On Mining Concurrency Defect-Related Reports from Bug Repositories

Frank Padberg and Philip Pfaffe
Karlsruhe Institute of Technology KIT
Karlsruhe, Germany
frank.padberg@kit.edu

Martin Blersch
FZI Forschungszentrum Informatik
Karlsruhe, Germany
blersch@fzi.de

***Abstract*— We present early findings of two ongoing case studies in which we automatically extract reports about concurrency defects from the MySQL and Apache bug repositories. To mine the unstructured reports, we apply keyword search and machine learning, using linear and non-linear classifiers. We analyze the results in detail and suggest some improvements for this mining task.**

Automated Bug Report Classification; Software Reliability Modeling; Concurrency Defects

I. INTRODUCTION

This paper reports on two mining case studies in progress.

Concurrency defects are notoriously difficult to detect and costly to debug. Many concurrency failures crash or hang the system, see, f.e., [Lu08] [Fonseca10]. Development teams need to know how many concurrency defects to expect, and at what times in the lifecycle. In our research project QUALICORE [QualiCore], we develop novel techniques for the early automated detection of concurrency defects from design artifacts [Padberg13a] [Padberg13b]. In the project, we also develop statistical models for the reliability of multicore software – this is where the mining of unstructured software engineering data comes into play. In this paper, we report on our progress with exploring this mining task.

For reliability modeling, specific information about the distribution of failures over time in an application is required. Such information usually gets computed based on the time stamps of *bug reports*. As our object of study, we currently analyze the bug repositories of large, open-source applications. Since we focus on the reliability of the "parallel part" of an application, we want to screen just the *concurrency-related* defects and failures. That means, our reliability models take as input only reports about failures that are concurrency defect-triggered; reports about general programming defects are deliberately left out. Due to the large size of the repositories, the relevant reports must be *identified automatically* applying data/text mining techniques.

The classification problem for bug reports has been addressed in related work, f.e., [Fonseca10] [Liu13]. The topic of mining from bug repositories has also been addressed in a keynote at the first MUD workshop [Lo10]. See the section on related work for more details.

In our context, we face a number of technical challenges when extracting concurrency-related bug reports from a repository:

- The bug repositories for large concurrent applications typically contain (tens of) thousands of bug reports. Hence, the concurrency-related reports must be extracted automatically.
- Bug reports in practice hardly ever contain a defect classification field. Hence, whether a report is concurrency-related or not must be identified from the contents of the main body of the report, which contains a mixture of natural language text, questions and answers, code fragments, failure-inducing input, and links to other software sources.
- For reliability modeling, we need as many reports about concurrency defects as possible. On the other hand, some "noise" in the extracted data is tolerable since the statistical models will smooth the data to some extent. That is, the automated extraction should provide a high recall and a good precision.

In the remainder of the paper, we present – preliminary – results and findings from our attempt to extract concurrency-related defect reports at a large scale from the bug repositories of MySQL and Apache.

II. RELATED WORK

[Fonseca10] automatically extract bug reports from the MySQL repository that pertain to concurrency defects, by searching for typical concurrency-related keywords. After extensive manual filtering, they obtain 80 reports of well-documented concurrency bug. The goal of the study is to analyze this instructive sample of reports in detail and understand the problems that lead to concurrency defects. In contrary, our goal is to extract as many concurrency defect-related reports as possible, no matter whether a good failure description and patch are included or not.

[Liu13] classify certain types of bug reports automatically, as the initial step for generating patches automatically. The study limits itself to three types of bugs: buffer overflows, null pointers, and memory leaks. They label a set of representative reports manually with the bug type, then apply machine learning to train a separate classifier for each bug type. Trained

on Linux kernel and Mozilla reports, the classifiers seem to achieve a good precision (80 per cent) when applied to Apache reports; the recall seems to be low, though (about 50 per cent). Due to lack of technical detail, the experimental classification results of the study cannot be reproduced.

[Lo10] also addresses text mining from bug reports in a keynote at the first MUD workshop. The goal is to identify duplicate reports based on textual similarities. The keynote makes a number of interesting observations, including: Bug reports suffer from incomplete and badly structured sentences; bug reports use special technical keywords; there often are different ways of describing the same issue; bug reports contain a mixture of content types; there is a variety of bug types each having its own characteristics. Some recommendations are: Apply robust text mining techniques; use parsers to split the contents according to type; use composite features. We arrive at similar findings in our case studies, see below.

