An Adaptive Filter-Framework for the Quality Improvement of Open-Source Software Analysis

Anna Hannemann, Michael Hackstein, Ralf Klamma, Matthias Jarke

Databases and Information Systems
RWTH Aachen University
Ahornstr. 55
D-52056 Aachen, Germany
{hannemann, hackstein, klamma, jarke}@dbis.rwth-aachen.de

Abstract: Knowledge mining in Open-Source Software (OSS) brings a great benefit for software engineering (SE). The researchers discover, investigate, and even simulate the organization of development processes within open-source communities in order to understand the community-oriented organization and to transform its advantages into conventional SE projects. Despite a great number of different studies on OSS data, not much attention has been paid to the data filtering step so far. The noise within uncleaned data can lead to inaccurate conclusions for SE. A special challenge for data cleaning presents the variety of communicational and development infrastructures used by OSS projects. This paper presents an adaptive filter-framework supporting data cleaning and other preprocessing steps. The framework allows to combine filters in arbitrary order, defining which preprocessing steps should be performed. The filter-portfolio can by extended easily. A schema matching in case of cross-project analysis is available. Three filters - spam detection, quotation elimination and core-periphery distinction - were implemented within the filter-framework. In the analysis of three large-scale OSS projects (BioJava, Biopython, BioPerl), the filtering led to a significant data modification and reduction. The results of text mining (sentiment analysis) and social network analysis on uncleaned and cleaned data differ significantly, confirming the importance of the data preprocessing step within OSS empirical studies.

1 Introduction

In the overview article "The Future of Research in Free/Open Source Software Development" (FOSSD) [Sca09] Scacchi identifies a range of research opportunities FOSSD provides for SE. The data collected over years and even decades within communities with up to hundred thousands of participants and large development histories alter fundamentally the costs and calculus of empirical SE. Further, the OSS project management taking place in publicly available Web resources (mailing lists, forums, etc.) provides new opportunities to extend conventional SE cost estimation techniques by rates for social and organizational capabilities. The introduction of new development paradigms like open innovation [Che03] implies integration of enduser communities into SE process. The OSS development presents a successful example of community-oriented organization, which

can be used to understand the motivation, role migration, inter-project social networking and community development.

Recognizing the advantages of OSS analyses for SE, a great amount of empirical studies were performed on OSS data in the last decade [GHH+09], [Sca10]. However, the quality of data analysis is directly dependent on the quality of the given data. OSS mailing lists can be messed up by spam. Message content can contain quotations and code clippings. Such "noise" can distort empirical studies executed on OSS data and lead to inaccurate conclusions for SE. During the decade of the knowledge mining in OSS, no general approach for data cleaning has been developed. Till now, some researchers perform data cleaning manually, others apply a combination of some partly automated procedures or even leave out the cleaning step completely. The variety of communication and development infrastructure used by OSS projects poses a special challenge for the automation of data preprocessing.

In this paper, we propose an adaptive and easily extensible Filtering-Framework (FF) for the quality improvement of OSS analysis. The FF also incorporates a schema-matching approach to deal with the variety of communication and development platforms used by different OSS projects. Within the FF, three filters have been implemented: quotation filter, spam filter, and core-periphery filter.

The filters were applied to three large-scale OSS projects from bioinformatics: BioJava[1], Biopython[2] and BioPerl[3]. The filtering led to a significant data modification and reduction. Sentiment and social network analyses were applied to both uncleaned and cleaned data. The analysis results based on uncleaned data differ significantly from those on cleaned data, confirming the importance of the data preprocessing step.

The rest of the paper is organized as follows. The knowledge mining tasks within OSS repositories and the strategies applied for data cleaning within those studies are presented in Section 2. Based on this overview the requirements for data cleaning process are defined. Section 3 describes an approach of our adaptive filtering-framework and its three basic filters for data reduction and content cleaning. The results applying our filtering framework to three OSS projects are shown in Section 4. These results are then discussed and different possibilities for framework extension, depending on a knowledge mining task, are specified in Section 5.

2 State of the Art

In [Sca09], the author presents an overview of opportunities OSS mining brings for SE. In most OSS empirical studies, the researchers use publicly available data within OSS management infrastructures. Presenting the results of a workshop on the future of OSS research held in 2010, Scacchi first describes five sample results on OSS development (OSSD): not only the knowledge extracted from OSS projects, but also the new techniques

[1] http://biojava.org
[2] http://biopython.org
[3] http://bioperl.org

and modeling methods developed for knowledge mining [Sca10]. Defining the research opportunities for OSSD and SE, the author addresses the development of the knowledge mining procedures and systems within and across the OSS projects repositories. But which knowledge mining procedures are relevant for the investigation of development process?

