Are Smell-Based Metrics Actually Useful in Effort-Aware Structural Change-Proneness Prediction?
An Empirical Study

Conference or Workshop Item

How to cite:


For guidance on citations see FAQs.

© 2018 IEEE
Version: Not Set
Are Smell-Based Metrics Actually Useful in Effort-Aware Structural Change-Proneness Prediction? An Empirical Study

Huihui Liu*, Yijun Yu†, Bixin Li*, Yibiao Yang‡, and Ru Jia§

*School of Computer Science and Engineering, Southeast University, Nanjing, China
†Centre for Research in Computing, The Open University, UK
‡State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
§School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia

Abstract—Bad code smells (also named as code smells) are symptoms of poor design choices in implementation. Existing studies empirically confirmed that the presence of code smells increases the likelihood of subsequent changes (i.e., change-proneness). However, to the best of our knowledge, no prior studies have leveraged smell-based metrics to predict particular change type (i.e., structural changes). Moreover, when evaluating the effectiveness of smell-based metrics in structural change-proneness prediction, none of existing studies take into account of the effort inspecting those change-prone source code. In this paper, we consider five smell-based metrics for effort-aware structural change-proneness prediction and compare these metrics with a baseline of well-known CK metrics in predicting particular categories of change types. Specifically, we first employ univariate logistic regression to analyze the correlation between each smell-based metric and structural change-proneness. Then, we build multivariate prediction models to examine the effectiveness of smell-based metrics in effort-aware structural change-proneness prediction when used alone and used together with the baseline metrics, respectively. Our experiments are conducted on six Java open-source projects with up to 60 versions and results indicate that: (1) all smell-based metrics are significantly related to structural change-proneness, except metric ANS in hive and SCM in camel after removing confounding effect of file size; (2) in most cases, smell-based metrics outperform the baseline metrics in predicting structural change-proneness; and (3) when used together with the baseline metrics, the smell-based metrics are more effective to predict change-prone files with being aware of inspection effort.

I. INTRODUCTION

Flower et al. [1] coined the term “code smells” and defined the concept as symptoms of bad design and implementation choices in source code. Since then, many empirical studies have investigated the impact of code smells on software quality properties, such as program understandability [2], change-/fault-proneness [3]–[6] and code maintainability [7]–[9]. These studies indicated that, when appeared in source code alone or together, code smells could significantly increase change-proneness.

Furthermore, the empirical results in the study of Romano et al. [10] have shown that some smells could even result in certain types of structural code changes (instead of changing arbitrary lines in the code), e.g., API changes (a category of change types, see Table II) are more likely to occur in the classes affected by Swiss Army Knife smell.

To the best of our knowledge, these results only indicated qualitatively that certain code smells could lead to textual changes or a particular type of structural changes. Software practitioners were not readily instructed on how to predict these change-prone modules (such as files or classes) based on the code smells, although it is obviously beneficial to do so. For example, if developers were aware of likely API changes due to code smells they could plan ahead of time to allocate resources for system-wide integration tests on the dependent modules. On the contrary, if only minor changes to statements inside a method’s body were predicted, unit testing would be sufficient because no further change impact can be expected. Differentiating the effort of such predicted would-be changes would add to the effective planning to be more prepared.

To achieve these goals, in this work, we consider mining the structural change history of files to build an effort-aware structural change-proneness prediction model based on five code smell metrics. Specifically, we employ univariate logistic regression approach to analyze the correlations between each smell-based metric and structural change-proneness (without discriminating particular change types); then we build multivariate prediction models to examine the effectiveness of smell-based metrics, when used alone and together with the baseline metrics (i.e. CK metrics), respectively, in predicting effort-aware structural change-proneness.

The experiments are conducted on six typical Java open-source projects with up to 60 versions. Our results show that: (1) smell-based metrics are significantly correlated to structural change-proneness; (2) in most cases, smell-based metrics outperform the baseline metrics in predicting structural change-proneness; and (3) when combined with the baseline metrics, smell-based metrics are more effective in predicting change-prone files with large LOC to inspect.

Our study provides valuable insight for researchers and practitioners to better understand the usefulness of five smell-based metrics in predicting structural change-prone files under the scenarios of ranking and classification, being aware of the
effort to inspect LOC in the files-to-change.

The rest of the paper is organized as follows. Section II discusses the related work. Section III introduces the study design including research questions, subject selection, data collection and prediction model construction, as well as model evaluation. Section IV reports the experimental results, answers our research questions. Section V examines the threats to validity of our study. Section VI concludes.

II. RELATED WORK

In this section, we introduce three aspects that mostly related to our research topic which consist of smell detection techniques, smell’s impact on quality and change prediction models.

A. Smells Detection

Code smells are symptoms of poor design and implementation choice. Fowler et al. [1] gave a brief description to 22 code smells and presented some refactoring techniques for their removal. After that, many approaches have been proposed for automatic smell detection, e.g., metrics-based techniques, visualization-based techniques and search-based techniques, as stated in recent literature reviews [11]–[14]. These literatures reveal that a notable portion (around 37%) of detection approaches are metrics-based, although this approach has its limitation. Based on aforementioned approaches, many smell detection tools (up to 84) have been developed [11], [12]. These tools include open-source tools, e.g., iPlasma [15], JDeodrant [16], CBSDetector [17] and AJCSD, and closed-source tools, e.g., Borland Together and InCode. In contrast to open-source tools, closed-source tools have the limitation that it is usually hard to know what smell detection strategies are employed behind them. However, none of these tools can detect all 22 code smells defined by Fowler [1].