III. MYSQL

The MySQL bug repository has grown to double its size since the [Fonseca10] study was published. It currently contains more than 25,000 closed reports. Considering closed reports only is common and ensures that defects are confirmed and descriptions are stable. For the purpose of reliability modeling, we aim at extracting as many concurrency-related reports as possible.

Henceforth, we shall focus on the subset of *closed* reports concerning the MySQL *server* versions, and simply call this "the repository."

A. Extraction Approach

To extract concurrency-related bug reports, [Fonseca10] searched the report bodies for keywords that are typically used when describing concurrency defects. The set of keywords was established in a trial-and-error approach. As stated in the study, the following keywords were included (among others that were not specified in the study):

lock, acquire, compete, atomic, concurrency, synchronization, etc.

We took this list as a starting point and applied the same keywords to the repository. Then, we randomly sampled from the reports extracted by the search ("hits") and checked manually whether they actually were concurrency-related. We found that quite often this was not the case, despite the match. In addition, we knew about certain concurrency defect-related reports that were not found by the keywords.

Hence, we iteratively refined the set of keywords in order to achieve a higher precision and recall. This is the list of search terms that we currently are using:

acquire(s) + lock, wrong + lock(ing), missing + lock, compete, atomic, concurrency, synchronization, "race condition(s)", deadlock(ed), concurrent, mutex, "read lock", "write lock"

Our search terms extracted 558 bug reports from the repository. Each report refers to one or more MySQL versions: Since the different versions often are based on common source code modules, many defects are relevant for several versions. Table 1 shows the number of hits for each version.

Table 1. Number of defect reports extracted ("hits") for MySQL

Version	Hits	Version	Hits
all	558	5.2	2
3.23	3	5.4	14
4.0	34	5.5	145
4.1	52	5.6	91
5.0	130	5.7	21
5.1	185	6.0	200

Some keywords, such as "concurrent", "deadlock", or "mutex", are more yielding than others, such as "atomic". Table 2 shows the number of hits for each keyword (note that for a number of reports, more than one keyword matched).

Table 2. Number of hits for the different keywords (MySQL)

Keyword(s)	**Hits**	**Keyword(s)**	**Hits**
all	558	synchronization	20
+acquire(s) +lock	11	"race condition(s)"	40
+wrong +lock(ing)	54	deadlock(ed)	142
+missing +lock	7	concurrent	161
compete	1	mutex	122
atomic	19	"read lock"	67
concurrency	19	"write lock"	26

B. Precision

We checked a random sample of about ten per cent of the extracted reports manually and found that the majority actually were concurrency-related. We are in the process of checking all 558 hits. Table 3 specifies our findings on precision for each MySQL version, as available so far. It seems that the precision of the search ranges between 70 and 85 per cent.

Table 3. Overall precision for some MySQL versions

MySQL Version	Hits	Concurrency-related	Precision
3.23	3	3	100%
4.0	34	29	85%
4.1	51	36	69%
5.0	130	93	71%
5.1	185	138	74%

For the versions 4.0, and 4.1, we checked how efficient the individual keywords are. Table 4 shows the results for those keywords that had less than 100 per cent precision.

Table 4. Keyword precision for some MySQL versions

Keyword(s)	**4.0**	**4.1**	**Keyword(s)**	**4.0**	**4.1**
+wrong +lock	67%	67%	mutex	100%	64%
+wrong +locking	50%	0%	"read lock"	80%	86%
deadlock	78%	75%	"write lock"	100%	67%
concurrent	100%	70%			

It seems that certain keywords require post-processing in order to increase their precision. In particular, the combination of "lock(ing)" and "wrong" seems prone to wrong hits. We are looking into this. In addition, for C/C++ code such as MySQL, "mutex" seems to occasionally appear as a parameter name in the code; similar for "concurrent". Such peculiarities should be

factored in when classifying bug reports automatically based on keywords.

C. Some Wrong Hits

We analyzed all 15 wrong hits for MySQL 4.1, to find out why exactly the keywords were misleading.

- In 6 cases, the keyword appeared as part of the code listed in the report, f.e., as a call parameter, a configuration parameter, or a MySQL command.
- In 4 cases, the keyword appeared in the code as an error string, or in the debugger output given in the report.
- In 1 case, the keyword appeared as a test case name mentioned in the report.
- In the remaining 4 cases, the keyword did appear in the bug report in some other way, but the defect was actually not related to a concurrency issue.

