2014 21st Asia-Pacific Software Engineering Conference

Mining Developer Mailing List to Predict Software Defects

Yu Zhang, Beijun Shen, Yuting Chen
School of Software
Shanghai JiaoTong University
Shanghai, 200240 China
{ruiko, bjshen, chenyt}@sjtu.edu.cn

***Abstract*—It has been studied that the communication among software stakeholders can be used to predict potential software defects. Yet researchers have rarely studied the relations between the software and the mailing lists of the developers. In this paper, we research on how to predict software defects by mining the mailing lists of the software developers. First, we extract both the structural and the unstructured information from mailing lists as metrics. The structural information is calculated through analyzing the social network hidden in the mailing lists, and the unstructured information is obtained through taking topical and textual analysis of the lists. Second, we design a mailing list-based approach to predicting software defects. We have also analyzed the software repository of several open source projects by linking their bug tracking databases to the mailing list archives. The experimental results provide empirical evidence that the mailing list metrics are related to software quality and can be used as predictors of defect-proneness. Furthermore, we found that (1) messages having certain structures may indicate some defect related files; (2) the sentiment and some topic-specific mailing models are of strong correlations with the software defects.**

Keywords-defect prediction; mailing list; software repository mining

I. INTRODUCTION

Communication plays an important role in software development and its quality assurance. Software, especially the open source software (OSS), is usually developed by teams consisting of a number of individuals ranging from the tens to the thousands [1]. Thus the better the quality of the communication among the teams is, the stronger the collaboration will be and the more possibility of success of the resulting software will be obtained. In addition, communication among software stakeholders may indicate the existence of defects in software and their locations. For example, Wolf et al. [2] present that some building failures of a software system can be predicted by taking social network analysis on the communication among developers.

On the other side, the effect of mailing lists on software quality is often neglected. Mailing list has become the primary communication channel for OSS development [3] in which it serves as the hub for project communication. However, most researches in the area of communication-based software assurance get communication data from some specific or private platforms. To the best of our knowledge, few researchers take advantages of mailing lists to detect or predict software defects and investigate their effects on software quality. For the abundant development information in mail repository, the mail feature should be a nice supplement for defect prediction.

In this paper, we research on how to predict software defects by mining the mailing lists of software developers. In order to do this, we first link mails to source code and then explore the relations between the mailing lists and software defects. In this step, we attempt to answer three research questions:

RQ1. ***Are certain specific structures of mailing lists associated with software defects?*** The answer is yes. In the research, Email contents are classified in some categories and threads of mails are organized in complex structures. We find that mails having some content structures or thread structures usually imply the existence of software defects in some files.

RQ2. ***Is there any relation between the developers' sentiment and the software quality?*** In the research, we adopt a Linguistic Inquiry and Word Count (LIWC) tool to recognize the developers' sentiment. Emotional processes can be quantized and calculated by using LIWC and the experimental result shows that the positive emotions usually lead to few defects.

RQ3. ***What concerns of mails are more likely to be defect-prone?*** We introduce a topic model to measure the Emails into some concerns, which are used as the input of machine learning-based defect prediction. Result shows that the topic-based metrics are often of a strong correlation with the defective files.

After conducting several experiments, we also have some findings with respect to the research questions. These findings help us obtain some metrics about communication behaviors hidden in the mailing lists. Next, we develop a mailing list-based approach to predicting software defects. With a set of mailing list-based metrics and source code, the approach can be used to indicate which artifacts are defective. Furthermore, we compare the prediction approach with some state-of-the-art approaches and make an optimization. By taking some combinations and optimizations, we improve the effectiveness of the prediction approach.

The paper is organized as follows. Section 2 discusses the related work. Section 3 presents the main activities of the prediction approach. Section 4 presents the case study and some research findings. Meanwhile some threats to validity are presented. Section 5 concludes this paper and points out the future research directions.

1530-1362/14 $31.00 © 2014 IEEE
DOI 10.1109/APSEC.2014.63

II. Related Work

A. Communication and E-Mails

Using data from IBM's Jazz TM project, Wolf et al. [2] have studied the communication structures of development teams with high coordination needs and explained the importance of communication. One main weakness is that IBM's Jazz TM was limited for extensive study. Similar problem exists in the study of using specific communication platform or tools in closed source software. Open source software development teams use general electronic means (emails, instant messaging, or forums) to communicate. Among these means mailing list becomes one of the primary communication channels for an OSS project.

Prior work focused on specific aspects of emails, including the handling of patches, traceability concerns, social networks, et al. Some important characteristics of emails have been found, such as the strong relationship between the level of email activity and the level of activity in the source code. Moreover, email threads cover a wide range of topics and implementation details are only in a portion of them [3]. Christian Bird et al. [4] have studied how to construct social networks of email correspondents by mining the email archives of some OSS projects, and explored the relationship of email activity and commit activity. These are successful attempts while the research methods are immature now. Bacchelli et al. [5] have dealt with the problem of recovering traceability links between emails and source code and created Miler, a tool infrastructure for building a statistically significant benchmark of links between e-mail and source code over six software systems. This work provides with an important foundation for the newcomers to research on mailing list. Wagstrom et al. [6] have gathered empirical social network data from several sources, including blogs, email lists and networking web sites, and built models about their social behavior on the network; the data was used in a construction of a simulation model which describes how users join and leave projects. On the basis of the above work, we focus on the little studied aspect of mining developers' mailing lists to predict the existence of software defects.

