Topic-based Software Defect Explanation

Tse-Hsun Chen[a], Weiyi Shang[b], Meiyappan Nagappan[c], Ahmed E. Hassan[a], Stephen W. Thomas[a]

[a] *Software Analysis and Intelligence Lab (SAIL)*
School of Computing, Queen's University, Canada
[b] *Concordia University, Canada*
[c] *Rochester Institute of Technology, USA*

Abstract

Researchers continue to propose metrics using measurable aspects of software systems to understand software quality. However, these metrics largely ignore the functionality, i.e., the *conceptual concerns*, of software systems. Such concerns are the technical concepts that reflect the system's business logic. For instance, while lines of code may be a good general measure for defects, a large file responsible for simple I/O tasks is likely to have fewer defects than a small file responsible for complicated compiler implementation details. In this paper, we study the effect of concerns on software quality. We use a statistical topic modeling approach to approximate software concerns as *topics* (related words in source code). We propose various metrics using these topics to help explain the file defect-proneness. Case studies on multiple versions of Firefox, Eclipse, Mylyn, and NetBeans show that (i) some topics are more defect-prone than others; (ii) defect-prone topics tend to remain so over time; (iii) our topic-based metrics provide additional explanatory power for software quality over existing structural and historical metrics; and (iv) our topic-based cohesion metric outperforms state-of-the-art topic-based cohesion and coupling metrics in terms of defect explanatory power, while being simpler to implement and more intuitive to interpret.

Key words:
code quality, topic modeling, LDA, metrics, cohesion, coupling

Email addresses: tschen@cs.queensu.ca (Tse-Hsun Chen), shang@encs.concordia.ca (Weiyi Shang), mei@se.rit.edu (Meiyappan Nagappan), ahmed@cs.queensu.ca (Ahmed E. Hassan), sthomas@cs.queensu.ca (Stephen W. Thomas)

Preprint submitted to Journal of Systems and Software *April 26, 2016*

1. Introduction

The cost of fixing software defects can be prohibitively high [69]. As a result, researchers have tried to uncover the possible reasons for software defects using different classes of software metrics, such as product metrics, process metrics, and project metrics [37, 44]. Indeed, such metrics have shown some success in explaining (i.e., correlation between the metrics and defects) the defect-proneness of software entities (e.g., methods, classes, files, or modules) [37]. However, these types of metrics do not take into account the actual *conceptual concerns* of a software system—the main technical concepts and business logic embedded within the files [49]. For example, an often-used metric, lines of code, may not always be a good general measure for defects: the largest file in one of our studied systems, Mylyn [3], for example, has 2,771 lines of code but no defects, while a much smaller file, with 23 lines of code, does contain a defect.

Recent studies propose a new class of metrics based on conceptual concerns [48, 49, 55, 60]. These studies approximate concerns using *statistical topic models*, such as latent Dirichlet allocation (LDA) [13]. Statistical topic models discover *topics* (i.e., sets of related words) within the source code files, which researchers use as surrogates for conceptual concerns. A prior study by Baldi *et al.* [7] shows that the topics that are generated by topic models have a strong agreement with that of other approaches like aspect mining. Recent studies [48, 49, 55, 60] on applying topic models on software systems provide initial evidence that topics in software systems are associated with the defect-proneness of source code files, opening up new possibilities for explaining why some files are more defect-prone than others.

In this paper, we build upon prior studies on software quality by considering the topics in source code files. We propose a set of topic-based metrics to study software quality: number of topics (NT), number of defect-prone topics (NDT), topic membership (TM), and defect-prone topic membership (DTM). We study how our proposed topic-based metrics can help better explain software defects. We also compare one of our metrics (NT), which measures cohesion in a software system using topics, with other state-of-the-art topic-based cohesion and coupling metrics. To the best of our knowledge, our study provides the first detailed comparison of the defect explanatory power of state-of-the-art topic-based cohesion and coupling metrics. We perform a detailed case study on multiple versions of four real-world software systems, with a focus on the following research questions:

RQ1: Are some topics more defect-prone than others?
We find that some topics, such as those related to new features and the core functionality of a system, may have a much higher cumulative defect density (CDDT) than others (average skewness in CDDT is 7.25, where a skewness of 1 is already considered highly skewed). We also find that defect-prone topics are likely to remain so over time, indicating that prior defect-proneness of a topic can be used to explain the future defect-proneness of topics and their associated files (Spearman correlation is 0.44–0.67).

RQ2: Can our proposed topic-based metrics help explain why some files are more defect-prone than others?
We find that our proposed topic-based metrics provide additional explanatory power (4 – 314% improvement) about the defect-proneness of files over existing state-of-the-art product and process metrics such as lines of code, code churn, and number of pre-release defects.

RQ3: How do our topic-based metrics compare with state-of-the-art topic-based cohesion and coupling metrics?
We find that our metric outperforms state-of-the-art topic-based cohesion and coupling metrics. Compared to state-of-the-art, our metric gives a larger improvement in defect explanatory power (8 – 55%) when using lines of code as a baseline metric. Thus, practitioners may benefit from including our metrics when analyzing software quality using cohesion and coupling.

This work extends our previous work [22]. First, we extend our case studies to include additional systems (NetBeans 4.0, 5.0, and 5.5.1). Second, we compare one of our topic-based metrics, which measures file cohesion, with state-of-the-art topic-based cohesion and coupling metrics (Section 5). We conduct a detailed comparison on how topic-based cohesion and coupling metrics help explain software defects. Finally, we study the sensitivity of the parameters that we use in our approach (Section 6). We have also made our datasets and results publicly available [21] and encourage researchers to replicate and verify our study.

The rest of this paper is organized as follows. Section 2 describes our approach to discover topics in source code files, and we define the topic-based metrics that we use to answer our research questions. Section 3 introduces the studied systems and outlines the design of our case studies. Section 4 and 5 present the result of our research questions. Section 6 discusses the parameter sensitivity of our approach. Section 7 talks about the potential threats to the validity of our findings, and Section 8 describes related work. Finally, Section 9 concludes the paper.

2. Proposed Approach

In this section, we outline our approach of using topics to explain defects. First, we briefly introduce topic modeling and describe how it can be applied to source code files to approximate conceptual concerns (i.e., main business logic). Next, we motivate and describe our new topic-based metrics.

2.1. Topic Modeling

Our goal is to determine which concerns are in each source code file. This information is often not available, since developers do not often manually categorize each file, and a file may contain several concerns [55]. In this paper, we approximate concerns using statistical topics, following the work of previous research [7, 55, 77]. In particular, we extract the *linguistic data* from each source code file, i.e., the identifier names and comments, which helps determine the functionality of a file [45]. We then treat the linguistic data as a corpus of documents, which we use as a basis for topic modeling.

In topic modeling, a *topic* is a collection of frequently co-occurring words in the corpus. Given a corpus of n documents $f_1, ..., f_n$, topic modeling approaches automatically discover a set Z of topics, $Z = \{z_1, ..., z_K\}$, as well as the mapping θ between topics and documents (see Figure 1). The number of topics, K, is an input that controls the granularity of the topics. We use the notation θ_{ij} to describe the topic membership value of topic z_i in document f_j.

Intuitively, the top words of a topic are semantically related and represent some real-world concept. For example, in Figure 1a, the three topics represent the concepts of “operating systems,” “computer networks,” and “user input,” respectively. The topic membership of a document then describes which of these concepts are present in that document: document f_1 is 30% about operating systems (i.e., $\theta_{11} = 0.3$) and 70% about computer network (i.e., $\theta_{21} = 0.7$).

More formally, each topic is defined by a probability distribution over all of the unique words in the corpus. Given two Dirichlet priors (used for computing Dirichlet distributions), α and β, a topic model will generate a

	Top words
z_1	*os, cpu, memory, kernel*
z_2	*network, speed, bandwidth*
z_3	*button click mouse right*

(a) Topics (Z).

	z_1	z_2	z_3
f_1	0.3	0.7	0.0
f_2	0.0	0.9	0.1
f_3	0.5	0.0	0.5
f_4	0.0	0.0	1.0

(b) Topic memberships (θ).

Figure 1: An example result of topic models, where three topics are discovered from four files. (a) The three discovered topics (z_1, z_2, z_3) are defined by their top (i.e., highest probable) words. (b) The four original source code files (f_1, f_2, f_3, f_4) are represented by topic membership vectors (e.g., $\{z_1 = 0.3, z_2 = 0.7, z_3 = 0.0\}$ for f_1).

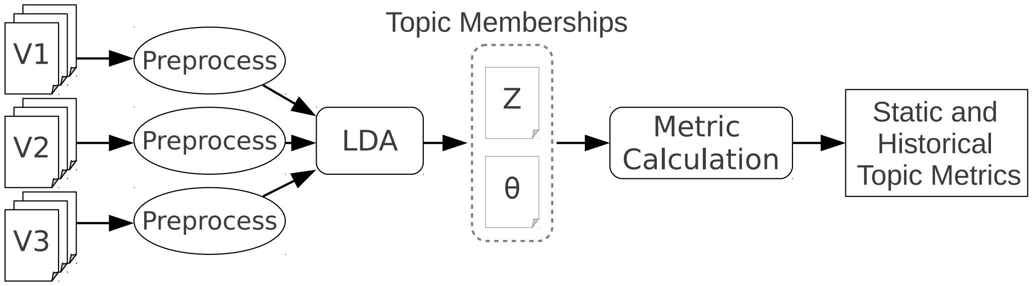


Figure 2: The process of calculating topic-based metrics. After preprocessing the source code, we run LDA on all versions of the source code files together, so we can study topics across versions. Using the topics and topic memberships that LDA returns, we calculate the topic-based metrics (see Section 2.2).

topic distribution, called θ_j, for each file f_j based on α, and generate a word distribution, called ϕ_i, for each topic z_i based on β. Choosing the right parameter values for K, α, and β is more of an art than a science, and depends on the desired granularity of the topics [73, 81]. Interested readers may refer to the original paper on LDA for the mathematical details [13].

2.2. Proposed Topic-based Metrics

To help explain the defect-proneness of source code files, we propose two categories of topic-based metrics: *static* and *historical*. Static topic metrics use a single snapshot of the software system, while historical metrics use the defect history of topics.

In the formulation of our topic-based metrics, we also consider traditional software metrics:

- $\mathsf{LOC}(f_j)$ The lines of code of file f_j.
- $\mathsf{PRE}(f_j)$ The number of pre-release defects that file f_j has. Pre-release defects are the defects found up to six months before the release of a given version.
- $\mathsf{POST}(f_j)$ The number of post-release defects that file f_j has. Post-release defects are the defects found up to six months after the release of a given version.

Before we define our topic-based metrics, we also summarize some of the above-mentioned LDA notations that we use for describing our metrics.

- z_i The topic i.
- Z The set of all topics.
- θ_{ij} The topic membership of topic z_i in source code file f_j.

Using these software metrics and the results of topic modeling, we propose the following topic-based metrics.

2.2.1. Cumulative Defect Density of a Topic

We define the cumulative defect density of a topic (CDDT), which we use to quantify the defect-proneness of topics. The defect density of a source code file is a well-known software metric, defined as the ratio of the number of defects in the file to its size. Using this ratio as our motivation, we define the cumulative pre-release defect density ($\mathsf{CDDT}_{\mathsf{PRE}}$) of a topic z_i as

$$\mathsf{CDDT}_{\mathrm{PRE}}(z_i) = \sum_{j=1}^{n} \theta_{ij} * \left(\frac{\mathrm{PRE}(f_j)}{\mathsf{LOC}(f_j)} \right), \tag{1}$$

where n is the total number of source code files and θ_{ij} is the topic membership of topic z_i in source code file f_j. Similarly, we define cumulative post-release defect density ($\mathsf{CDDT}_{\mathsf{POST}}$) of a topic z_i as

$$\mathsf{CDDT}_{\mathrm{POST}}(z_i) = \sum_{j=1}^{n} \theta_{ij} * \left(\frac{\mathrm{POST}(f_j)}{\mathsf{LOC}(f_j)} \right). \tag{2}$$

Since we only know the number of defects in each file, and we do not know the number of defects in each topic, we approximate the quality of a topic using CDDT. CDDT computes the topics defect density as the sum of topic defect density in each file. CDDT assumes that the ratio of the defect density in a file (and not the total number of defects) should be proportional to the topic membership in each individual file (and the same topic in different files may have different topic defect density values).

2.2.2. Static Topic-based Metrics

We propose static topic-based metrics to capture the number of topics in a file, and the topic membership of each file. We define the Number of Topics (NT) of a file f_j as

$$\mathsf{NT}(f_j) = \sum_{i=1}^{Z} I(\theta_{ij} \geq \delta), \tag{3}$$

where I is the indicator function that returns 1 if its argument is true, and 0 otherwise. δ is a cut-off threshold that determines if a topic plays an important role in a given file. δ is used only for removing topics that can be considered noise (e.g., some topics may have a membership value of 0.000001), as suggested in a prior study [81]. Thus, this threshold is not sensitive to the K (total number of topics) we choose for a particular system (see Section 6 for the effect of δ on the result). Note that if we choose K to be extremely small (e.g., 2 topics), δ may not be able to remove noise topics. However, choosing an extremely small K is not recommended as the information (topics) cannot be separated precisely [23, 81] (Section 6 provides the results of our parameter sensitivity analysis, which shows that using an extremely small K is not recommended). The NT metric measures the level of cohesion in a file: files with a large number of topics may be poorly designed or implemented, and thus are more likely to have defects [49].

The next static topic-based metric, Topic Membership (TM) of a file f_j, is defined as the topic membership values returned by the topic modeling approach:

$$\mathsf{TM}(f_j) = \theta_j. \tag{4}$$

The intuition behind this metric is that we assume different topics have different effects on the defect-proneness of a file. Some topics (e.g., a compiler-related topic) may increase the defect-proneness of a file, but other topics (e.g., an I/O-related topic) may actually decrease the defect-proneness. By using all the topic membership values, the TM metric captures the complete behavior of a file. In a prior work, Nguyen *et al.* [60] propose a similar metric using topic membership and LOC. However, they only consider five topics. Instead of considering an arbitrary and small number (five) of topics, we consider much more topics for TM (500), which may be more likely to capture finer-grained information. We also do not consider LOC in TM in order to eliminate the confounding effect LOC may cause in our analysis. Since TM is a vector of topic membership values, to avoid the problem of overfitting, we apply variable selection approaches in Section 4 when performing our analysis.

2.2.3. Historical Topic-based Metrics

We extend the static topic-based metric by considering the defect history of each topic. In order to calculate the number of defect-prone topics in a file, we define a *defect-prone topic* as a topic that has a higher CDDT than the average CDDT of all topics.

The set of defect-prone topics, B, is defined by

$$B = \{z_i \in Z \text{ s.t. } CDDT_{\text{PRE}}(z_i) > \mu(CDDT_{\text{PRE}}(Z))\}, \tag{5}$$

where $\mu(CDDT_{\text{PRE}}(Z))$ is the mean of the $CDDT_{\text{PRE}}$ of all topics. $\mathsf{CDDT_{POST}}$ usually have a larger impact on users, but the information is unknown until the system is released. Thus, we use $\mathsf{CDDT_{PRE}}$ to infer the quality of the topics (i.e., B) after the system is released.

We define the Number of Defect-prone Topics (NDT) in file f_j by

$$\mathsf{NDT}(f_j) = \sum_{i=1}^{K} I((z_i \in B) \wedge ((\theta_{ij}) \geq \delta)). \tag{6}$$

We define the Defect-prone Topic Membership (DTM) metric of file f_j as the topic memberships of defect-prone topics:

$$\mathsf{DTM}(f_j) = \theta_{ij} \text{ for all } i \text{ where } z_i \in B. \tag{7}$$

DTM is the topic memberships of all topics that are defined as defect-prone topics in Equestion 5 (i.e., topics in B). Similar to TM, DTM is a vector of topic memberships, so we also apply variable selection approaches in Section 4 when doing analysis.

3. Case Study Design

In this section, we introduce the systems that we use for our case study and we describe our analysis process, depicted in Figure 2.

3.1. Studied Systems

We focus on four large, real-world systems: Mylyn [3], Eclipse [1], Firefox [2], and NetBeans [4] (Table 1). For each system, we look at three consecutive versions (versions 1.0, 2.0, and 3.0 of Mylyn, versions 2.0, 2.1, and 3.0 of Eclipse, versions 1.0, 1.5, and 2.0 of Firefox, and versions 4.0, 5.0, and 5.5.1 of NetBeans). Eclipse is a popular IDE, which has extensive

Table 1: Statistics of the studied systems. The table shows the total lines of code, number of files, number of pre-release defects, number of post-release defects, and the used programming language in the systems.

	Lines of code (K)	No. of files	Pre. defects	Post. defects	Prog. language
Mylyn 1.0	127	833	1,047	712	Java
Mylyn 2.0	136	923	2,015	1,012	Java
Mylyn 3.0	165	1,115	2,045	480	Java
Firefox 1.0	2,841	5,523	638	454	C/C++
Firefox 1.5	3,111	5,879	716	946	C/C++
Firefox 2.0	3,205	5,942	1,134	453	C/C++
Eclipse 2.0	797	6,716	7,634	1,691	Java
Eclipse 2.1	987	7,799	4,975	1,182	Java
Eclipse 3.0	1,305	10,496	7,421	2,679	Java
NetBeans 4.0	915	4,253	630	311	Java
NetBeans 5.0	1,957	8,849	1,339	217	Java
NetBeans 5.5.1	3,302	16,383	883	795	Java

plugin architecture. Mylyn is a popular plugin for Eclipse that implements a task management system. Firefox is a well-known open-source web browser that is used by millions of users. Finally, NetBeans is a popular IDE that is implemented in Java.

3.2. Data Preprocessing

We first collect the source code files from each version of each studied system, then we preprocess the files using the preprocessing steps proposed by Kuhn *et al.* [45]. Namely, we first extract comments and identifier names from each file. Next, we split the identifier names according to common naming conventions, such as camel case and underscores. Finally, we apply stemming and remove common English-language stop words [76].

3.3. Topic Modeling

We use a popular topic modeling approach called latent Dirichlet allocation (LDA) [13]. We note other topic models can be used in our approach. We choose LDA because it is a generative statistical model, which helps alleviate model overfitting compared to other topic models such as Probabilistic LSI [40]. In addition, LDA has been shown to be effective for a variety of software engineering purposes, including analyzing source code evolution [78], calculating source code metrics [31], and recovering traceability links between source code and requirement documents [6]. Finally, LDA is fast and can easily scale to millions of documents [39].

