for open-source repositories.

Project	Rev.	Type	Indexing time (s)	Drawing time (s)	Tags
JUnit	1870	Git	37.5	1.0	7805
IText	5580	SVN	69.3	3.5	30824
JQuery	5618	Git	151.2	2.2	14048
JEdit	5765	SVN	232.4	6.2	18713
Bootstrap	9466	Git	441.1	4.2	19749
Valgrind	10716	SVN	102.8	8.4	39232
Django	18139	Git	353.0	50.9	56646
TortoiseSVN	22558	SVN	158.9	18.0	58389

4. IMPLEMENTATION

ConceptCloud is implemented in Java and uses the Play Web MVC Framework to create a web interface. ConceptCloud's main components are metadata extraction from the archive, concept lattice construction and tag cloud display from the resulting lattice.

Meta-data Extraction. ConceptCloud uses the SVNKit library to read log information from SVN repositories and the JGit library to clone a temporary copy and read from a Git repository. ConceptCloud performs basic lexical processing such as stop word removal and stemming on the free text fields that it extracts. The Apache Lucene Porter Stemmer implementation is used to handle the stemming. Since root words are not usually dictionary words we use the most common word that evaluates to a root word as the word's representation in the tag cloud.

Concept Lattice Construction. For the lattice construction, we use a method based on the Colibri Java library [9] which constructs concepts on the fly, so that we never need to compute the full lattice and are able to render an initial tag cloud quickly.

Tag Cloud Display. The interface presents the tag cloud in a viewer which can dynamically include and exclude certain categories of tags (e.g. message, filename for a repository) and multiple viewers can be displayed at the same time. Each tag is assigned a color according to the category of information that it represents (e.g. all filenames displayed in orange). In future we could also add links to tags that allow the opening of the original file source or the developer's GitHub page.

Evaluation. Table 1 shows details of applying ConceptCloud to several repositories. The context construction takes less than a few minutes on a standard laptop, even for larger repositories; note that Git repositories are indexed locally and need to be cloned and all times shown include network latency. Drawing the full tag cloud (which is shown on start-up) typically takes a few seconds, with slow-downs for larger clouds. However, cloud sizes quickly drop during navigation, and refined clouds are rendered with no significant delays. Moreover, we use caching to further improve the tool's reactivity, so that for example returning to the full cloud causes no significant delays.

5. RELATED WORK

Formal Concept Analysis. Godin et al. [8] introduced the use of concept lattices in information retrieval, including the construction of contexts over documents and keywords and the lattice-based navigation which ConceptCloud uses, but use manual keyword assignments. Carpineto and Romano [3] demonstrated how concept lattices can be built automatically from unstructured text documents.

Concept lattices have also been applied to software development repositories, in particular component libraries, but mostly as retrieval tools [10, 11]. Browsing was introduced by our own prior work [5], but that did not use tag clouds for navigation.

Mining Software Repositories. This field has produced many techniques and tools to solve specific problems such as the visualization of version histories. Codebook [2] is a social network inspired toolset to analyze information implicitly contained in software repositories. However unlike ConceptCloud, Codebook requires explicit regular expression queries to extract information, and does not directly support browsing. Hipikat [4] is another retrieval tool which also monitors multiple information sources (Bugzilla, CVS, email, newsgroups) and builds a uniform artifact database. Poshyvanyk and Marcus [12] use a combination of latent semantic indexing and concept lattices to find methods that are relevant to a bug report. Girba et al. [7] use concept analysis to detect co-change patterns in revision control systems.

6. CONCLUSIONS

Software archives contain a large amount of information about software projects and their development process. This information is however difficult to extract and there are a number of dedicated tools that are able to report on specific aspects of an archive. ConceptCloud presents software archive information in an intuitive browsable format making the information easily accessible to users and enabling them to investigate multiple aspects of the software projects. ConceptCloud is an easily accessible, easy to use and versatile tool for browsing software development archives.

7. REFERENCES

[1] http://square.github.io/crossfilter/.

[2] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. *ICSE*, pp. 125–134, 2010.

[3] C. Carpineto and G. Romano. Automatic construction of navigable concept networks characterizing text databases. *Topics in Artificial Intelligence*, LNCS 992, pp. 67–78, 1995.

[4] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Software Eng.*, 31(6):446–465, 2005.

[5] B. Fischer. Specification-based browsing of software component libraries. *Autom. Softw. Eng.*, 7(2):179–200, 2000.

[6] B. Ganter and R. Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[7] T. Gîrba, S. Ducasse, A. Kuhn, R. Marinescu, and R. Daniel. Using concept analysis to detect co-change patterns. *IWPSE*, pp. 83–89, 2007.

[8] R. Godin, E. Saunders, and J. Gecsei. Lattice model of browsable data spaces. *Information Sciences*, 40(2):89–116, 1986.

[9] D. N. Götzmann. Colibri/java, 2007. http://code.google.com/p/colibri-java/.

[10] C. Lindig. Concept-based component retrieval. *Working Notes of the IJCAI-95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 1995.

[11] Y. Park. Software retrieval by samples using concept analysis. *J. Systems and Software*, 54(3):179 – 183, 2000.

[12] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. *ICPC*, pp. 37–48, 2007.

[13] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival (ed.), *Ordered sets*, pp. 445–470. Reidel, 1982.