The Relationship between Commit Message Detail and Defect Proneness in Java Projects on GitHub

Jacob G. Barnett, Charles K. Gathuru, Luke S. Soldano, and Shane McIntosh
McGill University, Montréal, Canada
{jacob.barnett,charles.gathuru,luke.soldano}@mail.mcgill.ca,
shane.mcintosh@mcgill.ca

ABSTRACT

Just-In-Time (JIT) defect prediction models aim to predict the commits that will introduce defects in the future. Traditionally, JIT defect prediction models are trained using metrics that are primarily derived from aspects of the code change itself (e.g., the size of the change, the author's prior experience). In addition to the code that is submitted during a commit, authors write commit messages, which describe the commit for archival purposes. It is our position that the level of detail in these commit messages can provide additional explanatory power to JIT defect prediction models. Hence, in this paper, we analyze the relationship between the defect proneness of commits and commit message volume (i.e., the length of the commit message) and commit message content (approximated using spam filtering technology). Through analysis of JIT models that were trained using 342 GitHub repositories, we find that our JIT models outperform random guessing models, achieving AUC and Brier scores that range between 0.63-0.96 and 0.01-0.21, respectively. Furthermore, our metrics that are derived from commit message detail provide a statistically significant boost to the explanatory power to the JIT models in 43%-80% of the studied systems, accounting for up to 72% of the explanatory power. Future JIT studies should consider adding commit message detail metrics.

1. INTRODUCTION

Just-In-Time (JIT) defect prediction models (i.e., models that predict the commits that will introduce future defects) provide a more practical alternative to traditional defect prediction models (i.e., models that predict which modules like files or components will be defective in the future). JIT model predictions are advantageous because the predictions are constrained to a small region of the code (i.e., a commit). Moreover, the predictions can be easily assigned, since each commit has an author who can perform the inspection while design decisions are still fresh in their mind. JIT defect prediction has been successfully adopted by industrial software teams at Avaya [10], BlackBerry [13], and Cisco [15].

JIT defect prediction models have largely focused on metrics that are derived from aspects of the code change (e.g., # modified files). For example, Mockus and Weiss [10] predict defect-inducing changes in a large-scale telecommunication system using change measures. Kamei *et al.* [7] find that the addition of a variety of metrics that are extracted from commits helps to effectively predict defect-inducing changes. Moreover, Fukushima *et al.* [5, 6] show that JIT models can be applied in cross-project contexts.

While these JIT models focus on metrics that are derived from code elements, the commit messages that accompany the code changes have been largely overlooked. Work by Kim *et al.* [8] shows that metrics that indicate the presence or absence of terms in messages can be used in JIT models.

We suspect that commit messages that lack detail accompany commits that were themselves rushed. Indeed, rushed commits are likely to introduce defects.

In this paper, we use JIT models to study the relationship between commit message detail and defect proneness. We propose two new JIT metrics that estimate *commit message volume* (i.e., the length of the commit message) and *commit message content* (i.e., the amount of information in the commit message). Through analysis of the MSR challenge dataset [3], we address the following research questions:

(RQ1) Is there a relationship between commit message volume and defect proneness?
Our commit message volume metric, which measures the length of a commit message, contributes a statistically significant amount of explanatory power to the JIT models of 147 (43%) of the studied systems. Even when controlling for baseline code metrics, commit message volume contributes up to 25% of the explanatory power of our JIT models.

(RQ2) Is there a relationship between commit message content and defect proneness?
Our commit message content metric, which leverages spam filtering technology, contributes a statistically significant amount of explanatory power to the JIT models of 240 (80%) of the studied systems. Even when controlling for baseline code and commit message volume metrics, commit message content contributes up to 72% of the explanatory power of our JIT models.

Paper organization. The rest of the paper is organized as follows. Section 2 describes the case study design, while Section 3 presents the results. Section 4 draws conclusions.

