# A Novel Approach for Predicting the Probability of Inconsistent Changes to Code Clones Based LDA

**Lili Yin, Liping Zhang, Min Hou, Dongsheng Liu**
Computer and information engineering college Inner Mongolia normal university, Hohhot, China
yinliligood@126.com

**Abstract -** Inconsistent changes to code clones can create faults and, hence, lead to incorrect program behavior. Consequently, these clones increase the change effort when software is maintained. In order to improve software quality and to help programmers pre attention the hidden trouble of clone inconsistent changes. In this paper, Different from previous research, we predict the probability of inconsistent changes to clones based LDA. This paper expands the LDA (Latent Dirichlet Allocation) model application fields. The experiment on a large open source software system is presented. Experimental results show the feasibility of this technique.

**Index Terms -** Predicting Probability, Inconsistent Changes, LDA

## 1. Introduction

Research in software maintenance has shown that many programs contain a significant amount of duplicated (cloned) code. Such cloned code is considered harmful for two reasons: (1) multiple, possibly unnecessary, duplicates of code increase maintenance costs and, (2) inconsistent changes to cloned code can create faults and, hence, lead to incorrect program behavior [1-2].

To shed light on the situation, we investigated the effects of code cloning on program correctness. It is important to understand, that clones do not directly cause faults but inconsistent changes to clones can lead to unexpected program behavior. A particularly dangerous type of change to cloned code is the inconsistent bug fix. If a fault was found in cloned code but not fixed in all clone instances, the system is likely to still exhibit the incorrect behavior. To illustrate this, Fig. 1 shows an example, where a missing null-check was retrofitted in only one clone instance [3].

Previous studies were detecting inconsistent changes to code clones. Researchers calculated how many code clones are inconsistent changes in multiple versions of software. At the same time they proved that the inconsistent changes to clones is harmful to developers and maintainers. Therefore, it is important to predict the probability of inconsistent changes to code clones, in order to improve software quality and to help programmer pre attention the hidden trouble of clone inconsistent changes.

In this paper, we implement a mapping of multiple versions of the clone group to obtain a clone group evolution. Source code files which include code clones will be extracted according to the evolution history of each clone group. Then using the LDA model identifies themes of these source code files and judge the stability of file functions. Finally, we predict the probability of inconsistent changes to code clones.

Contributions of this paper:

- Unlike existing studies that researchers only consider a single feature of the code clones. The code clones in this paper are placed in the context and we enrich code clones feature information.
- LDA model is applied to the field of code clones for the first time. Simultaneously, we quantify the evolution information of code clones and predict the probability of inconsistent changes to code clones and provide data for our follow up predicting harmfulness of code clones.

## 2. Related Work

Hindle et al.apply the Link model to commit log messages in order to see what topics are being worked on by developers at any given time[4]. The authors apply the Link model (based on LDA) to a collection of commit logs over a period of 30 days, then link topics from successive periods using an 8-out-of-10 top-term similarity measure (i.e., if at least 8 of the 10 top words for a topic at period i are shared by a topic at period i+ 1, then the topics are considered the same). The authors find LDA to be useful in identifying activity trends and present several visualization techniques to understand the results.

Linstead et al. use the Hall model (based on LDA) to analyze source code evolution, claiming that LDA provides better results than LSI [5]. The authors present line plots of topic assignment percentages over time for two systems, Eclipse and ArgoUML. These plots reveal integration points and other changes that shape a project's lifetime. We build on this work by formalizing the approach, considering additional topic metrics to better understand topic change events, and providing a detailed, manual analysis of the topic change events to validate and characterize the results of the approach.

Above knowable, the current researchers don't pay much attention to the study predicting the probability of inconsistent changes to code clones.

