EPM–RT–2010-02

AN EMPIRICAL STUDY ON THE RELATION BETWEEN IDENTIFIERS AND FAULT PRONENESS

Venera Arnaoudova, Laleh Eshkevari, Rocco Oliveto,
Yann-Gaël Guéhéneuc, Giuliano Antoniol
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Août 2010

EPM-RT-2010-02

An Empirical Study on the Relation between
Identifiers and Fault Proneness

Venera Arnaoudova, Laleh Eshkevari, Rocco Oliveto
Yann-Gaël Guéhéneuc, Giuliano Antoniol
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Août 2010

Dépôt légal :
Bibliothèque nationale du Québec, 2009
Bibliothèque nationale du Canada, 2009

EPM-RT-2010-02
An Empirical Study on the Relation between Identifiers and Fault Proneness
par : Venera Arnaoudova, Laleh Eshkevari, Rocco Oliveto, Yann-Gaël Guéhéneuc, Giuliano Antoniol
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

An Empirical Study on the Relation between Identifiers and Fault Proneness

Venera Arnaoudova*, Laleh Eshkevari*, Rocco Oliveto†, Yann-Gaël Guéhéneuc‡, Giuliano Antoniol*
*SOCCER Lab. – DGIGL, École Polytechnique de Montréal, Québec, Canada
†SE@SA Lab – DMI, University of Salerno - Salerno - Italy
‡Ptidej Team – DGIGL, École Polytechnique de Montréal, Québec, Canada
{venera.arnaoudova, laleh.mousavi-eshkevari,yann-gael.gueheneuc}@polymtl.ca, roliveto@unisa.it, antoniol@ieee.org

***Abstract*—Poorly-chosen identifiers have been reported in the literature as misleading and increasing the program comprehension effort. Identifiers are composed of terms, which can be dictionary words, acronyms, contractions, or simple strings. We conjecture that the use of identical terms in different contexts may increase the risk of faults. We investigate our conjecture using a measure combining term entropy and term context-coverage to study whether certain terms increase the odds ratios of methods to be fault-prone. Entropy measures the *physical dispersion* of terms in a program: the higher the entropy, the more scattered across the program the terms. Context coverage measures the *conceptual dispersion* of terms: the higher their context coverage, the more unrelated the methods using them. We compute term entropy and context-coverage of terms extracted from identifiers in Rhino 1.4R3 and ArgoUML 0.16. We show statistically that methods containing terms with high entropy and context-coverage are more fault-prone than others.**

***Keywords*-Source code identifiers; fault models; program comprehension.**

I. Introduction

Program comprehension is preliminary to any maintenance activity because developers must first identify relevant code fragments before performing any activity. Source code of good quality in terms of comments and identifiers can surely ease such activities because developers use identifiers to build their mental models of the code under analysis. Thus, poorly-chosen identifiers could be misleading and increase the risk of faults.

Fault-prone entities, *i.e.*, classes, methods, and attributes, in object-oriented programs have been characterized by their internal characteristics. For example, several studies used metrics, such as Chidamber and Kemerer (CK) metrics suite [1], to build models to locate fault-prone entities [2], [3], [4]. These studies show that there are more faults in (1) complex [5] and (2) large entities (in term of LOCs), thus characterizing fault prone entities using only *structural data*. However, fault proneness is a complex phenomenon hardly captured by a single characteristic, such as complexity or size.

Several studies showed that identifiers impact program comprehension (*e.g.*, [6], [7], [8]) and code quality [9]. Indeed, identifiers must be sufficiently distinctive yet must relate to one another and to the context in which they appear [7]. We concur with Deißenböck and Pizka's observation that proper identifiers improve quality and that identifiers should be used consistently [7]. We consequently present, to the best of our knowledge, the first empirical study on the relation between the terms in identifiers, their spread in entities, and fault proneness.

By *term* we mean any substring in an identifier or to the identifier itself if it is not compound. Terms are obtained by splitting identifiers, for example with a Camel-case splitter [10], and they may be dictionary words (*e.g.*, *userAccount*), acronyms (*e.g.*, *uint* for unsigned integer), abbreviations (*e.g.*, *int* for integer), or any string (*e.g.*, *x11* for the window system).

We conjecture that a term should carry a single meaning in the context where it is used. Thus, terms used for different concepts and–or in different contexts may either reflect the program domain or some misunderstandings but would lead, in any case, to an increase in the developers' effort to understand the role of the associated entities, ultimately leading to faults.

We present a novel measure based on linguistic data and an empirical study to verify our conjecture. The novel measure quantifies terms from two aspects: *term entropy* and *context-coverage*. Term entropy is derived from entropy in information theory and measures the "physical" dispersion of a term in a program, *i.e.*, the higher the entropy, the more scattered the term is across entities. Term context-coverage is based on an Information Retrieval method and measures the "conceptual" dispersion of the entities in which the term appears, *i.e.*, the higher the context coverage of a term, the more unrelated are the entities containing it.

We perform an empirical study relating terms with high entropy and high context-coverage to the fault-proneness of the methods and attributes in which they appear. We analyze two widely studied open source programs, ArgoUML[1] and Rhino[2] because sets of manually-validated faults for these two programs exist in the literature. We show that there is a statistically significant relation between the "physical" and "conceptual" dispersion of terms and fault proneness.

Thus, the contributions of this paper are as follows:

- A novel measure characterizing the "physical" and "conceptual" dispersions of terms;
- An empirical study showing the relation between the proposed measure and entities fault proneness.

The rest of the paper is organized as follows. Section II presents related work. Section III introduces background definitions and defines the novel measure. Section IV describes our empirical study, reports, and discusses its results. Section V concludes and suggests future work.

II. Related Work

Our study relates to Information Retrieval (IR), fault proneness, and the quality of source code identifiers.

a) Entropy and IR-based Metrics: Several metrics based on entropy exist. Olague *et al.* [11] used entropy-based metrics to explain the changes that a class undergoes between versions of an object-oriented program. They showed that classes with high entropy tend to change more than classes with lower entropy. Yu *et al.* [12] combined entropy with component-dependency graphs to measure component cohesion. Entropy was also used by Snider [13] to measure the structural quality of C code by comparing the entropy of legacy program with that of a rewrite of the same program aimed at producing a well-structured system. The rewritten program had a much lower entropy that the legacy program.