We apply LDA to all versions of the preprocessed files of a system at the same time, an approach proposed by Linstead *et al.* [48]. For this study, we use K=500 topics for all studied systems. Lukins *et al.* [50] found that 500 topics is a good number for Eclipse and Mozilla, and we choose the same number for Mylyn and NetBeans. A prior study by Wallach *et al.* [81] has shown that choosing a larger K does not significantly affect the quality of the generated topics. The additional topics should be rarely used when LDA is assigning tokens to topics, and thus these topics may be considered noise [81]. On the other hand, choosing a small K may be more problematic, since the information (i.e., topics) cannot be separated precisely [81]. As a result, we chose to use a larger K (500), and the K was also suggested by a prior study [81]. The same intuition applies to running LDA on all versions of the same system. If a new feature only appears in V2 and not V1, then one of the topics (e.g., T1) may be present in V2. However, T1 will have a near-zero topic membership in V1, which should not affect the overall results.

Moreover, our metrics will remove the topics that are noise using a threshold or statistical approaches, so the remaining number of topics is much less. In fact, in the paper we find that the median number of significant topics (i.e., NT) is 5 to 9 for our studied systems (Section 4).

We use MALLET [56] as our LDA implementation. MALLET uses Gibbs sampling to approximate the joint distribution of topics and words. We run MALLET with 10,000 sampling iterations (II), and 1,000 of the iterations are used to automatically optimize α and β. The running time of MALLET ranges from about 100 minutes to 2,500 minutes depending on the size of the studied systems (we assign 10GB of memory to MALLET's Java process on a machine with 16 Intel 2.80GHz cores). However, the execution time should not be a problem when using our approach in practice, because we only need to apply LDA once per version. Studies [81, 82] have shown that using the optimized hyperparameters (i.e., α and β) will result in more robust models. In addition, we build the topics using both unigrams (single words) and bigrams (pairs of adjacent words), since bigrams help improve the performance for word assignments in topic modeling [16]. We set the membership threshold δ in Equations 3 and 6 to 1%. This value prevents topics with small, insignificant memberships in a file from being counted in the metrics of a file (i.e., noise removal).

In Section 6, we discuss the sensitivity of the results when these parameters and thresholds change.

4. Case Study Results

In this section, we present the results of our case study. We present each research question along three parts: the used approach to address the question; our experimental results; and a discussion of the results.

4.1. RQ1: Are some topics more defect-prone than other topics?

In this RQ, we study whether some topics are more defect-prone than others. Then, we may use topic information to help prioritize software quality assurance efforts.

4.1.1. Approach

We use Equation 2 to calculate the *cumulative post-release defect density* ($\mathsf{CDDT}_{\mathsf{POST}}$) for each topic in each studied software system. We evaluate this RQ in multiple directions: i) we provide a five-number summary and compute the skewness value of $\mathsf{CDDT}_{\mathsf{POST}}$; ii) and we perform single sample Kolmogorov-Smirnov non-uniformity tests to statistically determine if there is a significant difference between the defect densities of the various topics [29].

4.1.2. Results

We find that most topics in a system are not defect-prone (average skewness is 7.25). To verify our RQ, we want to see that most topics have a $\mathsf{CDDT}_{\mathsf{POST}}$ of zero or close to zero in the five-number summary, while some topics have a high $\mathsf{CDDT}_{\mathsf{POST}}$. If the $\mathsf{CDDT}_{\mathsf{POST}}$ is uniform, then topics will not give any extra information when studying software quality (since all topics would be equally defect-prone).

The average skewness for $\mathsf{CDDT}_{\mathsf{POST}}$ is 7.25 among the studied systems, where a skewness of one is already considered highly skewed. (Table 15 in Appendix A shows the five-number summary (minimum, first quartile, median, third quartile, maximum), skewness of the $\mathsf{CDDT}_{\mathsf{POST}}$, and skewness of the top 25% of the topics with the highest $\mathsf{CDDT}_{\mathsf{POST}}$ value.) The numbers indeed indicate that most topics have a low (almost zero) $\mathsf{CDDT}_{\mathsf{POST}}$, and the values are positively skewed (i.e., only a few topics have a high $\mathsf{CDDT}_{\mathsf{POST}}$). As an illustration for understanding skewness, Figure 3 show histograms of randomly-generated distributions that have a skewness of one and seven. As one can see, when the skewness is larger than one (positively skewed), most data points are on the left of the histogram, and the distribution is far away from normal. Thus, a skewness of 7.25 is considered extremely skewed.

We conduct the same study on the top 25% of the most defect-prone topics (i.e., 125 topics) in case the distribution is skewed by non-defect topics. Although the skewness is smaller than that of considering all topics, we find that the skewness is still extremely high (average skewness is 4.34).

Additionally, the number of topics and defect-prone topics per file for each system is consistent across versions (Table 2). We find that Mylyn has more defect-prone topics than the other three systems, while Firefox has the least number of defect-prone topics. Eclipse has the highest average $\mathsf{CDDT}_{\mathsf{PRE}}$ and $\mathsf{CDDT}_{\mathsf{POST}}$ among the four studied systems. NetBeans, on the other hand, has a similar median *number of topics* (NT) and *number of defect-prone topics* (NDT) to that of Eclipse. Appendix B shows hexbin plots of NDT vs NT for visualizing the relationship between NDT and NT. Hexbin plots add the information of the number of overlapping points on top of scatterplots, which are more suitable for visualizing the relationship between our metrics.

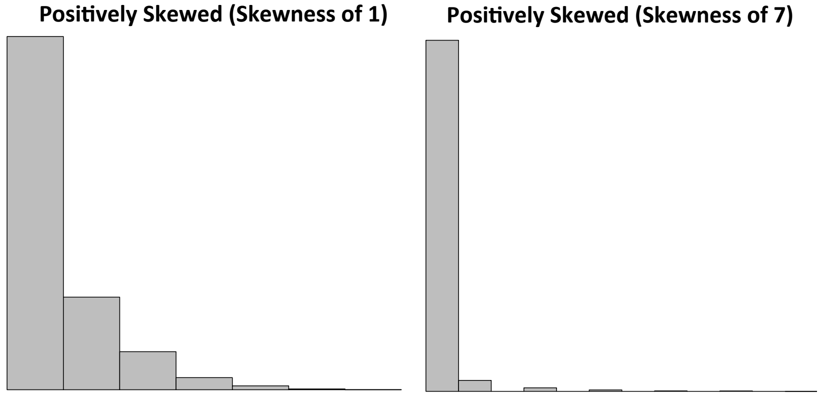


Figure 3: Histograms of randomly-generated distributions as examples to illustrate skewness. Since the data is randomly generated, the X-axis does not have any meaning, and the Y-axis shows the data frequency. The skewness values of the distributions are one (left) and seven (right). Positive skew indicates that the tail on the right side is longer than the left side.

Table 2: For each studied system, we show the mean $\mathsf{CDDT}_{\mathsf{PRE}}$ and $\mathsf{CDDT}_{\mathsf{POST}}$ across all topics ($\mu(CDDT_{\text{PRE}}$ and $\mu(CDDT_{\text{POST}}))$, the number and percentage of defect-prone topics (NDT), the median number of topics in each file (Med. NT), and the median number of defect-prone topics in each file (Med NDT).

	K	$\mu(D_{\text{PRE}})$	$\mu(D_{\text{POST}})$	NDT	Med. NT	Med. NDT
Mylyn 1.0	500	0.02	0.01	139 (27.8%)	9	7
Mylyn 2.0	500	0.04	0.02	137 (27.4%)	9	7
Mylyn 3.0	500	0.03	0.01	128 (25.6%)	9	7
Eclipse 2.0	500	0.22	0.03	122 (24.4%)	9	5
Eclipse 2.1	500	0.11	0.03	124 (24.8%)	9	5
Eclipse 3.0	500	0.17	0.06	136 (27.2%)	10	6
Firefox 1.0	500	0.00	0.00	106 (21.2%)	5	3
Firefox 1.5	500	0.00	0.00	111 (22.2%)	5	3
Firefox 2.0	500	0.00	0.00	82 (16.4%)	5	2
NetBeans 4.0	500	0.01	0.00	106 (21.2%)	9	5
NetBeans 5.0	500	0.01	0.00	103 (20.6%)	9	5
NetBeans 5.5.1	500	0.01	0.01	147 (29.4%)	9	6

Finally, we verify the non-uniformity illustrated in our result by applying the Kolmogorov-Smirnov test on the $\mathsf{CDDT_{POST}}$ of each version of each studied system. The Kolmogorov-Smirnov test assumes that the variables are independent. If the p-value of the Kolmogorov-Smirnov test is high, then the data is more likely to be uniformly distributed. However, we find that the p-values for all systems are significantly small (< 0.001)[1], indicating that the distribution of $\mathsf{CDDT_{POST}}$ values is indeed not uniform [72]. The same results hold when we apply Kolmogorov-Smirnov test on the top 25% defect-prone topics.

4.1.3. Discussion

To better understand why some topics are more defect-prone than others, we manually investigate the relevant words and files in these topics. We discuss the top three most and least defect-prone topics (i.e., according to $\mathsf{CDDT_{POST}}$) (Tables 16–19[2] in Appendix C). Since interpreting topics may not always be easy and may be subjective, we have provided the complete information of the generated topics and their CDDT values online for interested readers to verify [21].

Mylyn. Previously known as Mylar, Mylyn is an Eclipse plugin for task management. For the manually studied topics, we find that the topics with the highest defect densities are (i) those dealing with the Eclipse integration (topic 421); (ii) those that are related to tasks (e.g., development development) and the task UI (topics 164 and 168); and (iii) those dealing with the test suite of Mylyn (topic 400). In contrast, the least defect-prone topics deal with images and color (topics 405 and 178) and data compression (topic 175).

Eclipse. Among the manually studied topics in Eclipse, two of the most defect-prone topics (topics 496 and 492) in Eclipse 2.0 are about CVS plug-ins. We perform a manual analysis and discover that the build notes for this version indicate that the plug-ins supporting CVS-related functionalities were first introduced in this version, making it an active area of development. (In fact, according to Eclipse's defect repository, 17 defects relating to CVS

[1] even after a Bonferroni correction

[2] As explained in Section 3.3, we use both unigrams and bigrams when applying LDA. In the tables, our bigrams are depicted using two unigrams that are connected using a underscore.

remained unfixed after version 2.0). We perform another manual analysis and find that a similar story holds for Eclipse 2.1, where the integration with the Apache Ant build system was actively under development, which may lead to many defects in topic 131. Another set of defect-prone topics in Eclipse deals with low-level details such as memory operations and message passing (topics 462, 169, and 233). The least defect-prone topics in Eclipse include those about bit-wise operations (topic 116), arrays (topic 182), and parameter parsing (topic 192).

Firefox. For the manually studied topics, event handing (topic 101), which is responsible for dispatching events according to network protocol responses, is one of most defect-prone topics in Firefox 1.0 and 1.5. Another defect-prone topic in Firefox 2.0 deals with accessing saved states (topic 80). We perform a manual analysis and discover that the release notes of Firefox 2.0 indicate that new features were introduced, which allow the browser to restore previous sessions, and that the tabbed browsing functionality is updated.

Scanner Access Now Easy (SANE), an API that enables a scanner/digital camera application to be created with JavaScript, is one of the least defect-prone topics in all versions of Firefox (topic 280). Another topic that is not defect-prone deals with Base64 encoding (topic 359—the characters are segments of encoded characters), a known character standard.

NetBeans. After performing a manual analysis, we find that the NetBeans 4.0 release contained several new features, such as: project system based on Apache Ant (topic 219), code refactoring functionality, which uses tree-like structure to manipulate changes (topic 182), and GUI for controlling debug and build operations (topic 372). These topics appear to be the most defect-prone topics. Topics related to sending queries (topic 484), code completion (topic 57), parsing and manipulating Java code (topic 389), and using NetBeans to develop Mobile Information Device Profile (MIDP) application (topic 97) are also more defect-prone. From the release notes of NetBeans, we find a possible reason that these topics are more defect-prone: these features are relatively new in early versions of NetBeans and so they have more defects being reported after the release of the software. The least defect-prone topics in NetBeans are about handling XML files (topic 236), handing database metadata (dmd) and metadata adaptor (topic 455), and providing support for Java Platform (J2EE) customizer (topic 211). In addition, handling and storing software properties (topic 14), parser generator that produces syntax trees (topic 73), and helper classes for creating NetBeans GUI tests (topic

39) are less defect-prone.

To ensure that these defect-prone topics are not general topics (i.e., topics that exist in most files) in the studied systems, we compute the normalized entropy of the topic membership values of each detect-prone topic across all files. Information entropy [41] can be used to measure how general a topic is. If a topic appears only in a few files (i.e., the topic is specific), the entropy value will be low; otherwise, the entropy value will be high (i.e., the topic is general). We find that the defect-prone topics have a median entropy of 0.37–0.59, and a third quantile of 0.47–0.69. Thus, most of the defect-prone topics are not general topics that appear in most of the files in a system.

We find that most topics in a system are not defect-prone (average skewness across all 3 versions of all the 4 case study systems is 7.25, where a skewness of 1 is already considered highly skewed). We manually examine the top three most and least defect-prone topics, and we find that topics that are related to new features of a system tend to have a much higher $\mathsf{CDDT_{POST}}$ *than others.*

Do defect-prone topics remain defect-prone over time?

We find that the defect-prone topics in a previous release tend to be defect-prone in the later releases, with a correlation of 0.44 – 0.67. Previously, we found that some topics are more defect-prone than other topics. In order to verify that these topics are consistently defect-prone over time, we compute the Spearman rank correlation of the $\mathsf{CDDT_{POST}}$ values among different versions. Spearman rank correlation computes the correlation on the ranks of the topic $\mathsf{CDDT_{POST}}$, so a high correlation value will imply that the ranks of the topic defect-proneness are consistent across versions (i.e., the most defect-prone topics may still be the most defect-prone in the following version). Moreover, the metrics are highly skewed (as shown in Table 15, and p-values of Shapiro-Wilk normality test are all less than 0.05, meaning that the distributions of our metrics are statistically significantly different from a normal distribution). Therefore, the use of parametric correlation measures such as Pearson correlation, which have assumptions on the population distribution, are not suitable for our metrics.

Table 3 shows the correlation of $\mathsf{CDDT_{POST}}$ among different versions of a system. The correlation values are consistently medium to high between different versions, which indicates that a defect-prone topic is still likely to

Table 3: Spearman correlation coefficients of each topic's $\mathsf{CDDT}_{\mathsf{POST}}$ across software versions. The p-values are all less than 0.001.

	Mylyn 1.0	Mylyn 2.0	Mylyn 3.0
Mylyn 1.0	—	—	—
Mylyn 2.0	0.673	—	—
Mylyn 3.0	0.483	0.493	—

	Eclipse 2.0	Eclipse 2.1	Eclipse 3.0
Eclipse 2.0	—	—	—
Eclipse 2.1	0.529	—	—
Eclipse 3.0	0.438	0.530	—

	Firefox 1.0	Firefox 1.5	Firefox 2.0
Firefox 1.0	—	—	—
Firefox 1.5	0.536	—	—
Firefox 2.0	0.473	0.564	—

	NetBeans 4.0	NetBeans 5.0	NetBeans 5.5.1
NetBeans 4.0	—	—	—
NetBeans 5.0	0.612	—	—
NetBeans 5.5.1	0.588	0.563	—

Table 4: Spearman correlation coefficients of $\mathsf{CDDT}_{\mathsf{POST}}$ using the top 25% most defect-prone topics. Assuming top 25% of the most defect-prone topics in v1 is V, then cor(v1, v2) means that the correlation is computed using V in v1 and v2.

Mylyn		Eclipse	
cor(1.0, 2.0)	cor(2.0, 3.0)	cor(2.0, 2.1)	cor(2.1, 3.0)
0.44	0.46	0.13	0.26
Firefox		**Netbeans**	
cor(1.0, 1.5)	cor(1.5, 2.0)	cor(4.0, 5.0)	cor(5.0, 5.5.1)
0.47	0.42	0.20	0.23

be defect-prone in later versions. Similarly, in Tables 16–19, several of the top defect-prone topics are repeated across three versions of each studied software system.

To investigate further, we conduct the correlation analysis again on the top 25% most defect-prone topics (i.e., 125 topics). Because the top 25% most defect-prone topics may be different across versions, we study the correlation between most the defect-prone topics in an earlier version with *the same set of topics* in a later version. For example, if topics T1, T2, and T3 are identified as the top 25% most defect-prone topics in V1, then we will compute a rank correlation of the $\mathsf{CDDT}_{\mathsf{POST}}$ of T1, T2, and T3 in V1 with the $\mathsf{CDDT}_{\mathsf{POST}}$ of T1, T2, and T3 in V2. Table 4 shows the result of our correlation analysis. Although the strength of the correlation decreases compared to using all topics (excluding topics that have a close-to-zero $\mathsf{CDDT}_{\mathsf{POST}}$), we still see a moderate correlation (above 0.4) [14, 47] in Mylyn and Firefox. We find that Eclipse has the lowest correlation value between 2.0 and 2.1, but developers' focus on minor releases (i.e., 2.1) may be different from major releases (i.e., 2.0 and 3.0). Thus, we compute cor(2.0, 3.0), and the correlation value is 0.25. Based on our results, we note that, when considering only the top 25% most defect-prone topics, the defect-proneness of topics still has a moderate correlation across versions in two of the studied systems, and has a low correlation across versions in the other two studied systems.

The correlation of the $\mathsf{CDDT}_{\mathsf{POST}}$ *of each topic among different versions of a system is consistently from mid to high (0.44 – 0.67), which implies that defect-prone topics are likely to be defect-prone in later versions. This information can help practitioners allocate testing resources more effectively.*

4.2. RQ2: Can our proposed topic-based metrics help explain why some files are more defect-prone than others?

In the previous research questions, we have shown that topics have different levels of defect-proneness, and defect-prone topics tend to remain so over time. To provide evidence for practitioners that topics can help software quality assurance processes, we now examine the amount of additional deviance in post-release defects that our topic-based metrics can explain, with respect to traditional baseline metrics — *lines of code* (LOC), number of *pre-release defects* (PRE), and code churn. This analysis allows us to verify our intuition that topic-based metrics provide additional explanatory power over state-of-the-art metrics when studying post-release software defects.