B. Impact of Smells on Quality

Due to our topic focusing on change-proneness prediction, we only introduce studies of how smells affect code change-proneness. Khomh et al [5] investigated more than 12 smells on several projects and their results showed that the classes affected by code smells have higher change-proneness than other smell-free classes. Particularly, recent study by Palombaro et al. [6] further confirmed this result. Moreover, they also highlighted that the change-proneness of smelly classes would significantly decrease after removing corresponding smells in these classes. Olbrich et al [3], [4] conducted similar experiments focusing on God Class, Shotgun Surgery and Brain Class, and concluded that these smells did have higher change-proneness than smell-free components. However, this conclusion did not hold when the change size (sum of lines modified, added and deleted) was normalized by source lines of code. Besides, several researchers paid more attention on the effect of combined smells on the quality. Abbes et al. [18] conducted three experiments on 24 subjects and investigated whether the occurrence of antipatterns (alias to code smells) affected code understandability during the period of code comprehension and maintenance tasks. They concluded that although the occurrence of one antipattern did not significantly decrease developers’ performance, a combination of two antipatterns significantly hindered developers’ performance during code comprehension and maintenance tasks. Furthermore, Yamashita et al. [7], [9] reported that “coupled-smells” across different files could result in comparable negative maintenance issues as “collocated-smells” in the same file. In particular, some studies also investigated the relationship between code smells and structural change-proneness, rather than textual change-proneness [5], [10], [19]. Their empirical results indicated that code smells also had significant correlation with structural changes and even some smells could be more structurally change-prone as compared with other smells for a particular change category (see Table II).

C. Change Prediction Models

Tollin et al. [20] used coding rules violations to predict code changes through a set of machine learning models. Their results indicated that these classification models achieved satisfactory performance, especially when predicting changes in the next version. Lu et al. [21] used statistical meta-analysis to investigate the predictive capability of 62 OO metrics by performing empirical validation on 102 Java systems. Their study indicated that size metrics showed moderate predictive capability, inheritance-based metrics had the least predictive capability for change-prone class prediction. The predictive ability of coupling and cohesion metrics was lower than size metrics for change prediction tasks. In contrast to textual change-proneness prediction, Romano and Pinzger [22] used code metrics to predict change-prone Java interfaces. Soon after that, Giger et al. [23] employed code metrics (i.e., CK metrics) and network measures built on code dependency graph to predict particular type of changes (similar to change categories in Table II). Their results showed that the model built on CK metrics and network measures in combination achieved good prediction performance in terms of AUC, precision and recall indicators.

III. STUDY DESIGN

The goal of this experimental study is to construct a change-proneness prediction model using smell-based metrics and to evaluate the contribution of these metrics with respect to effort estimations such as LOC of changed files. In the following subsections, we propose three research questions and criteria to choose subject projects, illustrate the procedure of data collection, construct the effort-aware change-proneness prediction models using logistic regression and evaluate their performance under the scenarios of ranking and classification.
To investigate the effectiveness of smell-based metrics in effort-aware change-proneness prediction, we set up three research questions as follows:

**RQ1:** How are smell-based metrics correlated with structural change-proneness?

**RQ2:** Are smell-based metrics more or less effective than the commonly used code metrics in predicting change-prone files regarding each change category?

**RQ3:** Are the combination of smell-based metrics and commonly-used metrics more effective in predicting change-prone files regarding each change category?

The purpose of RQ1 is to determine whether or not each of the five smell-based metrics is potentially a useful predictor for change-proneness prediction. If so, those useful metrics can be employed alone or together with commonly used code metrics. As did in Giger et al.’s work [23], we choose the CK metrics [24] as the commonly used code metrics (i.e., baseline metrics), i.e., Weighted methods per class (WMC), Coupling between object classes (CBO), Lack of cohesion in methods (LCOM), Depth of inheritance tree (DIT), Number of children (NOC) and Response for class (RFC), since existing studies have shown that they are powerful to distinguish which files are more change-prone [21], [22]. RQ2 and RQ3 are set to assess the across-version predictive ability in the scenarios of ranking and classification.

Our empirical study is conducted on six typical Java open source projects, namely camel, derby, elasticsearch, hive, pig and lucene, with up to 60 releases hosted in GitHub. We use the following criteria to select subject projects: (1) long software version history with more than 2 years; (2) sizes ranging from 100 KLOC to 1MLOC; (3) the availability of mature releases from GitHub; (4) different domains that projects belong to, e.g., DataBase system, search engine, and research. Number of children (NOC) and Response for class (RFC), since existing studies have shown that they are powerful to distinguish which files are more change-prone [21], [22]. RQ2 and RQ3 are set to assess the across-version predictive ability in the scenarios of ranking and classification.

Our empirical study is conducted on six typical Java open source projects, namely camel, derby, elasticsearch, hive, pig and lucene, with up to 60 releases hosted in GitHub. We use the following criteria to select subject projects: (1) long software version history with more than 2 years; (2) sizes ranging from 100 KLOC to 1MLOC; (3) the availability of mature releases from GitHub; (4) different domains that projects belong to, e.g., DataBase system, search engine, and data analyzer. Due to limited space, more data information for replication, including release IDs and download URLs, can refer to our online appendix [25].

**B. Data Collection**

This section mainly introduces some techniques including how code metrics, code changes and smells are extracted from source code of a measured project with multiple releases. Fig. 1 shows the procedure of collecting these data. Specifically, all remote repositories of studied subjects are needed to be downloaded from GitHub and stored as local repositories, because accessing local repository not only promotes code parsing speed for calculating code metrics, but also is the requirement to extract structural changes using tool ChangeDistiller [26].

**1) Code Metrics Extraction and Smells Detection**

The CK metrics values mentioned in section III-A can be obtained using the tool ctk⁴ that are developed based on JDT⁵ and is reliable by our manual testing on several large projects.