Natural Science Foundation of Inner Mongolia (No.2011MS0906)
The National Natural Science Foundation of China (No.61363017)

 

```
Public String showElements(ModelElement[] elements,String nomsy){
  Boolean found = false;
  StringBuffer res=new StringBuffer();
  If(elements!=null){
     Index.getInstance().setCurrentRenderer(
       FlatReferenceRenderer.getInstance());
     For(int i=0;i<elements.length;i++){
       ModelElement el=elements[i];
       Res.append(showElementlink(el)).append(HTML.LINE_BREAK);
       Found = true;
     }
     Index.getInstance().resetCurrentRenderer();
  }
  If(found&&nomsg!=null&&nomsg.length()>0){
     Res.append(HTML.italics(nomsg));
  }
  Reeturn res.toString();
```

```
Public String showElements(ModelElement[] elements,String nomsy){
  Boolean found = false;
  StringBuffer res=new StringBuffer();
  If(elements!=null){
     Index.getInstance().setCurrentRenderer(
       FlatReferenceRenderer.getInstance());
     For(int i=0;i<elements.length;i++){
       ModelElement el=elements[i];
       Res.append(showElementlink(el)).append(HTML.LINE_BREAK);
       Found = true;
     }
     Index.getInstance().resetCurrentRenderer();
  }
     If(found && monsg.length()>0){
    Res.append(HTML.italics(nomsg));
  }
  Reeturn res.toString();
```

Figure 1 Missing null check on right side can cause exception

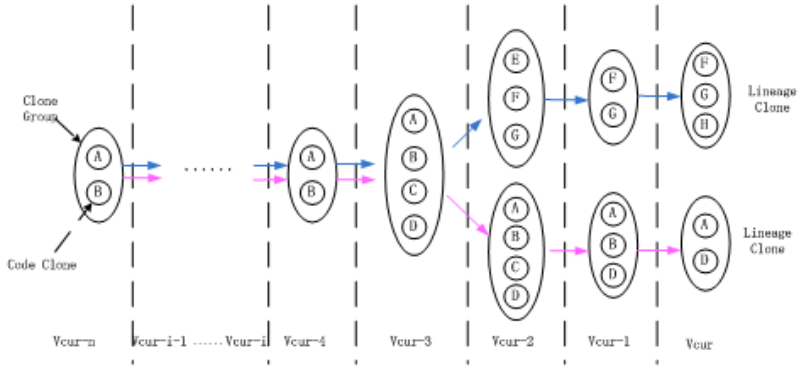


Figure 2 Adjacent version clone group mappings.

## 3. Study Approach

First of all, we build clone group mapping for multiple versions of software on the basis of existing clone detection tool. The specific mapping relationship is shown in Figure 2. Each branch in the figure constitutes a lineage clone. Lineage clone is a directed acyclic graph, which describes a clone group's immediate evolution history that has no branch. Next, source code files which include code clones will be extracted according to the evolution history of each clone group. Then using the LDA model identifies themes of these source code files. Finally, we predict the probability of inconsistent changes to code clones by calculating information entropy etc.

### *A. Building Clone Group Mapping*

The construction of cloning group mapping is primarily on the basis of calculating text similarity. The text similarity between two code snippets C1 and C2 is determined by calculating the common tokens sequence with respect to their token sizes. By considering tokens generated by FCD [6], we count the textual matches across versions. Equation (1) below describes the *TextSimilarity* function. Here $|C_1|$ and $|C_2|$ are the token sizes of code snippets of C1 and C2 respectively. $|C_1 \cap C_2|$ is the size of common ordered tokens between C1 and C2, calculated using the longest common subsequence (LCS) algorithm. We used a text similarity heuristic of 0.8.With this similarity threshold, the length and size of the genealogies are neither overestimated nor underestimated.

$$TextSimilarity(C_1, C_2) = \frac{2|C_1 \cap C_2|}{|C_1| + |C_2|} \quad (1)$$

Because code clones to convert into token is a lexical analysis process. Sometimes, the tokens that two code clones are not the same may be similar. So the text similarity score itself is not always enough to get better result. In snippet matching, on the other hand, we match the snippets based on the similarity of identifiers. Because the clone detection tools we used only detects clone group of type-1 and type-2. So each clone group contains almost exactly the same code snippets. Therefore, we can random extract a code fragment to a clone group. According to the following formula calculated the snippet matching.

$$SnippetMatching(S_i, S_j) = \left\{ \frac{LCS(S_i, S_j)}{len(S_i)} + \frac{LCS(S_i, S_j)}{len(S_j)} \right\} \Big/ 2 \quad (2)$$

*LCS* ($S_i$, $S_j$) represents the length of the longest common subsequence for two cloned fragments. *len* ($S_i$) represents the length of cloned fragment $S_i$. If the matching scores greater than the threshold 0.3, we establish mappings for the two clones group. On the contrary, we do not establish a mapping relationship.