4.2.1. Approach

Explaining Software Defects. As mentioned in Section 2.2, software metrics can be classified as *static* or *historical.* Static metrics, such as lines of code (LOC), are obtained from a single snapshot of the system [25]. On the other hand, historical metrics incorporate past information about the system, such as pre-release defects (PRE) and code churn (i.e, changes to the code) [11]. As such, in this research question, we build two sets of models: those based on static topic-based metrics — *number of topics* (NT) and *topic memberships* (TM); and those based on historical topic-based metrics — *number of detect-prone topics* (NDT) and *detect-prone topic membership* (DTM). For a baseline static metric, we choose LOC. Although LOC may not represent all static metrics, LOC is shown to be a good general software metric and has been used for benchmarking prior proposals of new metrics [27, 68]. In addition, LOC is shown to have a high correlation with other complexity metrics [42, 61]. For baseline historical metrics, we choose PRE and code churn because they have been shown to be good explainers for defects [12, 58], and have also been used as baseline models for comparing metrics [11]. Moreover, LOC, PRE, and code churn are shown to have the best explanatory power for defects, and are typically used as baseline metrics by other researchers when proposing new metrics [27, 35, 61].

Our goal here is not to predict post-release defects. Since topics provide a higher-level view of the system, using topics may better explain what is going on in the source code files [20]. Thus, we want to see how much improvement on explaining deviance (i.e., model fitness) in defects our topic-based metrics can bring to the baseline metrics. Using the results of our analysis, practitioners can better understand the key drivers for the quality of their software, and can focus on improving or tackling some of these drivers.

We use logistic regression with post-release defects as our dependent variable, and report the percent deviance explained (D^2) for each combination of independent variables (i.e., metric combinations). D is squared to show the magnitude of the model fitness. To control for the high skew in our metrics, we apply a log transformation on the metrics before the model is built. Here, the D^2 measure is similar to the adjusted R^2 measure in linear regression, except that D^2 quantifies the amount of deviance that a logistic regression model can explain.

Interpreting Results. A higher D^2 value generally indicates a better model fit. If adding our topic-based metrics to the model can increase D^2, our topic-based metrics can give extra information when understanding defects. Thus, we use D^2 as a relative measure, and our goal is to compare the improvement of model fitness (i.e., D^2) after adding our topic-based metrics. However, when the number of independent variables is large, D^2 may not be a good measure. As the number of independent variables increases, D^2 will always increase regardless of the quality of the model. Thus, we also use another measure called the Akaike information criterion (AIC). AIC is defined as:

$$AIC = 2k - 2ln(L),$$

where k is the number of estimated parameters in the model, and L is the maximized value of the likelihood function for the model. AIC can be used to compare the fitness of different models, as AIC penalizes more complex models (since they are more likely to suffer from overfitting) [17, 65]. Models with lower AIC scores are better.

Recall that by the definition of our TM and DTM metrics (Equations 4 and 7), each metric will produce many values for each file (K values in the case of TM, and $|B|$ values in the case of DTM). To avoid the problems of overfitting and multicollinearity, we use Principal Component Analysis (PCA) to reduce the dimensionality of the metrics [43]. PCA transforms the data into a smaller set of uncorrelated variables while still captures the patterns of the original data [43]. We choose the principal components (PCs) until either 90% of the variances are explained, or when the increase in variance explained by adding a new PC is less than the mean variance explained of all PCs [43].

We further perform stepwise regression on the PCs of TM and DTM metrics to reduce the number of independent variables to the smallest number of statistically significant independent variables in order to ensure that our models are stable across different data sets [26, 36]. Stepwise regression is a variable selection approach, which adds or removes variables to the model according to some criteria, which, in this paper, we choose to use the AIC score.

4.2.2. Results

Our proposed topic-based metrics provide additional defect explanatory power over the baseline metrics (18–314% over baseline static metrics and 4–76% over baseline historical metrics in terms of deviance explained). Moreover, more topics that a file has the higher that the chance the file is detect-prone. We present the results in Tables 5 and 6. Table 5 shows the results for our static topic-based metrics. We find that adding NT gives a significant improvement in the deviance explained. All models have statistically significant (p-value ≤ 0.05) improvements when NT is added to the model. In all the versions of Mylyn, Firefox, and NetBeans, NT gives at least an 18% increase in D^2 and 3% in AIC. However, we find that the performance of NT is not as high in Eclipse. TM, on the other hand, gives significant improvements in all four studied systems. We find that the TM metric improves the deviance explained by 61–314% and improves the AIC by 9–22%, compared to the baseline model. The reason that TM is better than NT may be that TM captures more underlying information of the topics when building the regression model.

Tables 5 and 6 also show the number of resulting PCs after our variable selection process. We see that the number of resulting PCs is less than 10 in all the TM and DTM models, which means that our approach is able to remove noise topics that do not contain much information. Moreover, overfitting is unlikely to happen due to using a small number of PCs on a large number of data points (i.e., files).

Table 5: D^2 improvement and AIC scores for static software metrics. The higher the D^2 the better the explanatory power; the lower the AIC scores the better the explanatory power. Our metrics are statistically significant in all models (i.e., p-value < 0.05). The numbers in the column $\%Change$ are the percentage D^2 increase or AIC score decrease compared to the baseline model. The best model of each version of the software is marked in bold.

System	**Model**	**D^2**	**% Change**	**AIC**	**% Change**
Mylyn 1.0	Base(LOC)	0.09		1047	
	Base+NT	0.14	+56%	991	+5%
	Base+TM (Selected 5)	***0.21***	***+133%***	***958***	***+9%***
Mylyn 2.0	Base(LOC)	0.14		1078	
	Base+NT	0.19	+36%	1020	+5%
	Base+TM (Selected 3)	***0.27***	***+93%***	***957***	***+11%***
Mylyn 3.0	Base(LOC)	0.13		1160	
	Base+NT	0.20	+54%	1072	+8%
	Base+TM (Selected 4)	***0.34***	***+162%***	***949***	***+18%***
Firefox 1.0	Base(LOC)	0.12		2375	
	Base+NT	0.16	+33%	2256	+5%
	Base+TM (Selected 7)	***0.20***	***+67%***	***2168***	***+9%***
Firefox 1.5	Base(LOC)	0.15		3475	
	Base+NT	0.21	+40%	3241	+7%
	Base+TM (Selected 7)	***0.28***	***+87%***	***2969***	***+15%***
Firefox 2.0	Base(LOC)	0.14		2225	
	Base+NT	0.17	+18%	2152	+3%
	Base+TM (Selected 4)	***0.24***	***+71%***	***1973***	***+11%***
Eclipse 2.0	Base(LOC)	0.18		4584	
	Base+NT	0.18	+0%	4576	+0%
	Base+TM (Selected 10)	***0.29***	***+61%***	***4004***	***+13%***
Eclipse 2.1	Base(LOC)	0.11		4805	
	Base+NT	0.11	+0%	4793	+0%
	Base+TM (Selected 8)	***0.28***	***+87%***	***2969***	***+15%***
Eclipse 3.0	Base(LOC)	0.14		7592	
	Base+NT	0.14	+0%	7590	+0%
	Base+TM (Selected 7)	***0.24***	***+71%***	***6800***	***+10%***
NetBeans 4.0	Base(LOC)	0.07		1529	
	Base+NT	0.10	+43%	1476	+3%
	Base+TM (Selected 8)	***0.29***	***+314%***	***1197***	***+22%***
NetBeans 5.0	Base(LOC)	0.07		1651	
	Base+NT	0.10	+43%	1595	+3%
	Base+TM (Selected 7)	***0.19***	***+171%***	***1477***	***+11%***
NetBeans 5.5.1	Base(LOC)	0.07		4916	
	Base+NT	0.11	+57%	4728	+4%
	Base+TM (Selected 9)	***0.17***	***+143%***	***4418***	***+10%***

Table 6: D^2 improvement and AIC scores for historical software metrics. The higher the D^2 the better the explanatory power; the lower the AIC scores the better the explanatory power. The numbers in the column $\%Change$ are the percentage D^2 increase or AIC score decrease compared to the baseline model. "*" indicates that the effect of our metrics is statistically significant (i.e., p-value < 0.05) when added to the baseline. The best model of each version of the software is marked in bold.

System	**Model**	**D^2**	**% Change**	**AIC**	**% Change**
Mylyn 1.0	Base(PRE+Churn)	0.21		918	
	Base+NDT*	0.24	+14%	886	+3%
	Base+DTM (Selected 5)*	***0.30***	***+43%***	***824***	***+10%***
Mylyn 2.0	Base(PRE+Churn)	0.22		987	
	Base+NDT*	0.23	+4%	971	+2%
	Base+DTM (Selected 7)*	***0.34***	***+55%***	***882***	***+11%***
Mylyn 3.0	Base(PRE+Churn)	0.28		957	
	Base+NDT	0.29	+4%	956	+0%
	Base+DTM (Selected 5)*	***0.36***	***+29%***	***910***	***+5%***
Firefox 1.0	Base(PRE+Churn)	0.14		2300	
	Base+NDT*	0.18	+29%	2204	+4%
	Base+DTM (Selected 7)*	***0.20***	***+43%***	***2152***	***+6%***
Firefox 1.5	Base(PRE+Churn)	0.20		3256	
	Base+NDT*	0.25	+25%	3081	+5%
	Base+DTM (Selected 7)*	***0.27***	***+35%***	***3006***	***+8%***
Firefox 2.0	Base(PRE+Churn)	0.23		2009	
	Base+NDT*	0.25	+9%	1951	+3%
	Base+DTM (Selected 6)*	***0.28***	***+22%***	***1893***	***+6%***
Eclipse 2.0	Base(PRE+Churn)	0.17		4605	
	Base+NDT*	0.20	+18%	4478	+3%
	Base+DTM (Selected 8)*	***0.30***	***+76%***	***3930***	***+15%***
Eclipse 2.1	Base(PRE+Churn)	0.15		4587	
	Base+NDT	0.15	+0%	4586	+0%
	Base+DTM (Selected 4)*	***0.19***	***+27%***	***4366***	***+5%***
Eclipse 3.0	Base(PRE+Churn)	0.17		7310	
	Base+NDT	0.17	+0%	7309	+0%
	Base+DTM (Selected 7)*	***0.24***	***+41%***	***6729***	***+8%***
NetBeans 4.0	Base(PRE+Churn)	0.28		1183	
	Base+NDT*	***0.31***	***+11%***	***1138***	***+4%***
	Base+DTM (Selected 5)*	0.31	+11%	1145.90	+3%
NetBeans 5.0	Base(PRE+Churn)	0.14		1524	
	Base+NDT*	0.17	+21%	1471	+4%
	Base+DTM (Selected 8)*	***0.21***	***+50%***	***1418***	***+7%***
NetBeans 5.5.1	Base(PRE+Churn)	0.13		4613	
	Base+NDT*	0.17	+31%	4367	+5%
	Base+DTM (Selected 9)*	***0.19***	***+46%***	***4307***	***+7%***

Table 6 shows the results for our historical topic-based metrics. We find that NDT gives a good improvement in D^2 and AIC over the baseline model. This implies that topics with high pre-release defects are more likely to have post-release defects, and having more defect-prone topics may have negative effects on the software quality. The improvement of adding NDT to the model is not as large for Eclipse 2.1 and 3.0. However, the improvements achieved by the DTM metric are consistent across all versions of all systems. DTM also helps explain defects more than NDT, except for NetBeans 4.0 where NDT has a lower AIC score than that of DTM. We find that overall DTM improves D^2 over the baseline model by 11–76% and improves AIC by 5–15%. Similar to TM, DTM contains a more general view of the distributions of topics, so the model performance may be better than the models built using NDT. In general, we find that models built using historical metrics have a higher model fitness than models built using static metrics in 11 out of 12 of the studied versions. However, historical metrics may not always be available, and may require time to collect. On the other hand, static metrics can be computed easily using the current snapshot of the system. In short, static metrics can be obtained easily but give a lower model fitness, while historical metrics may not always be available but give a higher model fitness.

4.2.3. Discussion

One possible explanation that the improvement of adding NDT to the model is not as high in Eclipse compared to other systems is because topics in Eclipse have higher defect densities. As shown in Table 2, Eclipse has a higher mean $\mathsf{CDDT}_{\mathsf{PRE}}$ and $\mathsf{CDDT}_{\mathsf{POST}}$ than the other studied systems. Since Eclipse topics are generally more defect-prone, NDT may fail to capture some relatively defect-prone topics that have a $\mathsf{CDDT}_{\mathsf{POST}}$ value smaller than the cut-off threshold (average of all topics' $\mathsf{CDDT}_{\mathsf{POST}}$), and thus the overall explanatory power of NDT decreases. In contrast, DTM contains more general information about all the defect-prone topics, which may better explain defects.

To see the effects of NT and NDT in our logistic regression models, Table 7 shows the average odds ratio of these metrics across the three versions of each studied system. Odds ratio [14] is used to interpret the effect of a metric in the regression model, and the odds ratio is computed as e^{coef}, where coef is the regression coefficient of the metric in the model. In our case, odds are defined as the ratio of the probability of being defect-prone and the probability of not being defect-prone, given one-unit increase of log-NT or log-NDT (we applied

Table 7: Odds ratio of the average regression coefficients of NT and NDT metrics.

	Mylyn	Firefox	Eclipse	NetBeans
NT	5.64	3.52	1.06	3.74
NDT	5.53	2.34	1.27	2.46

log transformation to the metrics to reduce the skewness when building the regression models). Thus, an odds ratio of one means that the metric has no effect on defect-proneness (probability of defect-prone and not defect-prone is the same), and an odds ratio of two means that a unit of increase in log-NT (or log-NDT), the chance of the file being detect-prone increases by 100%. In all studied systems, when log-NT increases by 1, a file will have a higher chance (6 – 564%) to be defect-prone; when log-NDT increases by 1, a file will have a higher chance (27 – 553%) to be defect-prone. As a result, when NT and NDT increases, the files will be more likely to be defect-prone.

Our findings show that the number of topics in a file has a positive relationship with defects, and files having more defect-prone topics are more likely to continue being more defect-prone.

All of our proposed topic-based metrics provide additional defect explanatory power over traditional state-of-the-art baseline metrics. We find that files containing more topics tend to be more defect-prone. We also find that files containing more defect-prone topics are even more likely to be defect-prone.

5. RQ3: How do our metrics compare with state-of-the-art topic-based cohesion and coupling metrics?

Maintaining a high cohesion and low coupling among source code files during development can help reduce maintenance costs and improve the reliability of a software system [30, 51]. Researchers have used various software structures, such as interactions among variables and methods, to measure cohesion and coupling in software systems [5, 8, 15, 18, 57]. A prior study [28] found that these cohesion and coupling metrics have a moderate to strong correlation with software defects. Recently, many studies have measured cohesion and coupling using a different approach, namely topic models [22, 31, 49, 53, 54, 64, 80]. These approaches measure cohesion and coupling using the topic similarity or scattering (i.e., topic is spread out over multiple files) in source code files. The topic-based cohesion and coupling metrics capture different information compared to the traditional cohesion and coupling metrics [64]. The NT metric that we propose in Section 4.2.1 also measures the level of cohesion of a source code file, i.e., more topics implies low cohesion, and in RQ2 we find that this metric is statistically significant when explaining software defects.

Prior studies [31, 49, 54, 80] focus on using different topic-based cohesion and coupling metrics to study software quality. Thus, in this RQ we want to compare the defect explanatory power of state-of-the-art topic-based cohesion and coupling metrics with our metric, NT. We compare cohesion and coupling metrics together because these two kinds of metrics are usually correlated together (e.g., high cohesion correlates with low coupling) [71]. To the best of our knowledge, this is the first study on comparing the defect explanatory power and the correlation among the topic-based cohesion and coupling metrics. We do not include our TM, NDT, and DTM metrics in the study in order to ensure a fair comparison. NDT and DTM consider the defect history of topics, while current state-of-the-art metrics only consider static information (i.e., source code). Additionally, TM and DTM contain more than one variable, because these two metrics are based on the topic membership values for a file (i.e., if we have 100 topics, then TM is still very likely to have more than one variable after PCA reduction and stepwise regression).

By identifying which metrics perform best, and whether the metrics are correlated with each other, practitioners can use the most effective combination of metrics and avoid possible problems of overfitting and multicollinearity. In addition, we can study whether NT offers additional explanatory power over the current state-of-the-art metrics.

5.1. Approach

We want to compare NT (Equation 3) with the cohesion and coupling metrics proposed by Ujhazi *et al.* [80], Liu *et al.* [49], and Gethers et al. [31], since these metrics use different topic modeling approaches. We begin with a correlation analysis, because these metrics each use topics to calculate cohesion and coupling. Thus, there may be some overlap in what they each

Table 8: Summary of the topic-based cohesion and coupling metrics that we compare in our study. Granularity indicates at which level the metric is computed.

Metric	Type	Granularity	Topic Model	Preferred Metric Value	Cited Paper
CLCOM5	cohesion	method	LSI	low	Ujhazi *et al.* [80]
CCBO	coupling	file	LSI	low	Ujhazi *et al.* [80]
RTC_s	coupling	file	RTM	low	Gethers *et al.* [31]
MWE	cohesion	method	LDA	high	Liu *et al.* [49]
NT	cohesion	file	LDA	low	This Paper

capture. Next, we measure how these metrics differ in terms of defect explanatory power. Table 8 shows the summary of these metrics. We briefly describe these metrics below.

Measuring Cohesion and Coupling using Latent Semantic Indexing. Ujhazi *et al.* [80] measure conceptual cohesion using a topic modeling approach called Latent Semantic Indexing (LSI). LSI is computed by applying singular value decomposition (SVD) on the term-document matrix of a corpus. The term-document matrix of a corpus is a matrix whose row corresponds to an individual document (i.e., source code file), and whose columns represent the unique words in the corpus. For example, the index $[i, j]$ in the matrix represents the number of occurrences of the word j in document i. In addition to word frequency count, other weight measures, such as $tf * idf$ (term frequency-inverse, document frequency), are often used [52].

$tf * idf$ is used to reflect the importance of a word in a document using the word's occurrence within and across documents. Thus, in this paper, we use $tf * idf$ instead of the raw word frequency count, since $tf * idf$ lowers the score of insignificant words. SVD then reduces the dimensions in the term-document matrix by selecting the K dimensions (i.e., K topics) with the largest singular values. Finally, by computing the cosine distance between each row (document), we obtain a similarity score between the documents.

Ujhazi *et al.* [80] propose a cohesion metric called CLCOM5, which is defined by computing the cosine similarity among all the *methods* within a source code file (i.e., each document in LSI is a method, and the corpus is the file). If a method is very similar to many other methods in the same file, then this file has low cohesion. If the similarity score between each pair of methods within the same file is larger than some threshold, then a score of one is added to the file (i.e., the file's coupling score plus one). The authors find that the optimal threshold for the cosine similarity score to be between 0.7 to 0.8 through empirical analysis. In this paper, we use the average of these thresholds, 0.75.