IR methods have also been used to define new measures of source code quality. Etzkorn *et al.* [14] presented a new measure for object-oriented programs that examines the implementation domain content of a class to measure its complexity. Patel *et al.* [15] and Marcus *et al.* [16] used Vector Space Model (VSM) and Latent Semantic Indexing (LSI) [17], respectively, to measure the semantic cohesion of a class. They used IR methods to compute the overlap of semantic information in implementations of methods, calculating the similarities among the methods of a class. Applying a similar LSI-based approach, Poshyvanyk and Marcus [18] defined new coupling metrics based on semantic similarity. Binkley *et al.* [19] also used VSM to analyze the quality of programs. Split identifiers extracted from entities were compared against the split identifiers extracted from the comments of the entities: the higher the similarity, the higher the quality of the entities. The metric was also applied to predict faults and a case study showed that the metric is suitable for fault prediction in programs obeying code conventions.

b) Metrics and Fault Proneness: Several researchers studied the correlations between static object-oriented metrics, such as the CK metrics suite [1], and fault proneness. For example, Gyimóthy *et al.* [3] compared the accuracy of different metrics from CK suite to predict fault-prone classes in Mozilla. They concluded that CBO is the most relevant predictor and that LOC is also a good predictor. Zimmermann *et al.* [5] conducted a case study on Eclipse showing that a combination of complexity metrics can predict faults, suggesting that the more complex the code is, the more faults in it. El Emam *et al.* [20] showed that the previous correlations between object-oriented metrics and fault-proneness are mostly due to the correlations between the metrics and size. Hassan [21] observed that a complex code-change process negatively affects programs. He measured the complexity of code change through entropy and showed that the proposed change complexity metric is a better predictor of faults than other previous predictors.

c) Identifiers and Program Comprehension: Haiduc and Marcus [8] studied several open-source programs and found that about 40% of the domain terms were used in the source code. Unfortunately, in collaborative environments, the probability of having two developers use the same identifiers for different entities is between 7% and 18% [22]. Thus, naming conventions are crucial for improving the source code understandability. Butler *et al.* [9] analyzed the impact of naming conventions on maintenance effort, *i.e.*, on code quality. They evaluated the quality of identifiers in eight open-source Java libraries using 12 naming conventions. They showed that there exists a statistically significant relation between flawed identifiers (*i.e.*, violating at least one convention) and code quality.

The role played by identifiers and comments on source code understandability has been empirically analyzed by Takang *et al.* [23], who compared abbreviated identifiers with full-word identifiers and uncommented code with commented code. They showed that (1) commented programs are more understandable than non-commented programs and (2) programs containing full-word identifiers are more understandable than those with abbreviated identifiers. Similar results have also been achieved

[1]http://argouml.tigris.org/
[2]http://www.mozilla.org/rhino/

by Lawrie *et al.* [24]. These latter studies also showed that, in many cases, abbreviated identifiers are as useful as full-word identifiers. Recently, Binkley *et al.* [25] performed an empirical study of the impact of identifier style on code readability and showed that Camel-case identifiers allow more accurate answers.

III. Identifiers, Entropy, and Contexts

We now detail the computations of term entropy and context-coverage. With no loss of generality, we focus on methods and attributes because they are "small" contexts of identifiers. Moreover, we consider attributes because they are often part of some program faults, *e.g.*, in Rhino they participate to 37% of the reported faults. However, the computation can be broaden by using classes or other entities as contexts for identifiers.

A. Data extraction

We extract the data required to compute term entropy and context-coverage in two steps. First, we extract the identifiers found in class attributes and methods, *e.g.*, , names of variables and of called methods, user defined types, method parameters. Extracted identifiers are split using a Camel-case splitter to build the term dictionary, *e.g.*, *getText* is split into *get* and *text*. We then apply two filters on the dictionary. First, we remove terms with a length less than two because their semantics is often unclear and because they most likely correspond to loop indexes (*e.g.*, I, j, k). Second, we prune terms appearing in a standard English stop-word list augmented with programming language keywords.

Second, the linguistic data is summarized into a $m \times n$ frequency matrix, *i.e.*, a *term-by-entity* matrix. The number of rows of the matrix, m, is the number of terms in the dictionary. The number of columns, n, corresponds to the number of methods and attributes. The generic entry $a_{i,j}$ of the term-by-entity matrix denotes the number of occurrences of the i^{th} term in the j^{th} entity.

B. Term Entropy

Shannon [26] measures the amount of uncertainty, or entropy, of a discrete random variable X as:

$$H(X) = -\sum_{x \in \varkappa} p(x) \cdot log(p(x))$$

where $p(x)$ is the mass probability distribution of the discrete random variable X and $\varkappa$ is its domain.

We consider terms as random variables with some associated probability distributions. We normalize each row of the term-by-entity matrix so that each entry is in $[0,1]$ and the sum of the entries in a row is equals to one to obtain a probability distribution for each term. Normalization is achieved by dividing each $a_{i,j}$ entry by the sum of all $a_{i,j}$ over the row i. A normalized entry $\widehat{a}_{i,j}$ is then the probability of the presence of the term t_i in the j^{th} entity. We then compute term entropy as:

$$H(t_i) = -\sum_{j=1}^{n} (\widehat{a}_{i,j}) \cdot log(\widehat{a}_{i,j}) \quad i = 1, 2, \ldots, m$$

With term entropy, the more scattered among entities a term is, the closer to the uniform distribution is its mass probability and, thus, the higher is its entropy. On the contrary, if a term has a high probability to appear in few entities, then its entropy value will be low.

C. Term Context Coverage

While term entropy characterizes the "physical" distribution of a term across entities, context-coverage measures its "conceptual" distribution in the entities in which the term appears. In particular, we want to quantify whether a same term is used in different contexts, *i.e.*, methods and–or attributes, with low textual similarity. Thus, the context coverage of term t_k (where $k = 1, 2, \ldots, m$) is computed as the average textual similarity of entities containing t_k:

$$CC(t_k) = 1 - \frac{1}{\begin{pmatrix} |C| \\ 2 \end{pmatrix}} \sum_{\substack{i = 1 \ldots |C|-1 \\ j = i+1 \ldots |C| \\ e_i, e_j \in C}} sim(e_i, e_j)$$

where $C = \{e_l | \widetilde{a}_{k,p} \neq 0\}$ is the set of all entities in which term t_k occurs and $sim(e_i, e_j)$ represents the textual similarity between entities e_i and e_j. Note that the number of summations is $\begin{pmatrix} |C| \\ 2 \end{pmatrix}$ because $sim(e_i, e_j) = sim(e_j, e_i)$.

