Database Techniques for the Analysis and Exploration of Software Repositories

Omar Alonso[1], Premkumar T. Devanbu, Michael Gertz
Department of Computer Science
University of California at Davis
oralonso@ucdavis.edu, devanbu@cs.ucdavis.edu, gertz@cs.ucdavis.edu

Abstract

In a typical software engineering project, there is a large and diverse body of documents that a development team produces, including requirement documents, specifications, designs, code, and bug reports. Documents typically have different formats and are managed in several repositories. The heterogeneity among document formats and the diversity of repositories make it often not feasible to query and explore the repositories in an integrated and transparent fashion during the different phases of the software development process.

In this paper, we present a framework for the analysis and exploration of software repositories. Our approach applies database techniques to integrate and manage different documents produced by a team. Tools that exploit the database functionality then allow for the processing of complex queries against a document collection to extract trends and analyze correlations, which provide important insights into the software development and maintenance process.

We present a prototype implementation using the Apache Web-server project as a case study.

1. Introduction

In a typical software engineering project, there is a large and diverse body of assets, usually documents that the development team produces over time. Those assets include requirement documents, specifications, design charts, code, bug reports, user manuals, etc. Different people create and update those heterogeneous types of documents that typically reside in different repositories. At different phases in a software development process, it would be useful to pose queries against these repositories. For example, one might ask about the status of a certain component, whether specified requirements have been met, or the amount of time that it took to fix a bug. The diversity of the data models and repositories from which these answers must be drawn complicates the task of processing such queries. This naturally augments the learning curve for new members of the team. It also makes it difficult to derive metrics and get insight into the software process.

In this paper, we present a framework for the analysis and exploration of software repositories. Our approach provides database techniques for the integration, processing, and management of different types of documents produced by the team over time.

Data integration enables us to manipulate different data sources within a single conceptual framework. An off-the-shelf query language and associated optimization and evaluation tools let us process the data to answer complex queries, even analytical questions to extract trends and correlations. The manageability of data is inherent to the use of a full-fledged modern database system.

Our contribution is a framework for incremental analysis and exploration of different data sources. Each data source is unique in certain ways so different techniques have to be applied to extract meaningful information. Finally, the combination of different data sources and extracted information within the framework allows the mining of the repositories.

We present a case study using the development mailing list of the Apache Web server project [1]. This is the main communication vehicle between the members of the team [2], [3]. We implemented a prototype that shows our ideas in practice using a commercial database product.

Current approaches on mining software repositories consist on ad-hoc scripts tailored to a particular data source. Those scripts manipulate the data source in the file system and produce metrics [5], [7], [12]. More recently, some research is starting to consider either database or information retrieval techniques to aid the manipulation of repositories [6], [8]. Finally, the social aspect is also an important component of the mining

[1] The author is also affiliated with Oracle Corp.

process like the identification of expert knowledge in a team [11].

2. Integration

In any particular software project the content arrives from different data sources in different formats. These sources can include project documentation using word processing template formats, web pages with miscellaneous information in different websites, email messages about project communication, reports from bug databases, etc. The content, except source code, is mainly free text with some rudimentary level of structure, if any. The idea is then to apply some structuring process to those data sources to produce more standardized data that would be easier to load and manipulate in a database. Once the content is in the database we can use all the existing techniques available to work on the sources like SQL to run queries, statistics packages to run analytics and so on.

A very arduous and time-consuming task is to extract information from a data source, modify it, and finally load it into the database. This step involves knowing the structure of the data source, the information that we want to extract, and the metadata that is possible to derive in the same process. For a web page, this implies parsing HTML, extracting the text of the document and metadata such as anchor text and other tags. In an email message, the main source of data is obviously the message but we also consider date, subject, and author to be important metadata.

Ultimately the goal is to transform the data extracted from a source into a more standard format that later will be loaded into the database using an existing tool. The final step is to generate all this information for a particular loading tool. Once this step is completed, the data is already in the database and it is possible to start the analysis phase.

In our case study, the integration is fairly straightforward and involves just one data source: email messages.