MSR'16, May 14-15, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4186-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2901739.2903496

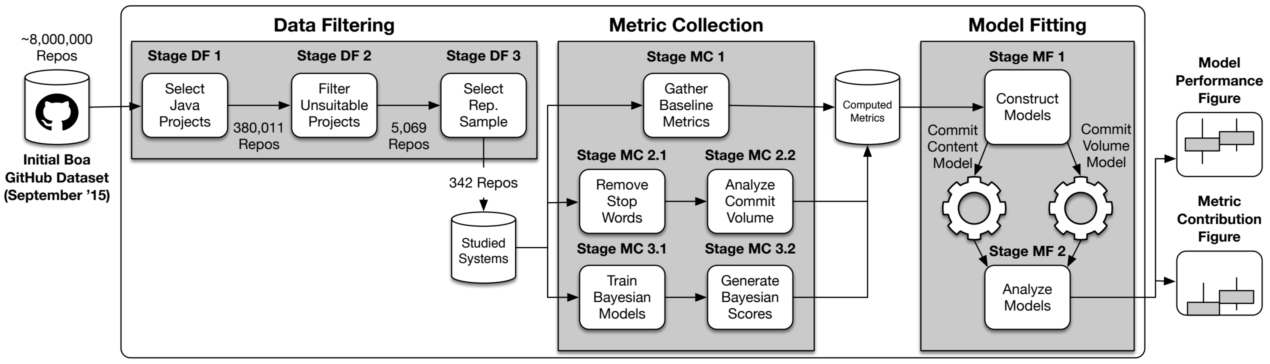

Figure 1: Our approach to study the relationship between commit message detail and defect proneness.

2. CASE STUDY DESIGN

The following section describes our 3-step approach to process the MSR challenge dataset in order to analyze the relationship between commit message detail and defect proneness. Figure 1 provides an overview of our approach. Below, we describe each step in our approach.

2.1 Data Filtering

The MSR challenge dataset contains 7,830,023 source code repositories, however, many of these repositories are not suitable for our analysis. Hence, we first filter the dataset using the three data filtering stages outlined in Figure 1.

DF 1: Select Java Projects. For this study, we focus on Java projects. Thus, our first filter removes repositories that do not contain the `Java` tag in the MSR challenge dataset. After applying this filter, 380,011 repositories survive.

DF 2: Filter Unsuitable Projects. Software forges like GitHub often contain projects that have not yet reached maturity [9]. To prevent immature projects from biasing our results, we apply two filters to our dataset. The first filter removes repositories with too few contributors to be considered a collaborative software project, while the second filter removes repositories with too few commits to be considered mature. In order to determine suitable cutoff values for the numbers of contributors and commits, we plot potential threshold values against the number of surviving repositories. Due to space constraints, we provide access to these figures online.[1] Based on these figures, we select thresholds of 4 contributors and 700 commits, which reduces our dataset to 5,069 repositories.

DF 3: Select Representative Sample. Due to computational constraints, it was necessary to select a smaller sample for analysis. To ensure that the sample that we select produces estimates that are within 5% bounds of the actual estimates, with a 95% confidence level, we use the sample size calculation of $s = \frac{z^2 p(1-p)}{0.05^2}$, where p is the proportion we want to estimate and $z = 1.96$. Since we did not know the proportion in advance, we use $p = 0.5$. After correcting for our population, we obtain a sample size of 342 repositories.

2.2 Metric Collection

After filtering the raw dataset, we collect metrics from the surviving repositories. The set of metrics includes *baseline metrics* (stage MC 1) and our *commit message detail metrics* (stages MC 2.x and MC 3.x). We provide a detailed description of each stage below.

MC 1: Gather Baseline Metrics. Several metrics are known to share a relationship with defect proneness [5–8, 10]. To control for the impact of several baseline metrics in our analysis, we compute the family of JIT metrics provided by *CommitGuru* [12], a JIT metric computation tool. These baseline metrics estimate the size, dispersion, and purpose of the commit, as well as the historical tendencies of the code being modified, and the experience of the author of the commit. CommitGuru also identifies the defect-introducing commits, which our models attempt to explain.