A low value of the context coverage of a term means a high similarity between the entities in which the term appears, *i.e.*, the term is used in consistent contexts.

To computed the textual similarity between entities we exploit LSI, a space reduction based method widely and successfully used in IR [17]. In particular, LSI applies a factor analysis technique to estimate the "latent" structure in word usage trying to overcome the main deficiencies of IR methods, such as synonym and polysemy problems. In particular, the non-normalized term-by-entity LSI projection into the entities subspace $\widetilde{a}_{i,j}$ captures the more important relations between terms and entities. The columns of the reduced term-by-entity matrix represent entities and can be thought of as elements of a vector space. Thus, the similarity between two entities can be measured by the cosine of the angle between the corresponding vectors.

D. Aggregated Metric

In this preliminary investigation we use the variable *numHEHCC* ("number of high entropy and high context coverage"), associated with all entities, to compute correlation, build linear as well as logistic models and contingency tables throughout the following case study:

$$numHEHCC(E_j) = \sum_{i=1}^{m} a_{ij} \cdot \psi(H(t_i) \geq th_H \wedge CC(t_i) \geq th_{CC})$$

where a_{ij} is the frequency in the term-by-entity matrix of term t_i and entity E_j $(j = 1, 2, \ldots, n)$ and $\psi()$ is a function returning one if the passed Boolean value is true and zero otherwise.

Thus, $numHEHCC$ represents the overall number of times any term with high entropy (value above th_H) and high context coverage (value above th_{CC}) is found inside an entity.

IV. Case Study

We now present a study of the term entropy and context-coverage measures following the Goal-Question-Metrics paradigm [27]. The *goal* of the study is to investigate the relation (if any) between term entropy and context-coverage, on the one hand, and entities fault proneness, on the other hand. The *quality focus* is a better understanding of characteristics likely to hinder program comprehension and to increase the risk of introducing faults during maintenance. The *perspective* is both of researchers and practitioners who use metrics to study the characteristic of fault prone entities.

The *context* of the study is two open-source programs: Rhino, a JavaScript/ECMAScript interpreter and compiler part of the Mozilla project, and ArgoUML, a UML modeling CASE tool with reverse-engineering and code-generation capabilities. We selected ArgoUML and Rhino because (1) several versions of these prorgrams are available, (2) they were previously used in other case studies [28], [29], and (3) for ArgoUML (from version 0.10.1 to version 0.28) and for Rhino (from version 1.4R3 to version 1.6R5), a mapping between faults and entities (attributes and methods) is available [29], [30].

A. Research Questions

Entropy and context coverage likely capture features different from size or other classical object-oriented metrics, such as the CK metrics suite [31]. However, it is well known that size is one of the best fault predictors [3], [32], [33] and, thus, we first verify that $numHEHCC$ is somehow at least partially complementary to size.

Second, we believe that developers are interested in understanding why an entity may be more difficult to change than another. For example, given two methods using different terms, all their other characteristics being equal, they are interested to identify which of the two is more likely to take part in faults if changed.

Therefore, the case study is designed to answer the following research questions:

- **RQ**$_1$ **– Metric Relevance:** Do term entropy and context-coverage capture characteristics different from size and help to explain entities fault proneness? This question investigates if term entropy and context-coverage are somehow complementary to size, and thus, quantify entities differently.
- **RQ**$_2$ **– Relation to Faults:** Do term entropy and context-coverage help to explain the presence of faults in an entity? This question investigates if entities using terms with high entropy and context-coverage are more likely to be fault prone.

Fault proneness is a complex phenomenon impossible to capture and model with a single characteristic. Faults can be related to size, complexity, and–or linguistic ambiguity of identifiers and comments. Some faults may be better explained by complexity while other by size or linguistic inconsistency of poorly selected identifiers. Therefore, we do not expect that **RQ**$_1$ and **RQ**$_2$ will have the same answer in all version of the two programs and will be universally true. Nevertheless, as previous authors [6], [7], [9], [19] we believe reasonable to assume that identifiers whose terms have with high entropy and high context-coverage hint at poor choices of names and, thus, at a higher risk of faults.

B. Analysis Method

To statistically analyze **RQ**$_1$, we computed the correlation between the size measured in LOCs and a new metric derived from entropy and context-coverage. Then, we estimated the linear regression models between LOCs and the new metric. Finally, as an alternative to the Analysis Of Variance (ANOVA) [34] for dichotomous variables, we built logistic regression models between fault proneness (explained variable) and LOCs and the proposed new metric (explanatory variables).

Our goal with **RQ**$_1$ is to verify whether term entropy and context-coverage capture some aspects of the entities at least partially different from size. Thus, we formulate the null hypothesis:

> H_{0_1}*: The number of terms with high entropy and context-coverage in an entity does not capture a dimension different from size and is not useful to explain its fault proneness.*

We expect that some correlation with size does exist: longer entities may contain more terms with more chance to have high entropy and high context-coverage.

Then, we built a linear regression model to further analyze the strength of the relation in term of unexplained variance, *i.e.*, $1 - R^2$. This model indirectly helps to verify that entropy and context-coverage contribute to explain fault proneness in addition to size.

Finally, we performed a deeper analysis via logistic regression models. We are not interested in predicting faulty entities but in verifying if entropy and context-coverage help to explain fault proneness. The multivariate logistic regression model is based on the formula:

$$\pi(X_1, X_2, \dots, X_n) = \frac{e^{C_0 + C_1 \cdot X_1 + \cdots + C_n \cdot X_n}}{1 + e^{C_0 + C_1 \cdot X_1 + \cdots + C_n \cdot X_n}}$$

where X_i are the characteristics describing the entities and $0 \leq \pi \leq 1$ is a value on the logistic regression curve. In a logistic regression model, the dependent variable π is commonly a dichotomous variable, and thus, assumes only two values $\{0, 1\}$, *i.e.*, it states whether an entity took part in a fault (1) or not (0). The closer $\pi(X_1, X_2, \dots, X_n)$ is to 1, the higher is the probability that the entity took part in a fault. An independent variable X_i models information used to explain the fault proneness probability; in this study we use a metric derived from term entropy and the context-coverage, $numHEHCC$, and a measure of size (LOCs) as independent variables.