This preliminary analysis provides some hints how splitting the body of a bug report according to content type – such as code snippet, failure-inducing input, failure description, and defect cause explanation – might improve the precision of the search. For classifying email contents, this approach is pursued in [Bacchelli12]. Considering the content type is in line with the suggestions given in [Lo10], and subject to further study.

D. Recall

Currently, we cannot make any substantiated statement about the recall of the keyword search. The MySQL bug repository is way too large to check this manually.

Unless development teams are willing to spent more effort on classifying defects when reported during development or maintenance, there seems to be little that research can do about estimating the ecall in large repositories.

For our primary purpose of reliability modeling, the situation seems not too bad, though: We expect that we can build useful reliability models already from a sufficiently large extract of the set of all concurrency defects.

IV. APACHE

A. Search-based Approach

Keyword-based search worked fairly well on the MySQL repository. We tried the same approach on the Apache repository, which contains almost 15,000 closed reports. For Apache, we used the following – slightly adapted – set of keywords:

deadlock, race condition(s), atomic, lock(s), locked, locking, unsynchronized, threading, synchronization, (multi)threaded, concurrency, mutex, atomicity.

515 hits were reported by the keyword search. This set of hits includes reports that actually were *not* concurrency-related, despite the match. For a random sample of 60 reports drawn from the hits, we had a low precision, below 50 per cent.

It seems that our concurrency-related keywords are less discriminative for Apache reports than for MySQL reports. We took a closer look at the Apache reports and identified several *potential* reasons for this:

- The Apache project has a more complex software structure than MySQL. Hence, identifying and describing the root cause for a failure is more difficult.
- The bug reports are filed not only by developers, but also by end users, as opposed to the MySQL project. Typically, it is hard for end users to describe the symptoms and runtime conditions of a failure precisely. Hence, the reports tend to be less specific.
- Developers who fix a defect don't always add as much technical information to user-reported defects as to developer-reported defects.
- The Apache project spans several programming languages and paradigms. The terminology that is being used in the reports often reflects the special terms of the language, not the general terms.

It might be helpful to tailor the keywords to the language (natural and technical) used by the authors of the report. This would require identifying, f.e., the programming language in the code snippets, and identifying whether the initial author of the report is a developer or an end user.

B. Voting-based Approach

Similar to MySQL, we often found that more than one concurrency-related keyword matched an Apache bug report. Hence, we tried a simple voting scheme: A report was marked as a hit, iff two or more keywords matched. Exploring this scheme on a sample of 30 Apache bug reports drawn randomly from the 515 reports retrieved by the keyword search, the precision increased to 60 per cent, but no more. We took this as a suggestion that it might make sense to try a learning-based approach, which would allow for more than one text fragment (keyword) as input for the classification.

C. Learning-based Approaches

To increase the precision, we are experimenting with *augmenting* the keyword search with machine learning-based approaches; that is, we aim at a two-stage extraction process. From the 515 (true and false) hits that resulted from the search, we randomly sampled 81 reports and subdivided them, roughly 2:1, into a training set and a test set, then trained linear and non-linear classifiers, using different sets of features.

Linear classifier – standard features. We first tried a linear classifier using unigrams, bigrams, and the corresponding relative term frequencies as features. This choice was inspired by standard text mining approaches [Feldman07]; we were aware that this would result in a large number of features. We implemented a linear "balanced winnow" classifier, which is known to be robust against having a large set of irrelevant features [Littlestone87]. Based on the [WEKA] winnow defaults, we set α=1.1, β= 0.9, and θ=1; the initial weights were set equal to $2\cdot\theta$ divided by the average length of the reports. We trained the classifier for 100 cycles. 57 reports were used

for training and 24 reports for testing. In the training set (testing set), 35 (17) reports were concurrency defects, 22 (7) were not.

The results were best when using the relative term frequencies as features. We achieved a precision of 77 per cent, but a recall of only 59 per cent on the test data – bearing in mind that we apply learning on top of the keyword search, this value is low. We are in the process of analyzing the results in more detail; f.e., we are checking which features receive a significant weight in the classifier.

Linear classifier – custom features. Next, we drastically reduced the set of features, trying to tailor them to the problem *before* training the classifier. We started from our set of keywords, the rationale being that a classifier can weigh and combine the keywords, contrary to a simple search. We ran a staged series of experiments, using the following features:

(i) binary values that indicate whether the keywords matched or not; (ii) the absolute frequencies of the keywords; (iii) the absolute frequencies of the keywords, plus the size of the bug report; (iv) the relative frequencies of the keywords.