3. Analysis and Exploration

Our methodology emphasizes on databases techniques for managing the contents of all data sources. There are several reasons why we want to use databases. They provide the basic infrastructure for large existing information systems in a wide range of application domains. Today's databases are more than just tables and SQL statements. They can manage multimedia content, and they provide functionality like back up, recovery, replication, partitioning, etc. All features that are useful for managing large-scale data collections. From a data management point of view, there are a number of benefits that we would like to take advantage of:

1. Uniform way to access data storage and other language access methods (JDBC, ODBC, SQL) are standards for managing data.
2. Access to advanced data management features (full text indexing, XML etc.): there is support for new technologies like XML and specialized indexes for managing text.
3. Analytics: most databases systems have built-in APIs for performing advanced statistical analysis that can be part of OLAP and data mining applications.

We would like to process the data sources (in this particular case an email data source), extract useful information and then load the content into an appropriate schema in the database. Instead of writing scripts to extract some data, we use standard languages like SQL or XPath to issue queries against the database. In the following, we discuss some types of queries that can pose, taking advantage of the capabilities we have just described.

In the software engineering domain there is a wide range of questions that we want to ask. We can partition the set in two ways: technical and social.

In the technical aspect we would like to know if a component has been tested, if the requirements for a feature are met, or if a bug was fixed to name a few.

In the social area, as we all know, software development involves people. Apart from productivity metrics like numbers of line/year per developer, it is also interesting to know who is the main owner of a piece of the system, who is more active in different phases, etc.

For both cases, one can trace people to assets (documents) once all this information is in the database. If we want to know who was the more productive developer we can issue a query for the number of code checked in for a particular person. Similar queries are possible to get the number of bugs open/fixed, etc.

At first this looks like information that should always be available at a glance but unfortunately, due to the heterogeneity of the data sources, it is not always the case. Without a proper database, basic facts are almost impossible to retrieve and correlate in an automated way.

So far, a query language can help when one is interested in information that we know it is possible to retrieve. For example, one might be interested in the number of developers in a project or the severity of new bugs after a release date. Those queries are typically reports on the status of the development process. The

argument for using a database here is that is easy to run any type of report.

Databases can also help in the discovery process. The first step is to add information retrieval to already existing data retrieval capabilities. Data retrieval, as presented earlier, returns objects with a clear defined condition. Information retrieval involves returning objects about a query and, because of its probabilistic nature, may contain minor errors or be ambiguous [9].

Full-text search over email messages is useful when we are looking if a particular topic has been covered. It is also possible to restrict the search for a particular time frame. In our email example, it is interesting to retrieve documents about a topic per year.

At this point we presented the functionality where a user can run a query against the database. It is also important to see the other side where the database can arrange content semi-automatically using statistical techniques.

There are different statistical techniques to discover data patterns. In our framework, we use clustering to discover grouping of documents [10]. In our case, we want to use clustering as an unsupervised classification of email messages that can give us an idea of what the communication of the development team is all about.

We believe that the analysis and exploration involves a number of techniques like query languages, information retrieval, and data mining, among others. Databases provide almost all those techniques in some sort of low-level interface. Making them work together to mine useful information is the challenge. In the following section, we present a prototype implementation of the ideas presented so far.

4. Prototype Implementation

This section describes Minero[2], a prototype for mining information from software repositories. There are two parts of the implementation: the backend and the user application. The backend consists of the schema creation, content and metadata extraction from email messages, and post-processing. The application is a web interface that has a number of features available like text searching, cluster browsing, and reporting, among others.

Minero runs on top of an Oracle 9.2 database with Solaris as the operating system. The database runs in one machine while the middle-tier resides in a separate one. The web application is a mix of PSP (PL/SQL Server Pages), PL/SQL, and JSP (Java Server Pages) code.

[2] www.db.cs.ucdavis.edu/minero

4.1. Backend

As mentioned earlier, there is a significant amount of work that needs to be done to extract information from a particular data source, making it available in a consistent format, and finally loading it into a database.

For the case of the Apache development mailing list, all email archives are available from the web [1]. Each archive consists of a large number of email messages. The first step is to run an email extractor script that parses the archive and creates a file per email plus all other metadata that is useful for later processing like author, date, etc. The extractor also generates a special file that a database utility will use to load all the content into the Oracle database. As part of the loading process there is the schema creation.

Now that the content is inside the database, we need to perform some post-processing tasks. The tasks range from simple clean up scripts to advanced information retrieval and mining operations using the Oracle Text API [4]. Figure 1 summarizes the loading and post-processing steps.