Once independent variables are selected, given a training corpus, the model estimation procedure assigns an estimated value and a significance level, p-value, to the coefficients C_i. Each C_i p-value provides an assessment of whether or not the i^{th} variable helps to explain the independent variable: fault proneness of entities.

Consequently, we expect that the logistic regression estimation process would assign a statistically relevant p-value to the coefficient of a metric derived from term entropy and context coverage, *i.e.*, lower than 0.05 corresponding to a 95% significance level.

With respect to our second research question (**RQ**$_2$) we formulate the following null hypothesis:

> H_{0_2}: *There is no relation between high term entropy and context coverage of an entity and its fault proneness.*

We use a prop-test (Pearson's chi-squared test) [34] to test the null hypothesis. If term entropy and context coverage are important to explain fault proneness, then the prop-test should reject the null hypothesis with a statistically significant p-value.

To quantify the effect size of the difference between entities with and without high values of term entropy and context coverage, we also compute the *odds ratio* (OR) [34] indicating the likelihood of the entities to have such high values for our metric. OR is defined as the ratio of the odds p of a fault prone entity to have high term entropy and high context coverage to the odds q of this entity to have low entropy and context coverage: $OR = \frac{p/(1-p)}{q/(1-q)}$. When $OR = 1$ the fault prone entities can either have high or low term entropy and context coverage. Otherwise, if $OR > 1$ the fault prone entities have high term entropy and high context coverage. Thus, we expect $OR > 1$ and a statistically significant p-value (*i.e.*, again 95% significance level).

C. Execution

We download several versions of Rhino for which faults were documented by Eaddy *et al.* [29] from the Mozilla Web site[3]. Versions of ArgoUML were downloaded from the Tigris Community Web site[4]. We selected the version of ArgoUML that has the maximum number of faulty entities (ArgoUML v0.16.) and one of the versions of Rhino, Rhino v1.4R3.

The selected version of ArgoUML consists of 97,946 lines of Java code (excluding comments and blank lines outside methods and classes), 1,124 Java files, and 12,423 methods and fields. Version 1.4R3 of Rhino consists of 18,163 lines of Java code (excluding comments and blank lines outside methods and classes), 75 files, 1,624 methods and fields. Indeed, Rhino v1.4R3 is the first and smallest Rhino version with documented faults; since our study in the earliest phase required a remarkable manual verification of data and results we selected v1.4R3 to simplify and minimize verification effort. In essence, all data and results where manually and carefully verified several times on Rhino v1.4R3. ArgoUML data were also verified but manual verification was limited to the ambiguous and critical cases discovered by Rhino analysis.

To create the term-by-entity matrix, we first parse the Java files of Rhino and ArgoUML to extract identifiers. We obtain terms by splitting the identifiers using a Camel-case split algorithm. We compute term entropy and context coverage using the approach presented in the previous section. We finally use existing fault mappings [29], [30] to tag methods and attributes and relate them with entropy and context coverage values. The following paragraphs detail each step.

1) Term extraction: We used a grammar for the Java programming language v1.5 and JavaCC[5] to generate

[3]https://developer.mozilla.org/
[4]http://argouml.tigris.org/
[5]https://javacc.dev.java.net/

a Java parser and extract identifiers. We verify the completeness and soundness of the grammar and of the extracted identifiers by parsing 11 versions of Rhino and 11 versions of ArgoUML. We omitted two versions of Rhino because their numbers of faulty entities was too low: two and six. In general Rhino versions have a very limited numbers of documented faults and this may possibly bias model estimation and derived conclusions; for example Rhino v1.4R3 contains 14 faulty entities. Only five Rhino and four ArgoUML versions generate some parse errors on more than 8% of files. We selected a Rhino version with 1% parsing errors (v1.4R3) and one ArgoUML (v0.16) with 4.7% parsing errors. Despite the parsing errors we choose v0.16 for ArgoUML as it was the version with the highest number of faulty entities, and v1.4R3 for Rhino as it has one of the lowest number of faulty entities with comparison to other versions and also, as noted above, it is the smallest version. The extracted identifiers were split considering both digits and underscore plus Camel-case aiming at extracting terms. The extracted terms are then filtered according to both stop-word function and list.

2) Mapping Faults to Entities: We reuse previous findings to map faults and entities. For Rhino the mapping of faults with entities was done by Eaddy *et al.* [29] for 11 versions of Rhino. We obtain the mapping which corresponds to Rhino v1.4R3 by extracting, for each fault, its reporting date/time[6] and its fixing date/time. Then, we keep only those faults which fall under one of the following two cases: (i) the reporting date of the fault was before the release date of v1.4R3 and its fixing date was after the release date of the same version, (ii) the reporting date of the fault is after the release date of v1.4R3 and before the release date of the next version (v1.5R1). As for ArgoUML, we also use a previous mapping between faults and classes [30]. For each class marked as faulty, we compare its attributes and methods with the attributes and methods of the same class in the successive version and keep those that were changed and mark them as faulty.

3) Mapping Entities to Entropy and Context Coverage: We identify entities with *high* term entropy and context coverage values by computing and inspecting the box-plots and quartiles statistics of the values on all Rhino versions and the first five versions of ArgoUML. The term context coverage distribution is skewed towards high values. For this reason we use 10% highest values of term context coverage to define a threshold identifying the high context coverage property. In other words, standard outlier definition was not applicable to context coverage. We do not observe a similar skew for the values of term entropy and, thus, the threshold for high entropy values is based on the standard outlier definition (1.5 times the inter-quartile range above the 75% percentile). We use the two thresholds to measure for each entity, the number of terms characterized by high entropy and high context coverage that it contains.

[6]https://bugzilla.mozilla.org/query.cgi

Table I
CORRELATION TEST FOR ARGOUML V0.16 AND RHINO V1.4R3.