Beside general statistical methods like Lines Of Code (LOC), the procedures use text mining, social network analysis or a combination of both. Already in 2004, Jensen and Scacchi in [JS04] explored the software process discovery in OSS in a more holistic, task-oriented and automated way. They proposed a process model which incorporates text analysis, link analysis, usage and update patterns. They emphasize the need to gather communication data from mailing lists and discussion forums, and not to limit the data source to the code repositories, which only reflect the activities involved in changes to the code. The authors warn against the "well-known problems in text analysis" (e.g. stemming). They also point out that the communication histories "may not accurately reflect" the OSSD and therefore the probabilistic methods are needed to filter the noise from the process instances. For example, the communication can be spread across different resources (e.g. mailing lists and forum); the data given can be outdated (FAQ, wikipage of an OSS project); the communication history can be distorted by spammer messages.

Recently there is a shift from the partly automated analyzing procedures to fully functional frameworks for knowledge mining in OSS [SGK+09], [HCC06]. The Alithea system for OSS analysis [SGK+09] follows a three tier architecture approach: Databases - Cruncher - Presentation Layer. The cruncher calculates several statistical metrics upon (some) database entries and offers the results to the presentation layer. Each metric manages its own database space to store the results and works independent from all other metrics. Whenever a new database entry is added all the metrics are triggered. This approach supports an additive analysis, easy extensibility of the metrics list, metric independence and nesting. However, data preprocessing is not realized within the Alithea system. FLOSSmole[4] [HCC06] addresses the problem of diverse data sources and formats. The project offers a huge data set of diverse OSS projects in multiple formats. The authors also identify the problem that many research teams perform similar analysis tasks on the same data sets and are all struggling with the same problems, for example: where to get necessary data, which datasets to include, or how to implement the analysis (including data cleaning) algorithm.

More and more, the need for automated data cleaning approaches is recognized by the OSS research community. In [BSH09], a study on importance of proper (pre)processing of mailing list data to ensure accurate research results is presented. The authors identify the following challenges for mailing lists mining: message extraction, duplicate removal, language support, attachments, quotes, signatures, thread reconstruction, resolving identity. Since 2010, there is even a workshop series on mining unstructured data with the motto: "Mining Unstructured Data is Like Fishing in Muddy Waters!" [BA10]. However, in most of the empirical studies, the cleaning and analysis is still performed by a bundle of partly automated scripts.

Similar to [MFH02], [BGD+07] apply a set of perl scripts to extract message elements

[4] http://flossmole.org/

and resolve aliasing. However, spam detection within the mailing list is not addressed. Thus, the *indegree* and *outdegree* values calculated in [BGD+07] based on total community size can be distorted, if the community is not cleaned from spammers. In [YNYK04] and [vKSL03], the investigations rely on the user dynamics. In [YNYK04], the issue of data cleaning from spam is not addressed at all, while in [vKSL03], a manual data cleaning from spam is reported. In [JKK11], the authors do extract the spam from the mailing before analyzing "newbies' first interaction". However, the process of spam detection is not described. In [BFHM11], the content-based social network analysis is applied only upon mailing lists of the OSS project R. The authors present a text mining plugin consisting of functions for thread reconstruction, quotation and signature deletion, transformation to British English, synonym replacement, stemming, and stop words. The latter four can be performed either separately or all at once. In another text-mining study of OSS mailing lists [RH07], the authors delete attachments, quoted replies, code and HTML. The transformed results are saved into a separate database. In [VR11], a multi-level ontology for an automated requirements classification within an OSS is designed. The first two levels of the classifier present common natural-language parsing procedure: tokenizer and part-of-speech tagger. Despite the text-mining procedure, the authors do not describe the quotation elimination from the mailing list messages they analyze. From these examples of text-mining, dynamic and social network analysis studies applied to the OSS projects we can conclude that the cleaning steps differ strongly from study to study, introducing variability into data and probably affecting the results. At the same time, different cleaning procedures overlap. Obviously, there is a need for an easily extensible and study-independent filtering-framework.

