Information and Software Technology 000 (2016) 1–19

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Visualizing and exploring software version control repositories using interactive tag clouds over formal concept lattices

Gillian J. Greene*, Marvin Esterhuizen, Bernd Fischer

CAIR, CSIR Meraka, Computer Science Division, Stellenbosch University, Stellenbosch, South Africa

ARTICLE INFO

Article history:
Received 29 January 2016
Revised 2 December 2016
Accepted 5 December 2016
Available online xxx

Keywords:
Formal concept analysis
Tag clouds
Browsing software repositories
Interactive tag cloud visualization

ABSTRACT

Context: version control repositories contain a wealth of implicit information that can be used to answer many questions about a project's development process. However, this information is not directly accessible in the repositories and must be extracted and visualized.

Objective: the main objective of this work is to develop a flexible and generic interactive visualization engine called ConceptCloud that supports exploratory search in version control repositories.

Method: ConceptCloud is a flexible, interactive browser for SVN and Git repositories. Its main novelty is the combination of an intuitive tag cloud visualization with an underlying concept lattice that provides a formal structure for navigation. ConceptCloud supports concurrent navigation in multiple linked but individually customizable tag clouds, which allows for multi-faceted repository browsing, and scriptable construction of unique visualizations.

Results: we describe the mathematical foundations and implementation of our approach and use ConceptCloud to quickly gain insight into the team structure and development process of three projects. We perform a user study to determine the usability of ConceptCloud. We show that untrained participants are able to answer historical questions about a software project better using ConceptCloud than using a linear list of commits.

Conclusion: ConceptCloud can be used to answer many difficult questions such as "What has happened in this project while I was away?" and "Which developers collaborate?". Tag clouds generated from our approach provide a visualization in which version control data can be aggregated and explored interactively.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Version control repositories contain a wealth of implicit information that can be used to answer many questions about a project's development process, such as "Who worked on these files?", "Which developers collaborate?", "What are the co-changed methods?", or "What has happened in this project while I was away?". Answering such questions is a daily task for software developers [1]. Developers also rely on examining the history of a software project to keep up with changes, understand coding decisions and debug [2]. In co-located teams new developers rely on members of the team to help them ramp-up [3] but in large open-source projects, where this is no longer possible, the repository information becomes a valuable resource for new developers as well [4].

While it is well-known that version control repositories are a valuable source of information, repository tools are not set up to provide insights into the history of a project directly and can only be used to see information about individual commits. Manually examining the last few commits to a software project in the version control repository is feasible for regular contributors of the project who are only seeking information about the most recent changes. However, manually examining the entire commit log of a repository in order to answer more complex questions becomes infeasible. Individual commits provide information about a single change to the project but only when a large number of commits is aggregated does the information become accessible to developers seeking answers to complex questions.

We develop ConceptCloud, an interactive tag cloud visualization engine for software repositories that aggregates commit data and lets users easily construct uniform visualizations of many different

* Corresponding author.
E-mail addresses: ggreene@cs.sun.ac.za (G.J. Greene), mhesterhuizen@cs.sun.ac.za (M. Esterhuizen), bfischer@cs.sun.ac.za (B. Fischer).

http://dx.doi.org/10.1016/j.infsof.2016.12.001
0950-5849/© 2016 Elsevier B.V. All rights reserved.

aspects of the project history. ConceptCloud makes use of a novel combination of tag clouds and an underlying concept lattice [5] to support *exploratory search* [6,7] tasks on software repositories. When users have no previous knowledge of a project or have not yet formulated a direct query their task becomes one of exploratory search instead of *direct search* or *retrieval* [7]. While there are already approaches supporting specific retrieval tasks and visualizing aspects of software repositories [8], support for exploratory search in software repository data remains unavailable. The goal of our work is to build a flexible and interactive visualization engine that allows users to visualize different aspects of a project interactively and therefore supports exploratory search tasks, instead of presenting the user with one static, pre-configured view.

An exploratory search approach can provide an overview of the repository data and allow the user to further investigate any aspects of the project which they might find interesting. Therefore, exploratory search approaches can support new developers on a project in understanding the project history and team structure. An exploratory approach can also be used to answer more general questions (e.g., "Which developers collaborate?") which cannot be formulated as a single search query which would be possible if the question was more focused (e.g., "Who collaborates with Alice?").

Tag clouds (or word clouds) are a simple visualization method for textual data where the frequency of each tag is reflected in its size. We use a tag cloud visualization to present aggregated software repository data, as tag clouds support exploratory search tasks and have been found to be effective when the information discovery task is wide [9]. While our tag cloud visualization may not be the optimal visualization for all aspects of the data, it is flexible enough to visualize many aspects of the software project such as developer expertise (e.g., which developers have worked on particular files or directories and would be good candidates to ask questions about this functionality), co-changed methods in a software project, project activity (e.g., in which years and months has there been a lot of development, and on which parts of the system), or developer collaboration (e.g., which developers are working together on which parts of the project) in a uniform way. Our interactive tag clouds allow developers to aggregate commits into groups and filter commits that apply to a certain topic, which has been noted by developers to be useful [2].