System	Correlation	p-values
ArgoUML	0.4080593	$\prec 2.2e-16$
Rhino	0.4348286	$\prec 2.2e-16$

Table II
LINEAR REGRESSION MODELS FOR RHINO V1.4R3 AND ARGOUML V0.16.

	Variables	Coefficients	p-values
Rhino ($R^2=0.1891$)	Intercept	0.038647	0.439
	LOC	0.022976	$\prec 2e-16$
Argo (R^2=0.1665)	Intercept	-0.0432638	0.0153
	LOC	0.0452895	$\prec 2e-16$

D. Results

We now discuss the results achieved aiming at providing answers to our research questions.

1) RQ_1 – Metric relevance: Table I reports the results of Pearson's product-moment correlation for both Rhino and ArgoUML. As expected, some correlation exists between LOC and $numHEHCC$ plus the correlation is of the same order of magnitude for both programs.

Despite a 40% correlation a linear regression model built between $numHEHCC$ (dependent variable) and LOC (independent variable) attains an R^2 lower than 19% (see Table II). The R^2 coefficient can be interpreted as the percentage of variance of the data explained by the model and thus $1-R^2$ is an approximations of the model unexplained variance. In essence Table II support the conjecture that LOC does not substantially explain $numHEHCC$ as there is about 80% (85%) of Rhino (ArgoUML) $numHEHCC$ variance not explained by LOC. Correlation and linear regression models can be considered a kind of sanity check to verify that LOC and $numHEHCC$ help to explain different dimensions of fault proneness.

The relevance of $numHEHCC$ in explaining faults, on the programs under analysis, is further supported by logistic regression models. Table III reports the interaction model built between fault proneness (explained variable) and the explanatory variables LOC and $numHEHCC$. In both models, $M_{ArgoUML}$ and M_{Rhino}, the intercept is relevant as well as $numHEHCC$. Most noticeably in Rhino the LOC

Table III
ARGOUML V0.16 AND RHINO V1.4R3 LOGISTIC REGRESSION MODELS.

	Variables	Coefficients	p-values
$M_{ArgoUML}$	Intercept	-1.688e+00	$\prec 2e-16$
	LOC	7.703e-03	$8.34e-10$
	numHEHCC	7.490e-02	$1.42e-05$
	LOC:numHEHCC	-2.819e-04	0.000211
M_{Rhino}	Intercept	-4.9625130	$\prec 2e-16$
	LOC	0.0041486	0.17100
	numHEHCC	0.2446853	0.00310
	LOC:numHEHCC	-0.0004976	0.29788

Table IV
ARGOUML V0.16 CONFUSION MATRIX.

ArgoUML	numHEHCC ≥ 1	numHEHCC = 0	Total
Fault prone	381	1706	2087
Fault free	977	9359	10336
Total	1358	11065	12423
p-value $\prec 2.2e-16$			
Odds ratio = 2.139345			

Table V
RHINO V1.4R3 CONFUSION MATRIX.

Rhino	numHEHCC ≥ 1	numHEHCC = 0	Total
Fault prone	6	8	14
Fault free	172	1438	1610
Total	178	1446	1624
p-value = 0.0006561			
Odds ratio = 6.270349			

coefficient is not statistically significant as well as the interaction term ($LOC : numHEHCC$). This is probably a fact limited to Rhino version 1.4R3 as for ArgoUML both LOC and the interaction term are statistically significant. Notice however, that in both models $M_{ArgoUML}$ and M_{Rhino} the LOC coefficient is, at least, one order of magnitude smaller than the $numHEHCC$ coefficient. This can partially be explained by the different range of LOC versus $numHEHCC$. On average in both programs method size is below 100 LOC and most often a method contains one or two terms with high entropy and context coverage. Thus, at first glance we can safely say that both LOC and $numHEHCC$ have the same impact in term of probability. In other words, the models in Table III clearly show that LOC and $numHEHCC$ capture different aspects of the fault proneness characteristic. Base on the reported results we can conclude that although some correlation exists between LOC and $numHEHCC$, statistical evidence allows us to reject, on the programs under analysis, the null hypothesis H_{0_1}.

2) RQ$_2$ – Relation to faults: To answer **RQ**$_2$, we perform prop-tests (Pearson's chi-squared test) and test the null hypothesis H_{0_2}. Indeed, (i) if prop-tests revel that *numHEHCC* is able to divide the population into two sub-populations and (ii) if the sub-population with positive values for *numHEHCC* has an odds ratio bigger than one, then *numHEHCC* may act as a risk indicator. For entities with positive *numHEHCC* it will be possible to identify those terms leading to high entropy and high context coverage, identifying also the contexts and performing refactoring actions to reduce entropy and high context coverage.

Tables IV and V show the confusion matrices for ArgoUML v0.16 and Rhino v1.4R3, together with the corresponding p-value and odds ratios. As the tables show, the null hypothesis H_{0_2} can be rejected.

We further investigate, with Tables VI and VII, the relation between $numHEHCC$ and odds ratio. These contingency tables compute the odds ratio of entities containing two or more terms with high entropy and high context coverage with those entities which only contain one high entropy and high context coverage term. They are not statistically significant, but the odds ratio is close to one, this seems to suggest that the real difference is between not containing high entropy and high context coverage terms and just containing one or more. The results allow us to conclude, on the analyzed programs, that there is a relation between high term entropy and context-coverage of an entity and its fault proneness.

E. Discussion

We now discuss some design choices we adopted during the execution of the case studies aiming at clarifying their rationale.

1) LSI subspace dimension: The choice of LSI subspace is critical. Unfortunately, there is not any systematic way to identify the optimal subspace dimension. However, it was observed that in the application of LSI to software artifacts repository for recovering traceability links between artifacts good results can be achieved setting $100 \leq k \leq 200$ [35], [36]. Therefore following such a heuristic approch we set the LSI subspace dimension equal to 100.

2) Java Parser: We developed our own Java parser, using a Java v1.5 grammar, to extract identifiers and comments from source code. Our parser is robust and fast (less than two minutes to parse any version of the studied programs, in average) but when applied, few files could not be parsed. Unparsed files include those developed on earlier versions of both ArgoUML and Rhino because of the incompatibility between the different versions of Java grammar.