Specifically, the following requirements can be identified:

Data-Structure Independence: Handling of different data sources and schemata.

Additive Filtering: Possibility to restrict the filtering only to newly added data.

Filter Nesting Possibility to combine arbitrary filters in any order.

Consistent Data Format Consistent structure of results.

Consistent and Easy-to-Use Interface Provide a list of available filters, database instances and required parameters; All filters should be invokable in exactly the same way.

Extensibility: Only the functionality of a new filter has to be implemented, while data exchange between the databases, the framework and the end user should be part of a framework.

Adaptive Database Insertion A parameterized query generator should be included, serving as a mediator between databases and filters.

3 Adaptive-Filtering Approach

OSS projects maintain several management mediums. In order to perform different kinds of analysis upon several OSS projects, there is a need for a filtering approach, which is applicable in the same way on data from any of these mediums. An *artifact* is defined as an elementary contribution in such a management element, like one post in a forum or one mail in a mailing list. The framework can be provided with a set of artifacts that need to be filtered. As these artifacts define different properties, a mapping from semantic meaning – known by the framework – to the corresponding property of the artifact is needed.

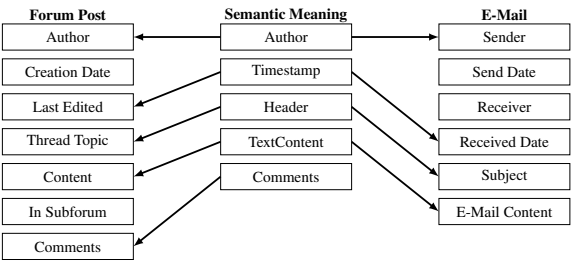

Figure 1: An Exemplary Mapping for Forums and Mailing Lists

Figure 1 gives an exemplary mapping for forums and mailing lists. In order to add a new artifact type or a new project, a database connection and a mapping table have to be specified. Given this information, the filters of the framework can be extended for any additional data sources.

A multi-threading approach, as presented in Figure 2, can speed up the filtering. The concept is to check when the last time this filter was run on the given data source. If there was no data update in the meanwhile, then the results of the last filtering are returned. If there was a data update after the last filtering, the system can clean the new data either in a synchronous or in an asynchronous way.

In the asynchronous way, the already filtered data subset is directly returned to the requester so that the next analysis task can be started immediately upon the pre-filtered data. In parallel, the filtering process upon the non-filtered data is applied. If the data was not previously filtered at all, the requester will get the data after filtering in subsets of a predefined size. With this technique, the requester accesses the first results very quickly and can already start analyzing preprocessed artifacts, while further filtering is continued in the background.

In the synchronous way, all new artifacts are filtered first, and then the whole data set is returned to the requester. This alternative is applied, if the requester needs all artifacts at once for further analysis/filtering (for example social network analysis), but it significantly increases the requesters waiting time for the results.

This approach is quite different to the one presented for Alithea [SGK+09], where the computation starts on each update. In our case, the filtering is triggered on demand, as a large total amount of filters is expected, but only a small subset is needed for each analysis

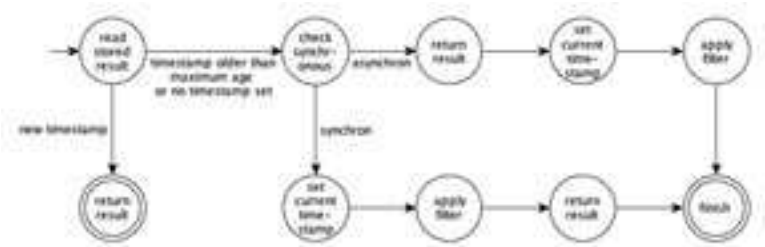

Figure 2: Finite State Machine to Receive Filtered Data

task. Additionally, filters may be designed for a special type of input data (e.g. user-list vs. mailing-list). Hence, computing a filter for user lists on a mailing list would create unnecessary overhead. A further argument for this approach is that in many cases, only certain data subsets are significant for the analysis. These subsets can be created using fast and simple filters on the entire dataset. The more time-consuming filtering can then be applied only to this subset. Again this prevents computing unused overhead. Also filtering on a daily basis is rare in research, mostly it occurs in intervals alternating intensive and no analysis.