B. Defect Prediction

To build defect prediction models, researchers have investigated different factors in a software system and identified the ones that are most related to the defect-prone files. These factors include code complexity, change/process complexity, semantic dependencies between software modules, organization of the development team, socio-technical aspects of software development, and so on.

In socio-technical aspects, Wolf [2] has proposed method to predict software failures using social network analysis on developer communication. Developer-module [7] and social organization [8] have also been investigated in the past studies. Nguyen et al [9] have implemented a topic-based approach to defect prediction. Topic model has been adopted to measure the concerns in source code which are then used as the input for machine learning-based defect prediction models. This approach can be used for reference meaning of the mailing list-based prediction of defects.

III. Prediction Approach

A. Approach Overview

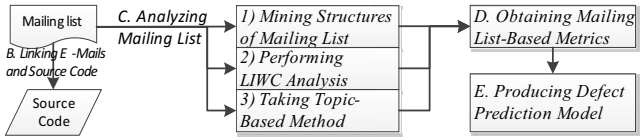


Figure. 1. Approach Overview

In this section, we introduce an approach to mining mailing lists to predict software defects. The workflow of the approach is shown in Figure. 1. Section III.B introduces the activity of linking emails and source code as the first step of the prediction. Sections III.C presents three activities of answering the three research questions (i.e., RQ1, RQ2, and RQ3), respectively. Sections III.D and III.E present the activities of obtaining the mailing list-based metrics and building the defect prediction model, respectively.

B. Linking E-Mails and Source Code

Knowledge can be implicitly expressed in non-code artifacts, such as the documentation, wikis, forums, e-mails, and so on. Since our goal is to study how to use the mailing lists to predict defective file, it is crucial to link the lists with source code so that the source files that are possibly flawed can be directly found and verified by exploring the links.

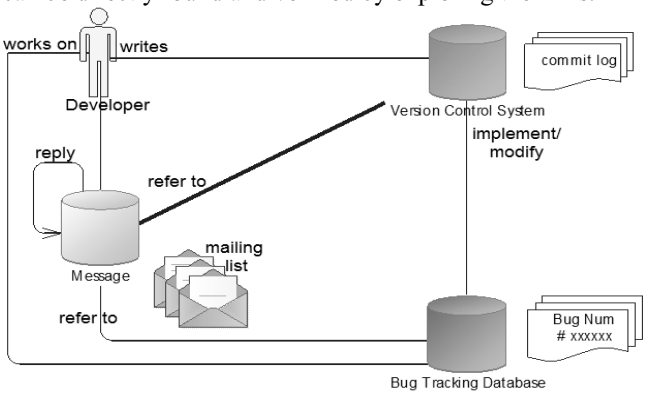


Figure. 2. Artifact linkage schema

Improved from the schema in [10], an artifact linkage schema (see Figure. 2) is used to represent a project's implicit group memory. There are three types of artifacts represented in the schema: the bug and feature descriptions (e.g., items in Bugzilla), the source file revisions (e.g., checked in a svn source repository), and the messages posted on developer forums (e.g., newsgroups and mailing lists). These artifacts are created by project members, represented by 'Developer' in the figure. Entries in the project's issue-tracking system are a locus because they typically represent a logical unit of work of the project. Source revisions are checked into the source repository to respond to one entry; mailing list messages often contain a discussion item that either results in a new entry or relates to an existing one.

There exist a number of methods, ranging from lightweight methods on the basis of regular expressions to full-fledged information retrieval method [5], for building

the direct links between the source code and messages. Since the lightweight linking methods are easy to implement and their precisions are comparatively high, we adopt a lightweight linking method which uses regular expressions about the entity name, case sensitive, and punctuation to filter the mail content with source code information.

Eg. (.*) (\s|\.|\\|/) <packageTail> (\.|\\|/) < EntityName > ((\.(java|class))|(\s))+ (.*)

C. Analyzing Mailing List

1) *Mining Structures of Mailing List:* Figure. 3 shows a message fragment and its structure. On the top part, there shows an email with two replies. The detail of the original email in the box is given below, where different icons represent different contents. RQ1 investigates the relationship between the structures of mailing lists and the software defects. Two kinds of structures of mailing list are defined here: the "content structure" and the "thread structure". The former represents the internal composition of the mail body, and the later represents the external aspect of the emails. Specially, when investigate the thread structure of an email, we focus on the structural relationship between the email and the other emails.