3) Statistical Computations: All statistical computations were performed in R[7]. The computations took

[7]http://www.r-project.org/

Table VI
ArgoUML v0.16 confusion matrix.

ArgoUML	numHEHCC $\geq$ 2	numHEHCC = 1	Total
Fault prone	198	183	381
Fault free	511	466	977
Total	709	649	1358
p-value = 0.9598			
Odds ratio = 0.9866863			

Table VII
Rhino v1.4R3 confusion matrix.

Rhino	numHEHCC $\geq$ 2	numHEHCC = 1	Total
Fault prone	3	3	6
Fault free	75	97	172
Total	78	100	178
p-value = 1			
Odds ratio = 1.293333			

Table VIII
Odds change due to LOC ($numHEHCC$=1) and $numHEHCC$(LOC=10) for ArgoUML v0.16 and Rhino v1.4R3.

Changing variable	Δ	Odds change ArgoUML	Odds change Rhino
LOC	1	1.007448705	1.003657673
	10	1.077034036	1.037184676
	50	1.449262781	1.200274163
numHEHCC	1	1.074742395	1.270879652
	2	1.155071215	1.61513509
	10	2.056097976	10.99117854
	50	36.74675785	160406.2598

about one day for both programs, where the most expensive part of the computation in terms of time and resources was the calculation of the similarity matrix. We believe that neither extensibility nor scalability are issues: this study explains the fault phenomenon and is not meant to be performed on-line during normal maintenance activities. In the course of our evaluation, we realized that the statistical tool R yields different results when used in different software/hardware platforms. We computed the results of our analysis on R on Windows Vista/Intel, Mac OS X (v10.5.8)/Intel, and RedHat/Opteron, and we observed some differences. All results provided in this paper have been computed with R v2.10.1 on an Intel computer running Mac OS. We warn the community of using R and possibly other statistical packages on different platforms because their results may not be comparable.

4) Object-oriented Metrics: We studied the relation between our novel metric, based on term entropy and context coverage, and LOC, which is among the best indicator of fault proneness [3], [32], [33] to show that our metric provides different information. We did not study the relation between our metric and other object-oriented metrics. Of particular interest are coupling metrics that could strongly relate to term entropy and context coverage. However, we argue, with the following thought-experiment, that term entropy and context coverage, on the one hand, and coupling metrics, on the other hand, characterize different information. Let us assume the source code of a working software system, with certain coupling values between classes and certain entropy and context coverage values for its terms. We give this source code to a simple obfuscator that mingles identifiers. The source code remains valid and, when compiled, results in a system strictly equivalent to the original system. Hence, the coupling values between classes did not change. Yet, the term entropy and context coverage values most likely changed.

F. Threats to Validity

This study is a preliminary study aiming at verifying that our novel measure, based on term entropy and context coverage, for two known programs (ArgoUML v0.16 and Rhino 1.4R3), is related to the fault proneness of entities (methods and attributes) and, thus, is useful to identify fault prone entities. Consider Table VIII; for a fixed $numHEHCC$ value (one) an increase of ten for LOC will not substantially change the odds (7.7% for ArgoUML; 3.7% for Rhino[8]) while an increase of 50 increases the odds but not significantly (44.9% for ArgoUML; 20% for Rhino) in comparison to the variation of $numHEHCC$ (for a fixed value of LOC=10). For instance, in the case of ArgoUML for a fixed size of entities, one unit increase of $numHEHCC$ has almost the same odds effect than an increase of 10 LOCs. In the case of Rhino, for a fixed size of entities, one unit increase of $numHEHCC$ has more effect than an increase of 50 LOCs. Table VIII suggests that indeed an entity with ten or more terms with high entropy and context coverage dramatically change the odds and, thus, the probability of the entities to be faulty. Intuition as well as reported evidence suggest that term entropy and context coverage are indeed useful.

Threats to *construct validity* concern the relationship between the theory and the observation. These threats in our study are due to the use of possibly incorrect fault classifications and–or incorrect term entropy and context coverage values. We use manually-validated faults that have been used in previous studies [29]. Yet, we cannot claim that all fault prone entities have been correctly tagged or that fault prone entities have not been missed. There is a level of subjectivity in deciding if an issue reports a fault and in assigning this fault to entities. Moreover, in the case of ArgoUML, we used the mapping of faults to classes provided in [30]. In

[8]Although the coefficient for LOC is not significant, it was taken into account for the calculation of odds.

order to map the faults to entities we compared faulty classes with their updated version in the consecutive release, and we marked as faulty those entities that were modified. However, the changes could be due to a maintenance activity other than fault fixing, such as refactoring. Our parser cannot parse some Java files due to the incompatibility between the different versions of Java grammar, but errors are less than 4.7% in the studied program and thus do not impact our results. Another threat to validity could be the use of our parser to compute the size of entities. In the computation we took into account the blank lines and comments inside method bodies. We also used a threshold to identify "dangerous" terms and compute $numHEHCC$. The choice of threshold could influence the results achieved. Nevertheless, analyses performed with other thresholds did not yield different or contrasting results.

Threats to *internal validity* concern any confounding factor that could influence our results. This kind of threats can be due to a possible level of subjectiveness caused by the manual construction of oracles and to the bias introduced by the manual classification of fault prone entities. We attempt to avoid any bias in the building of the oracle by reusing a previous independent classification [29], [30]. Also, we discussed the relation and lack thereof between term entropy and context coverage and other existing object-oriented metrics.

Threats to *external validity* concern the possibility of generalizing our results. The study is limited to two programs, ArgoUML 0.16 and Rhino 1.4R3. Results are encouraging but it pays to be cautious. Preliminary investigation on the ten ArgoUML and eleven Rhino releases show that $numHEHCC$ is complementary to LOC for fault explanation. The results of both ArgoUML and Rhino are summarized in Figure 1. Overall, although the approach is applicable to other programs, we do not know whether or not similar results would be obtained on other programs or releases. Finally, although we did not formally investigate the measures following the guidelines of measurement theory [37], we derived them from well-known definitions and relations and we plan to study their formal properties as part of our future work while addressing the threats to external validity.