Two distinct types of filters can be implemented in our framework: **Dataset Reduction Filters** and **Artifact Transformation Filters**. For performance issues, **Content Cleaning Filters**, a subset of Artifact Transformation Filters, are distinguished in this paper as well. All types return artifacts in a predefined format. In order to construct a complex filter, it is possible to define a sequence of filters $f_1, f_2, \ldots f_n$, which is then applied to the data. The results of filter f_1 serve as the data source for f_2, leading to sequences of arbitrary many filters in an arbitrarily order. A user just needs to trigger the last filter of the filter sequence as for each further filter in the sequence the updating process described above is triggered as a cascade. In contrast to Alithea system [SGK+09], which allows only independent metrics, in our framework filters can be implemented which require a pre-filtered data format.

3.1 Dataset Reduction Filters

In order to analyze an OSS project, the data set should be reduced to those artifacts which are significant for the analysis goal. E.g. if the analysis goal is to identify communities of users collaborating on the same issues based on their communication, only discussions should be taken into account. For this issue, Dataset Reduction Filters (DRF) are designed:

$$f_i \in DRF : ARTIFACT \rightarrow ARTIFACT, a \mapsto \begin{cases} a, & \text{if conditions are met.} \\ \emptyset, & \text{otherwise.} \end{cases} \quad (1)$$

These filters are used to remove artifacts that do not match certain conditions defined by the filter. Referring to the community identification example before, one of these filters

checks if an artifact is part of a discussion, then it is kept, otherwise it is removed. As a reference implementation a spam-filter is presented based on the Bayes Decision Rule (BDR):

$$\text{BDR}: WORDS \rightarrow CLASSES, w_1^N \mapsto \hat{c} = \arg\max_c \{\log(pr(c)) + \sum_{w=1}^{W} N_w \log(p(w|c))\}$$

For this rule a set of N words is needed, delivered by the text content of the artifact.

$$w_1^N := a.TextContent$$

For spam-filtering, only two classes are relevant: $SPAM$ and HAM.

The decision is based on the minimal posterior risk that the set of words w_1^N belongs to class c with the assumption that only the amount of words matter, the ordering is not taken into account. The equation presents a simplified way to decide, if an artifact is most likely *SPAM* or not. But still one problem is left: the exact distributions of all words among the classes is not known. These distributions are derived from manually labeled artifacts, the assumption is that for a random subset the same distribution of words holds as for the complete set. Hence the subset has to be large enough to fulfill this assumption. One special case to be considered is if no maximum can be found: an artifact has equal probabilities for *SPAM* and *HAM*. The behavior is defined via a parameter "onEqual" for each filtering process individually, leading to the following filter function:

$$f: ARTIFACT \rightarrow ARTIFACT, a \mapsto \begin{cases} a, & \text{if BDR}(a) = HAM \\ \emptyset, & \text{if BDR}(a) = SPAM \\ \text{onEqual}(a), & \text{otherwise.} \end{cases}$$

where: BDR : $ARTIFACT.TextContent \rightarrow CLASSES$,
$w_1^N \mapsto \hat{c} = \arg\max_c \{\sum_{w=1}^{W} N_w \log(p(w|c))\}$

3.2 Artifact Transformation Filters

For many tasks, preprocessing of the data is necessary, e.g. aliasing of users for mailing-list archives. The same preprocessing is required for several tasks, hence it is useful to store the intermediate result. As in the source data, these intermediate results are not considered, they have to be stored in an additional location. The filter-framework introduces Artifact Transformation Filters (ATF) to provide this functionality:

$$f_i \in ATF: ARTIFACT \rightarrow ARTIFACT, a \mapsto a' \tag{2}$$

These filters are used to modify parts of the artifact. This modification can either be overwriting an attribute of or adding an additional attribute to the artifact.

The results of an ATF are again artifacts, which can be used for further filtering. However, the user has to be careful as the modified attribute may influence the behavior of filters

based on it. In order to reduce redundant data storage only the modified attribute is stored by the filter, all other attributes are requested from the original artifact, or earlier filters respectively. An additional benefit of this storage method is that ATFs might be skipped during computation of filter sequences, if the ATF results do not affect the behavior of the requesting filter. For example, a spam filter is not affected by an ATF for aliasing and therefore these could work in parallel and the latter could get ahead of the ATF. As soon as the transformed attribute is requested, the requester has to wait for the ATF to finish.