⁴https://github.com/MauricioAniche/ck  
⁵http://www.eclipse.org

<table>
<thead>
<tr>
<th>Smell Type</th>
<th>Short Smell Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Field</td>
<td>A class contains more than one public field.</td>
</tr>
<tr>
<td>Data Class</td>
<td>A class contains data but not behavior related to the data.</td>
</tr>
<tr>
<td>Large Class</td>
<td>A class has grown too large with more than specified LOC.</td>
</tr>
<tr>
<td>Middle Man</td>
<td>A class delegating to other classes most of their methods.</td>
</tr>
<tr>
<td>Refused Parent</td>
<td>A subclass not using the protected methods of its superclass.</td>
</tr>
<tr>
<td>Speculative General</td>
<td>An unused class, method, field or parameter.</td>
</tr>
<tr>
<td>Data Clump</td>
<td>Some fields together appeared in a couple of classes.</td>
</tr>
<tr>
<td>Divergent Change</td>
<td>Different parts occur changes in class for different reasons.</td>
</tr>
<tr>
<td>Feature Entry</td>
<td>A method is more interested in other class than its own.</td>
</tr>
<tr>
<td>Long Method</td>
<td>A long method exists if it has extreme LOC.</td>
</tr>
<tr>
<td>Long Parameter List</td>
<td>A method having a long list of parameters.</td>
</tr>
<tr>
<td>Message Chains</td>
<td>A long chain of method call to implement a complex role.</td>
</tr>
<tr>
<td>Shotgun Surgery</td>
<td>A single change in a method needs many scattered changes.</td>
</tr>
<tr>
<td>Switch Case</td>
<td>A switch-case was misused to implement polymorphism.</td>
</tr>
</tbody>
</table>

In this paper, we consider 14 code smells listed in Table I, where the first column describes smell types and the second column shows their short descriptions. Noting that some smells have their aliases, e.g., Large Class is also named as God Class in some literatures [15]. We choose these smells for two reasons. First, they can be well measured based on the combination of code metrics and hence can be automatically identified by tools. Second, several systematic review papers [11]–[14] revealed that these smells were widely investigated by researchers in the past decade. In this paper, two opensource smell detection tools, namely BSdetctor [17] and AJCSD⁶ are employed to identify these 14 kinds of code smells for Java programs. The smell detection strategies behind two tools above are described in [15], [17].

(2) Structural Changes Extraction

The procedure of fine-grained change extraction is divided into two steps. First, JGit⁷ tool is employed to extract each pair of modified files having identical name from any two successive versions $v_i$ and $v_{i+1}$ of a given project (denoted by $f_i$ and $f_{i+1}$ in Fig. 1, respectively). Second, we utilize ChangeDistiller tool [26] to extract fine-grained changes from file $f_i$ and $f_{i+1}$. We choose this tool to detect fine-grained structural changes due to its powerful capability of identifying up to 48 types of fine-grained changes for Java code.

(3) Smell-based Metrics Calculation

In this paper, we use smell-based metrics as predictors in prediction model. A point to note that, these smell-based metrics are all derived from the evolutionary history of smelly files, rather than only one version snapshot. In previous study, Tabo et al proposed [27] four smell-based metrics (originally namely antipattern-based metrics), i.e., Average

⁶https://github.com/MelihAltintas/AutomaticJavaCodeSmellDetector  
⁷http://www.eclipse.org/jgit/
Number of Smells (ANS), Smell Complexity Metric (SCM), Smell Recurrence Length (SRL) and Smell Cumulative Pair-wise Differences (SCPD). In the context of our study, the first three smell-based metrics are selected and SCPD metric is not considered due to highly relevance to metric ANS. In addition, we newly define two smell-based metrics which leverage structural change information during the period of file evolution. The first one is Average Structural Changes (ASC) representing the average changes over the file’s evolutionary history where only changes occurred in a smelly file belonging to a certain version are taken into account. The other smell metric is Number of Distinctive Contributors (NDC). More definition details about these smell metrics refer to a technical report in our online appendix [25].

C. Model Construction and Evaluation

(1) Variable Description

The goal of this study is to assess the actual predictive power of smell-based metrics for effort-aware change-proneness prediction towards each fine-grained change type. However, the number of fine-grained change types considered in tool ChangeDistiller [26] reaches up to 48 in total and our initial investigation reveals that some change types have very lower occurrences in our dataset e.g., adding attribute modifiability. As a result, in order to have higher frequencies of some change types in our experiments, we combine several change types into one change type category according to their semantics. Specifically, all fine-grained change types are classified into four change (type) categories: api, state, functionality and statement, showed in Table II. In this paper, the modeling technique we choose is logistic regression where its moderate predictive ability in discriminating between change-prone and not change-prone files in previous studies [21]. With these variables, we can build a corresponding change prediction model.

\[
Pr(Y = 1|X_1, X_2, ..., X_n) = \frac{e^{\beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n}}{1 + e^{\beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n}}
\]  \hspace{1cm} (1)

where independent variables \(X_1, X_2, ..., X_n\) represent the metrics considered in this paper; while dependent variable \(Y\) only takes binary value of 1 or 0. \(\beta_i\) \((i = 0, 1, ..., n)\) is the regression coefficient and can be estimated through the maximization of log-likelihood. \(Pr(Y = 1|X_1, X_2, ..., X_n)\) stands for the probability of \(Y = 1\), which indicates a file (class) being structurally change-prone with respect to a particular change category. Similar to work in [28], [29], we use the adjusted \(OR\), the odds ratio associated with one standard deviation increase, to provide an intuitive insight into the impact of the dependent variable \(X_i\). \(OR(X_i)\) can be used to compare the relative magnitude of the effects of different independent variables, as the same unit is increased. For a given independent variable \(X_i\) (i.e., one metric in this paper), \(OR > 1\) means that the independent variable \(X_i\) is positively associated with the dependent variable \(Y\); \(OR = 1\) means that independent variable does not affect the dependent variable, while \(OR < 1\) indicates that independent variable is negatively associated with the dependent variable. The univariate logistic regression model is a special case of the multivariate logistic regression model, where there is only one independent variable.