We generate tags directly from the data that we extract from software repositories, instead of relying on user-generated labels as tags for particular content, as often done in Web 2.0 applications (such as Flickr's early tag cloud view). The data available in a version control archive is often large (for example, more than 500,000 revisions for the Linux [10] repository) and so we allow the user to make incremental refinements (i.e., navigate) in the tag cloud in order to generate smaller, more detailed visualizations. The navigation in our tag clouds is crucial for facilitating exploratory search tasks. Navigation using tag clouds has previously been explored using a Bayesian approach [11]; however, navigation in our browser is supported by a novel combination of tag clouds and concept lattices [5,12,13].

We conjecture that a concept lattice [5] provides a high level of internal structure for the repository data and therefore allows users to explore the data through multiple navigation paths. Concept lattices have been shown to be useful for browsing data [14–16] but large lattices do not provide a suitable data visualization because the relationships between the concepts are difficult to identify in a large Hasse diagram. Therefore, we make use of a concept lattice to facilitate navigation in the more intuitive and scalable tag cloud visualization.

Fig. 1 shows an overview of our approach. We construct a formal context from data in a version control archive (see Section 4.1) and generate a concept lattice directly from the context. Note that we have used a small illustrative example as larger context tables and lattices (for example, one derived from the Linux repository) become incomprehensible. Our refinement-based navigation algorithm (see Section 3.4) then enables interactive repository browsing through our tag cloud interface (see Section 4.1). Our navigation algorithm maintains a *focus concept* in the underlying lattice which represents the user's current tag selection. We derive the tag cloud visualization from the current focus concept and update it after each navigation step. Navigation is driven by the user's selection (or de-selection) of tags in the tag cloud. Fig. 1(i) shows the initial focus concept generating the first tag cloud, after the selection of tag "Alice" the focus moves to (ii) and the tag cloud is updated.

By using different objects in the formal contexts (see Section 3.2) that are used to construct concept lattices, we are able to generate tag clouds that provide different perspectives on the same underlying data in the same familiar visualization. Our foundation in formal concept analysis allows us to change the objects easily to get different insights on the same repository.

We have implemented our approach in the ConceptCloud browser (available at www.conceptcloud.org) which includes advanced visualizations, such as multiple interlinked tag clouds. Section 5 shows the application of ConceptCloud to three different repositories.

In this paper, we extend our previous work [17] by providing a formalization for our formal context construction from repositories (see Section 2), combining multiple archives (such as issue databases and version control repositories) in the same context in order to support data fusion (see Section 3.2.5) and developing a browser scripting language for ConceptCloud to support advanced customizations (see Section 4.2.4). We have also conducted additional evaluation in the form of a user study (see Section 7).

2. Modeling software repositories

We use a simple repository model derived from Hindle and Germán's SCQL [18] to formalize how we construct the contexts that underpin our browser: a *repository* is simply a collection of *versions* of a set of *files* that are grouped into *revisions*. Note that we follow the SVN terminology [19] here. Hindle and Germán [18] refer to versions as revisions, while revisions are called modification requests; elsewhere revisions are called transactions.

A *version* $v \in \mathcal{V}$ denotes the abstract state of a *file* $f \in \mathcal{F}$ created by an *author* $a \in \mathcal{A}$ at a *time* $t \in \mathcal{T}$. We ignore the actual file contents and only use meta-data and abstract modifications. Versions constitute a *version history* if they are ordered by a precedence relation $\prec$ that holds only between versions of the same file and is compatible with the file creation times. We say that *vevolves into* v' if $v \prec v'$ holds; two versions v_1 and v_2 are *merged* into v if $v_1 \prec v$ and $v_2 \prec v$.

Definition 1. Let $\mathcal{V} \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{A}$ be a set of *versions* over files $\mathcal{F}$ and $\prec \subseteq \mathcal{V} \times \mathcal{V}$ be an irreflexive partial order. $(\mathcal{V}, \prec)$ is called a *version history* iff $v = (f, t, a) \in \mathcal{V}$, $v' = (f', t', a') \in \mathcal{V}$, and $v \prec v'$ imply $f = f'$ and $t < t'$.

A *revision* r is a set V of file versions that are committed to the *repository* $\mathcal{R}$ at time t by an author a; on commit, some meta-data (i.e., author, time, and an additional *log message* $l \in \mathcal{L}$) is stored together with the versions. We assume that each revision $r \in \mathcal{R}$ contains only one version of a file (which need not be the most recent version), and that each revision is uniquely determined by an abstract identifier $id(r)$.

Definition 2. Let $(\mathcal{V}, \prec)$ be a version history and $\mathcal{R} \subseteq \mathbb{P}(\mathcal{V}) \times \mathcal{T} \times \mathcal{A} \times \mathcal{L}$ be a set of *revisions*. $\mathcal{R}$ is called a *repository* iff $r = (V, t_r, a_r, l) \in \mathcal{R}$ and $v = (f, t_v, a_v) \in V$ imply $t_v \leq t_r$ and $v \not\prec v'$ for all $v' \in V$.

Download English Version:

https://daneshyari.com/en/article/4972260

Download Persian Version:

https://daneshyari.com/article/4972260

Daneshyari.com