A core-periphery filter (CPF) was implemented as one example for the ATF filter type adding an additional attribute. This filter iterates over a list of artifacts, most likely users, and takes the network as input, in form of a database table or a filter. At first, CPF computes values for each node in the network based on its out-degree and orders the nodes by this value in ascending order: $\{N_1, N_2, \ldots, N_n\}$ with out-degree(N_i) $\leq$ out-degree(N_{i+1})$\forall i \in \{1, \ldots n-1\}$. Iteratively CPF removes the first node from this ordered list and determines two values:

k_{i-1}**:** The value CPF assigned in the last iteration, starting with $k_0 = 0$.

o_i**:** The out-degree of the removed node, considering only edges leading to nodes still in the list.

The highest of these two values is then assigned to the node which was just removed: $k_i = \max(k_{i-1}, o_i)$. Besides that, CPF determines the two succeeding nodes N_j, N_{j+1} with the largest difference between their assigned values k_j and k_{j+1}. After this iteration the nodes N_i are mapped to the artifacts in the dataset. For the successfully mapped nodes the attribute *COREPERIPHERY* is added. The assigned value is either *Core* or *Periphery* based on the node's assigned value k_i:

$$CPF : ARTIFACT \rightarrow ARTIFACT, a \mapsto a' \tag{3}$$

$$\text{where: } a'.\text{COREPERIPHERY} = \begin{cases} \text{Core} & \text{if } a \mathrel{\widehat{=}} N_i \wedge k_i \geq k_j \\ \text{Periphery} & \text{if } a \mathrel{\widehat{=}} N_i \wedge k_i < k_j \end{cases}$$

During the evaluation we experienced that the last two nodes in the list get assigned a much higher value than all other nodes. Those nodes represent managers of OSS projects. Therefore we restrict $j < n - 2$, such that these two are not used to determine the largest difference, leading to better results.

3.3 Content Cleaning Filters

The analysis of longer texts is known to be a very time-consuming task. In order to enhance the performance of the analysis algorithm, unnecessary information, e.g. HTML-tags, should be extracted beforehand. Additionally some algorithms need the text in a special format, e.g. "part-of-speech"-tagged, which could also be done beforehand. For these issues, the Content Cleaning Filters (CCF) are designed as a subset of ATFs.

$$f_i \in CCF : ARTIFACT \rightarrow ARTIFACT, a \mapsto a' \quad (4)$$

These filters are used to transform the text content of artifacts. This transformation is considered as a special case, as the text content in general consumes much more space than all other attributes. To keep the overhead for the general case as low as possible this transformation is stored in a different table, offering more space for the value.

A quotation filter is presented as a reference implementation for the CCF filter type. Most web-based communication tools offer a possibility to quote other users. For analysis of user communication, these quotations are of high importance. However, for analysis of the content they are superfluous and lead to blurred results, as the same content gets analyzed again every time it is quoted. These quotes most of the times follow a certain pattern depending on the tool used for communication e.g. in many forums:

[QUOTE]Some quoted text[/QUOTE]

Hence, the quotation filter is applied to search for these patterns. If one or more patterns are found in the text content of an artifact, a copy of the text is generated in which the quoted parts are removed. The cleaned message is stored in the database. This behavior can be mathematically described as follows:

$$QF : WORDS \rightarrow WORDS, w \mapsto \begin{cases} \emptyset, & \text{if } w \text{ inside a pattern.} \\ w, & \text{otherwise.} \end{cases}$$

Hence the function for this filter is defined as follows:

$$f : ARTIFACT \rightarrow ARTIFACT, a \mapsto a'$$
$$\text{where: } a'.TextContent = QF(a.TextContent)$$

4 Validation in three OSS Projects

To validate our adaptive-filtering approach, it has been applied to the mailing lists of three large-scale OSS projects – BioJava[5], Biopython[6], and BioPerl[7].

Bayesian Spam Filter. In order to evaluate the spam filter, a test set of 1000 mails was extracted randomly out of the overall 60.190 mails. They were manually filtered and labeled. Afterwards the same data subset was automatically filtered, and the results were compared with the manual classification, shown in Table 1.

According to this evaluation result, no ham was wrongly removed by the filter. Around half of the spam mails were automatically identified. Figure 3 shows an excerpt of the spam messages and spammer ratio in the BioJava project from June 2003 to June 2005.