### *B. Using LDA Extract the Themes to Set of Lineage Clone Group*

Source code files which include code clones are extracted from a lineage clone that originated in the same group. These files are defined as the set of lineage clone

group. Using LDA model extract the themes to set of lineage clone group. Then we get the probability of each theme in each file.

To apply LDA on code clone, we consider a software system as a collection of documents and each document is associated with a set of concepts (i.e., topics). The mapping between LDA model and code cloning entities is shown in Table 1.

TABLE 1 Mapping LDA to Code Cloning

| LDA Model | Code Cloning Entities |
|---|---|
| word | Identifiers and comments extracted from source code, which comprise the vocabulary set |
| document | source code files of a clone code |
| set of documents | source code files are included in a clone group. |
| corpus | set of lineage clone group |

For the LDA computation, we used MALLET version 2.0.6 [7]. MALLET is a highly scalable Java implementation of the Gibbs sampling algorithm. We ran for 1,000 sampling iterations, the first 100 of which were used for parameter optimization [8]. We allowed MALLET to use hyper-optimization for α and β input parameters, which are smoothing parameters for the model.

For any given corpus, there is no provably optimal choice for K [9]. The choice is a trade-off between coarser topics (smaller K) and finer-grained topics (larger K). Setting K to extremely small values results in topics that contain multiple concepts (imagine only a single topic, which will contain all of the concepts in the corpus!), while setting K to extremely large values results in topics that are too fine to be meaningful and only reveal the idiosyncrasies of the data. Our goal in this study is to discover topics of medium granularity, so we seek a non-extreme value for K.

Appendix A provides a brief replication guide for our study.

### C. *Predicting the Probability of Inconsistent Changes to Code Clones*

We quantify the evolution information of code clones by calculating the average probability, entropy, etc. Then we predict the probability of inconsistent changes to code clones. The following example in Figure 3 introduced attribute values need to calculate.

Let us consider the following example of a group of cloning, which consists of two documents implementing three different topics (see Figure 3).


Figure 3 Probability distributions of three topics in two files

We can observe that the second topic (vertical filling line pattern in Figure 3), which is relevant to the first document with probability pt=0.7 and to the second document with probability pt=0.5, is the dominating topic in the documents (as compared to the other two topics).

*(1) Average weight(AVG_w)*

In this paper, the weight value is defined as the average probability of each topic in the selected file. Average weights are important parameters to calculation maximum weight entropy in the next. Average weight is calculated as follows:

$$AVG\_w(t_i) = \frac{\sum_{d=1}^{n} p_{t_i}^{d}}{n} \tag{3}$$

The $p_{t_i}^{d}$ is the probability of topic $t_i$ in a document d. Its value can be gained by the LDA model. $n$ is the number of selected files, in our example (figure 2): $AVG\_w(t_1) = (0.1+0.2)/2 = 0.15$, $AVG\_w(t_2) = 0.5$.

*(2 )Topic_Distribution(TD)*

Topics distribution is the entropy. Entropy essentially reflects the degree of uncertainty. The higher entropy of a theme is, the more similar probability value of the topics in each file is. The probability distribution of topics in the file can reflect the relation degree of file functions. Therefore, through the entropy represent related degree between files function. In order to compute entropy for a topic $t_i$ we need to transform document-topic probability distributions $P_t$ to topic-document distributions $q_{t_i}$ as the following:

$$q_{t_i}^{d_j} = \frac{p_{t_i}^{d_j}}{\sum_{d=1}^{n} p_{t_i}^{d}} \tag{4}$$

$d_j$ is the j-th file, in our example (figure 3), $q_{t_3}^{1} = 0.2/(0.2+0.5) = 0.286$, $q_{t_3}^{2} = 0.5/(0.2+0.5) = 0.714$.