(3) Analysis Method

In this paper, univariate logistic regression is used to evaluate the correlation between individual smell-based metrics and change-proneness. Meanwhile, multivariate logistic regression is employed to build three kinds of models based on CK metrics, smell-based metrics or these metrics in combination, respectively, when the particular change category is chosen as dependent variable.

<table>
<thead>
<tr>
<th>Category</th>
<th>Ingredients Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>api</td>
<td>Changes that involve the declaration of a class (e.g., class renaming and class API changes) and signature of method (e.g., modifier changes, method renaming, return type changes, changes of the parameter list).</td>
</tr>
<tr>
<td>state</td>
<td>Changes that affect object states of a class (e.g., fields addition and deletion).</td>
</tr>
<tr>
<td>functionality</td>
<td>Changes that affect the functionality of a class (e.g., methods addition and deletion).</td>
</tr>
<tr>
<td>statement</td>
<td>Changes that modify executable statements (e.g., statements insertion and deletion) among condition expressions in control structures and the modification of else-parts.</td>
</tr>
</tbody>
</table>
(3.1) Univariate logistic regression analysis for RQ1
For RQ1, our goal is preliminarily to find which smell-based metrics are potentially useful variables in change-proneness prediction models. Therefore, we do not discriminate the specific change types and particular versions, that is, all kinds of fine-grained change types (i.e., 48 in total) are accumulated in a file and the data from multiple versions of a considered project are combined as a unique dataset In this study, a smell-based metric is considered to be significantly related with change-proneness at the significance level $\alpha$ of 0.05, as suggested in [28]. When performing univariate logistic regression analysis for RQ1, we use Cook’s distance to identify the influential observations [30]. If the Cook’s distance of an observation is equal to or more than 1, such observation is considered as influential and is recommended to removed from dataset [30]. Moreover, for each smell-based metric, we use OR to quantify its effect on the change-proneness. This OR allows us to compare the relative magnitude of the effects of individual metrics on the change-proneness. In previous study, module size (i.e., file’s LOC in this study) might have a potential effect on the relationships between smell-based metrics and change-proneness [4], [8]. Therefore, there is a need to remove the potentially confounding effect of module size in order to understand their actual relationships, as suggested by Zhou et al [31]. For each smell-based metric, if corresponding p-value is less than 0.05 and OR is larger than 1, such metric is regarded as having significant positive correlation with change-proneness.

(3.2) Multivariate logistic regression analysis for RQ2 and RQ3
In order to answer RQ2 and RQ3, we perform a forward stepwise variable selection procedure to build three kinds of multivariate logistic regression models, i.e., “B” model built on CK metrics, “S” model built on smell-based metrics and “B+S” model built on their combination. One common problem in multivariate logistic regression analysis is multicollinearity in which two or more independent variables are highly linearly related. These highly correlated predictors may lead to inaccurate coefficient estimation in logistic regression model. Variance inflation factor (VIF) is a widely used indicator of multicollinearity. In this study, cut-off 10 is recommended to deal with multicollinearity in a regression model [32]. In addition, similar to analysis in RQ1, all raw data are checked for the presence of influential observations using Cooks distance [30]. After building the above models, we compare the prediction effectiveness for two pairs of models, i.e., “S” vs “B” and “B+S” vs “B”. In this paper, we only use across-version prediction to generate effectiveness data, because such method is supposed to be more suitable and practical for industrial software practitioners. Specifically, across-version prediction uses a model trained on earlier versions to predict change types in follow-up versions within the same project.

Ranking and classification are two typical scenarios of applying change-proneness prediction. In both scenarios, we evaluate the effectiveness of smell-based metrics in the context of effort-aware change-proneness prediction where we use LOC of a file to measure the effort required to inspect it. The intuition is that a larger file takes a longer time to review than a smaller one, hence smaller files should be inspected earlier if the possibility of being change-prone is the same. Therefore, in the ranking scenario, files are ranked in descending order according to their relative change risk, which is defined as the ratio of the predicted possibility of change-proneness to its LOC. In the classification scenario, files are first classified into two categories: high-risk and low-risk. After that, those files classified as high-risk are targeted for software quality enhancement. Due to the limited space, only condensed points are introduced and more details about following two performance indicators are described in a technique report in our online appendix [25].

- CE in Ranking Scenario

We borrowed the cost-effectiveness measure CE from Arisholm et al.’s work [33] to evaluate the effort-aware ranking effectiveness of a change-proneness prediction model. CE measure is based on the concept of the LOC-based Alberg diagram showed in Fig. 2, where the x-axis is the cumulative percentage of LOC of the files selected from the file ranking list and the y-axis correspondingly represents the cumulative percentage of changes for a given change category. Consequently, each change-proneness prediction model corresponds to a curve in the diagram. Fig. 2 shows the ranking performance of a prediction model $m$, where, $m$ may represent in our context the “B” model, the “S” model or the “B+S” model. To compute CE, we also consider two additional curves, which respectively correspond to “random” model and “optimal” model. In the “random” model, files are randomly selected to inspect. In the “optimal” model, files are sorted in decreasing order according to their actual change densities (the ratio of overall changes about a particular change category occurred in the file to the LOC of that file). Based on this diagram, the effort-aware ranking effectiveness of the prediction model $m$ is defined as follows:

$$CE_{\pi}(m) = \frac{A_{\pi}(m) - A_{\pi}(\text{random model})}{A_{\pi}(\text{optimal model}) - A_{\pi}(\text{random model})}$$

where $A_{\pi}(x)$ is the area under the curve corresponding to model $x$ ($m$, random model, optimal model) for a given $\pi$ percentage of LOC. The range of $CE_{\pi}(m)$ is $[-1,1]$ and larger value means a better ranking effectiveness while a negative $CE_{\pi}(m)$ indicates that the model is inferior to the random model. The cut-off value $\pi$ varies between 0
and 1, depending on the amount of available resource for inspecting files. In practice, practitioners are more interested in the ranking performance of a prediction model at the top fraction. Therefore, we choose the \( CE_\pi(m) \) at \( \pi = 0.1 \) and \( \pi = 0.2 \) to evaluate the performance of each model.