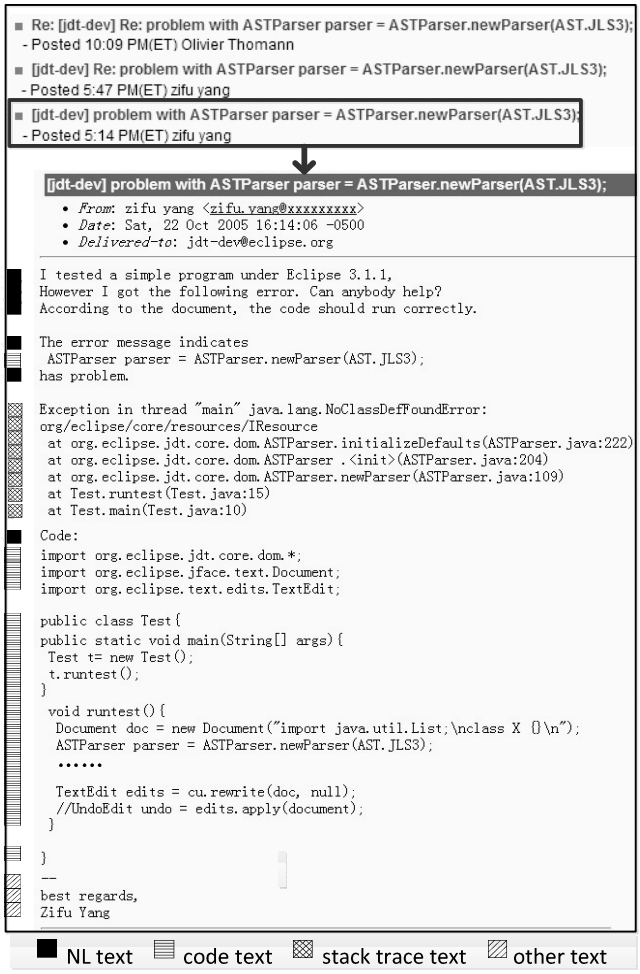


Figure. 3. Message structure

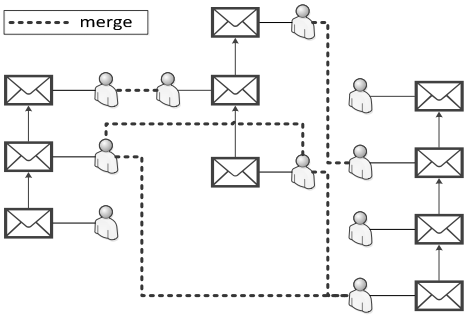


Figure. 4. Network graph about message thread structure

A mail body is usually formed with mixed content, which can be classified into five categories (i.e., text, junk, code, patch, and stack trace [11]). Different kinds of compositions may denote different purposes or effects, so defects may be discovered by exploring some special content structures. We use four kinds (i.e., NL text, code, stack trace and other text) to discover the rule.

On the other hand, a message is often dependent. An email is usually followed by a thread of replies which carries about the same issue or share their solutions to one problem. In addition, the messages sent by the same author during a short period of time may hold some common features. All these situations do exist in practice, which drives us to describe the thread structure of a mailing list at a viewpoint of network graph. Network graph is widely used in the study of social software engineering [12, 13]. But it has rarely been used to represent the relationship among messages. Figure. 4 shows the network graph designed to express the thread structure of mails. In this figure, each mail can be assigned with a directed edge from its reply to it. A mail node is connected with its author node by using an undirected edge so that the relationship among different threads are shown. Social Network Analysis provides several network measures such as degree centrality, closeness centrality, betweenness centrality, etc. These measures define the characteristics of each node and thus can be used as the metrics for evaluation.

2) Performing LIWC Analysis: We choose Linguistic Inquiry and Word Count (LIWC) to perform analysis in order to answer RQ2. LIWC is based on a function of counting words or particles. However, the semantics of the text may be lost if we only count the words because word count programs cannot catch sarcasm, irony, or a given contextual meaning for words. The LIWC uses a psychometrically-based dictionary that has been validated by independent judges and used in a number of experiments by Pennebaker and others.

Our primary goal is to take the LIWC tool as a predictor and classifier to understand the intricacies of the software development. LIWC tool works like this: words like 'maybe' are associated with tentativeness and words like 'important' are associated with certainty, etc. The LIWC tool simply counts words that are contained in its dictionary. The dictionary is grouped into some basic linguistic and psychometric dimensions with each word belonging to one or more dimensions. We only use partial function about

psychometric of LIWC to conduct experiment on the Eclipse's developer mailing lists, where Emails are taken as input and output contains the sentiment of the text.

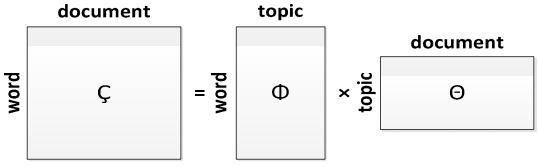
An activity is mainly designed at a viewpoint of the emotion such that we can check whether the message content reflects the developer's feelings, either positive or negative. It is expected that the negative emotions may lead to more defects hidden in the code. Next, we analyze the empirical data, identify critical points based on the relationship between emotional processes and defects, and validate whether the measures can become a predictor of defects.