The distribution of the topic $t_i$ is calculated as follows:

$$TD(t_i) = \frac{\sum_{d=1}^{n} -q_{t_i}^{d} \times \log(q_{t_i}^{d})}{\log n} \tag{5}$$

In our example, $TD(t_1)$=0.92, $TD(t_2)$=0.88, $TD(t_3)$=0.86.

*(3) File Function Correlation(FFC)*

Each file has more than one topic. The entropy of every topic reflects the probability distribution of the topic in the file. If relevant degree value is higher, the file function is more similar. Conversely, the function of these files is more dispersed. It also means that clones in the files are scattered. Computation formula is as follows:

$$FFC = \sum_{i=1}^{|t|} TD(t_i) \tag{6}$$

$|t|$ is the number of extracting topics, in our sample, $FFC$=0.92+0.88+0.86=2.66, $0 \le FFC \le |t|$.

*(4) Maximal Weighted Entropy(MWEwe)*

At last we need to further calculate the maximum weighted entropy to obtain the main function. Because the choice of each documents set is carried out in the adjacent version and the value of two files' sets which most relevant

functions are different is similar. The FFC value of two sets of files is similar, but its most relevant functions may be different. So we can not judge the stability of file's functions just by FFC values in the process of version evolution. Considering this situation, need to calculate MWE:

$$MAX_{we} = \max_{1 \le i \le |t|}(AVG\_w(t_i) \times TD(t_i)) \quad (7)$$

$|t|$ is the number of extracted topics, in our sample, $MAX_{we} = \max\{0.15 \times 0.92, 0.5 \times 0.88, 0.35 \times 0.86\} = 0.440$.

According to the step B, we obtain the probability for each topic in each file. Then we can calculate the Average weight, Topic-Distribution and File Function Correlation, Maximal Weighted Entropy. The FFC value reflects file functions' degree of similarity. The higher *FFC* value is, the more similar the two documents function is. If source code files which include code clones change frequently in the evolution history, the *FFC* values we calculate will be small. This shows that these code clones have high flexibility and the probability of inconsistent changes to code clones will be high. The probability is calculated as follows:

$$probability = \sum_{i=1}^{n} \frac{1}{n} \times f(FFC_i) \quad (8)$$

$$f(FFC_i) = 1 - \frac{1}{|t|} FFC_i \quad (9)$$

We need to explain that every set of lineage clone group has been repeated step B and C after step A.

## 4. Experiment and Result Analysis

*A. Experiment*

We chose open source software Bluefish to complete the approach proposed in this paper, because the size of the software appropriate to verify experiment results by manual, as shown in table 2. Lines of clones are counted according to detect function clones. The number of clone groups is statistical results that filter out code clones less than five lines.

TABLE 2 Experimental Systems

| Software Title | Soft Size | Lines of Clones | Number of Clone Groups |
|---|---|---|---|
| bluefish-2.2.4 | 66459 | 3478 | 56 |
| bluefish-2.2.3 | 65731 | 3461 | 56 |
| bluefish-2.2.1 | 63712 | 3622 | 53 |
| bluefish-2.0.3 | 57168 | 2069 | 33 |
| bluefish-2.0.0 | 53160 | 1710 | 24 |
| bluefish-1.0.7 | 50786 | 14241 | 21 |
| bluefish-1.0.6 | 50778 | 14241 | 21 |
| bluefish-1.0.5 | 50741 | 14241 | 21 |

Table 2 shows the input data of this experiment. Next, we construct cloned group mappings. With the last version (bluefish-2.2.4) clone group is starting to move back mapping process. The mapping results are shown in Table 3. Each row is the clone group id number in each version and each column is the evolution of clone groups and "-" is the end of the evolution process. For example, bluefish-2.2.4 clone group No.0 is evolved from bluefish-2.2.3 clone group No.0.