MC 2.1: Remove Stop Words. To yield more meaningful volume values, we first remove common stop words from each commit message. To do so, we use the stop word removal feature of the Natural Language Toolkit library [1].

MC 2.2: Analyze Commit Volume. To study the relationship between commit message volume and defect proneness, we count the words in each commit message, and use it as an additional explanatory variable in our JIT models.

MC 3.1: Train Bayesian Models. To estimate the content of a commit message, similar to Shihab *et al.* [14], we train SpamBayes classifiers [11], i.e., Naïve Bayes classifiers that are often used in email spam filtering. Our rationale is that defective and non-defective commit messages may have contrasting content, which we can leverage.

When training our SpamBayes classifiers, we want to ensure that they are not overly specific to the systems that they will be applied to. To do so, we use an approach inspired by leave-one-out cross-validation. We train the SpamBayes classifier for a given repository using the commit message data of the 341 other repositories. This process is repeated 342 times, yielding a new SpamBayes classifier for each repository in our dataset.

MC 3.2: Generate Bayesian Scores. The SpamBayes classifiers produced by stage MC 3.1 are used on each commit message in each repository to produce a *spam probability score*, i.e., the likelihood that the commit message is spam. We add these spam probability scores as an explanatory variable in our JIT models to address RQ2.

2.3 Model Fitting

After computing our baseline and commit message detail metrics, we are ready to fit our models. Our model fitting

[1]http://is.gd/lM4B4i

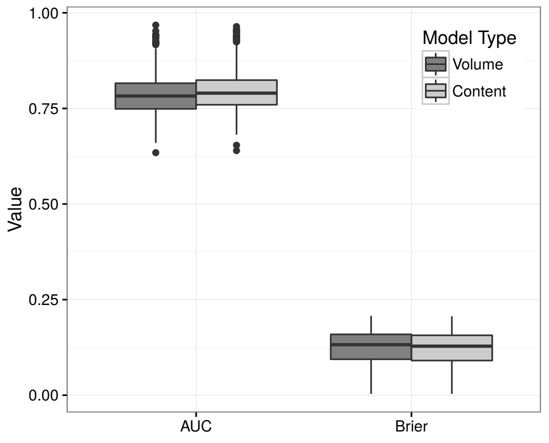

Figure 2: Performance of each model by type.

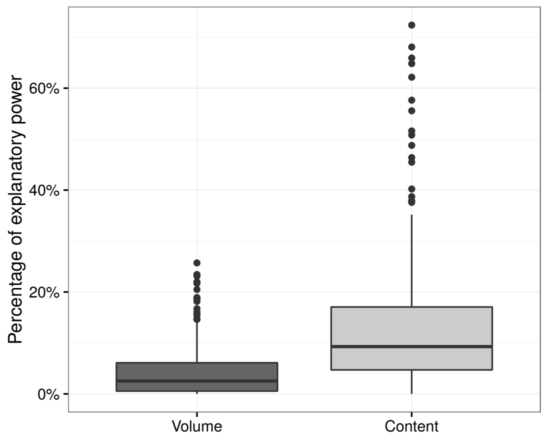

Figure 3: Explanatory power of our metrics that estimate commit message detail.

process is further divided into model construction and model analysis stages. We describe each stage below.

MF 1: Construct Models. Redundant variables in our JIT models will interfere with each other, distorting the modeled relationship between the explanatory variables and defect proneness. We mitigate the effect of redundant variables prior to constructing our models. Similar to prior work [9], we do so using an automated variable redundancy analysis provided by the `redun` function in the `rms R` package. The `redun` function builds preliminary models that explain each explanatory variable using the other explanatory variables. Explanatory variables for which these preliminary models perform well ($R^2 \geq 0.9$) are not included when we fit our JIT models. After removing redundant variables, we fit our JIT models with the surviving metrics using the logistic regression modeling technique.