Ujhazi *et al.* also propose a coupling metric called CCBO at the *file level* (i.e., each document in LSI is a file, and the corpus is all the files in the software system). If a file is similar to many other files in the same software system, then this file is highly coupled. The authors compute the cosine similarity among all the source code files in a software system, and assign a coupling score of one if the score is larger than a threshold (same threshold range as that in CLCOM5).

Measuring Coupling using Relational Topic Models. Gethers *et al.* [31] propose a coupling metric using a variant of LDA, called relational topic model (RTM) [19]. RTM predicts the linking probability among documents using the underlying topics. If the topics in one file are very similar to the topics in another file, then these files will be highly coupled. Let $\mathsf{RTC}(f_1, f_2)$, the Relational Topic-based Coupling between f_1 and f_2, be the probability that a link exists between file f_1 and f_2, as outputted by the RTM model. We then define the relational topic coupling RTC_s for a source code file f_i as

$$\mathsf{RTC}_s(f_i) = 1/n * \sum_{j=1}^{n} \mathrm{RTC}(f_i, f_j) \text{ where } i \neq j, \tag{8}$$

where n is the number of files in the system.

Measuring Cohesion using Topic Distributions and Occupancies. Liu *et al.* [49] propose a cohesion metric using LDA, called Maximal Weighted Entropy (MWE). MWE measures the level of cohesion in software systems at the method level. For each topic, Liu *et al.* [49] compute the topic occupancy and distribution in a source code file, and MWE is the product of the maximum occupancy and distribution among all topics. The occupancy of topic z_i in all the methods of a source code file is defined as

$$O(z_i) = 1/m(f) * \sum_{j=1}^{m(f)} \theta_{ij}, \tag{9}$$

where $m(f)$ is the number of methods in the source code file f, and θ_{ij} is the topic membership value of topic z_i in method j. Occupancy measures the average of a topic's membership value across all methods in a file. In other words, occupancy answers the question: on average, how much does this topic exist in this file?

Liu *et al.* [49] use information entropy to measure the distribution of topics in a source code file. Information entropy measures the uncertainty of the topics [41]. If a topic is distributed uniformly in the methods of a file, then the entropy value will be low; otherwise, the entropy value will be high. The Shannon information entropy function E is defined as

$$E = -\sum_{i=1}^{n} p(x_i) * log_e p(x_i), \tag{10}$$

where $p(x_i)$ is the probability value of outcome x_i (i.e., topic-document probability).

The distribution $D(z_i)$ of topic z_i in a file is defined as

$$D(z_i) = 1/log(m(f)) * E(z_i), \tag{11}$$

where $E(z_i)$ is the information entropy value for z_i computed across all the methods in a file.

Finally, MWE of a source code file f is defined as

$$\mathsf{MWE}(f) = max_{z_i \in Z}(O(z_i) * D(z_i)), \tag{12}$$

where Z is the set of topics in a file.

Implementation and Experiment Procedures. We implement and compute all of the above-mentioned metrics using K = 500 and II = 10,000, the same as our previous research questions. These parameters may slightly affect the results, but we want to do a controlled experiment where the parameters are the same. For implementing RTC_s, we use α=50/K and β=0.1, as is recommended by the software package that Gethers *et al.* [31] use. For implementing MWE, we use MALLET to optimize α and β. Our experiment procedures are as follows:

1. We first examine the pair-wise correlation between all topic-based metrics and LOC, and study whether these topic-based metrics are capturing different information than LOC. We use LOC as our baseline metric and remove any metrics that are highly correlated to LOC, because most software complexity metrics are highly correlated with LOC, and LOC is considered one of the best metrics for explaining defects [42, 61].

2. We next study how much D^2 and AIC improvement each metric gives over a base LOC model. We use LOC as the baseline model, since these metrics are all using a single snapshot of the current software system.
3. We finally perform a PCA analysis on LOC and all the topic-based cohesion and coupling metrics, except NT. We use the resulting PCs as the baseline model, and examine the D^2 improvement when NT is added to the model. By doing this experiment, we can examine whether NT can bring any additional improvements to all of LOC and state-of-the-art topic-based cohesion and coupling metrics combined when explaining defects.

5.2. Results and Discussion

5.2.1. Correlation Analysis

Most metrics are not highly correlated with* LOC*, except* CLCOM5*. We perform a Spearman correlation analysis to remove any metrics that are highly correlated with LOC (i.e., > 0.6) to ensure that all the topic-based metrics are capturing different information relative to LOC. We use the Spearman correlation because all of the metrics are not normally distributed (the p-values of the Shapiro-Wilk Normality test are all smaller than 0.05).

In Tables 20–23 in Appendix D, we see that CLCOM5 is highly correlated with LOC in every studied system, with a coefficient between 0.63–0.78. Due to such high correlation with the baseline metric, we remove CLCOM5 from further analyses. We also find that NT has a relatively strong negative correlation with MWE (mostly around -0.5). However, neither of these two metrics is highly correlated with LOC. Even if they are highly correlated, one metric may still outperform the other when explaining defects, so we will keep these two metrics and perform a more detailed comparison below. Furthermore, NT is considerably simpler metric to measure compared to MWE, and we are curious about NT's overall performance compared to MWE, the more complex metric.

5.2.2. Improvement in Defect Explanatory Power of Each Topic-base Cohesion and Coupling Metric

NT ***gives the most improvement over the baseline metrics (30% on average) compared to other topic-based cohesion and coupling metrics (3 – 12% on average).*** Tables 9 and 10 shows that our NT metric generally gives the greatest improvement (30% on average), except for Eclipse 2.0 and 3.0, and NetBeans 4.0; however, no topic-based cohesion

Table 9: D^2 improvement and AIC scores for topic-based cohesion and coupling metrics. The numbers in the column $\%Change$ are the percentage D^2 increase or AIC score decrease compared to the baseline model. The best model of each version of the software is marked in bold. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

System	**Model**	**D^2**	**% Change**	**AIC**	**% Change**
Mylyn 1.0	Base(LOC) *	0.06		958	
	Base+NT *	***0.09***	***+50%***	***925***	***+3%***
	Base+CCBO	0.06	+0%	959	+0%
	Base+RTC$_s$ *	0.09	+50%	931	+3%
	Base+MWE *	0.08	+33%	940	+2%
Mylyn 2.0	Base(LOC) *	0.10		947	
	Base+NT *	***0.14***	***+40%***	***908***	***+4%***
	Base+CCBO	0.10	+0%	949	+0%
	Base+RTC$_s$	0.10	+0%	949	+0%
	Base+MWE *	0.12	+20%	923	+3%
Mylyn 3.0	Base(LOC) *	0.11		1075	
	Base+NT *	***0.17***	***+55%***	***1004***	***+7%***
	Base+CCBO *	0.13	+18%	1059	+1%
	Base+RTC$_s$	0.11	+0%	1077	+0%
	Base+MWE *	0.13	+18%	1058	+2%
Firefox 1.0	Base(LOC) *	0.12		2330	
	Base+NT *	***0.17***	***+42%***	***2189***	***+6%***
	Base+CCBO *	0.13	+8%	2299	+0%
	Base+RTC$_s$ *	0.13	+8%	2301	+0%
	Base+MWE *	0.15	+25%	2254	+3%
Firefox 1.5	Base(LOC) *	0.15		3400	
	Base+NT *	***0.22***	***+47%***	***3115***	***+8%***
	Base+CCBO	0.15	+0%	3399	+0%
	Base+RTC$_s$	0.15	+0%	3400	+0%
	Base+MWE *	0.17	+13%	3305	+3%
Firefox 2.0	Base(LOC) *	0.14		2167	
	Base+NT *	***0.18***	***+29%***	***2070***	***+4%***
	Base+CCBO	0.14	+0%	2168	+0%
	Base+RTC$_s$	0.14	+0%	2168	+0%
	Base+MWE *	0.17	+21%	2099	+3%
Continued in Table 10.					

Table 10: *Continued from Table 9.* D^2 improvement and AIC scores for topic-based cohesion and coupling metrics. The numbers in the column $\%Change$ are the percentage D^2 increase or AIC score decrease compared to the baseline model. The best model of each version of the software is marked in bold. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

System	Model	D^2	% Change	AIC	% Change
Eclipse 2.0	Base(LOC) *	0.12		4197	
	Base+NT *	0.13	+8%	4175	+0%
	Base+CCBO *	***0.13***	***+8%***	***4156***	***+0%***
	Base+RTC$_s$ *	0.13	+8%	4195	+0%
	Base+MWE *	0.13	+8%	4197	+0%
Eclipse 2.1	Base(LOC) *	0.10		4111	
	Base+NT *	0.10	+0%	4106	+0%
	Base+CCBO	0.10	+0%	4113	+0%
	Base+RTC$_s$	0.10	+0%	4112	+0%
	Base+MWE *	***0.10***	***+0%***	***4096***	***+0%***
Eclipse 3.0	Base(LOC) *	0.11		6717	
	Base+NT	0.11	+0%	6719	+0%
	Base+CCBO *	0.11	+0%	6710	+0%
	Base+RTC$_s$	0.11	+0%	6719	+0%
	Base+MWE *	***0.11***	***+0%***	***6681***	***+0%***
NetBeans 4.0	Base(LOC) *	0.04		1062	
	Base+NT	0.04	+0%	1064	+0%
	Base+CCBO *	***0.04***	***+0%***	***1058***	***+0%***
	Base+RTC$_s$	0.04	+0%	1065	+0%
	Base+MWE *	0.04	+0%	1064	+0%
NetBeans 5.0	Base(LOC) *	0.06		872	
	Base+NT *	***0.08***	***+33%***	***849***	***+3%***
	Base+CCBO	0.06	+0%	871	+0%
	Base+RTC$_s$	0.06	+0%	873	+0%
	Base+MWE	0.06	+0%	867	+0%
NetBeans 5.5.1	Base(LOC) *	0.04		3617	
	Base+NT *	***0.08***	***+50%***	***3475***	***+4%***
	Base+CCBO *	0.04	+0%	3614	+0%
	Base+RTC$_s$	0.04	+0%	3618	+0%
	Base+MWE *	0.04	+0%	3606	+0%

and coupling metrics give improvement in these systems. Note that, since some of the metrics are computed at the method level (then aggregated to obtain a file level metric), source code files that do not have methods are excluded. As a result, the D^2 and AIC score for the baseline model are slightly different from that of RQ2 (dataset is slightly different due to exclusion of some files that do not have methods). We find that 364 files were excluded from all versions of Mylyn (a total of 37 defects are excluded); 790 files were excluded from all versions of Firefox (17 defects); 6,236 files were excluded from all versions of Eclipse (330 defects); and 9,143 files were excluded from all versions of NetBeans (469 defects).

MWE gives the second best improvement over the baseline model (on average 12%). RTC_s, which gives 6% improvement on average, is not statistically different from the baseline model in most of the studied systems: low level of statistical significance indicates that the effect of RTC_s is likely due to chance. CCBO gives about 3% improvement, but is not statistically significant in 7 out of 12 versions of the four studied system.

From the results, we can see that NT outperforms other metrics whenever topic-based cohesion and coupling metrics can help explain software defects. Also, NT is statistically significant in most of the systems. This implies that the effect of NT most likely does not happen by chance, and that it is more consistent than other topic-based cohesion and coupling metrics (other metrics, in general, are not statistically significant in many studied systems). Another advantage of NT is that NT can be applied to source code files that do not have any methods, whereas MWE can only work on the files that have methods. In our case study, several defects are removed in Eclipse and NetBeans (330–469 defects) because these defects are in the files that do not have any methods. If we use MWE, then these defects will be ignored.

NT is also more intuitive and much simpler to measure than the other studied metrics (e.g., RTC_s took more than 2 weeks to compute). By examining how many topics a source code file has, practitioners can determine if a refactoring of such file is required to increase file cohesion and decrease maintenance difficulty.

NT *gives the best improvement in terms of explaining defects, and is statistically significant in most systems.* NT *is easier to interpret and much simpler to implement than other metrics, which can also help practitioners during maintenance (i.e., to identify and refactor source code files that contain many topics).*

5.2.3. Does NT *improve over state-of-the-art topic-based cohesion and coupling metrics when explaining defects?*

Adding* NT *gives 3–97% improvements to the models that are built using* LOC *and all other state-of-the-art topic-based cohesion and coupling metrics combined. In Tables 9 and 10, we have shown how each metric helps explain defects, and NT generally gives the best improvement over the baseline model. However, since some metrics have some overlapping information (i.e., moderate correlation), we want to study whether NT gives additional improvement in explaining defects over all other state-of-the-art topic-based cohesion and coupling metrics, and LOC.

PCA transforms the metrics (i.e., independent variables) into a set of uncorrelated PCs, so all the PCs are not correlated with each other. Since some topic-based cohesion and coupling metrics may be correlated, PCA solves the problem of multicollinearity and the order of the metrics in regression models does not matter anymore [32, 46]. Thus, we build a baseline model using the PCs we transformed from all topic-based metrics (except NT) and LOC. We report the D^2 improvement in explaining defects when NT is added to the baseline model (i.e., the PCA model) in Table 11. We also report the regression coefficient and the level of statistical significance (p-value). We find that adding NT gives statistically significant improvement to the baseline model (LOC and state-of-the-art topic-based cohesion and coupling metrics), giving 3–97% improvement over the baseline model.

The coefficient of NT is always positive, except in Eclipse 2.0. Positive coefficients imply that if a file has more topics, then it is more likely to be defect-prone. We do not report the coefficients for the cases where NT is not statistically significant.

Since NT and MWE have a moderate correlation relationship, we perform the same analysis for MWE, and study how much improvement MWE gives. Namely, we compute the PCs of LOC, CCBO, RTC_s, and NT, and build a regression model using these PCs. Then, we add MWE to the model and examine the improvement that MWE gives over the baseline model. Table 12

Table 11: D^2 improvement when NT is added to the baseline model that is composed of PCs of LOC and other state-of-the-art topic-based cohesion and coupling metrics (i.e., CCBO, RTC_s, and MWE). "*" indicates a p-value < 0.05. "**" indicates a p-value < 0.01. "***" indicates a p-value < 0.001.

System	Base	Base+NT	% Inc.	NT Coeff.	p-val
Mylyn 1.0	0.10	0.11	14.74	0.89	***
Mylyn 2.0	0.12	0.15	16.94	1.20	***
Mylyn 3.0	0.14	0.18	28.57	1.57	***
Firefox 1.0	0.16	0.18	11.04	1.00	***
Firefox 1.5	0.17	0.23	31.21	1.67	***
Firefox 2.0	0.17	0.19	12.94	1.01	***
Eclipse 2.0	0.14	0.14	2.92	-0.41	***
Eclipse 2.1	0.10	0.10	0.00	—	—
Eclipse 3.0	0.12	0.12	0.00	—	—
Netbeans 4.0	0.05	0.05	0.00	—	—
Netbeans 5.0	0.07	0.09	37.31	1.23	***
Netbeans 5.5.1	0.04	0.09	97.67	1.76	***

shows the result of this MWE analysis. The column *Base + MWE* has the same result as the column *Base + NT*, because these two columns report the D^2 of the full model. We include this column in the table to ease the comparison.

We can see that the baseline models in Table 12 have higher D^2 values than the ones in Table 11, which implies that including NT in the model explains more defects than including MWE in the model. In addition, MWE gives less improvement to the baseline models when compared to NT in Table 11. MWE gives an additional explanatory power in Eclipse 2.1, 3.0, and NetBeans 4.0, but these software systems also exclude many source code files that do not have methods (i.e., the dataset is reduced). Even though MWE does not consider files without methods, if we consider those files, NT gives 43% improvement in D^2 to the baseline model in NetBeans 4.0 (Table 5).

Overall, we find that although MWE and NT have a moderate correlation, NT still provides additional explanatory power to the baseline models, while being much simpler to implement, and it works on files without methods.

Table 12: D^2 improvement when MWE is added to the baseline model that is composed of PCs of LOC, CCBO, NT, and RTC_s. "*" indicates a p-value < 0.05. "**" indicates a p-value < 0.01. "***" indicates a p-value < 0.001. The column *Base+MWE* has the same number as the column *Base+NT* in Table 11, because they both refer to the same full model.