We used the same linear classifier, training set, and test set as before. From the learning curves (not shown), 100 training cycles apparently were sufficient. Table 5 shows the precision and recall on the test data.

Table 5. Precision and recall for different feature sets (Apache)

Features	**Linear Class.**		**Neural Net**	
	Prec	Rec	Prec	Rec
binary (i)	80%	47%	74%	76%
absolute freq. (ii)	91%	64%	83%	59%
abs. freq. + size (iii)	88%	47%	88%	57%
relative freq. (iv)	78%	64%	81%	69%

The absolute frequencies performed best; their precision is good, yet, their recall is not. A comparison of (iii) and (iv) indicates that it is difficult to learn relative frequencies by adding the report size as a feature to the absolute frequencies.

Non-linear classifier. Finally, we quickly tried a non-linear classifier (neural net [WEKA] [Hall09]) on the same data set of 81 reports. We wanted to gain insight into the question whether some features carry non-linear information about the target.

We used a default configuration for the neural net, as provided by the library function: one hidden layer with 9 neurons, and backpropagation with a learning rate of 0.3 and a momentum of 0.2. The subdivision into training and test set was done automatically (3-fold cross-validation) by the library. Table 5 shows the precision and recall values for the neural net, using the same feature sets as before.

The maximum precision that we achieved on the test data was close to 90 per cent, but with a low recall below 60 per cent. Overall, as opposed to the linear classifier, the binary features provided the best balance between precision and recall with the neural net. We are in the process of analyzing these results in detail, in particular, how the features are being weighed by the classifier. This is ongoing work.

V. CONCLUSIONS

We presented early findings of two – ongoing – case studies that aim at extracting the concurrency-related defect reports from a large bug repository, automatically. The case studies are part of our research on reliability modeling for multicore software. From the perspective of mining unstructured software data, we consider this to be an instructive application domain, because the requirements on the precision and recall seem manageable.

We applied a search-based approach using specific concurrency keywords with some success. We found that some keywords work better than others, that the keywords need to be adapted to the repository somewhat, and that some keywords require post-processing for a good precision.

To improve the precision, we currently are exploring learning-based approaches. Initial results are interesting and encouraging, but we think that we need to better tailor the feature set and learning technique to the problem.

We are looking forward to the discussions and advice from the data mining experts at the MUD workshop.

ACKNOWLEDGMENT

The QUALICORE project is supported by research grant no. 01|S11011 from the German Federal Ministry of Science and Education BMBF.

REFERENCES

[Bacchelli12] Bacchelli, A., Dal Sasso, T., D'Ambros, M., and Lanza, M., "Content classification of development emails," 34th Int. Conf. on Softw. Eng. (ICSE'12), 375-385.

[Feldman07] Feldman, R., and Sanger, J., *The Text Mining Handbook*, Cambridge University Press, 2007.

[Fonseca10] Fonseca, P., Li, C., Singhal, V., and Rodrigues, R., "A study of the internal and external effects of concurrency bugs," Int. Conf. on Dependable Systems and Networks (DSN '10), 221-230.

[Hall09] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., "The WEKA Data Mining Software: An Update," SIGKDD Explorations Newsletter, vol. 2, no. 1 (2009), 10-18

[Littlestone87] Littlestone, N., "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," Machine Learning, vol. 2, no. 4 (1987), 285-318

[Liu13] Liu, C., Yang, J., Tan, L., and Hafiz, M., "R2Fix: Automatically Generating Bug Fixes from Bug Reports," 6th Int. Conf. on Softw. Testing, Verification and Validation (ICST'13), 282-291.

[Lo10] Lo, D., "Mining Execution Traces and Bug Reports: Challenges and Opportunities," 1st Int. Workshop on Mining Unstructured Data at WCRE (MUD'10), Keynote Presentation.

[Lu08] Lu, S., Park, S., Seo, E., and Zhou, Y., "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08), 329–339.

[Padberg13a] Padberg, F., and Denninger, O., "Multicore-Softwarefehler im Visier. Automatische Fehlererkennung in Entwürfen paralleler Programme," Objektspektrum, vol. 20, no. 1 (2013), 72-76

[Padberg13b] Padberg, F., Carril, L. M., Denninger, O., and Blersch, M., "On Detecting Concurrency Defects Automatically at the Design Level," 20th Asia Pacific Softw. Eng. Conf. (APSEC '13), accepted.

[QUALICORE] http://www.qualicore-projekt.de/

[WEKA] http://www.cs.waikato.ac.nz/ml/weka