MF 2: Analyze Models. After fitting our models, we analyze the accuracy of the fit and the contribution of our commit message detail metrics. Since the performance estimates of our models are derived from the same data that they were trained with, the estimates will be inflated if the model is overfit. We take overfitting into account by subtracting the *bootstrap-calculated optimism* [4] from initial performance estimates. The optimism is calculated as follows:

1. From the original dataset with n commits, select a new (bootstrap) sample of n commits with replacement.
2. In the bootstrap sample, refit the model using the same model formula as was used in the original fit.
3. Apply the model that was trained using the bootstrap sample to both the bootstrap and original datasets and calculate the performance estimate on each.
4. The optimism is the difference between the performance estimate in the bootstrap and original samples.

In our study, we repeat the above optimism calculation 1,000 times and subtract the average optimism from the performance estimates of the original model fits.

We also estimate the contribution of our commit message metrics using a "drop one" approach [2]. This approach measures the impact of an explanatory variable on a JIT model by measuring the difference in the performance of models built using: (1) all explanatory variables (the full model), and (2) all explanatory variables except for the variable under test (the dropped model). A χ^2 maximum likelihood test is applied to the resulting models to detect whether the variable under test improves model performance to a statistically significant degree ($\alpha = 0.05$). Since we have 342 systems in total, we apply Bonferroni correction to control for family-wise errors, which adjusts $\alpha = \frac{0.05}{342} = 1.46 \times 10^{-4}$.

3. CASE STUDY RESULTS

In this section, we discuss the results of our case study. We first discuss the performance of both our commit message volume (RQ1) and content (RQ2) models, and then we present the results of our research questions. For each RQ, we first describe our approach and then discuss the results.

3.1 JIT Model Performance

Our models outperform naïve baseline classifiers. Figure 2 shows the optimism-adjusted AUC (larger values are better) and Brier score values (smaller values are better) of our volume and content models. Our models achieve AUCs that range between 0.65-0.96 and Brier scores that range between 0.01-0.21. Since a random classifier would achieve an AUC of 0.5 and a Brier score of 0.25, this indicates that our models outperform random guessing models by at least 15 and 4 percentage points in terms of AUC and Brier score, respectively.

RQ1: Is there a relationship between commit message volume and defect proneness?

Approach. We add our commit volume metric to the set of baseline metrics, and analyze JIT models that we fit for each studied system using the steps outline in Section 2.

Results. **The commit message volume metric contributes a significant amount of explanatory power to the JIT models of 43% of the studied systems**. The drop one χ^2 tests indicate that our volume metric achieves a p-value below the Bonferroni-corrected α value (i.e., 1.46×10^{-4}) in 147 of the studied systems. Therefore, we can reject the null hypothesis that the volume metric does not significantly contribute to the fit of our JIT models in 43% of the studied systems.

The volume metric contributes up to 25% of explanatory power in our JIT models. Figure 3 shows the distribution of the percentage of the explanatory power that, of the studied explanatory metrics, is only attributable to the volume metric (according to our drop one tests). The values range between 0%-35%, with a median of 4%.

Through analysis of the top five repositories where the volume metric contributes the largest amount of explanatory power, we find that commit volume has a large interquartile range, with the highest being 10. On the other hand, analysis of the bottom five repositories where the volume metric contributes the smallest amount of explanatory power, we find that commit volume has a small interquartile range, maxing out at 6.

The explanatory power of 43% of the studied systems are improved to a statistically significant degree by adding our commit message volume metric. Furthermore, up to 25% of the explanatory power of our JIT models is only attributable to the commit message volume metric.

RQ2: Is there a relationship between commit message content and defect proneness?