	Base	Base+MWE	% Inc.	MWE Coeff.	p-Val
Mylyn 1.0	0.11	0.11	2.83%	—	—
Mylyn 2.0	0.14	0.15	5.07%	-1.08	**
Mylyn 3.0	0.18	0.18	0.00%	—	—
Firefox 1.0	0.18	0.18	2.26%	-1.15	***
Firefox 1.5	0.23	0.23	0.44%	-0.65	*
Firefox 2.0	0.18	0.19	4.92%	-1.60	***
Eclipse 2.0	0.14	0.14	2.92%	-0.89	***
Eclipse 2.1	0.10	0.10	3.00%	-0.88	***
Eclipse 3.0	0.11	0.12	6.31%	-1.48	***
Netbeans 4.0	0.04	0.05	29.27%	-1.77	***
Netbeans 5.0	0.09	0.09	1.10%	—	—
Netbeans 5.5.1	0.08	0.09	1.19%	0.73	*

NT *gives improvements to* LOC *and all other state-of-the-art topic-based cohesion and coupling metrics. Although* NT *has a moderate correlation with* MWE*,* NT *gives more improvement than* MWE*. In addition, our previous finding (if a file has more topics then it is more likely to be defect-prone) still holds when controlling for* LOC *and other cohesion and coupling measures.*

6. Sensitivity Analysis for the Parameters of Our Approach

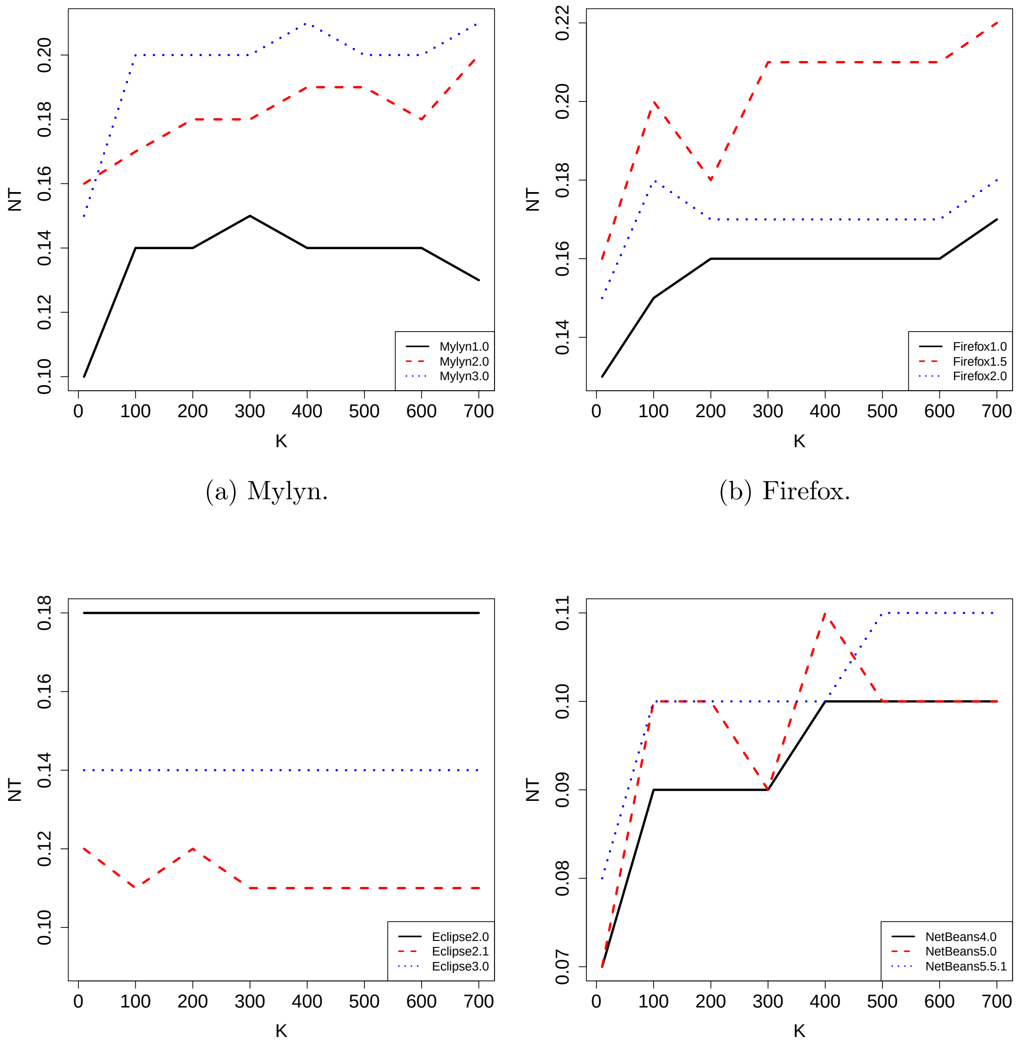
In our approach, we use several parameter values for LDA and our proposed topic-based metrics: two Dirichlet priors for smoothing (α and β), the number of iterations (II), the number of topics (K), and δ in NT and NDT. We perform a parameter sensitivity analysis to see how these parameters affect the defect explanatory power of our topic-based metrics. We do not change the LDA parameters of the topic-based cohesion and coupling metrics in Section 5, since we are only interested in comparing how different topic-based cohesion and coupling metrics can help explain defects when controlling for the LDA parameters.

In particular, we use our previous setting as a baseline (II=10,000, K=500, and δ=1%), and we change the value of each parameter to two lower and higher values (except for K, which we try more values). In short, we choose 8,000, 9,000, 11,000, and 12,000 for II, 10, 100, 200, 300, 400, 600 and 700 for K, and 0.25%, 0.5%, 2% and 4% for δ. We report the D^2 of the model after adding NT or NDT. Note that since the number of variables is the same across the models, D^2 is stable. Thus, we report D^2 because it is easier to interpret than AIC. We report only the result for NT and NDT due to their sensitivity on the parameters. For example, K may have a direct effect on NT and NDT, so changing K may affect the explanatory power of these metrics in the model. We exclude TM and DTM in this study because after changing the LDA parameters, the number of PCs resulted in TM and DTM may be different. In addition, as the LDA parameters change, the topics may also change (e.g., some topics may be merged if we use a small K) [13], so TM and DTM may be very different from our baseline, which makes the results incomparable. In this analysis, for α and β, we use the values as optimized by MALLET [56].

We find that, in general, when K is extremely small (i.e., 10), our metrics give little or no improvement over the baseline, while the results are more stable when K is larger. In addition, we find that when $delta$ is smaller, the results are relatively more stable, and varying II does not have a considerable impact on D^2. We also find that the results of our RQ1 still hold (most topics are not defect-prone, and defect-prone topics tend to be defect-prone in future versions). The skewness and the correlation of topic defect densities across different versions remain high. We manually check the topic label of the most defect-prone topics when K and II change, and we notice that the topic labels are very similar to the labels in Tables 16–19.

Since we see a larger variation in our topic-based metrics when changing K, we plot the D^2 of NT and NDT when choosing different values of K (Figures 4 and 5). We find that, even though the studied systems have various sizes, using a smaller K for smaller systems (e.g., use the same K for Mylyn and Eclipse) may not give considerably better results (as also suggested by Wallach *et al.* [81] that using a larger K may be better than using a smaller K). Our sensitivity analysis shows that the ratio between K and the size of the system may not have a considerable impact on the explanatory power of the topic-based metrics. Moreover, Wallach *et al.* [81] found that choosing a larger K does not significantly affect the quality of the generated topics. They found that the additional topics should be rarely used when LDA is assigning tokens to topics, which may be the reason that using the same K for Mylyn yields comparable performance to NetBeans, Eclipse, and Firefox. Our results also show that choosing a small K (i.e., 10) has negative results on the generated metrics, because topics in the source code cannot be separated precisely. However, when choosing a larger K, the generated metrics have similar performance after removing noise topics using a threshold.

Tables 24 – 27 in Appendix E show the complete results of our sensitivity analysis. We list the baseline D^2 of the regression models, and report the new D^2 when the parameter is changed. In most cases, D^2 values remain unchanged. When we change δ (which is used for removing noise) to a larger value, D^2, in general, decreases. A higher δ value may remove some topics that are not noise. However, when we change δ to a smaller value, the results are relatively more stable. The reason may be that most noise topics have a near-zero topic membership, so using a smaller δ does not have a considerable impact on the overall model.



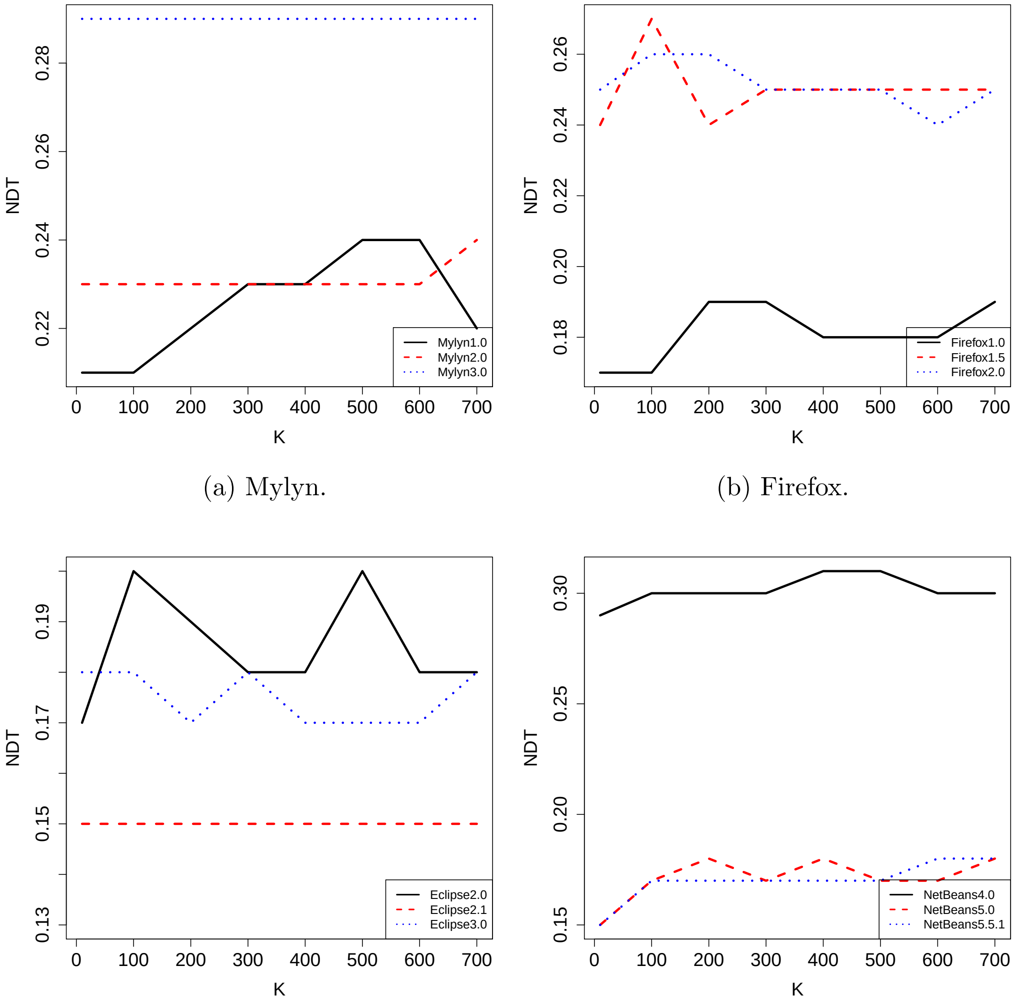
(a) Mylyn.

(b) Firefox.

(c) Eclipse.

(d) NetBeans.

Figure 4: Trend of NT over various values of K.



(a) Mylyn.

(b) Firefox.

(c) Eclipse.

(d) NetBeans.

Figure 5: Trend of NDT over various values of K.

7. Threats to Validity

The results of our case study provide an initial evaluation of using topics to explain software defects, and we show that our topic-based metric outperforms state-of-the-art topic-based cohesion and coupling metrics. However, we note the following threats to the validity of our findings.

7.1. Internal Validity

In this paper, we study the effect of topic-based metrics in regression models in order to understand their relationship with defects. However, there may be other confounding variables that are not observed. However, prior studies have shown that topic-based metrics are usually different from other traditional metrics [31, 49, 64]. In addition, we conduct correlation analysis to minimize the impact of multicollinearity.

7.2. External Validity

We considered three versions of Mylyn, Firefox, Eclipse, and NetBeans, and answered our research questions based on these systems. However, the results that we found on these systems may not necessarily generalize to all software systems. Nevertheless, we try to choose systems with different sizes and purposes to increase the generalizability of our result. Although Eclipse and NetBeans are both IDEs, we do not find many similarities. Most topic-based metrics have statistical significance in NetBeans, but not Eclipse. Thus, the performance of topic-based metrics may not have the same performance on systems in the same domain, and further studies are needed to understand the phenomenon.

7.3. Construct Validity

Parameter and Threshold Choices. Our approach involves the choice of several parameters, and there is no automated technique to find the optimal values for them. However, as shown in Section 6, our results are not particularly sensitive to the parameters that we choose. Nevertheless, further research is required to understand the effects of these parameters on the results.

Representing Concerns as Topics. Topic models such as LDA are based on machine learning techniques, which involve some probabilistic algorithms. Therefore, each computation may result in slightly different topic distributions. We use a relatively large number of Gibbs sampling iterations (II=10,000) to approximate topics distributions. Using such a high value of II, we should get more stable results. This value of II is also used in other studies when applying topic models to source code files [74, 77, 78]. As suggested by prior studies [50, 81] we choose K as 500. A larger K is able to separate the topics more accurately and additional topics will be rarely used when LDA is assigning tokens to topics (i.e., very low topic membership values) [81]. A smaller K may be more problematic, since the information may not be separated precisely [81]. The results of our sensitivity analysis also show that when K is small (i.e., 10), our topic-based metrics give little or no improvement over the baseline. We also find that our results are not particularly sensitivity to the K we used (except when K =10), even if the system is small and K is large. We apply several thresholds or statistical approaches to remove noise topics, so the remaining topics can better present the underlying concerns in the source code.

Noise in the Defect Data. We obtain the defect information from commit logs, following a similar approach proposed by Sliwerski *et al.* [70]. We make the assumption that the defect information that we obtain is correct. For example, when a developer fixes a file, he/she always includes certain related change messages (i.e., *defect fixed*, or *bug #1234*) when committing changes to the file. The data may contain some noise, which may affect our results. However, noise is unavoidable and it exists in near-perfect datasets [59]. Hassan *et al.* [38] compared automatic change message classification with classification done by six professional software developers, and found a high correlation (>0.8) between an automated classifier and a human. In summary, we believe the defect data that we use in this paper is as accurate as possible, given the limited information we can get for the studied software systems.

Choosing Baseline Metrics. Due to the nature of our topic-based metrics, we compare static topic metrics with LOC and compare historical topic-based metrics with PRE and CHURN. Potentially we can combine LOC, PRE, and CHURN together as our baseline (although the historical information may not always be present). We find that when using these three metrics as our baseline metrics, adding our topic-based metrics still gives a statistically significant improvement. We find that at least one of the topic-based metrics is statistically significant for each version of the studied systems.

Choosing Different Values of K for LSI. We chose to use the same K for LSI when comparing topic-based metrics. However, since topics in LSI have different mathematical meanings than topics in LDA, we investigate CCBO and CLCOM5 using different values of K. In particular, we use the heuristic proposed by Kuhn *et al.* [45] to compute K: $(m * n)^{0.2}$, where m is the total number of files and n the total number of terms over all files. Table 13 shows the K computed using the heuristic, and the improvement of

Table 13: Improvement of CCBO (LSI-based metrics) over the baseline (using *LOC*) when computed using the *K* suggested by Kuhn *et al.* [45]. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

System	K	**Model**	$\mathbf{D}^2$	**% Change**
Mylyn 1.0	–	Base(LOC) *	0.06	–
	500	Base+CCBO	0.06	+0%
	19	Base+CCBO *	0.07	+17%
Mylyn 2.0	–	Base(LOC) *	0.10	–
	500	Base+CCBO	0.10	+0%
	19	Base+CCBO *	0.12	+20%
Mylyn 3.0	–	Base(LOC) *	0.11	–
	500	Base+CCBO *	0.13	+18%
	21	Base+CCBO *	0.13	+18%
Firefox 1.0	–	Base(LOC) *	0.12	–
	500	Base+CCBO *	0.13	+8%
	49	Base+CCBO *	0.15	+25%
Firefox 1.5	–	Base(LOC) *	0.15	–
	500	Base+CCBO	0.15	+0%
	50	Base+CCBO *	0.18	+20%
Firefox 2.0	–	Base(LOC) *	0.14	–
	500	Base+CCBO	0.14	+0%
	51	Base+CCBO *	0.18	+29%
Eclipse 2.0	–	Base(LOC) *	0.12	–
	500	Base+CCBO *	0.13	+8%
	39	Base+CCBO *	0.14	+17%
Eclipse 2.1	–	Base(LOC) *	0.10	–
	500	Base+CCBO	0.10	+0%
	41	Base+CCBO	0.10	+0%
Eclipse 3.0	–	Base(LOC) *	0.11	–
	500	Base+CCBO *	0.11	+0%
	44	Base+CCBO	0.11	+0%
NetBeans 4.0	–	Base(LOC) *	0.04	–
	500	Base+CCBO *	0.04	+0%
	36	Base+CCBO	0.04	+0%
NetBeans 5.0	–	Base(LOC) *	0.06	–
	500	Base+CCBO	0.06	+0%
	45	Base+CCBO *	0.06	+0%
NetBeans 5.5.1	–	Base(LOC) *	0.04	–
	500	Base+CCBO *	0.04	+0%
	53	Base+CCBO	0.04	+0%

Table 14: Average correlation between NT and POST, and NDT and POST across the three versions of each studied system.

	Eclipse	Mylyn	Firefox	NetBeans
Avg. Cor. bwt. NT and POST	0.03	0.32	0.15	0.09
Avg. Cor. bwt. NDT and POST	0.01	0.24	0.22	0.14

the new CCBO (computed using the new K) over the baseline. We still find that CLCOM5 to be highly correlated with LOC; however, we found that the new CCBO in general gives a larger improvement over the baseline. The new CCBO is better in seven out of 12 studied systems compared to using the old CCBO (computed using $K = 500$). Nevertheless, our NT still gives the largest improvement over the baseline among all studied topic-based metrics. Our study shows the same K may not work well for different types of topic models (e.g., LDA versus LSI), and future should consider our results when doing analysis using topic models.

Results in Eclipse when Compared to Other Systems. We find that topic-based cohesion and coupling metrics give less improvement in Eclipse compared to other studied software systems. Moreover, the coefficients of NT is small and almost zero in Eclipse (as shown in Table 7, the average coefficient for three studied versions of Eclipse is 0.06), which differs with our findings in other studied software systems. We perform a brief study to understand how NT and NDT are different in Eclipse than in the other studied systems. We find that the variances and skewnesses of NT and NDT are not considerably different across different studied systems. However, we find that the Spearman correlation between NT and POST, and the correlation between NDT and POST in Eclipse are smaller than the other studied systems (Table 14). Lower correlations in Eclipse may give initial evidence as to why the improvement of NT and NDT in explaining defects is not as good compared to other studied systems.

8. Related Work

8.1. Applying Topic Models to Software Engineering Tasks

Recently, many researchers have used topic modeling approaches to understand software systems from a different point of view than from the traditional structural and historical views [23]. For example, Kuhn *et al.* [45] used Latent Semantic Indexing (LSI) to cluster the files in a software system according to the similarity of word usage. Maskeri *et al.* [55] were the first to apply LDA to source code to uncover its conceptual concerns. Prior studies used topics to study the evolution of concerns in the source code [48, 74, 77, 78].

Other uses of topic models in software engineering include concept location [24, 50, 63, 66, 67], traceability link recovering [6], building source code search engines [79], and even test case prioritization [75].

8.2. Using Topic Models to Study Software Defects

A few recent studies have tried to establish a link between topics and defects. For example, Liu *et al.* [49] propose a new metric, called Maximal Weighted Entropy (MWE), to measure the level of cohesion in a software system. A more detailed description of MWE is in Section 5.1. Nguyen *et al.* [60] use LDA to predict defects. The authors first apply LDA to the studied systems using K=5 topics, and for each source code file they multiply the topic memberships by the file's LOC. As a result, the authors obtain five topic variables for each file, and use these variables to build a prediction model. Using this approach, the authors provide initial evidence that it is possible to explain defects using topic-based metrics. In this paper, we use a different set of topic-based metrics. We are interested in explaining defects while also controlling for the standard defect explainers, i.e., LOC, churn, and pre-release defects. We compute the $\mathsf{CDDT}_{\mathsf{POST}}$ of each topic, and study how the defect-proneness of topics evolves overtime. In addition, we consider a larger number of topics in order to capture more accurate and detailed conceptual concerns. We use PCA to extract the most effective topics and avoid the possible problem of multicollinearity and minimize the effects of overfitting. We compare our topic-based metrics with other state-of-the-arts topic-based metrics.