- **ER in Classification Scenario**

We use the effort reduction (ER) measure, originally called LOC inspection reduction (LIR) [34], to evaluate the effort-aware classification effectiveness of a change-proneness prediction model. The ER measure represents the ratio of the reduced LOC to inspect by using a classification model compared with a random selection to achieve the same recall of changes. To simplify the presentation, we assume that the system under analysis consists of \( n \) files. Let \( l_i \) be the LOC of file \( f_i \) and \( c_i \) be the number of changes about a change category occurred in it, \( 1 \leq i \leq n \). For a given prediction model \( m \), let \( p_i \) \( (1 \leq i \leq n) \) be 1 if the file \( f_i \) is predicted as high-risk by \( m \), and 0 otherwise. In the classification scenario, only those files predicted to be high-risk will be inspected for operations, which is consistent to findings by Peters and 1. Depending on the amount of available resource for inspecting files. In practice, practitioners are more interested in the ranking performance of a prediction model at the top fraction. Therefore, we choose the \( CE_\pi(m) \) at \( \pi = 0.1 \) and \( \pi = 0.2 \) to evaluate the performance of each model.

\[ ER(m) = \frac{Effort(random\ model) - Effort(m)}{Effort(random\ model)} \]  

where \( Effort(m) \) is the ratio of the total LOC in those predicted high-risk files to the total LOC in the system, i.e., \( Effort(m) = (\sum_{i=1}^{n} (l_i \times p_i))/\sum_{i=1}^{n} l_i \); while \( Effort(random\ model) \) is the proportion of LOC needed to inspect to the total LOC in the system that a random selection model needs to achieve the same recall of changes as the prediction model \( m \), i.e., \( Effort(random\ model) = (\sum_{i=1}^{n} (c_i \times p_i))/\sum_{i=1}^{n} c_i \). Noting that our aim in classification scenario is to classify each file into two categories, while logistic regression model returns a prediction probability. Thus, we need to choose a threshold for the prediction model \( m \). In the literature, there are two popular methods to determine the classification threshold. The first method is called balanced-pf-pd (BPP) method. This method employs the ROC curve corresponding to model \( m \) to determine the classification threshold. More details about BPP can be found in [35]. The second method is balanced-classification-error (BCE) method. Unlike BPP, BCE chooses the threshold to roughly equalize two classification error rates: false positive rate and false negative rate. As stated by Schein et al. [36], such an approach has the effect of giving more weight to errors from false positives in imbalanced data sets. This is especially important for change data, as they are typically imbalanced, i.e., most files do not occur any structural changes. For the simplicity of presentation, the effort reduction metrics under the BPP and BCE thresholds are respectively called “ER-BPP” and “ER-BCE”. For a given model, we use them to evaluate the effort-aware classification effectiveness of a prediction model. To avoid obtain misleading ER metric, we also use an additional metric “ER-AVG”, proposed by Zhou et al. [31], which represents the average effort reduction of the model over all possible thresholds on the test data set.

### IV. Results and Analysis

In this section, we try to answer three research questions proposed in section III-A based on experimental results. Due to limited space, we only report evaluation results of \( CE_\pi \) at \( \pi = 0.2 \) for effort-aware api prediction model in ranking scenario and \( ER-AVG \) in classification scenario. Results from other prediction models where dependent variable respectively takes on remainder change categories (i.e., state, functionality, statement) can refer to our online appendix [25].

**RQ1: How are the smell-based metrics correlated with structural change-proneness?**

We use results from univariate regression analysis to answer RQ1. For each smell-based metric, we test its correlation with structural change-proneness for each analyzed project. Noting that we only carry out correlation analysis at project level, i.e., do not discriminate specific version data. In Table VI, the column “Metric” represents the name of smell-based metrics, the column “Coeff.”, “p-value” and “OR” represent the estimated regression coefficient, the statistical significance of the coefficient from Z test and the odds ratio associated with one standard deviation increase, respectively. From Table VI, we can see that all smell-based metrics have significant correlation with structural change-proneness because their \( p \)-values are all less than 0.05 and OR values are more than 1. This indicates that files with a higher value of smell-based metric tend to be change-prone. Particularly, the OR values from metrics ANS and SKL are mostly larger than 2 in studies projects.

A representative example is the class MulticastProcessor from project camel. This class implements the multicast pattern to send a message exchange to a number of endpoints in which each endpoint receives a copy of the message exchange. In the starting version 2.0M2, the LOC of this class is 292 and it is affected by two smell instances, i.e., Speculative Generality(1) and Shotgun Surgery(1). When it evolves from version 2.0M2 to 2.3.0 through 2.1.0, the number of smells affecting it increases to 6 by adding 4 instances of Message Chains, accordingly, the structural changes underwent during this period are 7, 35 and 75, respectively. In subsequent version 2.6.0, the magnitude of LOC quickly reaches to 1016 (two times as many as prior one) and 7 new smell instances are newly added, i.e., Large Class(1), Speculative Generality(1), Message Chains(1), Divergent Change(2), Long Parameter List(2), which lead to average 45 structural changes in follow-up versions (i.e., 2.6.0 → 2.7.0 → 2.10.5 → 2.14.0). From the evolutionary history of this file, we can find that, the more smells a file contains, the more likely it will be subject to structural changes, which confirmed previous findings in [10], [19]. Therefore, the metric of Average Number of Smells (ANS) is indeed a suitable indicator for potentially change-prone files. On the other hand, we also find that once the smells are introduced, the developers may be aware of code smells, but are not willing to remove these smells by refactoring operations, which is consistent to findings by Peters and
Zaidman [37] as well as Tufano et al. [38]. Such long lifespan of smells (because of delayed refactoring behaviour) may also explain the significant relation between metric Smell Recurrence Length (SRL) and structural changes with OR ranging from 1.6 to 2.3, indicated in Table VI and VII.