[5] http://biojava.org
[6] http://biopython.org
[7] http://bioperl.org

		Correct	
		Ham	Spam
	in test set	976	24
Selected	Ham	976 (100.00%)	13 (54.17%)
	Spam	0 (0.00%)	11 (45.83%)

Table 1: Bayes Spam Filtering

		Correct	
		Normal	Quotation
	in test set	11662	21646
Selected	Normal	11115 (95.31%)	1365 (6.31%)
	Quotation	547 (4.69%)	20281 (93.69%)

Table 2: Quotation Filtering

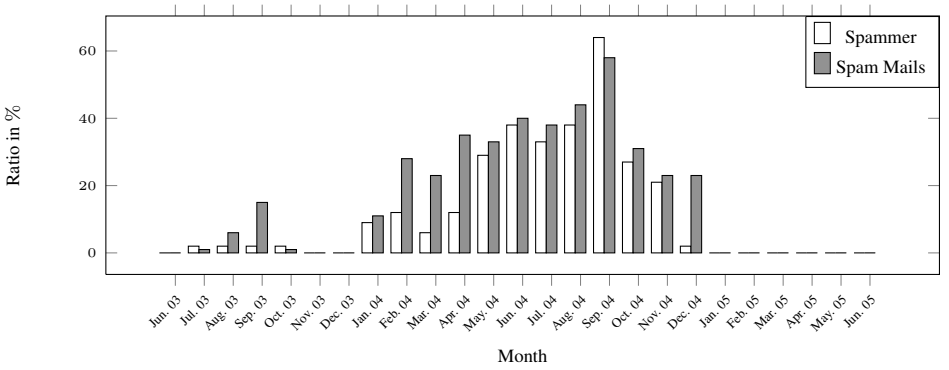

Figure 3: Spam distribution in BioJava Project

Obviously, the amount of spam is neither constant – fluctuating between 0% and 60% – nor insignificant – covering up to 60% of the communication. Similar fluctuations could be measured in all projects. The considerable and non-monotone variation of the spam volume is consistent with previous findings in spam analysis [Faw03].

Quotation Filter The Quotation Filter presented in Section 3 was also applied on these three OSS projects. Again, to evaluate the filter a test set of 100 mails was extracted randomly and the quotations were removed manually. In total, this test set included 33308 words, of which 21646 words were in quoted text, presenting 65% of the original data. This manually filtered set was compared word by word with the results of automatic filtering. Table 2 displays the results of the comparison.

The significant modification of data collected form OSS repositories after filtering gives evidence to the following general conclusions concerning spam within mailing lists: A *non-constant fraction of spammers* over the years was detected in diverse projects. Due to the variable level of spamming activity a *distortion of dynamics* occurs in the uncleaned data, hiding the important changes within the community. In addition, a *non-negligible rise of spammers* up to 60% could be observed. Hence, a major part of uncleaned data is spam not relevant for OSS analysis. Further, the *analysis could be accelerated* by removing spam from the input data and, thus, significantly reducing the data set. Additionally, the *identification of spammers* or other user classes can be applied to investigate their influence on the OSS project success [HKJ12].

Based on the results of the quotation filter we can conclude: A *non-negligible amount of quotations* – about 65% of the content – can be detected in user communication. These

quotations form a large amount of text which is contained multiple times in the overall communication. But they do not add any additional content information. Hence, *text-mining* can be significantly *speeded-up* by removing the quotations and thereby reducing the input data. Additionally, *textual classification improvements* could be achieved as the quotations belong to other authors and therefore should not be taken into account.