Approach. Following the same approach as RQ1, we fit JIT models for each studied system using the baseline metrics and our commit message content metric. Furthermore, we also include the commit message volume metric of RQ1 in the set of baseline metrics for RQ2 to control for the potentially confounding impact of commit message volume on our content models.

Results. **The spam probability score contributes a significant amount of explanatory power to the JIT models of 80% of the studied systems.** The drop one χ^2 tests indicate that our spam probability score achieves a p-value below the Bonferroni-corrected α value (i.e., 1.46×10^{-4}) in 272 of the studied systems. Therefore, we can reject the null hypothesis that the spam probability score does not significantly contribute to the fit of our JIT models in 43% of the studied systems.

The Bayesian content score model contributes up to 72% of the explanatory power of our models. Figure 3 shows the distribution of the percentage of the explanatory power that, of the studied explanatory metrics, is only attributable to the spam probability score (according to our drop one tests). The values range between 0%-72%, with a median of 10%.

Through analysis of the top five repositories where the spam probability score contributes the largest amount of explanatory power, we find that commit volume has a large median value of 24 words. On the other hand, analysis of the bottom five repositories where the spam probability score contributes the smallest amount of explanatory power, we find that commit volume has a small median value of 6 words. Since more data is being provided to the SpamBayes classifiers, the spam probability score appears to be more useful in the systems with longer commit messages.

The explanatory power of 80% of the studied systems are improved to a statistically significant degree by adding our commit message content metric. Furthermore, up to 72% of the explanatory power of our JIT models is only attributable to the commit message content metric.

4. CONCLUSIONS

JIT models, which aim to predict the commits that will introduce future defects, are typically trained using code-based metrics. In this paper, we add metrics that estimate the level of detail in commit messages to JIT models with code-based metrics. We find that 43% and 80% of the JIT models of the studied systems are significantly improved by adding metrics that measure commit message volume and content, respectively. Future JIT studies should consider adding metrics that measure the commit message detail.

5. REFERENCES

[1] S. Bird, E. Loper, and E. Klein. *Natural Language Processing with Python.* O'Reilly Media Inc, 2009.

[2] J. M. Chambers and T. J. Hastie, editors. *Statistical Models in S.* Wadsworth and Brooks/Cole, 1992.

[3] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. In *Proc. of the 35th Int'l Conf. on Software Engineering (ICSE)*, pages 422–431, 2013.

[4] B. Efron. How Biased is the Apparent Error Rate of a Prediction Rule? *Journal of the American Statistical Association*, 81(394):461–470, 1986.

[5] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi. An Empirical Study of Just-in-Time Defect Prediction using Cross-Project Models. In *Proc. of the 11th Working Conf. on Mining Software Repositories (MSR)*, pages 172–181, 2014.

[6] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan. Studying Just-In-Time Defect Prediction using Cross-Project Models. *Empirical Software Engineering*, To appear, 2016.

[7] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *Transactions on Software Engineering (TSE)*, 39(6):757–773, 2013.

[8] S. Kim, E. J. Whitehead, Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *Transactions on Software Engineering (TSE)*, 34(2):181–196, 2008.

[9] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering*, In press, 2015.

[10] A. Mockus and D. M. Weiss. Predicting Risk of Software Changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[11] T. Myer and B. Whately. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Proc. of the 7th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, 2004.

[12] C. Rosen, B. Grawi, and E. Shihab. Commit guru: Analytics and risk prediction of software commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 966–969, New York, NY, USA, 2015. ACM.

[13] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Zhang. An Industrial Study on the Risk of Software Change. In *Proc. of the 20th Symposium on the Foundations of Software Engineering (FSE)*, pages 62:1–62:11, 2012.

[14] E. Shihab, A. Ihara, Y. , Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042, 2012.

[15] M. Tan, L. Tan, S. Dara, and C. Mayeux. Online Defect Prediction for Imbalanced Data. In *Proc. of the 37th Int'l Conf. on Software Engineering (ICSE)*, volume 2, pages 99–108, 2015.