3) *Taking Topic-Based Method:* For investigating RQ3, we adopt a topic based method to study the concerns of emails. Automated methods have been extensively studied in text mining and machine learning area to extract topics from documents. The basic idea is to discover the latent structure of document by recognizing the words' co-occurrence in the corpus. We collect mails associated with different parts (i.e., components) of eclipse. There exist some components-specific concerns in the emails and we find that some concerns are defect-prone. For example, the topics of the mail contents are usually similar when the developers discuss about the potential defects within a component.

Probabilistic topic model contains a series of algorithms which are designed to discover the hidden topic structure in the large-scale document. A topic model analyzes the words in the original text to discover the hidden topics and links topics to the evolution of themes over time. A generative model can be used to describe the document and topic. The so-called generative model denotes that an article is formed by a process such that every word in an article selects a certain topic by a certain probability, and selects a certain word from this topic with a certain probability. So the probability of each word appearing in it is:

$$p(word|document) = \sum_{topic} p(word|topic) \times (topic|document). \quad (1)$$

This probability formula can be represented by using a "document - word" matrix:



where word frequency means the probability of each word to appear, the "topic-word" matrix represents the probability of each word to appear in each topic, and the "document-topic" matrix represents the probability of each topic to appear in each document. Topic models have one main advantage that no training data is required when the models are built in discovering structures from unstructured data. Given unstructured data we want to explore, what to do is to set some parameters such as the number of topics. We use LDA (Latent Dirichlet Allocation) [14], a simple topic model on the basis of a document composed by a number of topics. LDA models the documents in the same corpus as generated by some or all of the given N topics and words in each document come from the word-topic distribution.

D. Obtaining Mailing List-Based Metrics

[15] summarizes four studies of historical metrics including code metric and process metrics:

- Change metrics — Bugs are usually caused by changes.
- Previous defects — Past defects can predict future defects.
- Source code metrics — Complex components are hard to change, and hence error-prone.
- Entropy of changes — Complex changes are more error-prone than simpler ones.

And two novel source code metrics were defined:

- Churn (Source code metrics) — Source code metrics are a better approximation of code churn.
- Entropy (Source code metrics) — Source code metrics better describe the entropy of changes.

For a more comprehensive summary of metrics, [17] defines four categories of metrics: code-related metrics, process-related metrics, organizational metrics, and geographical metrics. Network metrics belongs to the category of organizational metrics, which denotes that the networks between developers and modules can be analyzed for predicting failures. The idea is similar to our design of thread structure of mails, but our metrics from other research methods cannot be classified into this type. We adopt the next metrics as social tech metrics:

- Mail Content Metrics (MCM) — Special type of content predict future defects.
- Mail Network Metrics (MNM) — Key emails about communication among core developers may link to more defective files.
- Emotion Metrics (EM) — Mails with passive emotion may indicate potential defects.
- Topic-based Metrics (TM) — Particular topics of mail text can indicate defect-prone files.

MCM and MNM are designed from RQ1, EM and TM are designed from RQ2 and RQ3 respectively. It is noted that whether these metrics are available is determined according to the answers to the research questions. The precise meanings of these metrics are confirmed in our study.

E. Producing Defect Prediction Model

We adopt a regression model to examine the relationship between the mail related metrics and the post-release bugs. The independent variables (used in the prediction) are the set of metrics for each class, while the dependent variable (the predicted one) denotes that whether a class is defective or not with respect to the number of post-release defects. Principal component analysis is employed to avoid the problem of multicollinearity among the independent variables.

For comparison and evaluation, we generalize the linear regression model [15], the regression tree model (M5P) [9], and the logistic regression model [16] to predict defects. Results in [15] are served as the benchmark to evaluate the predictive factor of the new model. Since social tech metrics

are not discussed in [15], another benchmark in [16] which takes social tech metrics in prediction model is selected for comparison. We also adopt the *10-fold cross-validation* strategy in the study, i.e., 90% of the dataset is used to build the prediction model, and the others to evaluate the accuracy of the model.

IV. CASE STUDY

A. Process

Our case study is performed in four phases: data collection, data analysis, validation and defect prediction. Figure. 5 shows the overall process of the case study.

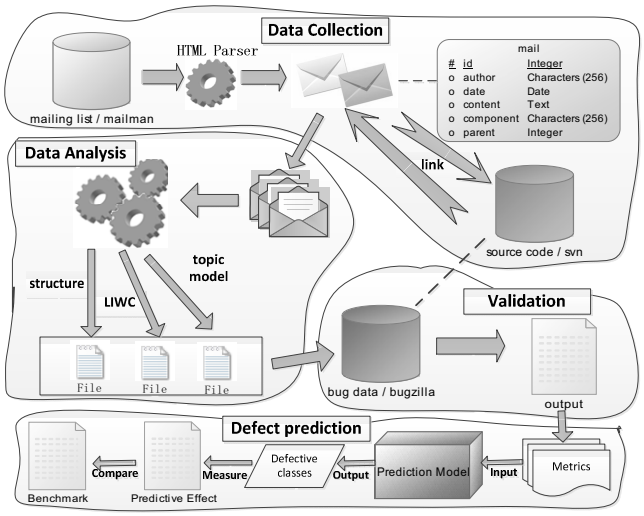


Figure. 5. Process of case study

- Data collection: Data set in our research is collected from several components of eclipse whose mailing lists are managed by Mailman. Mailman provides a web page view. We use a Java library called HTML Parser to parse HTML code, in order to collect the mails from web pages. Then all messages are recorded by its corresponding author, date, and module. Next, these emails are linked to their related files/classes by the lightweight method and filtrated. From table I, we can see that the link rate is not high, ranges between 13.9%~19.8%. We conducted an extra manual sampling so that the error link rate is 0%, the missing link rate is less than 5% and the correct rate is achieved.