V. Conclusion

In this paper, we presented a novel measure related to the identifiers used in programs. We introduced term entropy and context-coverage to measure, respectively, how rare and scattered across program entities are terms and how unrelated are the entities containing them. We provide mathematical definitions of these concepts based on terms frequency and combined them in a unique measure. We then studied empirically the measure by relating terms with high entropy and high context-coverage with the fault proneness of the entities using these terms. We used ArgoUML and Rhino as object programs because previous work provided lists of faults. The empirical study showed that there is a statistically significant relation between attributes and methods whose terms have high entropy and high context-coverage, on the one hand, and their fault proneness, on the other hand. It also showed that, albeit indirectly, the measures of entropy and context coverage are useful to assess the quality of terms and identifiers.

Future work includes empirical studies of the relations between high entropy and context coverage with other evolution phenomena, such change proneness or participation to occurrences of anti-patterns and–or design patterns. It also includes using the measures to provide hints to the developers on the best choice of identifiers while programming. We also plan to relate *numHEHCC* and other term entropy and context coverage derived metrics with a larger suite of object-oriented metrics and study interaction between OO metrics and metric proposed in this work.

Acknowledgements. This work has been partly funded by the NSERC Research Chairs in Software Change and Evolution and in Software Patterns and Patterns of Software as well as Antoniol's and Guéhéneuc's individual NSERC Discovery Grants and Arnaoudova's FQRNT Ph.D. scholarship.

References

[1] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[2] M. Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system," *IEEE Trans. on Software Engineering*, vol. 26, no. 8, pp. 786–796, August 2000.

[3] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction." *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

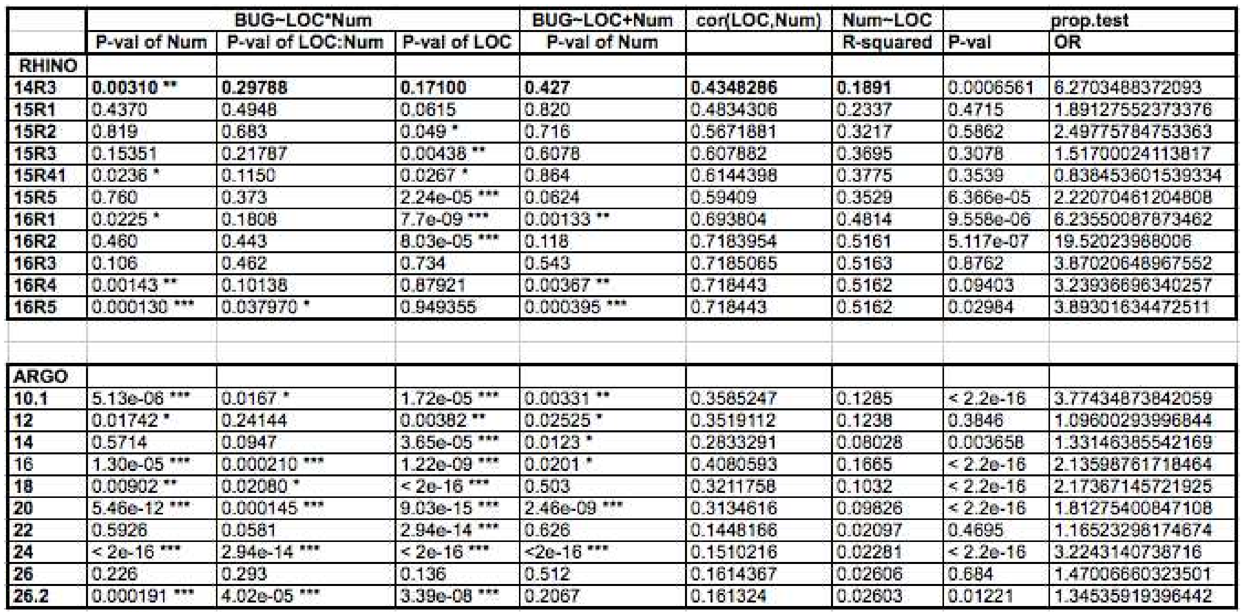
[4] T. Khoshgoftaar., B. Allen, K. Kalaichelvan, and N. Goel, "Early quality prediction: a case study in telecommunications," *IEEE Software*, vol. 13, no. 1, pp. 65–71, 1996.

[5] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, May 2007.

[6] B. Caprile and P. Tonella, "Restructuring program identifier names," in *Proceedings of 16th IEEE International Conference on Software Maintenance*. San Jose, California, USA: IEEE CS Press, 2000, pp. 97–107.

[7] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, 2006.

[8] S. Haiduc and A. Marcus, "On the use of domain terms in source code," in *Proceedings of 16th IEEE International Conference on Program Comprehension*. Amsterdam, the Netherlands: IEEE CS Press, 2008, pp. 113–122.