However, the results shown in Table VI may not reflect the true correlations of the investigated metrics with structural change-proneness, as the potentially confounding effect of file size is not taken into account [31]. Table 10 reports the univariate analysis results after removing the confounding effect of file size. As can be seen, most smell-metrics except ANS and SCM in camel and hive, are still significantly related to structural change-proneness. For example, the smell metric ASC (Average Structural Changes) is significantly correlated to structural changes due to OR > 1. This result is expected and also confirms the basic assumption towards “modules changing often in the past will be most likely change-prone in the future as well”. However, for smell metric ANS (Average Number of Smells) and SCM (Smell Complexity Metric), their significant correlations with structural changes disappear in project hive after removing confounding effect of file size. We do not find extreme biased distribution about these two metrics and LOC of files in all versions of hive after manually inspecting their source code. The probable reason is due to other factors driving structural changes, rather than code quality enhancement, for example, user’s requirement for new features. These results suggest that the characteristics captured by smell-metrics are mostly different from file size and can be used as change-proneness indicators.

Overall, our univariate logistic regression analysis results indicate that smell-based metrics are, in most cases, significantly related to structural changes occurred in files regardless of whether the confounding effect of file size is removed or not.

**RQ2: Are the smell-based metrics more effective to predict change-prone files regarding each change category than CK metrics?**

In order to answer RQ2, we first build corresponding models “B” and “S” as described in section III-C. Then we compare the prediction effectiveness of the “B” and “S” models in ranking and classification scenarios under across-version prediction. Fig. 3a shows a series of boxplots which describe the distribution of the CE values at cutoff $\pi=0.2$ and ER-AVG values generated from the across-version api change-proneness predictions for models “B” and “S”. From Fig. 3a, the model built on smell-based metrics tends to perform generally better than the model built using CK metrics, specifically, we can obtain the following observations:

- **Ranking Performance**

For derby, hive and pig, their “S” models gain significant advantage over “B” models towards $CE_{0.2}$ with p-value < 0.05 when predicting api change-proneness. The effect sizes are all large in terms of Cliff’s $\delta$ (0.51 < $\delta$ < 1.68). For camel, the median CE of “S” model is larger than “B” model, but not significantly due to the p-value more than 0.05 in Wilcoxon signed-rank test and trivial Cliff’s $\delta$ (0.19). In particular, for elasticsearch and lucene, we can observe that the “S” model has a comparable median CE than “B” model and their median values are fluctuating around zero, indicating the ranking performance of these two models is similar to random model.

- **Classification Performance**

For camel, derby, hive and pig, the “S” model has a higher median ER-AVG as compared with “B” model. The p-values in Wilcoxon signed-rank test are all less than 0.05 and their effect sizes are all large in terms of Cliff’s $\delta$ (0.56 < $\delta$ < 1.23). For elasticsearch, the median ER-AVG of “S” model is slightly lower than “B” model with p-value more than 0.05 and trivial Cliff’s effect size ($\delta = 0.11$). For lucene, although the model “S” model has significantly ER-AVG than the one of “B” model (p-value < 0.05), the effect size in terms of Cliff’s $\delta$ is moderate ($\delta = 0.36$). After deeply investigating the causes behind the improvement of performances, we did found that the model built by smell-based metrics was able to classify more than 75% of smell instances in studied projects. These results highlight that these smell-based metrics are actually useful when predicting the change-proneness (particularly, the api change-proneness) of a smelly file. An interesting example regarding this characteristic is represented by class MapReduceOper (from version 0.11.0 of pig), Reply and BinaryOperatorNode (from version 10.10.1.1 of derby), as showed in Table III. These three classes are all affected by code smells. As expected, they all suffer from Large Class and Shotgun Surgery, particularly, class Reply also contains smells Data Clump(1), Long Method(1) and Switch Case(2), while class BinaryOperatorNode contains smells Message Chains(1) and Divergent Change(1). Noting that, the api changes occurred in these classes account for more than 13%, 50% and 51%, respectively. However, the basic model classifies these three classes as not change-prone based on CK metrics, showed in Table III, but the model “S” using smell-based metrics can classify these classes as change-prone. More interestingly, we also find that the model “S” even can correctly classify 76% of non-smelly files from predicted version as change-prone, shifting by 8% as compared with model “B”. This classification ability could be explained by the fact that smell-based metrics leverage information from evolutionary history of files, especially regarding structural changes occurred in the past. However, we also observe that, for elasticsearch, their median ER-AVG values are around zero, which indicates their classification performance is similar to random model.

In general, smell-based metrics provide an improvement over CK metrics under ranking and classification scenarios for effort-aware api change-proneness prediction.

**RQ3: Are the combination of smell-based metrics and commonly used code metrics more effective to predict change-prone files regarding each change category than CK metrics alone?**

In order to answer to RQ3, we first use the method described in section III-C to build the models “B” and “B+S” on each version data set. Then, we use the similar way did for RQ2
to compare the prediction effectiveness of the “B” and “B+S” models in ranking and classification scenarios under across-version change-proneness prediction. Fig. 3b shows paralleled boxplots which describe the distribution of the CE$s$ at cutoff $\pi=0.2$ and ER-AVGs generated from the across-version api change-proneness predictions for “B” and “B+S” models. Table IV quantitatively describes the performance comparison results in ranking scenario. In Table IV, the second and third columns represent the median $CE_{0.2}$ for the “B” model and the “B+S” model, respectively. The fourth and the fifth columns are respectively the percentage of the improvement for the “B+S” model over the “B” model and the effect sizes in terms of the Cliff’s $d$. The sixth column represents the p-value in the Wilcoxon tests. The means of columns in Table V are similar to the ones in Table IV. From Fig. 3b, Table IV and Table V, we have the following observations:

- **Ranking Performance**

  It is obvious that the “B+S” model significantly outperforms the “B” model for all subject projects due to their p-values less than 0.05. Specifically, for all studied projects, the “B+S” model has larger median $CE_{0.2}$ than the “B” model. For camel, derby, hive and pig, the effect sizes are large in terms of Cliff’s $d$ is moderate $(0.47 < d < 1.65)$, while for elasticsearch and lucene, their effect sizes are small $(|d| < 0.2)$. On average, the “B+S” model leads to about 90% improvement over the “B” model. Noting that, in Table IV, the entries in column of “improve” with underline means that the corresponding baseline “B” model has negative median $CE_{0.2}$, indicating inferior to random model, and hence its median $CE_{0.2}$ is replaced with the median $CE_{0.2}$ of “S” model.