TABLE I. INFORMATION OF DATA SETS

Component	Number of Mails	Mails been Linked	Link Rate	Description
jdt-dev	751	117	15.6%	Eclipse JDT general developers list
jwt-dev	1644	231	13.9%	Java Workflow Tool developers list
eclipse-dev	9807	1944	19.8%	General development mailing list of Eclipse project

- Data analysis: This phase includes three analysis activities explained in Section 3. And three methods are designed to answer the three research questions.
- Validation: The output in the previous phase is a set of files which are predicted to be defective. In this phase we verify the correctness of the result by comparing the predictive defective files to the actual defective files (Eclipse bug data is offered by Zeller).
- Defect prediction: After validation, research questions are answered and hence, the mailing lists-related social tech metrics and prediction model can be constructed. Then a prediction can be carried out to obtain the predictive effects with the benchmarks.

B. Findings

RQ1. Are certain specific structures of mailing lists associated with software defects? Table II shows the results as correlation between content structure and defects. In our experiments, two kinds of content structures are chosen, including code text and stack trace text. Both of the two kinds of text are likely related to bugs, because in most of the emails, the author adds codes or stack trace texts to show the problem in the project. Then the problem may turn into a new bug. In the table, the rate of each kind of content is simply worked out by calculating lines of text. Top ten emails of the rate ranking are selected to analyze. From the result table, it can be seen that emails with high rate of code lines are always linked to files with bug, so are the emails with high rate of stack trace lines.

TABLE II. DEFECTIVE RATE OF TOP TEN SPECIFIC MAIL STRUCTURES

Component	Code Text Rate	Defective Rate	Stack Tracing Text Rate	Defective Rate	Average Defective Rate
jdt-dev	71.3%	70%	47.1%	80%	35.0%
jwt-dev	68.2%	60%	40.5%	60%	24.9%
eclipse-dev	64.7%	70%	44.7%	70%	21.5%

We apply network graph analysis to study the thread structure of emails. For each component, we create a network graph which contains emails in a specific period. Through a series of evaluation, we select betweenness centrality measure to describe the result. It equals to the number of shortest paths from all vertices to all others that pass through that node. The measure is given by the formula:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (2)$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v. The betweenness centrality of all nodes is calculated and then the author nodes are discarded because emails are linked to source files with a uniform method. Emails are sorted by their betweenness centrality, where we retrieve the top ten emails in each component (see Table III). By calculating the rate of defective files linked from top ten emails, we find that in most instances, emails in network graph with high betweenness centrality have the highest probability of linking to the defective files.

TABLE III. DEFECTIVE RATE OF TOP TEN CENTRALITY EMAILS

Component	Average Betweenness Centrality	Defective Rate of File	Average Defective Rate
jdt-dev	0.47	60%	35.0%
jwt-dev	0.52	50%	24.9%
eclipse-dev	0.62	70%	21.5%

The answer to RQ1 is yes according to the result. Defects are related to some specific structures of mailing lists, including both the content and the thread structures. For the former, developers write different kinds of email contents for different purposes, and code and stack tracing text, especially the stack tracing lines, always relate to program bugs and problems. For the latter, betweenness centrality denotes that the possibility of the case that one node is passed by the shortest path connecting another two nodes. Thus a node with high centrality is more important. These key nodes are defect-prone in that they have more influence on the whole systems. On the other hand, more serious problems are concerned by more developers. Above all, the structures of mailing lists reflect the characteristic of emails and specific structures certainly indicate defects.

RQ2. Is there any relation between the developers' sentiment and the software quality? The use of the LIWC tool is not difficult. Table IV shows the analysis result of an email using the tool. We concentrate on the "Positive emotions" and the "Negative emotions" in the result table since the experiment is carried out on the basis of the emotions. By taking a great deal of measurement, it indicates that the value of the positive emotions is larger than that of the negative ones. Thus we assign a threshold to judge whether a particular text is positive or not. The threshold value is assigned by analyzing the results of the formal texts and the existing mailing lists.

Table V shows the result of the study on the research question 2. We compare the differences between positive emotion and negative emotions with the threshold value (which is 2 in experiment). It can be seen that the emails whose emotion difference value is larger than the threshold value tend to be linked with less defective files and those whose emotion value is less than threshold value tend to be linked with more defective files.