TABLE 3 Clone Groups Mapping Between Versions

| Software Title | Clone Group ID | | | | |
|---|---|---|---|---|---|
| bluefish-2.2.4 | 0 | 1 | 2 | 3 | ..... |
| bluefish-2.2.3 | 0 | 1 | 2 | 3 | ..... |
| bluefish-2.2.1 | 0 | - | 1 | 2 | ..... |
| bluefish-2.0.3 | 0 | - | 1 | 2 | ..... |
| bluefish-2.0.0 | 0 | - | 0 | 1 | ..... |
| bluefish-1.0.7 | - | - | - | 0 | ..... |
| bluefish-1.0.6 | - | - | - | 0 | ..... |
| bluefish-1.0.5 | - | - | - | 0 | ..... |

Next, using LDA model extracts themes to set of lineage clone group. Figure 4 is a portion of themes' information extracted. We can see that "topic id=0"means detailed information of topic 0. Table 4 is the probability of each theme in each file. We assuming five themes were extracted.

We got the probability of inconsistent changes to code clones by calculating information entropy etc. according to the probability of each theme in each file, as shown in table 5. 0.00023 in the table is the probability of inconsistent changes to code clones that have mapping relationship between bluefish-2.2.1 and bluefish-2.2.3. In this way, we obtain the probability of inconsistent changes that each clone group of bluefish-2.2.4 traces evolution every step. Table 5 selected clone groups of long evolution history to show the results, because clone group NO.1 and NO.4 of bluefish-2.2.4 just have two versions evolution information. They are too short and will be filtered out.

*B. Result Analysis*

We can see that the probability of inconsistent changes have hardly any greater than 0.6. On the one hand, because we have chosen some small software and most of all clone groups containing codes are from the same source file. So the probability of files change is small in the evolution. On the other hand, because even if some source code file that is contained by one clone group has changed in the evolution, according to the formula (8) shows that the value of results to improve the range is not large. Although the final probability scores are almost all less than 0.5, the size discrimination of inconsistent changes is quite obvious. So they will not affect the final result of the experiment.

## 5. Conclusion

In this paper, we propose an approach that can predict the probability of inconsistent changes to code clones for the first time. LDA is also first applied to the code clone. We extend application fields of LDA model. This method is based on the detection result of clone groups. First of all, according to the text similarity construct clone groups mapping cross versions. Secondly, we extract themes to set of lineage clone group with the LDA model and get the probability of each theme in each file. Finally, we predict the probability of inconsistent changes to code clones by applying the theory of information entropy and calculating average probability etc.. In the future, we will study predicting harmfulness of code clones according to the experimental results.


Figure 4 Identification file topics sample

TABLE 4 Topic Probability in Each File Sample

| File | Topic0 | Topic1 | Topic2 | Topic3 | Topic4 | Topic5 |
|---|---|---|---|---|---|---|
| file:/e:/test/10-1qpel.c | 0.00002 | 0.00002 | 0.00001 | 0.4697 | 0.00002 | 0.5303 |
| file:/e:/test/10-2dsputil.c | 0.122 | 0.3487 | 0.0355 | 0.00009 | 0.1306 | 0.3632 |
| file:/e:/test/11-1qpel.c | 0.00001 | 0.00001 | 0.00001 | 0.4718 | 0.00001 | 0.5281 |
| file:/e:/test/11-2dsputil.c | 0.1176 | 0.354 | 0.0348 | 0.00009 | 0.1319 | 0.3616 |
| file:/e:/test/8-1qpel.c | 0.00001 | 0.00001 | 0.00001 | 0.4632 | 0.00001 | 0.5367 |
| file:/e:/test/8-2dsputil.c | 0.1278 | 0.36 | 0.0416 | 0.0001 | 0.1161 | 0.3543 |
| file:/e:/test/9-1qpel.c | 0.0001 | 0.00001 | 0.00001 | 0.4614 | 0.00001 | 0.5384 |
| file:/e:/test9-2dsputil.c | 0.1231 | 0.3503 | 0.0358 | 0.0003 | 0.1271 | 0.3635 |