- **Classification Performance**

  From Fig. 3b and Table V, we find that, for all studied projects, the “B+S” model has a larger ER-AVG as compared with “B” model with p-values less than 0.05. Their effect sizes are large $(0.49 < d < 1.16)$ except in elasticsearch $(d = 0.37)$. Although the performance of model “B+S” for elasticsearch is not very satisfactory, it is still better than the ones of model “B” and “S”. The class `TransportIndexAction` from package `org.elasticsearch.action.index` belongs to such case. The performance achieved by model “B+S” is shifted by +44% (from 0.45 to 0.65, while 0.27 achieved by model “B”). This indicates that smell-based metrics can provide supplementary information in comparison to traditional CK metrics. Besides, on average, the “B+S” model provides an improvement by 193.84%. Again, for camel, lucene and pig, their median ER-AVGs in Table V are not from “B” model but from “S” model because their “B” models are inferior to random model (ER-AVG<0).

Overall, the above observations indicate that the “B+S” model outperforms the “B” model in effort-aware api change-proneness prediction under ranking and classification scenarios. This demonstrates that the smell-based metrics considered in this paper is actually useful predictors in effort-aware change-proneness prediction.

Finally, it is worth noting that, we only report results and analysis of api change-proneness prediction in ranking and classification scenarios. For other change categories, we generally achieve similar conclusions. More details about their prediction results can refer to our online appendix [25].

<table>
<thead>
<tr>
<th>project</th>
<th>B</th>
<th>B+S</th>
<th>improve [%]</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>camel</td>
<td>0.0313</td>
<td>0.0277</td>
<td>12.5%</td>
<td>0.0001</td>
</tr>
<tr>
<td>derby</td>
<td>0.0367</td>
<td>0.0885</td>
<td>141.19%</td>
<td>0.5613</td>
</tr>
<tr>
<td>elasticsearch</td>
<td>0.0181</td>
<td>0.0421</td>
<td>133.07%</td>
<td>0.2851</td>
</tr>
<tr>
<td>hive</td>
<td>0.0392</td>
<td>0.0992</td>
<td>152.63%</td>
<td>1.6523</td>
</tr>
<tr>
<td>lucene</td>
<td>0.0100</td>
<td>0.0156</td>
<td>56.59%</td>
<td>0.2338</td>
</tr>
<tr>
<td>pig</td>
<td>0.0406</td>
<td>0.0520</td>
<td>37.97%</td>
<td>1.2596</td>
</tr>
</tbody>
</table>

**V. Threats to Validity**

This section discusses the threats to validity that can affect the results of our empirical study.

Threats to **construct validity** concern the relationship between theory and observation. In our study, this threat involves the data collection of independent and dependent variables. In our prediction model, independent variables represent CK metrics and smell-based metrics. CK metrics are extracted from tool ck, while code smells are detected using tool CBSDetector [17] and AJCSD. The tool ck and AJCSD are open-source tools that are developed based on JDT and they are adequately tested by our lab members. Thus, the data from these tools are reliable. As for fine-grained changes detection, we employ a state of the art tool ChangeDistiller [26] utilizing AST-based matching algorithm with higher detection accuracy. This tool has been used by many researchers for empirical studies [10], [19], [23]. Therefore, the construct validity of data in prediction model is satisfactory.

Threats to **internal validity** concerns the factors that could influence our observations. We did not manually examine many files containing relatively high numbers of structural
TABLE VI: Results of Univariate Logistic Regression Analysis Before Removing the Potentially Confounding Effect of File Size (LOC)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANS</td>
<td>0.285</td>
<td>&lt;0.001</td>
<td>2.447</td>
<td>0.108</td>
<td>&lt;0.001</td>
<td>2.191</td>
<td>0.385</td>
<td>&lt;0.001</td>
<td>2.200</td>
<td>0.085</td>
<td>&lt;0.001</td>
<td>2.256</td>
<td>0.152</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>SCM</td>
<td>9.620</td>
<td>&lt;0.001</td>
<td>2.244</td>
<td>10.283</td>
<td>&lt;0.001</td>
<td>1.926</td>
<td>6.806</td>
<td>&lt;0.001</td>
<td>1.491</td>
<td>20.336</td>
<td>&lt;0.001</td>
<td>1.545</td>
<td>3.214</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>SRL</td>
<td>0.057</td>
<td>&lt;0.001</td>
<td>2.290</td>
<td>0.065</td>
<td>&lt;0.001</td>
<td>2.366</td>
<td>0.053</td>
<td>&lt;0.001</td>
<td>1.856</td>
<td>0.062</td>
<td>&lt;0.001</td>
<td>2.260</td>
<td>0.021</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>ASC</td>
<td>0.056</td>
<td>&lt;0.001</td>
<td>2.860</td>
<td>0.033</td>
<td>&lt;0.001</td>
<td>2.164</td>
<td>0.053</td>
<td>&lt;0.001</td>
<td>1.730</td>
<td>0.034</td>
<td>&lt;0.001</td>
<td>1.401</td>
<td>0.030</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>NDC</td>
<td>0.227</td>
<td>&lt;0.001</td>
<td>1.854</td>
<td>0.221</td>
<td>&lt;0.001</td>
<td>2.004</td>
<td>0.816</td>
<td>&lt;0.001</td>
<td>1.491</td>
<td>0.193</td>
<td>&lt;0.001</td>
<td>2.054</td>
<td>0.068</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