TABLE IV. LIWC TOOL OUTPUT

LIWC Dimension	Your Data	Personal Texts	Formal Texts
Self-references (I, me, my)	0.00	11.4	4.2
Social words	5.04	9.5	8.0
Positive emotions	4.20	2.7	2.6
Negative emotions	0.00	2.6	1.6
Articles (a, an, the)	9.24	5.0	7.2
Big words (> 6 letters)	38.66	13.1	19.6

TABLE V. DEFECTIVE RATE OF EMAILS WITH DIFFERENT EMOTIONS

Component	Defective Rate below Threshold	Defective Rate above Threshold
jdt-dev	70%	20%
jwt-dev	70%	10%
eclipse-dev	60%	20%

We can answer research RQ2 now. There exist some relations between developers' sentiment and software quality. However, the conclusion is not always correct and sometimes the result is volatile in our experiments. The reasons for these problems are various. First, nature language may not be simply analyzed using a tool and a developer may not only express one feeling in her mail. Second, some key words or variable names in code such as 'if', 'new' can have some influence on the result. However, the final results prove that emotion and defects are relative.

RQ3. What concerns of mails are more likely to be defect-prone? This question is answered by a topic model which can tell us the concerns of emails. In LDA, topics are collections of words that co-occur frequently in the corpus. Table VI shows five topics extracted from the Eclipse JDT mailing list contents. Each column entry presents a topic and its top 10 words which are of the highest co-occurrence frequencies. It can be seen that topic has its own concern. However, this point is not very obvious in table VI, which will be discussed later.

For each email, non-standard use of some word in it should be processed by splitting or removing. After pre-processing, each email is represented as a collection of words and the system is represented by a collection of emails. We apply the LDA topic-modeling technique on that collection with a number of topics K = 5, a number of iterations R = 50, and two hyper parameters (α= 0.5 and β= 0.1). Using the discovered structure, emails can be linked through the topics assigned to [9], and the result provides with the numbers of words assigned for each of the topics in each email. We take them as the topic-based metrics.

We calculated the correlation between the topic-based metrics and the defective rate of the files linked by emails. As Table VII shows, it is obvious that high defective rate indicates high correlation. It can be seen that the topic-based metrics have correlations with the number of bugs at different levels. Specifically, T3 has a quite high correlation while T1 has a much lower one. This suggests that topic T3 is more defect-prone than T1, i.e. in general, source files related to T3 are more probably to be defective than those related to T1. From topic 3, we can find that it is concerned about source code and bug in this component. We hypothesize that, in Eclipse JDT, bug discussion topic (T4) is more defect-prone. In this way the third research question can be answered.

TABLE VI. DEFECTIVE RATE OF MAILS WITH DIFFERENT EMOTIONS

Rank	*Topic 1*	*Topic 2*	*Topic 3*	*Topic 4*	*Topic 5*
1	java	mailing	jdt	eclipse	java
2	working	class	compiler	completion	code
3	problem	version	method	point	line
4	annotation	find	public	work	project
5	extends	jdt-core-dev	change	type	files
6	variable	default	bug	create	jdt-core-dev
7	time	context	source	build	extension
8	list	classpath	error	void	file
9	functionality	support	code	way	core
10	args	respond	default	static	found

TABLE VII. CORRELATION OF TOPIC-BASED METRICS AND DEFECTS

Component	T1	T2	T3	T4	T5
Number of Defective File	4	8	13	7	9
Defective Rate of Files	0.10	0.20	0.32	0.17	0.22

The experiment confirms the hypothesis that the components-specific concerns obtained from emails correlate to the defect-proneness of source files at different levels. However, it is not clear that how the topic-based metrics can be used to predict bugs. Although five topics are extracted from the jdt mailing lists, it is difficult to say what they are concerned about. On the other words, the five topics are difficult to distinguish. One possible reason for this is that after linking of the mails to source files, only a small number of mails are left and they have something in common. Thus a sensitive method for analyzing the topic model is necessary in future.

C. Application and Evaluation

All the three aforementioned methods for analyzing mailing lists generate some metrics to predict defects. Here we obtain these metrics to predict defects and evaluate the predictive capability and effectiveness.

1) Extraction of Metrics: The first step is to extract the metrics. Four types of metrics have been prepared. After analyzing the results in the last section, we adopt the metrics shown in Table VIII.

TABLE VIII. DEFINITIONS AND TYPE OF METRICS

Metric	Definition	Type
LOCM	Line of Code in Mail	Mail Content Metrics (MCM)
LOSTM	Line of Stack Trace in Mail	
ROCM	Rate of Code in Mail	
ROSTM	Rate of Stack Trace in Mail	
DCOM	Degree Centrality of Mail Network	Mail Network Metrics (MNM)
BCOM	Betweenness centrality of Mail Network	
ESOM	Emotion sum of Mail	Emotion Metrics (EM)
EDOM	Emotion difference of Mail	
TM1~TM5	Top Five Topic-based Metrics	Topic-based Metrics (TM)

LOCM and LOSTM can be obtained by counting the lines of the content. ROCM and ROSTM are calculated through dividing the total lines of content by the lines of the content. DCOM denotes the degree centrality of a vertex v, for a given graph G:=(V, E) with |V| vertices and |E| edges, DCOM equals to deg (v). BCOM is defined in Section III.B. ESOM and EDOM represent the sum and difference of Positive emotions and Negative emotions values, respectively. TM1~TM5 is the similarity of topics of each email text with T1~T5. T1~T5 is the five topics extracted from all of the Eclipse JDT mailing list contents by LDA.