Chen *et al.* [22] use topics to explain software defects. They find that there is only a few defect-prone topics in a software system, and topics can give additional defect explanatory to traditional state-of-the-art metrics. We expand this research by including more systems, comparing the metrics with state-of-the-art, and provide a sensitivity analysis of the parameters and thresholds that are used in their approach.

Prior studies capture cohesion using latent semantic indexing (LSI) [53, 54], and show that it is possible to predict defects using the level of cohesion in a file [54]. Poshyvanyk *et al.* [64] also use LSI to capture level of coupling by the cosine similarity score among files. Ujhazi *et al.* [80] propose a cohesion and a coupling metric, based on previous work [53, 54, 64], and provide a parametric version of the metrics (which we compare our NT metric with). In addition, Gethers *et al.* [31] use relational topic models to uncover coupling among files, and use the level of coupling for prioritizing file inspection work. We compare our metric with the metrics by Ujhazi *et al.* [80] and Gethers *et al.* [31] in Section 5. We do not compare the metrics by Marcus *et al.* [53, 54] and Poshyvanyk *et al.* [64], since Ujhazi *et al.* propose new metrics based on their metrics.

8.3. Understanding and Tuning the Parameters of Topic Model

Since the performance of topic models is correlated with the topic model parameters, a number of studies try to understand the impact of these parameters when applying topic models on software engineering tasks. Grant *et al.* [33, 34] propose a heuristic to measure the quality of the generated topics on source code files. Binkley *et al.* [10] applied LDA on artificial data to understand the effect of each LDA parameter on the resulting topics. Panichella *et al.* [62] propose an approach to search for the best set of LDA parameters according to clustering distances. Biggers *et al.* [9] tried different combination of LDA parameters and input for LDA-based feature location techniques, and provided some recommendation for the parameters when using LDA for feature location. Since most prior studies usually try to understand the effect of LDA parameters for a specific software engineering task [23], it is difficult to generalize the findings on other tasks. Thus, in this paper, we also include a sensitivity analysis where we vary the LDA parameters and thresholds. We found the parameters and thresholds do not have significant impact on how well the resulting topic-based metrics explain defect.

9. Conclusions and Future Work

In this paper, we aim to understand the relationship between the *conceptual concerns* in source code files, i.e., their technical content, with their defect-proneness. To do so, we captured the concerns in each file using *topics*, and proposed new metrics based on these topics. In particular, we considered the defect history of each topic, which we hypothesized would help better explain the defect-proneness of the files.

To evaluate our new metrics, we performed a detailed case study on multiple versions of four large, real-world systems: Mylyn, Mozilla Firefox, Eclipse, and NetBeans. The highlights of our study results include:

- A small number of topics is much more defect-prone than others.
- The defect-proneness of a topic tends to hold over time.
- The more topics a file has, the higher the chances it has defects.
- The more *defect-prone* topics (determined using pre-release defects) a file has, the higher the chances that it has post-release defects.
- Our proposed topic-based metrics (number of topics, topic membership, number of defect-prone topics, and defect-prone topic membership) provide additional defect explanatory power over existing static (i.e., LOC) and historical (i.e., PRE and churn) metrics, suggesting that our metrics provide additional information about the quality of the code. Further studies should consider using such metrics alongside traditional metrics for building defect prediction models.
- Our NT metric, which measures the level of cohesion in a file, outperforms other topic-based cohesion and coupling metrics. Practitioners may benefit from including our metric when studying software cohesion and coupling using topic models.

Our goal is to examine the improvement in the defect explanatory power to traditional static and historical metrics using topics. Since topics are derived from the source code files and can also be combined with pre-release defects to discover defect-prone topics, we examine the improvement of our topic-based metrics on static and historical metrics separately. It is possible to combine different topics metrics, such as NT and NDT, in our defect explanation models. However, we leave the full investigation of all possible metric combinations to future work. We also plan to consider using other information sources, such as defect reports or mailing lists, to help explain defects. For instance, we can add topics derived from the commit messages to the corresponding files.

Acknowledgements

We thank Dr. Yasutaka Kamei for providing us the bug datasets of the studied systems that are used in this paper.

References

[1] Eclipse. http://www.eclipse.org/, 2012.

[2] Mozilla firefox. http://www.mozilla.org/, 2012.

[3] Mylyn. http://www.eclipse.org/mylyn/, 2012.

[4] Netbeans. http://netbeans.org/, 2012.

[5] Edward B. Allen and Taghi M. Khoshgoftaar. Measuring coupling and cohesion: An information-theory approach. In *Proceedings of the 6th International Symposium on Software Metrics*, pages 119–, 1999.

[6] H. U Asuncion, A. U Asuncion, and R. N Taylor. Software traceability with topic modeling. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 95–104, 2010.

[7] Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead, and Sushil K. Bajracharya. A theory of aspects as latent topics. In *Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications*, pages 543–562, 2008.

[8] James M. Bieman and Byung-Kyoo Kang. Measuring design-level cohesion. *IEEE Transactions on Software Engineering*, 24(2):111–124, 1998.

[9] Lauren R. Biggers, Cecylia Bocovich, Riley Capshaw, Brian P. Eddy, Letha H. Etzkorn, and Nicholas A. Kraft. Configuring latent dirichlet allocation based feature location. *Empirical Softw. Engg.*, 19(3):465–500, June 2014.

[10] David Binkley, Daniel Heinz, Dawn Lawrie, and Justin Overfelt. Understanding lda in source code analysis. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 26–36, 2014.

[11] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th Symposium on the Foundations of Software Engineering and the 13rd European Software Engineering Conference*, pages 4–14, 2011.

[12] S. Biyani and P. Santhanam. Exploring defect data from development and customer usage on software modules over multiple releases. In *Proceedings of the 9th International Symposium on Software Reliability Engineering*, pages 316–320, 1998.

[13] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003.

[14] S. Boslaugh and P.A. Watters. *Statistics in a Nutshell: A Desktop Quick Reference*. In a Nutshell (O'Reilly). O'Reilly Media, 2008.

[15] Lionel C. Briand, John W. Daly, and Jürgen Wüst. A unified framework for cohesion measurement in object-orientedsystems. *Empirical Software Engineering*, 3(1):65–117, July 1998.

[16] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479, December 1992.

[17] Anderson David R Burnham Kenneth P. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods Research*, 33:467–479, November 2004.

[18] Heung Seok Chae, Yong Rae Kwon, and Doo-Hwan Bae. A cohesion measure for object-oriented classes. *Software Practice & Experience*, 30(12):1405–1431, October 2000.

[19] Jonathan Chang and David Blei. Relational topic models for document networks. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, 2009.

[20] Tse-Hsun Chen. Studying software quality using topic models. Master's thesis, School of Computing, Queen's University, 2013.

[21] Tse-Hsun Chen. http://sailhome.cs.queensu.ca/replication/topicExplain, 2014.

[22] Tse-Hsun Chen, S. W. Thomas, Meiyappan Nagappan, and A. E. Hassan. Explaining software defects using topic models. In *Proceedings of the 9th Working Conference on Mining Software Repositories*, 2012.

[23] Tse-Hsun Chen, StephenW. Thomas, and Ahmed E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, pages 1–77, 2015.

[24] Brendan Cleary, Chris Exton, Jim Buckley, and Michael English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14(1):93–130, 2008.

[25] S G Crawford, A A McIntosh, and D Pregibon. An analysis of static metrics and faults in c software. *Journal of Systems and Software*, 5:37–48, 1985.

[26] E.E. Cureton and R.B. D'Agostino. *Factor Analysis: An Applied Approach*. Lawrence Erlbaum Associates, 1993.

[27] Marco DAmbros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. pages 31–41, 2010.

[28] Marc Eaddy, Thomas Zimmermann, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, and Alfred V. Aho. Do crosscutting concerns cause defects? *IEEE Transactions on Software Engineering*, 34:497–515, July 2008.

[29] R.M. Feldman and C. Valdez-Flores. *Applied Probability and Stochastic Processes*. Springer, 2010.

[30] N.E. Fenton. *Software metrics: a rigorous approach.* Chapman & Hall, 1991.

[31] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in object-oriented software systems. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–10, 2010.

[32] M.A. Golberg and H.A. Cho. *Introduction to regression analysis.* 2004.

[33] Scott Grant and James R. Cordy. Estimating the optimal number of latent concepts in source code analysis. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, pages 65–74, 2010.

[34] Scott Grant, James R. Cordy, and David B. Skillicorn. Using heuristics to estimate an appropriate number of latent topics in source code analysis. *Science of Computer Programming*, 78(9):1663 – 1678, 2013.

[35] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, October 2005.

[36] C.T. Haan. *Statistical methods in hydrology.* Iowa State University Press, 1977.

[37] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic review of fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, PP(99), 2011.

[38] Ahmed E. Hassan. Automated classification of change messages in open source projects. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 837–841, 2008.

[39] Matthew D. Hoffman, David M. Blei, and Francis R. Bach. Online learning for latent dirichlet allocation. In *Neural Information Processing Systems*, pages 856–864, 2010.

[40] T. Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 50–57, 1999.

[41] S. Ihara. *Information Theory for Continuous Systems.* World Scientific, 1993.

[42] Graylin Jay, Joanne E. Hale, Randy K. Smith, David P. Hale, Nicholas A. Kraft, and Charles Ward. Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship. *Journal of Software Engineering and Applications*, 2:137–143, 2009.

[43] I.T. Jolliffe. *Principal component analysis.* Springer-Verlag, 2002.

[44] Stephen H. Kan. *Metrics and Models in Software Quality Engineering.* Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2002.

[45] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49:230–243, March 2007.

[46] M.H. Kutner, C. Nachtsheim, and J. Neter. *Applied linear regression models.* 2004.

[47] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 1977.

[48] E. Linstead, C. Lopes, and P. Baldi. An application of latent Dirichlet allocation to analyzing software evolution. In *Proceedings of Seventh International Conference on Machine Learning and Applications*, pages 813–818, 2008.

[49] Yixun Liu, D. Poshyvanyk, R. Ferenc, T. Gyimothy, and N. Chrisochoides. Modeling class cohesion as mixtures of latent topics. In *Proceedings of the 25th International Conference on Software Maintenance*, pages 233 –242, 2009.

[50] Stacy K. Lukins, Nicholas A. Kraft, and Letha H. Etzkorn. Bug localization using latent dirichlet allocation. *Information and Software Technology*, 52:972–990, September 2010.

[51] A. Macro and J.N. Buxton. *The craft of software engineering.* 1987.

[52] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[53] Andrian Marcus and Denys Poshyvanyk. The conceptual cohesion of classes. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 133–142, 2005.

[54] Andrian Marcus, Denys Poshyvanyk, and Rudolf Ferenc. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, 34(2):287–300, March 2008.

[55] Girish Maskeri, Santonu Sarkar, and Kenneth Heafield. Mining business topics in source code using latent Dirichlet allocation. In *Proceedings of the 1st India Software Engineering Conference*, pages 113–120, 2008.

[56] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[57] T. Menzies, A Butcher, D. Cok, A Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, 39(6):822–834, June 2013.

[58] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of 27th International Conference on Software Engineering*, pages 284–292, 2005.

[59] Thanh H. D. Nguyen, Bram Adams, and Ahmed E. Hassan. A case study of bias in bug-fix datasets. In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 259–268, 2010.

[60] Tung Thanh Nguyen, Tien N. Nguyen, and Tu Minh Phuong. Topic-based defect prediction. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 932–935, 2011.

[61] A. Oram and G. Wilson. *Making Software: What Really Works, and Why We Believe It*. O'Reilly Series. O'Reilly Media, Incorporated, 2010.

[62] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 522–531, 2013.

[63] D. Poshyvanyk, Y. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, pages 420–432, 2007.

[64] Denys Poshyvanyk and Andrian Marcus. The conceptual coupling metrics for object-oriented systems. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 469–478, 2006.

[65] Adrian Raftery. Bayesian model selection in social research (with discussion). *Sociological Methodology*, 25:111–163, 1995.

[66] Shivani Rao and Avinash Kak. Retrieval from software libraries for bug localization: A comparative study of generic and composite text models. In *Proceeding of the 8th Working Conference on Mining Software Repositories*, pages 43–52, 2011.

[67] Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Using structural and textual information to capture feature coupling in object-oriented software. *Empirical Software Engineering*, 16(6), 2011.

[68] J. Rosenberg. Some misconceptions about lines of code. In *Proceedings of the 4th International Symposium on Software Metrics*, pages 137–142, 1997.

[69] Sandra A. Slaughter, Donald E. Harter, and Mayuram S. Krishnan. Evaluating the cost of software quality. *Communications of the ACM*, 41:67–73, August 1998.

[70] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, 2005.

[71] I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011.

[72] J.H. Stapleton. *Models for probability and statistical inference: theory and applications*. 2008.

[73] S. W. Thomas. *Mining software repositories with topic models*. PhD thesis, School of Computing, Queen's University, 2012.

[74] S. W. Thomas, B. Adams, D. Blostein, and A. E. Hassan. Studying software evolution using topic models. *Science of Computer Programming*, pages 1–23, 2013.

[75] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, pages 1–31, 2012.

[76] Stephen W. Thomas. Preprocessing tool. https://github.com/doofuslarge/lscp, 2012.

[77] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Validating the use of topic models for software evolution. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 55–64, 2010.

[78] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 173–182, 2011.

[79] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent Dirichlet allocation for automatic categorization of software. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pages 163–166, 2009.

[80] Bela Ujhazi, Rudolf Ferenc, Denys Poshyvanyk, and Tibor Gyimothy. New conceptual coupling and cohesion metrics for object-oriented systems. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, pages 33–42, 2010.

[81] H. Wallach, D. Mimno, and A. McCallum. Rethinking LDA: Why priors matter. *Proceedings of Neural Information Processing Systems, Vancouver, BC*, 2009.

[82] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 977–984, 2006.

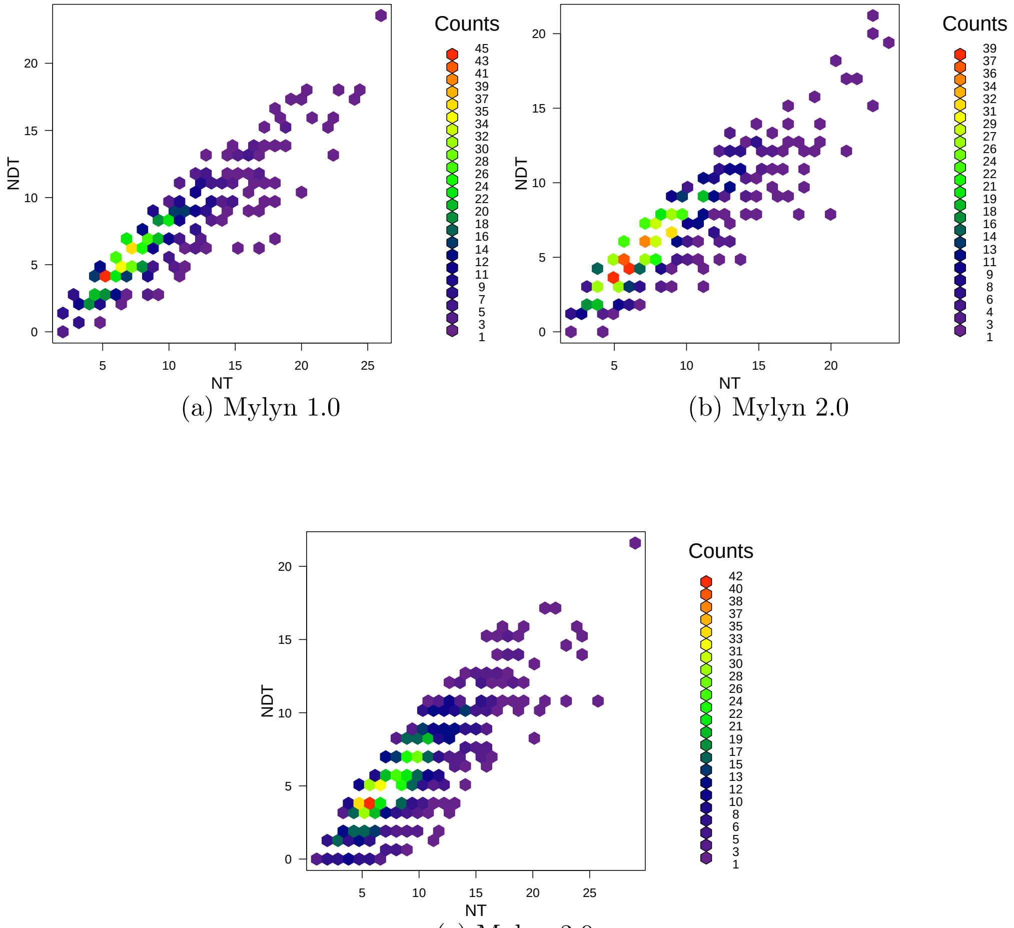
A. A Five-number Summary and Skewness of $\mathsf{CDDT}_{\mathsf{POST}}$

Table 15: A five-number summary and skewness of the $\mathsf{CDDT}_{\mathsf{POST}}$ of topics in each version in the studied systems. The $\mathsf{CDDT}_{\mathsf{POST}}$ is highly skewed, and most of the topics have a $\mathsf{CDDT}_{\mathsf{POST}}$ value close to zero.

	Min.	1st Qu.	Median	3rd Qu.	Max.	Skewness	Skewness of top 25% defect-prone topics
Mylyn 1.0	0.00	0.00	0.00	0.01	0.33	7.18	4.54
Mylyn 2.0	0.00	0.00	0.01	0.02	0.50	7.41	5.19
Mylyn 3.0	0.00	0.00	0.00	0.01	0.18	6.00	4.10
Eclipse 2.0	0.00	0.00	0.01	0.03	1.66	13.28	7.64
Eclipse 2.1	0.00	0.00	0.01	0.03	1.16	7.92	4.63
Eclipse 3.0	0.00	0.01	0.02	0.06	1.25	4.90	3.24
Firefox 1.0	0.00	0.00	0.00	0.00	0.07	6.44	3.61
Firefox 1.5	0.00	0.00	0.00	0.00	0.09	5.88	3.23
Firefox 2.0	0.00	0.00	0.00	0.00	0.07	7.96	4.14
NetBeans 4.0	0.00	0.00	0.00	0.00	0.17	10.29	6.12
NetBeans 5.0	0.00	0.00	0.00	0.00	0.06	5.62	2.98
NetBeans 5.5.1	0.00	0.00	0.00	0.01	0.13	4.08	2.67

B. Hexbin Plots of NDT vs NT

In the hexbin plots, the data points (i.e., NT and NDT) are bounded by hexagons, and the colour of the hexagon represents the frequency of the data points.