As a simple first step, a few SQL scripts group email messages by date and year. Now we can issue queries that can give us information about the traffic on the development list (stored in the table `dev_ml`) in a particular time frame.

```
select count(tk) from dev_ml
where mdate = '1995';
```

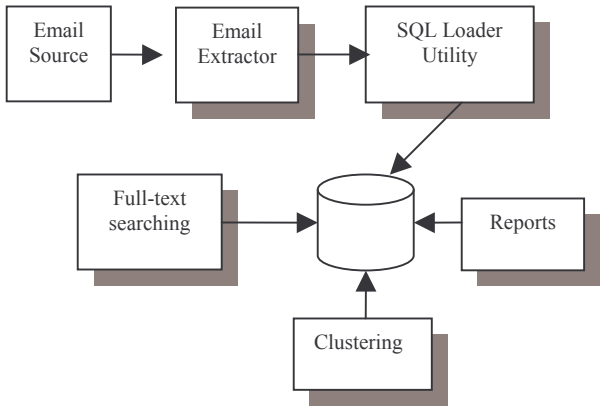

Figure 1. Extraction, loading, and post-processing of Email messages.

Moving into more advanced features, we want to make all the content searchable. For this, we create a couple of full text indices that will allow users to search

for emails about particular topics. Oracle provides an index type for full text searching that is available for querying using an extension of SQL. Because of the structure of an email message, we decided to create one text index on the subject only and a second one on the actual content. For example, to create a text index on a table we issue the following command:

```
create index devml_idx on dev_ml(text)
indextype is context
parameters ('filter null_filter');
```

A full-text search query for the term Java then has the following form:

```
select title, author, date
from dev_ml
where contains(text,'Java')>0;
```

The final step is to apply text mining techniques to the collection for identification of patterns. One way of doing that is to run a clustering algorithm that produces a flat partition of the content. The clusters will give us a "snapshot" of the content, and we can use them to refine searches, browse content, and produce a more accurate classification. The clusters can describe the areas of main interest in the mailing list. Minero uses a *k*-Mean clustering package that is also available with the database. Usually the running time of this cluster algorithm is $O(n*k*l)$ where n is the number of documents, k is the number of clusters, and l the number of iterations. For this prototype, we set $k = 200$ and $l = 6$. The output consists of tables that contain the description, metrics, and other data about the clusters and documents that belong to them.

4.2. Web application

We have built a front-end web application for users to explore and discover information from the mailing list. The front-end consists of a user interface where the user can search for email messages, browse clusters, run reports, identify main contributors to the list, and browse the entire collection by different attributes.

The user interface is based on a two-view model where the left side shows structure, and the right side, content of the collection of email messages. Minero presents the search results in two structures: a list of hits sorted by relevance score or a list of cluster descriptions that the user can open to see the hits. The right frame shows content (email messages) and message operations. The operations are: message with highlighted query terms, main themes, and gist (document summary).

Figure 2 shows a particular cluster about "processes, threads, MPM" and all the related email messages (documents). Again, users can view the email messages on the right frame and perform operations on them.