2) Measures for Evaluation: We choose 91 adjacent versions of eclipse to conduct our experiment. IR measures: precision ($\frac{|TP|}{|TP|+|FP|}$), recall ($\frac{|TP|}{|TP|+|FN|}$) and f-measure ($f1 = \frac{2*precision*recall}{precision+recall}$) are used to evaluate the prediction result. TP (true positives) is a set of the defective files correctly predicted. The set FN (false negatives) contains the defective files not predicted, while FP (false positives) links normal files presented as the defective ones. Spearman correlation is used to evaluate the predictive capability. It is a robust technique that can be applied even when the association between the measures is non-linear [8]. The Spearman correlation is computed on two lists (classes ordered by the actual number of bugs and those by the number of predicted bugs) and is an indicator of the similarity of their order.

3) Experimental Results: For comparison and evaluation, the paper established linear regression model and logistic regression model respectively. The linear regression model is represented by formula $f(M_1, M_2, \dots M_k) = \omega_0 + \sum_{i=1}^{k} \omega_i * M_i$, the function $f(M_1, M_2, \dots M_k)$ in which means the situation of defects. Then a logistic regression equation $g(x) = 1/(1 + exp(-f(M_1, M_2, \dots M_k))$ is used for mapping the range of linear regression function to the domain probability values [0,1].

In Table IX, the left part shows the predictive capability of the model via Spearman correlation between the predicted and actual numbers of bugs in [15]. The right part shows the result, which is low when using each metric alone. Moreover, result with combination of metrics is lower than the best result achieved using the past approaches. In the last line, the combination of the metrics in Table VIII and some past metrics gains the best result. Predictors do not involve the social-based metrics in [15], and thus we choose [16] to take a comparison. In table X, the above part presents the prediction results obtaining using social network metrics [16]. Relatively, our combined metrics show the result with IR measures.

TABLE IX. PREDICTION RESULTS AND COMPARISON PART1

Predictor	Spearman	Predictor	Spearman
Change metrics	0.381	MCM	0.432
Bug metrics	0.434	MNM	0.416
Code metrics	0.395	EM	0.428
History of Complexity metrics	0.416	TM	0.389
Churn of code metrics	0.442	—	—
Entropy of code metrics	0.425	Our best combination	0.437
Their best combination	0.448	Best combination	0.526

TABLE X. PREDICTION RESULTS AND COMPARISON PART2

Research	Metric Type	Precision	Recall	F-measure
Social Network	Dependency	0.789	0.625	0.697
	Contribution	0.764	0.688	0.724
	Combined	0.754	0.708	0.730
	Socio-technical	0.753	0.761	0.757
Mailing List	MCM	0.804	0.781	0.792
	MNM	0.763	0.746	0.754
	EM	0.775	0.762	0.768
	TM	0.713	0.748	0.730
	Combined	0.831	0.812	0.821

In summary, compared with some state-of-the-art approaches, our approach has the higher predictive capability in some aspects and achieves better result after combination. This suggests that, mail-related metrics are useful to be predictors of defect-proneness. On the other hand, we can

learn that the predictive effect is not exciting when using our email related metrics alone, but the model can be improved after taking some combination.

It is remarkable that the predictive effects can be different when the different email-related metrics are adopted. Considering that we only use the data from jdt-dev in prediction model and ignore the data from jwt-dev and eclipse-dev, the results are not obvious. But it is clear that the structural metrics provide better effects of prediction than the unstructured metrics. Unstructured text is the main part of emails and it contains more information. So it is expected to get better results from the unstructured information. The idea that taking unstructured text into consideration is affirmed, but the methods adopted should be improved, especially the topic model analysis method. The number of the topic and ranked words need to be obtained carefully and the definition of the topic metrics be adjusted. In summary, the results of defect prediction can be further improved.

D. Threats to Validity

In our design, we only analyze the files which are linked by the mailing lists. Since the number of mails that can be linked to some files may only be a small portion of all of the mails, it is noted that only a small part of emails are left after linking. It denotes that some defective files will be neglected during this phase. The recall must be influenced for this reason. On the other hand, our results are only valid for the eclipse project and may not be generalized to some other projects. Studies need to be further conducted in order to validate the results.

V. Conclusion

This paper has shown our exploration of the relationship between communication and software quality and of prediction of software defects through mining mailing lists. To reach this goal, we use a lightweight approach to linking mails collected from eclipse community to the corresponding source code. Then we apply three methods to analyze mails to investigate the effects of some metrics derived from mailing lists. We also make use of these metrics to predict the existence of defects in software.

Our results show that the developers' mailing lists may indicate software defects in some respects. First, software defects may be correlated with some specific structures of mailing lists. Second, developers' sentiment may relate to software quality. Third, Emails with some concerns are more likely to be defect-prone. Some metrics extracted from mailing lists can also be used to improve the prediction model.