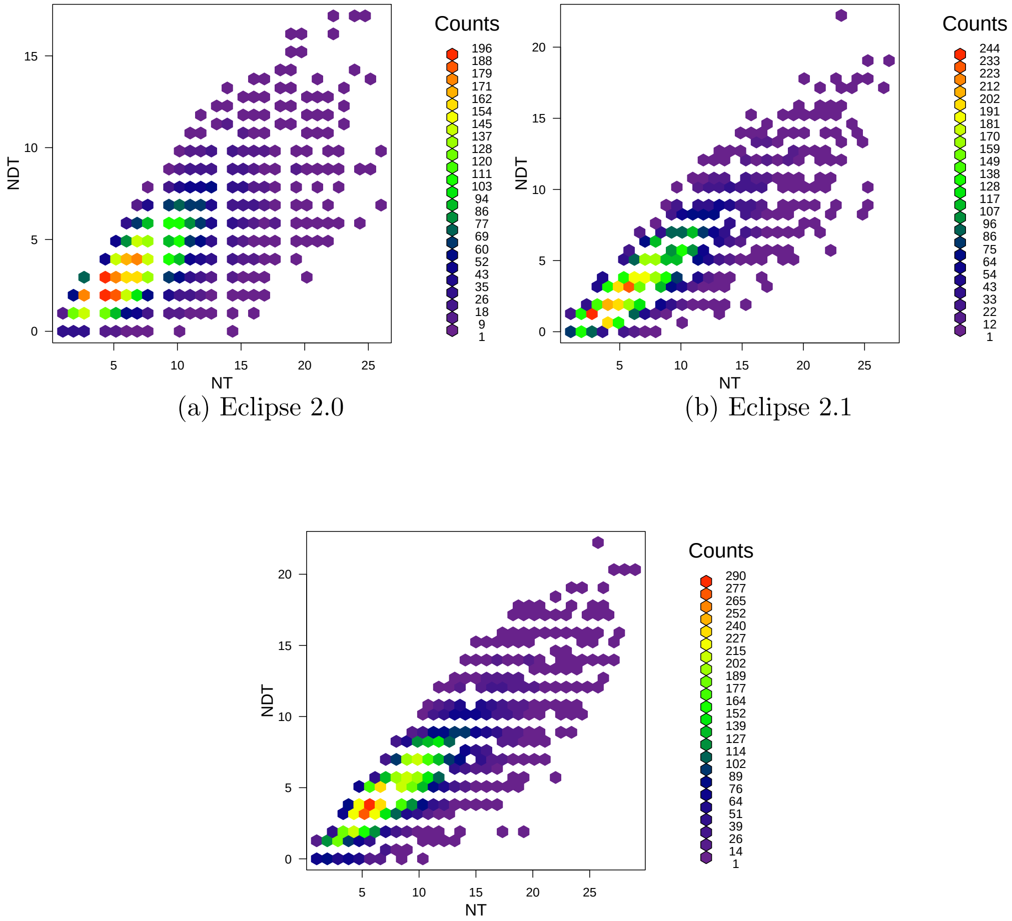


(a) Mylyn 1.0

(b) Mylyn 2.0

(c) Mylyn 3.0

Figure 6: Hexbin plots of NDT vs NT for Mylyn.

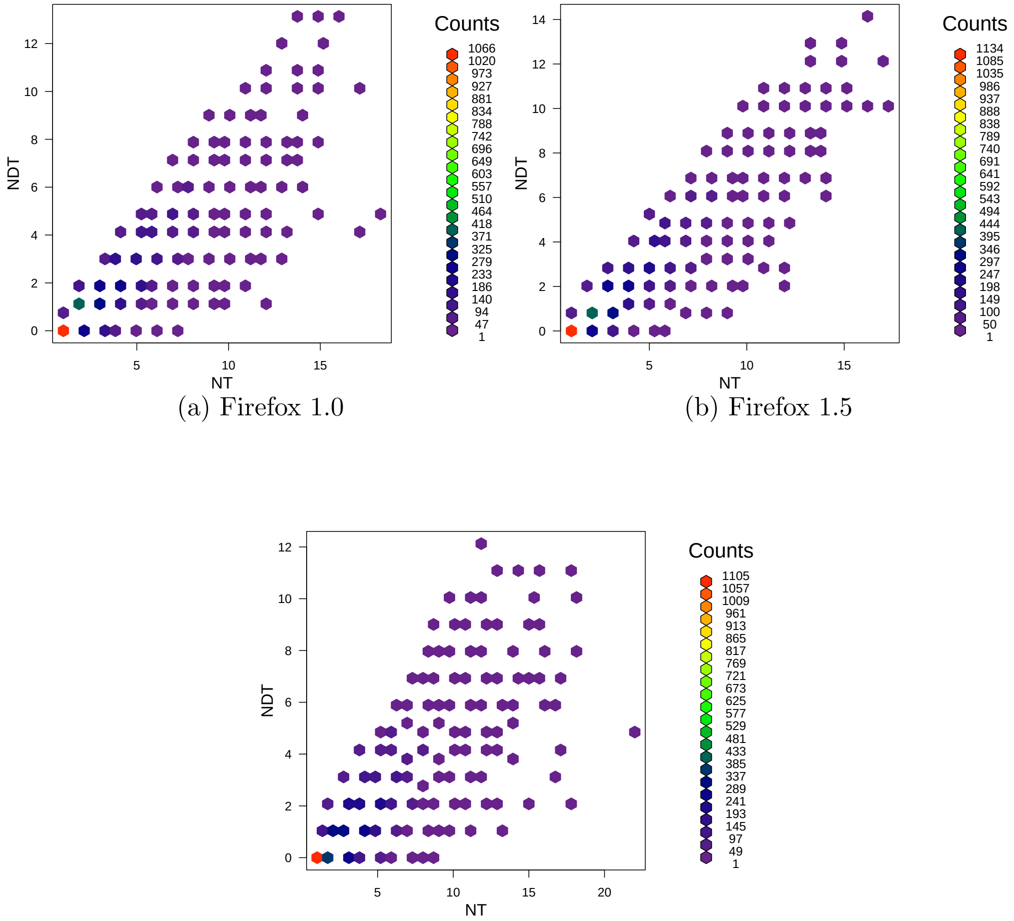


(a) Eclipse 2.0

(b) Eclipse 2.1

(c) Eclipse 3.0

Figure 7: Hexbin plots of NDT vs NT for Eclipse.

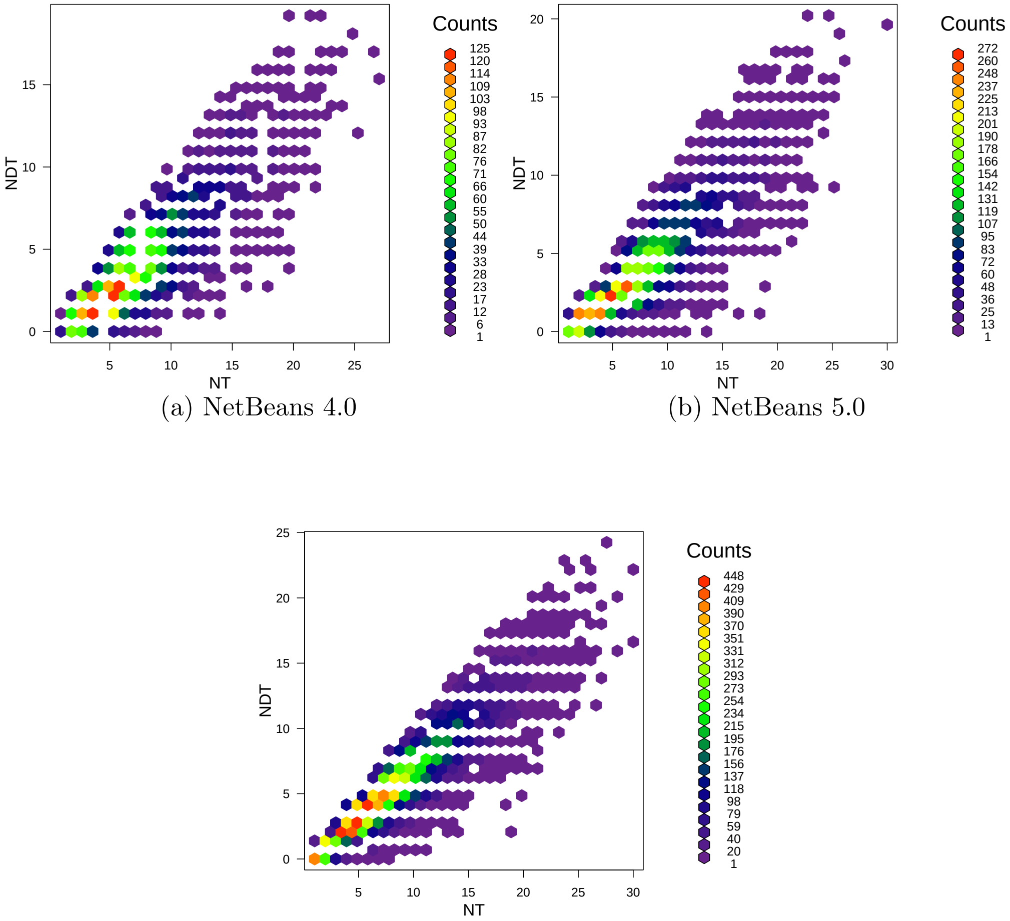


(a) Firefox 1.0

(b) Firefox 1.5

(c) Firefox 2.0

Figure 8: Hexbin plots of NDT vs NT for Firefox.



(a) NetBeans 4.0

(b) NetBeans 5.0

(c) NetBeans 5.5.1

Figure 9: Hexbin plots of NDT vs NT for NetBeans.

C. Top Words and Defect Densities of the Most/Least Defect-prone Topics

Table 16: Top words and defect densities of the most/least defect-prone topics in our studied systems. The number on the left-hand side of each column represents the topic ID.

	Most Defect-Prone			Least Defect-Prone	
	Top words	Density		Top words	Density
Mylyn 1.0					
421	mylar, eclips, eclips_mylar, mylar_intern, mylar_task	0.334	405	src, dest, base, imag, fragment, imag_pattern	<0.001
164	task, list, task_list, task_ui, ui, plugin	0.182	178	lower_color, part, put_light green_lower, jface, medium	<0.001
400	test, suit, test_suit, add_test, add, suit_add	0.180	175	monitor, gzip, configur, key, bugzilla_attribut, iter	<0.001
Mylyn 2.0					
143	task, eclips, eclips_mylyn, mylyn, ui, task_ui	0.502	405	src, dest, base, imag, fragment, imag_pattern	<0.001
457	eclips, mylyn, eclips_mylyn, intern, mylyn_intern, core	0.244	178	lower_color, part, put_light, green_lower, jface, medium	<0.001
164	task, list, task_list task_ui, ui, plugin	0.207	175	monitor, gzip, configur, key, bugzilla_attribut, iter	<0.001
Mylyn 3.0					
143	task, eclips, eclips_mylyn, mylyn, ui, task_ui	0.181	178	lower_color, part, put_light, green_lower, jface, medium	<0.001
457	eclips, mylyn, eclips_mylyn, intern, mylyn_intern, core	0.111	310	aa, comparison_check, comparison, check, check_aa, aa_comparison	<0.001
168	repositori, task_repositori, task_core, repositori	0.092	6	select, caller, calle, editor, part, foo	<0.001
Continued in Table 17.					

Table 17: *Continued from Table 16.* Top words and defect densities of the most/least defect-prone topics in our studied systems. The number on the left-hand side of each column represents the topic ID.

	Most Defect-Prone			**Least Defect-Prone**	
	Top words	Density		Top words	Density
Eclipse 2.0					
174	express, method, declar, ast, node, astnod	1.663	116	0xff, 0xff_0xff, src, dst, 0xa, 0xf	<0.001
496	option, local, seccion, folder, ccv_core, local option	0.691	146	printer, data, printer_data, code, error, dispos	<0.001
492	team, eclips_team, eclips, ccv, intern_ccv, team_intern	0.325	316	run, line, offset, style, length, item	<0.001
Eclipse 2.1					
143	form, toolkit, dfm, nfm, ui, eclips_ui	1.164	192	arg, vtbl, arg_arg, guid, iidfrom, system	<0.001
131	ant, eclips, task, eclips_ant, ui, intern	0.779	325	token, scribe, align, print, scribe_print, space	<0.001
233	bundl, recourc, resourc_bundl, kei, bundl_resourc, messag	0.572	116	0xff, 0xff_0xff, src, dst, 0xa, 0xf	<0.001
Eclipse 3.0					
462	memori, block,render, memori_block, view, address	1.247	330	packet, print, id, command, stream, spy	<0.001
169	transfer,data,code, transfer_data, java, object	0.708	182	array, constant, array_dim, dim, pixbuf, paramet	<0.001
131	ant, eclips, task, eclips_ant, ui, intern	0.700	270	pt, ph, pt_arg, arg, pg, wm	<0.001

Continued in Table 18.

Table 18: *Continued from Table 17.* Top words and defect densities of the most/least defect-prone topics in our studied systems. The number on the left-hand side of each column represents the topic ID.

	Most Defect-Prone			**Least Defect-Prone**	
	Top words	Density		Top words	Density
Firefox 1.0					
462	list, val, isvgvalu, modifi, observ, imethodimp	0.067	359	ghhd, sbz_yxkgd, yxkgd, sbz, yxkgd_ghhd, vghle_sbz	<0.001
381	frame, svgframe, comptr, queri, kid, add	0.038	280	sane, plugin, zoom, sane_plugin, instanc, error	<0.001
101	rv, rv_rv, comptr, nsresult, fail, fail_rv	0.038	361	child, border, spec, num, color, col	<0.001
Firefox 1.5					
305	elem, rv, length, map, rv_rv, comptr	0.088	359	ghhd, sbz_yxkgd, yxkgd, sbz, yxkgd_ghhd, vghle_sbz	<0.001
413	xform, elem, model, wrapper, instanc, xform_xpath	0.087	280	sane, plugin, zoom, sane_plugin, instanc, error	<0.001
101	rv, rv_rv, comptr, nsresult, fail, fail_rv	0.078	335	ck, rv, pr, log, modlog, log_modlog	<0.001
Firefox 2.0					
168	param, info, pruint, xpttype, val, count	0.071	359	ghhd, sbz_yxkgd, yxkgd, sbz, yxkgd_ghhd, vghle_sbz	<0.001
305	elem, rv, length, map, rv_rv, comptr	0.061	100	frame, pfd, span, psd, width, line	<0.001
80	access, state, retval, shell, node, comptr	0.048	280	sane, plugin, zoom, sane_plugin, instanc, error	<0.001

Continued in Table 19.

Table 19: *Continued from Table 18.* Top words and defect densities of the most/least defect-prone topics in our studied systems. The number on the left-hand side of each column represents the topic ID.

	Most Defect-Prone			**Least Defect-Prone**	
	Top words	Density		Top words	Density
NetBeans 4.0					
182	model, node, type, tree, tree_model, unknown	0.168	236	node, layout, properti, form, code, buf	<0.001
372	grid, bag, grid_bag, constraint, bag_constraint, awt	0.058	211	level, sourc_level, platform_kei, modul, chang, version	<0.001
219	project, helper, ant, properti, project_helper, evalu	0.050	455	prop, set, adaptor, dmd, sqlexcept, max	<0.001
NetBeans 5.0					
57	path, cp, classpath, recourc, implement, java	0.058	236	node, layout, properti, form, code, buf	<0.001
484	queri, javadoc, binari, root, file, binari_queri	0.045	211	level, sourc_level, platform_kei, modul, chang, version	<0.001
375	artifact, path, librari, project, ant, ant_artifact	0.044	455	prop, set, adaptor, dmd, sqlexcept, max	<0.001
NetBeans 5.5.1					
97	midp, compon, present, vmd, modul_vmd, modul	0.127	14	properti, pw, tf, properti_sheet, sheet, text	<0.001
389	token, token_token, offset, sequenc, lexer, token_id	0.125	73	jj, kind, cur, state, activ, po	<0.001
142	test, suit, junit, test_test, test_suit, netbean	0.110	39	jelli, mbean, constant, jelli_constant, nfwo, helper	<0.001

D. Pair-wise correlation between all topic-based cohesion and coupling metrics and LOC

Table 20: Pair-wise correlation between all topic-based cohesion and coupling metrics and LOC. "*" indicates a p-value < 0.05.

			Mylyn 1.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.49*	—	—	—	—
RTCs	0.38*	-0.35*	—	—	—
CCBO	-0.15*	0.13*	-0.04	—	—
CLCOM5	0.12*	0.01	-0.04	-0.10*	—
LOC	0.38*	-0.37*	0.04	-0.17*	0.63*
			Mylyn 2.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.48*	—	—	—	—
RTCs	0.02	-0.05	—	—	—
CCBO	-0.15*	0.12*	0.04	—	—
CLCOM5	0.12*	-0.02	-0.07*	-0.10*	—
LOC	0.37*	-0.39*	-0.03	-0.16*	0.64*
			Mylyn 3.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.51*	—	—	—	—
RTCs	0.00	-0.03	—	—	—
CCBO	-0.19*	0.14*	0.00	—	—
CLCOM5	0.15*	-0.01	0.00	-0.06*	—
LOC	0.42*	-0.36*	0.04	-0.15*	0.66*

Continued in Table 21.

Table 21: *Continued from Table 20.* Pair-wise correlation between all topic-based cohesion and coupling metrics and LOC. “*” indicates a p-value < 0.05.

			Firefox 1.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.63*	—	—	—	—
RTCs	0.04*	0.12*	—	—	—
CCBO	-0.53*	0.49*	0.16*	—	—
CLCOM5	0.34*	-0.30*	-0.26*	-0.24*	—
LOC	0.45*	-0.51*	-0.31*	-0.33*	0.78*
			Firefox 1.5		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.62*	—	—	—	—
RTCs	-0.01	0.01	—	—	—
CCBO	-0.52*	0.48*	0.00	—	—
CLCOM5	0.32*	-0.28*	0.01	-0.21*	—
LOC	0.43*	-0.49*	0.02	-0.30*	0.78*
			Firefox 2.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.63*	—	—	—	—
RTCs	0.02	-0.02	—	—	—
CCBO	-0.51*	0.48*	-0.02	—	—
CLCOM5	0.33*	-0.28*	0.00	-0.20*	—
LOC	0.44*	-0.49*	0.00	-0.30*	0.78*

Continued in Table 22.