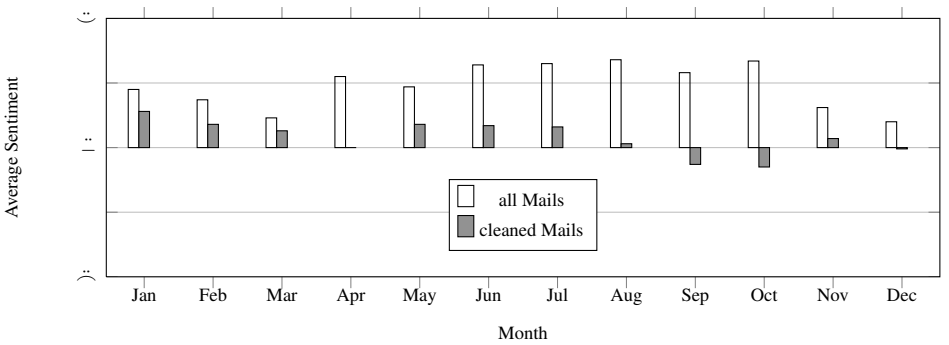

Figure 4: Sentiment Analysis for BioJava in year 2004

Furthermore, the influence of this filtering on OSS analysis was investigated. The mailing lists were cleaned, using the spam and the quotation filter. First, two measures of social network analysis - indegree and density - were applied to uncleaned and cleaned data. In [BGD+07], *indegree* was defined as "the proportion of the total population that has responded to this candidate since his/her first post". In 2004, 951 different users have posted an email to the BioJava mailing lists. The spam filter identified only 626 non-spammers - one third people less then in the original data. The indegree values calculated using uncleaned data are about one and half times smaller than indegree values calculated using data after filtering. The density, a basic measurement of social network analysis, calculated on the cleaned data is increased by a factor of 2.3.

Next, a sentiment analysis, a text-mining procedure which allows to estimate the sentiment of a message (either positive or negative) [Liu11], was performed on the original and filtered data. The average sentiment of all three OSS projects under study per month was calculated by subtracting the number of negatively labeled mails from the number of positively labeled mails. The values were normalized by the total number of mails per month. Figure 4 shows the sentiment of the BioJava community in 2004. The white bars represent the sentiment estimated on uncleaned data, while the grey bars represent the sentiment estimated on cleaned data. The latter is in the most month less positive. This shift can be explained by the positive nature of spam advertisement messages. Additionally, there is an effect resulting from quotation deletion. It insignificantly influences the average, but shifts many mails from positive to negative sentiment or vice versa. For instance, the following mail[8] is labeled as negative, however, after removing the quotations it is labeled as positive:

[8] http://lists.open-bio.org/pipermail/biojava-l/2004-March/004512.html

```
Maximilian H. wrote:
> Hallo everyone,
> Another question, this time a rather stupid one: How can I get rid of
> those."Originally:."/."Now:.". messages of the demos? [...]
I found some code in my copy in org.biojava.bio.gui.sequence.GuiTools
that is printing these messages (lines .33 and .34). These lines are
commented out in the version in CVS (revision 1.4). [...]
Matthew
```

This effect is due to the fact that someone describes a bug or a problem with negative emotion. The answer contains a solution with positive sentiment, but quotes the negative description as well. The description is often much longer than the solution, hence the mail would be recognized as negative in total. If the problem description is removed, only the solution part is analyzed and gets labelled as positive. The same situation can also occur the other way round: someone announces a nice feature (positive) and gets a bug report back (negative).

5 Conclusion and Outlook

Publicly available OSS communication and development histories provide a range of new opportunities for empirical studies of development process, which are not possible within conventional SE projects. However, the quality of a study is strongly depended on the quality of the data used for analysis. A "noise" within OSS data can lead to inaccurate conclusions for SE. Within this paper, an adaptive filter-framework for the quality improvement of OSS analysis has been presented. The framework uses a semantic mapping approach and a plugin technique for different database systems. Therefore, new data sources can be added quickly. It offers an easily extensible set of elementary filters, to clean and preprocess data sets. By using sequences of these elementary filters more complex filters can be constructed. Any filter sequence can be individually constructed/reused for each data set. Reusability is provided by a strict interface to the filters and a consistent data format within the framework: Each filter can be accessed in the same way, independent of the underlying data set and filtering function. In order to reduce redundant filtering to a minimum, an incremental filtering approach is implemented. Data sources of OSS projects are continuously growing. The approach allows to filter only newly arrived data, which has not been filtered so far. Quotation and spam filters of the framework were validated in three large-scale OSS projects - BioJava, Biopython, and BioPerl. A significant data reduction after both spam and quotation filtering was measured. A non-monoton rise of spammers up to 60% could be observed. The quotations form a large amount of text - about 65% - within OSS mails. Thus, the cleaned data set is much smaller than the original one. The results of text mining and social network analysis on the uncleaned OSS data significantly differ from the results of the same procedures on the cleaned data. The importance of data preprocessing step for OSS analysis was confirmed.

In future, we plan to enhance the presented framework by further frequently required filters. Plenty of studies apply text mining for OSS analysis. Despite different goals of the studies, in most cases, several preliminary preprocessing steps of textual data are similar. For example, part-of-speech tagging can be realized as a new filter. However, OSS

communication brings some new challenges to part-of-speech tagging: First, users often posts source code snippets, which do not follow a natural language and therefore cannot be tagged correctly. Next, many artifacts contain informal language and domain-related abbreviations, which are also hard to tag correctly with a training set in formal language. To handle each of these two challenges, a separate filter can be designed.