Our future work is to improve the efficiency of mail analysis methods, such as the topic-based prediction method. Another future work is to evaluate and justify the approach using some large-scale software systems and taking empirical studies.

Acknowledgment

Beijun Shen is the corresponding author. This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242, 91118004, 61100051).

References

[1] Nachiappan Nagappan, Brendan Murphy, Victor Basili, "The influence of organization structure on software quality: an empirical case study," Proc. Intl Conf. on Soft. Engineering (ICSE 08), ACM, May. 2008, pp. 521-530, doi:10.1145/1368088.1368160.

[2] Timo Wolf, Adrian Schroter, Daniela Damian, Thanh Nguyen, "Predicting Build Failures using Social Network Analysis on Developer Communication," Proc. Intl Conf. on Soft. Engineering (ICSE 09), IEEE, May. 2009, pp. 1-11, doi:10.1109/ISSRE.2009.17.

[3] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, Arie van Deursen, "Communication in Open Source Software Development Mailing List," Proc. Mining Soft. Repositories (MSR 13), IEEE, May. 2013, pp.277-286, doi:10.1109/MSR.2013.6624039.

[4] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, Anand Swaminathan, "Mining email social networks," Proc. Mining Soft. Repositories (MSR 06), ACM, May. 2006, pp. 137-143 doi:10.1145/1137983.1138016.

[5] Alberto Bacchelli, Michele Lanza, Romain Robbes, "Linking e-mails and source code artifacts," Proc. Intl Conf. on Soft. Engineering (ICSE 10), May. 2010, pp.375-384, doi: 10.1145/1806799. 1806855.

[6] P. A.Wagstrom, J. D.Herbsleb, K.Carley. "A social network approach to free/open source software simulation." Proc. Intl Conf. on Open Source Systems (OSS 05), Jul. 2005, pp. 16-23, doi:10.1.1.178.4984.

[7] Martin Pinzger, Nachiappan Nagappan, Brendan Murphy, "Can Developer-Module Networks Predict Failures?" Proc. Intl Symp. on Foundations of Soft. engineering (SIGSOFT/FSE 08), ACM, Nov. 2008, pp. 2-12, doi:10.1145/1453101.1453105.

[8] Bettenburg, A.E. Hassan, "Studying the Impact of Social Structures on Software Quality," Proc. Intl Conf. on Program Comprehension (ICPC 10), IEEE, Jun. 2010, pp. 124-133, doi:10.1109/ICPC.2010.46.

[9] Tung Thanh Nguyen, Tien N. Nguyen, Tu Minh Phuong, "Topic-based Defect Prediction (NIER Track)," Proc. Intl Conf. on Soft. Engineering (ICSE 11), IEEE, May. 2011, pp. 932-935, doi: 10.1145/1985793.1985950.

[10] Bikram Sengupta, Satish Chandra, Vibha Sinha, "Hipikat: Recommending Pertinent Software Development Artifacts," Proc. Intl Conf. on Soft. Engineering (ICSE 06), IEEE, May. 2006, pp.408-418, doi: 10.1109/ICSE.2003.1201219.

[11] Alberto Bacchelli, Tommaso Dal Sasso, Marco D'Ambros, Michele Lanza, "Content Classification of Development Emails," Proc. Intl Conf. on Soft. Engineering (ICSE 12), IEEE, Jun. 2012, pp. 375-385, doi:10.1109/ICSE.2012.6227177.

[12] Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli, Sasa Nesic, "A Survey of Social Software Engineering," Proc. Intl Conf. on Automated Soft. Engineering (ASE/KBSE 08), IEEE, Sept. 2008, pp. 1-12, doi: 10.1109/ASEW.2008.4686304.

[13] Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, Michalis Faloutsos, "Graph-Based Analysis and Prediction for Software Evolution," Proc. Intl Conf. on Soft. Engineering (ICSE 12), IEEE, Jun. 2012, pp. 419-429, doi: 10.1109/ICSE.2012.6227173.

[14] Stacy K. Lukins, Nicholas A. Kraft, Letha H. Etzkorn, "Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation," Proc. Working Conference on Reverse Engineering (WCRE 08), IEEE, Oct. 2008, pp. 155-164, doi:10.1109/WCRE.2008.33.

[15] M. D'Ambros, M. Lanza, R. Robbes, "An extensive comparison of bug prediction approaches," Proc. Mining Soft. Repositories (MSR 10), IEEE, May. 2010, pp. 31-41, doi:10.1109/MSR.2010.5463279.

[16] Christian Bird, Nachiappan Nagappan, Harald Gall Brendan Murphy, Premkumar Devanbu, "Putting It All Together: Using Socio-technical Networks to Predict Failures," Proc. Intl Symp. on Soft. Reliability Engineering (ISSRE 09), IEEE, Nov. 2009, pp. 109-119, doi:10.1109/ISSRE.2009.17.

[17] Hideaki Hata, Osamu Mizuno, Tohru Kikuno, "Bug prediction based on fine-grained module histories," Proc. Intl Conf. on Soft. Engineering (ICSE 12), IEEE, Jun. 2012, pp. 200-210, doi:10.1109/ICSE.2012.6227193.