Table 22: *Continued from Table 21.* Pair-wise correlation between all topic-based cohesion and coupling metrics and LOC. “*” indicates a p-value < 0.05.

Eclipse 2.0

	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.47*	—	—	—	—
RTCs	0.35*	-0.39*	—	—	—
CCBO	-0.49*	0.29*	-0.15*	—	—
CLCOM5	0.10	-0.01*	0.07*	0.01*	—
LOC	0.35*	-0.40*	0.23*	-0.16*	0.70*

Eclipse 2.1

	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.48*	—	—	—	—
RTCs	0.00	0.00	—	—	—
CCBO	-0.47*	0.29*	0.00	—	—
CLCOM5	0.14*	-0.01*	0.01	0.02*	—
LOC	0.40*	-0.38*	0.01	-0.15*	0.71*

Eclipse 3.0

	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.47*	—	—	—	—
RTCs	-0.01	0.00	—	—	—
CCBO	-0.39*	0.31*	0.03*	—	—
CLCOM5	0.16*	-0.01	0.02	0.04*	—
LOC	0.41*	-0.38*	0.01	-0.15*	0.71*

Continued in Table 23.

Table 23: *Continued from Table 22.* Pair-wise correlation between all topic-based cohesion and coupling metrics and LOC. “*” indicates a p-value < 0.05.

			NetBeans 4.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	0.00	—	—	—	—
RTCs	0.02	-0.30*	—	—	—
CCBO	-0.01	0.21*	-0.09	—	—
CLCOM5	0.06*	0.00	0.11*	0.02	—
LOC	0.08*	-0.31*	0.22*	-0.09*	0.68*
			NetBeans 5.0		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.41*	—	—	—	—
RTCs	0.00	0.00	—	—	—
CCBO	-0.38*	0.14*	-0.01	—	—
CLCOM5	0.19*	0.00	0.00	-0.02	—
LOC	0.40*	-0.33*	-0.01	-0.06	0.68*
			NetBeans 5.5.1		
	NT	MWE	RTCs	CCBO	CLCOM5
NT	—	—	—	—	—
MWE	-0.47*	—	—	—	—
RTCs	-0.01	0.00	—	—	—
CCBO	-0.37*	0.17*	0.00	—	—
CLCOM5	0.21*	0.00	0.01	-0.03*	—
LOC	0.44*	-0.32*	0.01	-0.10*	0.67*

E. Results of the parameter sensitivity analysis

Table 24: Results of the parameter sensitivity analysis. The baseline parameters and their values are shown in the table. For each system, we show the D^2 when the parameter changes. The values in the parentheses indicate the increase/decrease from the baseline D^2 score. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

Baseline Values: K=500, II=10,000, δ=0.01							
NT				*NDT*			
Lower		Higher		Lower		Higher	
Mylyn 1.0 Baseline D^2: LOC+NT = 0.14				*Baseline PRE+CHURN+NDT = 0.24*			
K=10	0.10* (-0.04)	—	—	K=10	0.21 (-0.03)	—	—
100	0.14* (—)	—	—	100	0.21 (-0.03)	—	—
200	0.14* (—)	—	—	200	0.22* (-0.02)	—	—
300	0.15* (+0.01)	K=600	0.14* (—)	300	0.23* (-0.01)	K=600	0.24* (—)
400	0.14* (—)	700	0.13* (-0.01)	400	0.23* (-0.01)	700	0.22* (-0.02)
II=8K	0.14* (—)	II=11K	0.14* (—)	II=8K	0.24* (—)	II=11K	0.24* (—)
9K	0.14* (—)	12K	0.14* (—)	9K	0.24* (—)	12K	0.24* (—)
δ=0.0025	0.14* (—)	δ=0.02	0.14* (—)	δ=0.0025	0.24* (—)	δ=0.02	0.22* (-0.02)
0.005	0.14* (—)	0.04	0.13* (-0.01)	0.005	0.24* (—)	0.04	0.21* (-0.03)
Mylyn 2.0 Baseline D^2: LOC+NT = 0.19				*Baseline PRE+CHURN+NDT = 0.23*			
K=10	0.16* (-0.03)	—	—	K=10	0.23* (—)	—	—
100	0.17* (-0.02)	—	—	100	0.23* (—)	—	—
200	0.18* (-0.01)	—	—	200	0.23* (—)	—	—
300	0.18* (-0.01)	K=600	0.18* (-0.01)	300	0.23* (—)	K=600	0.23* (—)
400	0.19* (—)	700	0.20* (+0.01)	400	0.23* (—)	700	0.24* (+0.01)
II=8K	0.19* (—)	II=11K	0.19* (—)	II=8K	0.23* (—)	II=11K	0.23* (—)
9K	0.19* (—)	12K	0.19* (—)	9K	0.23* (—)	12K	0.23* (—)
δ=0.0025	0.19* (—)	δ=0.02	0.19* (—)	δ=0.0025	0.24* (+0.01)	δ=0.02	0.22* (-0.01)
0.005	0.19* (—)	0.04	0.19* (—)	0.005	0.24* (+0.01)	0.04	0.22 (-0.01)
Mylyn 3.0 Baseline D^2: LOC+NT = 0.20				*Baseline PRE+CHURN+NDT = 0.29*			
K=10	0.15* (-0.05)	—	—	K=10	0.29* (—)	—	—
100	0.20 (—)	—	—	100	0.29* (—)	—	—
200	0.20 (—)	—	—	200	0.29* (—)	—	—
300	0.20* (—)	K=600	0.20* (—)	300	0.29* (—)	K=600	0.29* (—)
400	0.21* (+0.01)	700	0.21* (+0.01)	400	0.29* (—)	700	0.29* (—)
II=8K	0.20* (—)	II=11K	0.20* (—)	II=8K	0.29 (—)	II=11K	0.29* (—)
9K	0.20* (—)	12K	0.20* (—)	9K	0.29 (—)	12K	0.29* (—)
δ=0.0025	0.19* (-0.01)	δ=0.02	0.20* (—)	δ=0.0025	0.29* (—)	δ=0.02	0.29* (—)
0.005	0.20* (—)	0.04	0.18* (-0.02)	0.005	0.29* (—)	0.04	0.28 (-0.01)
Continued in Table 25.							

Table 25: *Continued from Table 24.* Results of the parameter sensitivity analysis. The baseline parameters and their values are shown in the table. For each system, we show the D^2 when the parameter changes. The values in the parentheses indicate the increase/decrease from the baseline D^2 score. “*” indicates the metric is statistically significant (i.e., p-value < 0.05).

Baseline Values: K=500, II=10,000, δ=0.01											
		NT						*NDT*			
Lower			Higher			Lower			Higher		
Firefox 1.0 Baseline D^2: LOC+NT = 0.16						*Baseline PRE+CHURN+NDT = 0.18*					
K=	10	0.13* (-0.03)		—	—	*K*=	10	0.17* (-0.01)		—	—
	100	0.15* (-0.01)		—	—		100	0.17* (-0.01)		—	—
	200	0.16* (—)		—	—		200	0.19* (+0.01)		—	—
	300	0.16* (—)	*K*=	600	0.16* (—)		300	0.19* (+0.01)	*K*=	600	0.18* (—)
	400	0.16* (—)		700	0.17* (+0.01)		400	0.18* (—)		700	0.19* (+0.01)
II=	8K	0.16* (—)	*II*=	11K	0.16* (—)	*II*=	8K	0.18* (—)	*II*=	11K	0.18* (—)
	9K	0.16* (—)		12K	0.16* (—)		9K	0.18* (—)		12K	0.18* (—)
δ=	0.0025	0.16* (—)	*δ*=	0.02	0.15* (-0.01)	*δ*=	0.0025	0.18* (—)	*δ*=	0.02	0.17* (-0.01)
	0.005	0.16* (—)		0.04	0.15* (-0.01)		0.005	0.18* (—)		0.04	0.16* (-0.02)
Firefox 1.5 Baseline D^2: LOC+NT = 0.21						*Baseline PRE+CHURN+NDT = 0.25*					
K=	10	0.16* (-0.05)		—	—	*K*=	10	0.24* (-0.01)		—	—
	100	0.20* (-0.01)		—	—		100	0.27* (+0.02)		—	—
	200	0.18* (-0.03)		—	—		200	0.24* (-0.01)		—	—
	300	0.21* (—)	*K*=	600	0.21* (—)		300	0.25* (—)	*K*=	600	0.25* (—)
	400	0.21* (—)		700	0.22* (+0.01)		400	0.25* (—)		700	0.25* (—)
II=	8K	0.21* (—)	*II*=	11K	0.20* (-0.01)	*II*=	8K	0.25* (—)	*II*=	11K	0.25* (—)
	9K	0.20* (-0.01)		12K	0.21* (—)		9K	0.25* (—)		12K	0.25* (—)
δ=	0.0025	0.21* (—)	*δ*=	0.02	0.20* (-0.01)	*δ*=	0.0025	0.26* (+0.01)	*δ*=	0.02	0.24* (-0.01)
	0.005	0.21* (—)		0.04	0.19* (-0.02)		0.005	0.26* (+0.01)		0.04	0.23* (-0.02)
Firefox 2.0 Baseline D^2: LOC+NT = 0.17						*Baseline PRE+CHURN+NDT = 0.25*					
K=	10	0.15* (-0.02)		—	—	*K*=	10	0.25* (—)		—	—
	100	0.18* (+0.01)		—	—		100	0.26* (+0.01)		—	—
	200	0.17* (—)		—	—		200	0.26* (+0.01)		—	—
	300	0.17* (—)	*K*=	600	0.17* (—)		300	0.25* (—)	*K*=	600	0.24* (-0.01)
	400	0.17* (—)		700	0.18* (+0.01)		400	0.25* (—)		700	0.25* (—)
II=	8K	0.17* (—)	*II*=	11K	0.17* (—)	*II*=	8K	0.25* (—)	*II*=	11K	0.25* (—)
	9K	0.17* (—)		12K	0.17* (—)		9K	0.25* (—)		12K	0.25* (—)
δ=	0.0025	0.17* (—)	*δ*=	0.02	0.17* (—)	*δ*=	0.0025	0.25* (—)	*δ*=	0.02	0.24* (-0.01)
	0.005	0.17* (—)		0.04	0.16* (-0.01)		0.005	0.25* (—)		0.04	0.24* (-0.01)
Continued in Table 26.											

Table 26: *Continued from Table 25.* Results of the parameter sensitivity analysis. The baseline parameters and their values are shown in the table. For each system, we show the D^2 when the parameter changes. The values in the parentheses indicate the increase/decrease from the baseline D^2 score. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

Baseline Values: K=500, II=10,000, δ=0.01											
NT						*NDT*					
Lower			Higher			Lower			Higher		
Eclipse 2.0 Baseline D^2: LOC+NT = 0.18						*PRE+CHURN+NDT = 0.20*					
K=	10	0.18* (—)		—	—	K=	10	0.17 (-0.03)		—	—
	100	0.18* (—)		—	—		100	0.20* (—)		—	—
	200	0.18* (—)		—	—		200	0.19* (-0.01)		—	—
	300	0.18* (—)	K=	600	0.18* (—)		300	0.18 (-0.02)	K=	600	0.18* (-0.02)
	400	0.18 (—)		700	0.18* (—)		400	0.18* (-0.02)		700	0.18* (-0.02)
II=	8K	0.18* (—)	II=	11K	0.18* (—)	II=	8K	0.20* (—)	II=	11K	0.20* (—)
	9K	0.18* (—)		12K	0.18* (—)		9K	0.20* (—)		12K	0.20* (—)
δ=	0.0025	0.18* (—)	δ=	0.02	0.18* (—)	δ=	0.0025	0.18* (-0.02)	δ=	0.02	0.21* (+0.01)
	0.005	0.18* (—)		0.04	0.18* (—)		0.005	0.19* (-0.01)		0.04	0.21* (+0.01)
Eclipse 2.1 Baseline D^2: LOC+NT = 0.11						*PRE+CHURN+NDT = 0.15*					
K=	10	0.12* (+0.01)		—	—	K=	10	0.15 (—)		—	—
	100	0.11* (—)		—	—		100	0.15 (—)		—	—
	200	0.12* (+0.01)		—	—		200	0.15* (—)		—	—
	300	0.11 (—)	K=	600	0.11* (—)		300	0.15* (—)	K=	600	0.15 (—)
	400	0.11* (—)		700	0.11 (—)		400	0.15 (—)		700	0.15* (—)
II=	8K	0.11* (—)	II=	11K	0.11* (—)	II=	8K	0.15 (—)	II=	11K	0.15 (—)
	9K	0.11* (—)		12K	0.11* (—)		9K	0.15 (—)		12K	0.15 (—)
δ=	0.0025	0.11* (—)	δ=	0.02	0.11* (—)	δ=	0.0025	0.15* (—)	δ=	0.02	0.15 (—)
	0.005	0.11* (—)		0.04	0.11* (—)		0.005	0.15* (—)		0.04	0.15 (—)
Eclipse 3.0 Baseline D^2: LOC+NT = 0.14						PRE+CHURN+NDT = 0.17					
K=	10	0.14 (—)		—	—	K=	10	0.18* (+0.01)		—	—
	100	0.14 (—)		—	—		100	0.18* (+0.01)		—	—
	200	0.14 (—)		—	—		200	0.17* (—)		—	—
	300	0.14* (—)	K=	600	0.14* (—)		300	0.18* (+0.01)	K=	600	0.17 (—)
	400	0.14 (—)		700	0.14* (—)		400	0.17* (—)		700	0.18* (+0.01)
II=	8K	0.14* (—)	II=	11K	0.14 (—)	II=	8K	0.17 (—)	II=	11K	0.17 (—)
	9K	0.14* (—)		12K	0.14* (—)		9K	0.17 (—)		12K	0.17 (—)
δ=	0.0025	0.14* (—)	δ=	0.02	0.14 (—)	δ=	0.0025	0.17 (—)	δ=	0.02	0.18* (+0.01)
	0.005	0.14* (—)		0.04	0.14 (—)		0.005	0.17 (—)		0.04	0.18* (+0.01)
Continued in Table 27.											

Table 27: *Continued from Table 26.* Results of the parameter sensitivity analysis. The baseline parameters and their values are shown in the table. For each system, we show the D^2 when the parameter changes. The values in the parentheses indicate the increase/decrease from the baseline D^2 score. "*" indicates the metric is statistically significant (i.e., p-value < 0.05).

Baseline Values: K=500, II=10,000, δ=0.01											
NT						*NDT*					
Lower			Higher			Lower			Higher		
NetBeans 4.0 Baseline D^2: LOC+NT = 0.10						*PRE+CHURN+NDT = 0.31*					
K=	10	0.07 (-0.03)		—	—	*K*=	10	0.29* (-0.02)		—	—
	100	0.09* (-0.01)		—	—		100	0.30* (-0.01)		—	—
	200	0.09* (-0.01)		—	—		200	0.30* (-0.01)		—	—
	300	0.9* (-0.01)	*K*=	600	0.10* (—)		300	0.30* (-0.01)	*K*=	600	0.30* (-0.01)
	400	0.10* (—)		700	0.10* (—)		400	0.31* (—)		700	0.30* (-0.01)
II=	8K	0.10* (—)	*II*=	11K	0.10* (—)	*II*=	8K	0.31* (—)	*II*=	11K	0.30* (-0.01)
	9K	0.10* (—)		12K	0.10* (—)		9K	0.30* (-0.01)		12K	0.31* (—)
δ=	0.0025	0.11* (+0.01)	*δ*=	0.02	0.09* (-0.01)	*δ*=	0.0025	0.30* (-0.01)	*δ*=	0.02	0.30* (-0.01)
	0.005	0.10* (—)		0.04	0.08* (-0.02)		0.005	0.30* (-0.01)		0.04	0.30 (-0.01)
NetBeans 5.0 Baseline D^2: LOC+NT = 0.10						Baseline PRE+CHURN+NDT = 0.17					
K=	10	0.07 (-0.03)		—	—	*K*=	10	0.15* (-0.02)		—	—
	100	0.10* (—)		—	—		100	0.17* (—)		—	—
	200	0.10* (—)		—	—		200	0.18* (+0.01)		—	—
	300	0.09* (-0.01)	*K*=	600	0.10* (—)		300	0.17* (—)	*K*=	600	0.17* (—)
	400	0.11* (+0.01)		700	0.10* (—)		400	0.18* (+0.01)		700	0.18* (+0.01)
II=	8K	0.10* (—)	*II*=	11K	0.10* (—)	*II*=	8K	0.17* (—)	*II*=	11K	0.17* (—)
	9K	0.10* (—)		12K	0.10* (—)		9K	0.17* (—)		12K	0.17* (—)
δ=	0.0025	0.11* (+0.01)	*δ*=	0.02	0.10* (—)	*δ*=	0.0025	0.18* (+0.01)	*δ*=	0.02	0.17* (—)
	0.005	0.11* (+0.01)		0.04	0.8* (-0.02)		0.005	0.18* (+0.01)		0.04	0.15* (-0.02)
NetBeans 5.5.1 Baseline D^2: LOC+NT = 0.11						*Baseline PRE+CHURN+NDT = 0.17*					
K=	10	0.08* (-0.03)		—	—	*K*=	10	0.15* (-0.02)		—	—
	100	0.10* (-0.01)		—	—		100	0.17* (—)		—	—
	200	0.10* (-0.01)		—	—		200	0.17* (—)		—	—
	300	0.10* (-0.01)	*K*=	600	0.11* (—)		300	0.17* (—)	*K*=	600	0.18* (+0.01)
	400	0.10* (-0.01)		700	0.11* (—)		400	0.17* (—)		700	0.18* (+0.01)
II=	8K	0.11* (—)	*II*=	11K	0.11* (—)	*II*=	8K	0.17* (—)	*II*=	11K	0.18* (+0.01)
	9K	0.11* (—)		12K	0.11* (—)		9K	0.18* (+0.01)		12K	0.17* (—)
δ=	0.0025	0.12* (+0.01)	*δ*=	0.02	0.10* (-0.01)	*δ*=	0.0025	0.19* (+0.02)	*δ*=	0.02	0.16* (-0.01)
	0.005	0.11* (—)		0.04	0.8* (-0.03)		0.005	0.18* (+0.01)		0.04	0.14* (-0.03)