References

[BA10] Bettenburg N.; Adams B.: Workshop on Mining Unstructured Data (MUD) because "Mining Unstructured Data is Like Fishing in Muddy Waters"! In Reverse Engineering (WCRE), 2010 17th Working Conference on, pages 277 –278, oct. 2010.

[BFHM11] Bohn A.; Feinerer I.; Hornik K.; Mair P.: Content-Based Social Network Analysis of Mailing Lists. The R Journal, 3(1):11–18, June 2011.

[BGD+07] Bird C.; Gourley A.; Devanbu P.; Swaminathan A.; Hsu G.: Open Borders? Immigration in Open Source Projects. In Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07, pages 6–, Washington, DC, USA, 2007. IEEE Computer Society.

[BSH09] Bettenburg N.; Shihab E.; Hassan A.: An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, pages 539 –542, sept. 2009.

[Che03] Chesbrough H. W.: Open Innovation: The new imperative for creating and profiting from technology. Boston: Harvard Business School Press, 2003.

[Faw03] Fawcett T.: "In vivo" spam filtering: a challenge problem for KDD. SIGKDD Explor. Newsl., 5(2):140–148, December 2003.

[GHH+09] Godfrey M. W.; Hassan A. E.; Herbsleb J.; Murphy G. C.; Robillard M.; Devanbu P.; Mockus A.; Perry D. E.; Notkin D.: Future of Mining Software Archives: A Roundtable. IEEE Softw., 26(1):67–70, January 2009.

[HCC06] Howison J.; Conklin M.; Crowston K.: FLOSSmole: A collaborative repository for FLOSS research data and analyses. International Journal of Information Technology and Web Engineering, 1(3):17–26, 07 2006.

[HKJ12] Hannemann A.; Klamma R.; Jarke M.: Soziale Interaktion in OSS. Praxis der Wirtschaftsinformatik, February 2012.

[JKK11] Jensen C.; King S.; Kuechler V.: Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists. In System Sciences (HICSS), 2011 44th Hawaii International Conference on, pages 1 –10, jan. 2011.

[JS04] Jensen C.; Scacchi W.: Data Mining for Software Process Discovery in Open Source Software Development Communities. In Proceedings Workshop on Mining Software Repositories, 2004.

[Liu11] Liu B.: Web Data Mining. Springer, 2 edition, 2011.

[MFH02] Mockus A.; Fielding R. T.; Herbsleb J. D.: Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol., 11(3):309–346, July 2002.

[RH07] Rigby P. C.; Hassan A. E.: What can OSS mailing lists tell us? A preliminary psychometric text analysis of the Apache developer mailing list. In Fourth International Workshop on Mining of Software Repositories, 2007.

[Sca09] Scacchi W.: Understanding Requirements for Open Source Software. In Lyytinen K.; Loucopoulos P.; Mylopoulos J.; Robinson B., editors, Design Requirements Engineering: A Ten-Year Perspective, pages 467–494. Springer-Verlag, 2009.

[Sca10] Scacchi W.: The Future Research in Free/Open Source Software Development. In Proc. ACM Workshop on the Future of Software Engineering Research (FoSER), pages 315–319, Santa Fe, NM, November 2010.

[SGK+09] Spinellis D.; Gousios G.; Karakoidas V.; Louridas P.; Adams P. J.; Samoladas I.; Stamelos I.: Evaluating the Quality of Open Source Software. Electronic Notes in Theoretical Computer Science, 233:5 – 28, 2009. Proceedings of the International Workshop on Software Quality and Maintainability (SQM 2008).

[vKSL03] von Krogh G.; Spaeth S.; Lakhani K. R.: Community, joining, and specialization in open source software innovation: a case study. Research Policy, 32(7):1217–1241, 7 2003.

[VR11] Vlas R.; Robinson W. N.: A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In Proceedings of the 44th Hawaii International Conference on System Sciences - 2011, 2011.

[YNYK04] Ye Y.; Nakakoji K.; Yamamoto Y.; Kishida K.: The Co-Evolution of Systems and Communities in Free and Open Source Software Development. In Koch S., editor, Free/Open Source Software Development, pages 59–82. Idea Group Publishing, Hershey, PA, 2004.