TABLE VII: Results of Univariate Logistic Regression Analysis After Removing the Potentially Confounding Effect of File Size (LOC)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
<th>OR</th>
<th>Coeff.</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANS</td>
<td>0.008</td>
<td>0.185</td>
<td>1.023</td>
<td>0.036</td>
<td>&lt;0.001</td>
<td>1.220</td>
<td>0.366</td>
<td>&lt;0.001</td>
<td>2.028</td>
<td>-0.018</td>
<td>&lt;0.001</td>
<td>2.218</td>
<td>0.055</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>SCM</td>
<td>-0.914</td>
<td>0.002</td>
<td>0.942</td>
<td>2.598</td>
<td>&lt;0.001</td>
<td>1.141</td>
<td>5.760</td>
<td>&lt;0.001</td>
<td>1.382</td>
<td>-10.398</td>
<td>&lt;0.001</td>
<td>0.609</td>
<td>3.893</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>SRL</td>
<td>0.038</td>
<td>&lt;0.001</td>
<td>1.632</td>
<td>0.050</td>
<td>&lt;0.001</td>
<td>1.828</td>
<td>0.050</td>
<td>&lt;0.001</td>
<td>1.761</td>
<td>0.061</td>
<td>&lt;0.001</td>
<td>2.219</td>
<td>0.020</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>ASC</td>
<td>0.016</td>
<td>&lt;0.001</td>
<td>1.296</td>
<td>0.006</td>
<td>&lt;0.001</td>
<td>1.112</td>
<td>0.046</td>
<td>&lt;0.001</td>
<td>1.599</td>
<td>0.012</td>
<td>&lt;0.001</td>
<td>1.503</td>
<td>0.028</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>NDC</td>
<td>0.120</td>
<td>&lt;0.001</td>
<td>1.337</td>
<td>0.142</td>
<td>&lt;0.001</td>
<td>1.464</td>
<td>0.739</td>
<td>&lt;0.001</td>
<td>1.430</td>
<td>0.188</td>
<td>&lt;0.001</td>
<td>2.000</td>
<td>0.064</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

(a) B vs S evaluation under across-version change-proneness prediction with respect to api.
(b) B vs B+S evaluation under across-version change-proneness prediction with respect to api.

Fig. 3: B vs S and B vs B+S evaluation under across-version change-proneness prediction with respect to api.

Changes which also contained other smells beyond our consideration in this paper. Such an examination may reveal other smells that have a greater impact on structural changes, and which may be confounding our results. Yet, we filter out some test files in which changes may not have been driven by our considered code smells, but test smells [39]. The second threat to the internal validity of our study is the unknown effect of the deviation of the independent variables from the normal distribution. In our study, we used the raw data to build the logistic regression models when investigating RQ2 and RQ3. In other words, we did not take into account whether the independent variables follow a normal distribution. The reason is that, in logistic regression, there is no assumption related to normal distribution. In addition, we also consider the confounding effect of file size (LOC) in answering RQ3, as did in RQ1. After removing the potentially confounding effect of file size, however, our conclusions about api change-proneness prediction still hold.

Threat to external validity concerns the generalization of our findings. To mitigate this threat, we conducted our experiments on six open-source systems of different size and from different domains. The experimental results drawn from these subject are quite consistent. Furthermore, the data sets collected from these systems are large enough to draw statistically meaningful conclusions. We believe that our research provides a deeper understanding about the usefulness of smell-based metrics for effort-aware change-proneness prediction. Nonetheless, we do not claim that our findings can be generalized to all systems, as the subject systems under study might not be representative of systems in general. To mitigate this threat, we hope that other researchers will replicate our study across a wide variety of systems in the future.

VI. CONCLUSION

This paper reports an empirical study where five smell-based metrics have been examined for predicting effort-aware structural change-proneness. Experiments have been conducted on six typical Java open-source projects with up to 60 versions. Our findings indicate that, statistically, all the five smell-based metrics are positively correlated with structural changes which may be confounding our results. Yet, we filter out some test files in which changes may not have been driven by our considered code smells, but test smells [39]. The second threat to the internal validity of our study is the unknown effect of the deviation of the independent variables from the normal distribution. In our study, we used the raw data to build the logistic regression models when investigating RQ2 and RQ3. In other words, we did not take into account whether the independent variables follow a normal distribution. The reason is that, in logistic regression, there is no assumption related to normal distribution. In addition, we also consider the confounding effect of file size (LOC) in answering RQ3, as did in RQ1. After removing the potentially confounding effect of file size, however, our conclusions about api change-proneness prediction still hold.

Threat to external validity concerns the generalization of our findings. To mitigate this threat, we conducted our experiments on six open-source systems of different size and from different domains. The experimental results drawn from these subject are quite consistent. Furthermore, the data sets collected from these systems are large enough to draw statistically meaningful conclusions. We believe that our research provides a deeper understanding about the usefulness of smell-based metrics for effort-aware change-proneness prediction. Nonetheless, we do not claim that our findings can be generalized to all systems, as the subject systems under study might not be representative of systems in general. To mitigate this threat, we hope that other researchers will replicate our study across a wide variety of systems in the future.
In the future work, we plan to validate our results in more subject projects and consider other classifiers to build more accurate change-proneness prediction models.

REFERENCES


[17] Tracy Hall, Min Zhang, David Bowes, and Yi Sun. Some code smells have a significant but small effect on faults. ACM Transactions on Software Engineering and Methodology, 23(4):33, 2014.