	BUG~LOC*Num			BUG~LOC+Num	cor(LOC,Num)	Num~LOC	prop.test	
	P-val of Num	P-val of LOC:Num	P-val of LOC	P-val of Num		R-squared	P-val	OR
RHINO								
14R3	**0.00310 ****	**0.29788**	**0.17100**	**0.427**	**0.4348286**	**0.1891**	0.0006561	6.2703488372093
15R1	0.4370	0.4948	0.0615	0.820	0.4834306	0.2337	0.4715	1.89127552373376
15R2	0.819	0.683	0.049 *	0.716	0.5671881	0.3217	0.5862	2.49775784753363
15R3	0.15351	0.21787	0.00438 **	0.6078	0.607882	0.3695	0.3078	1.51700024113817
15R41	0.0236 *	0.1150	0.0267 *	0.864	0.6144398	0.3775	0.3539	0.838453601539334
15R5	0.760	0.373	2.24e-05 ***	0.0624	0.59409	0.3529	6.366e-05	2.22070461204808
16R1	0.0225 *	0.1808	7.7e-09 ***	0.00133 **	0.693804	0.4814	9.558e-06	6.23550087873462
16R2	0.460	0.443	8.03e-05 ***	0.118	0.7183954	0.5161	5.117e-07	19.52023988006
16R3	0.106	0.462	0.734	0.543	0.7185065	0.5163	0.8762	3.87020648967552
16R4	0.00143 **	0.10138	0.87921	0.00367 **	0.718443	0.5162	0.09403	3.23936696340257
16R5	0.000130 ***	0.037970 *	0.949355	0.000395 ***	0.718443	0.5162	0.02984	3.89301634472511
ARGO								
10.1	5.13e-06 ***	0.0167 *	1.72e-05 ***	0.00331 **	0.3585247	0.1285	< 2.2e-16	3.77434873842059
12	0.01742 *	0.24144	0.00382 **	0.02525 *	0.3519112	0.1238	0.3846	1.09600293996844
14	0.5714	0.0947	3.65e-05 ***	0.0123 *	0.2833291	0.08028	0.003658	1.33146385542169
16	1.30e-05 ***	0.000210 ***	1.22e-09 ***	0.0201 *	0.4080593	0.1665	< 2.2e-16	2.13598761718464
18	0.00902 **	0.02080 *	< 2e-16 ***	0.503	0.3211758	0.1032	< 2.2e-16	2.17367145721925
20	5.46e-12 ***	0.000145 ***	9.03e-15 ***	2.46e-09 ***	0.3134616	0.09826	< 2.2e-16	1.81275400847108
22	0.5926	0.0581	2.94e-14 ***	0.626	0.1448166	0.02097	0.4695	1.16523298174674
24	< 2e-16 ***	2.94e-14 ***	< 2e-16 ***	<2e-16 ***	0.1510216	0.02281	< 2.2e-16	3.2243140738716
26	0.226	0.293	0.136	0.512	0.1614367	0.02606	0.684	1.47006660323501
26.2	0.000191 ***	4.02e-05 ***	3.39e-08 ***	0.2067	0.161324	0.02603	0.01221	1.34535919396442

Figure 1. Summary of all results for different versions of ArgoUML and Rhino.

[9] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Relating identifier naming flaws and code quality: An empirical study," in *Proceedings of the 16th Working Conference on Reverse Engineering*. Lille, France: IEEE CS Press, 2009, pp. 31–35.

[10] D. Lawrie, H. Feild, and D. Binkley, "Syntactic identifier conciseness and consistency," in *Sixth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2006) 27-29 September 2006 Philadelphia Pennsylvania USA*.

[11] H. M. Olague, L. H. Etzkorn, and G. W. Cox, "An entropy-based approach to assessing object-oriented software maintainability and degradation - a method and case study," in *Software Engineering Research and Practice*, 2006, pp. 442–452.

[12] Y. Yu, T. Li, N. Zhao, and F. Dai, "An approach to measuring the component cohesion based on structure entropy," in *Proceedings of the 2nd International Symposium on Intelligent Information Technology Application*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 697–700.

[13] G. Snider, "Measuring the entropy of large software systems," HP Laboratories Palo Alto, Tech. Rep., 2001.

[14] L. H. Etzkorn, S. Gholston, and W. E. Hughes, "A semantic entropy metric," *Journal of Software Maintenance: Research and Practice*, vol. 14, no. 5, pp. 293–310, 2002.

[15] S. Patel, W. Chu, and R. Baxter, "A measure for composite module cohesion," in *Proceedings of 14th International Conference on Software Engineering*. Melbourne, Australia: ACM Press, 1992, pp. 38–48.

[16] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, 2008.

[17] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[18] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *Proceedings of 22nd IEEE International Conference on Software Maintenance*. Philadelphia Pennsylvania USA: IEEE CS Press, 2006, pp. 469 – 478.

[19] D. Binkley, H. Feild, D. Lawrie, and M. Pighin, "Software fault prediction using language processing," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 99–110.

[20] K. E. Emam, S. Benlarbi, N. Goel, and S. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Trans. on Software Engineering*, vol. 27, no. 7, pp. 630–650, July 2001.

[21] A. E. Hassan, "Predicting faults using the complexity of code changes," in *ICSE*. Vancouver, Canada: IEEE Press, 2009, pp. 78–88.

[22] G. Butler, P. Grogono, R. Shinghal, and I. Tjandra, "Retrieving information from data flow diagrams," in *Proceedings of 2nd Working Conference on Reverse Engineering*. Toronto, Ontario, Canada: IEEE CS Press, 1995, pp. 84–93.

[23] A. Takang, P. Grubb, and R. Macredie, "The effects of comments and identifier names on program comprehensibility: an experiential study," *Journal of Program Languages*, vol. 4, no. 3, pp. 143–167, 1996.

[24] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *Proceedings of 14th IEEE International Conference on Program Comprehension*. Athens, Greece: IEEE CS Press, 2006, pp. 3–12.

[25] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To CamelCase or Under score," in *Proceedings of 17th IEEE International Conference on Program Comprehension*. Vancouver, British Columbia, Canada: IEEE CS Press, 2009.

[26] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York, NY 10158-0012: Wiley Series in Telecommunications John Wiley & Sons., 1992.

[27] V. Basili, G. Caldiera, and D. H. Rombach, *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[28] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. D. Penta, "An empirical study on the evolution of design patterns," in *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIG-*

SOFT symposium on The foundations of software engineering. New York NY USA: ACM Press, 2007, pp. 385–394.

[29] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho, “Do crosscutting concerns cause defects?” *IEEE Transaction on Software Engineering*, vol. 34, no. 4, pp. 497–515, 2008.

[30] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, “An empirical study on the maintenance of source code clones,” *Empirical Software Engineering*, vol. 15, no. 1, pp. 1–34, Jan 2010.

[31] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Trans. of Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[32] L. Briand, J. Wüst, J. W. Daly, and V. Porter, “Exploring the relationships between design measures and software quality in object-oriented systems,” *Journal of Systems and Software*, vol. 51, pp. 245–273, 2000.

[33] Y. Zhou and H. Leung, “Empirical analysis of object-oriented design metrics for predicting high and low severity faults,” *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, 2006.

[34] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[35] A. Marcus and J. I. Maletic, “Recovering documentation-to-source-code traceability links using latent semantic indexing,” in *Proceedings of 25th International Conference on Software Engineering*. Portland, Oregon, USA: IEEE CS Press, 2003, pp. 125–135.

[36] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Recovering traceability links in software artefact management systems using information retrieval methods,” *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, 2007.

[37] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach (2nd Edition)*. Boston: Thomson Computer Press, 1997.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873