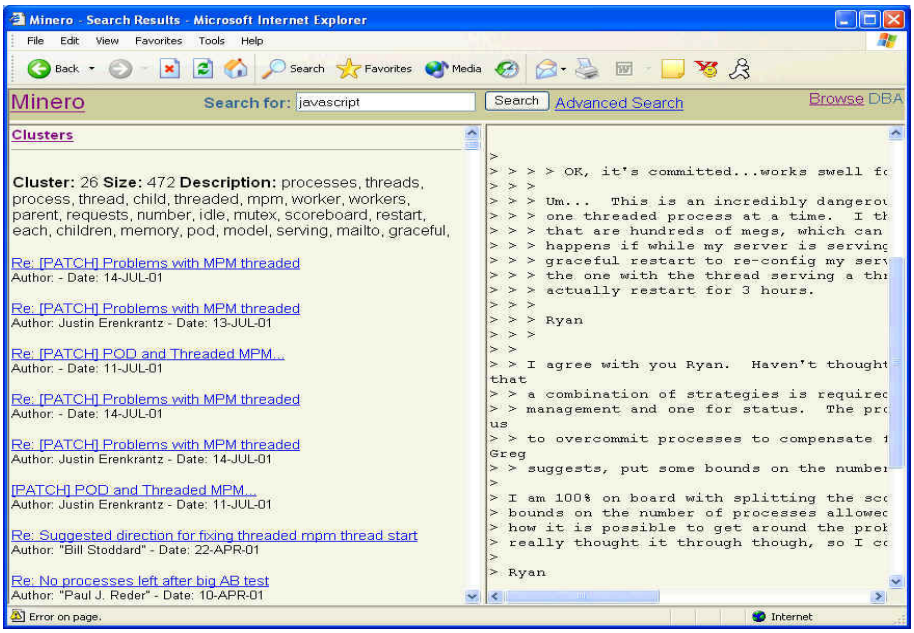

Figure 2. Using clusters to examine the Email collection

5. Results

The current database contains about 72,000 email messages. The content is searchable and can be restricted by date. The collection is partitioned into clusters that are available for browsing and discovery. There are a number of reports available for mailing list participation and overall message traffic per year.

Although our implementation is restricted to email messages only, we report similar findings to [12] in terms of the size and participation of the development community.

Going a little bit further and using time as a variable, we can report participation in the list per year using the following query:

```
select author, count(tk) cnt from dev_ml
where mdate = '1998'
group by author
order by cnt desc;
```

The past and present results are consistent with the public information about current and past team members [1]. We can also report that the main contributors have been doing so consistently over the life of the project.

Other results that we consider interesting from the software engineering perspective are as follows:

- We can identify two defined trends in the collection clustering. One is obviously technology. Examples are clusters about C/Unix, security, memory management, and threads. The second one is about process where the clusters are about releases, beta versions, patches, and the famous voting scheme for new features.
- As in any project, the early stages contain lots of process activities. Between the first and second year, there is a lot of traffic about the voting process, which later became stable.
- People change work places but continue participating on the list. This shows the commitment to certain open source projects.

6. Conclusions and Future Work

We presented a framework for the analysis and exploration of software repositories that relies on database techniques. We presented a prototype implementation of the ideas discussed so far. The project uses commercial database technology and it can be applied to other open source projects. We illustrated that a database is very convenient for this kind of projects. It allows us to define a schema where we can later perform queries or run more sophisticated mining techniques.

We were able to answer some software process questions like participation on the development list and also discover the technologies behind Apache using clustering.

We plan to continue working on Minero from the system perspective and on the overall methodology. We also plan to integrate the current schema with the source code repository and later the bug database so we can have a unified and integrated view of the project and the documents accompanying the project development.

7. References

[1] Apache Web server project http://httpd.apache.org

[2] R. Fielding and G. Kaiser, "The Apache HTTP Server Project". *IEEE Internet Computing*, 1(4), July/August 1997.

[3] R. Fielding "Shared Leadership in the Apache Project". *Comm. of the ACM*. Vol. 42, No. 4, April 1999.

[4] Oracle Text 9.2 Reference Guide (2003).

[5] D. German "Using software trails to rebuild the evolution of software", *Int. Workshop on Evolution of Large-scale Industrial Software Applications*. The Netherlands, September 2003.

[6] T. Zimmermann *et al*. "Mining Versions Histories to Guide Software Changes", *Proceedings of ICSE*, Scotland, UK, May 2004.

[7] A. Mockus, R. Fielding, and J. Herbsleb "Two Case Studies of Open Source Software Development: Apache and Mozill". *ACM TOSEM*, Vol. 11, No. 3 July 2002.

[8] S. Kawaguchi *et al* "Automatic Categorization Algorithm for Evolvable Software Archive", *Proceedings of IWPSE,* September 2003.

[9] R. Baeza-Yates and B. Ribeiro-Neto *Modern Information Retrieval*, Addison-Wesley (1999).

[10] A. Jain, M. Murty, and P. Flynn "Data Clustering: A Review". *ACM Computing Surveys*, Vol 31, No. 3, September 1999.

[11] A. Mockus and J. Herbsleb "Expertise Browser: A Quantitative Approach to Identifying Expertise", *Proceedings of ICSE,* Orlando FL. 2002.

[12] A. Mockus, R. Fielding, and Herbsleb "A Case Study of Open Source Software Development: The Apache Server", *Proceedings of ICSE* Limerick, Ireland, IEEE, 2000.