TABLE 5 the Probability of Inconsistent Changes to Code Clones

| Clone groups of final version (bluefish-2.2.4) | Version of bluefish | | | | | | |
|---|---|---|---|---|---|---|---|
| | 2.2.3 | 2.2.1 | 2.0.3 | 2.0.0 | 1.0.7 | 1.0.6 | 1.0.5 |
| clone group NO.0 of bluefish-2.2.4 | 0.00024 | 0.00023 | 0.00217 | 0.00351 | - | - | - |
| clone group NO.2 of bluefish-2.2.4 | 0.00058 | 0.00043 | 0.00095 | 0.00189 | - | - | - |
| clone group NO.3 of bluefish-2.2.4 | 0.07068 | 0.21819 | 0.00023 | 1 | 0.47028 | 0.09523 | 0.08917 |
| clone group NO.5 of bluefish-2.2.4 | 0.06427 | 0.00560 | 0.10010 | 0.04304 | 0.20096 | 0.01171 | 0.00062 |
| ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... |

## Appendix a Replication Guide

For the sake of completeness, we include a guide for replicating our study.

1. Collect source code data. This can be performed by either checking out copies from the open source software.

2. Preprocess the data. Isolate source code identifiers and comments. Split the words based on common naming schemes. Convert all letters to lower case. Remove stop words. Stem each word. Remove overly common words (those that appear in more than 80% of the documents) and rare words (less than 2%).

3. Transform the data into MALLET format. If input-dir is the name of the top-level directory containing the preprocessed source code documents, and ${MALLET-BIN} is the path to the MALLET executable, then the command

${MALLET-BIN} import-dir --input input-dir --output data. mallet --keep-sequence

will create the output file data. mallet.

4. Discover the topics. Run the command

${MALLET-BIN} train-topics \
--input data. mallet --num-topics K \
--num-iterations 10000 --optimize-burn-in 1000 \
--optimize-interval 100 \
--output-doc-topics allfiles.txt \
--output-topic-keys topics.dat \
--xml-topic-phrase-report topic-phrases.xml

substituting the number of topics, iterations, etc. as desired.

## References

[1] R.Koschke. Survey of research on software clones. In Duplication, Redundancy, and Similarity in Software. Dagstuhl Seminar Proceedings, 2007.

[2] C. K. Roy and J. R. Cordy. A survey on software clone detection research. Technical Report 541, Queen's University at Kingston, 2007.

[3] Juergens E, Deissenboeck F, Hummel B, et al. Do Code Clones Matter? //Proc. of the 31st International Conference on Software Engineering. IEEE press, pp. 485-495, 2009.

[4] Hindle, M.W.Godfrey, R.C.Holt, What's hot and what's not: windowed developer topic analysis, in: Proceeding soft the 25th International Conference on Software Maintenance, 2009, pp. 339–348.

[5] E. Linstead, C. Lopes, P. Baldi, An application of latent Dirichlet allocation to analyzing software evolution, in: Proceedings of the 7th International Conference on Machine Learning and Applications, 2008, pp. 813–818.

[6] Qingqing Shi, Liping Zhang, Fanjun Meng, Dongsheng Liu, A Novel Detection Approach for Statement Clones, 2013 4th IEEE International Conference on Software Engineering and Service Science, May 23-25, 2013, Beijing, China, 2013.

[7] A.K.McCallum, Mallet: a machine learning for language toolkit. URL http://mallet.cs.umass.edu, 2002.

[8] T.L. Griffiths, M. Steyvers, Finding scientific topics, Proceedings of the National Academy of Sciences 101 (2004) 5228–5235.

[9] H.M. Wallach, I. Murray, R. Salakhutdinov, D. Mimno, Evaluation methods for topic models, in: Proceedings of the 26th International Conference on Machine Learning, 2009, pp. 1105–1112.

[10] Liu Y, Poshyvanyk D, Ferenc R, et al. Modeling class cohesion as mixtures of latent topics//Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. IEEE, 2009: 233-242.

[11] Thomas S W, Adams B, Hassan A E, et al. Studying software evolution using topic models. Science of Computer Programming, 2012.

[12] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation, Journal of Machine Learning Research 3 (2003) 993–1022.