Physical and Conceptual Identifier Dispersion: Measures and Relation to 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.**

***Index Terms*—Source Code Identifiers; Program Comprehension; Fault Models; Information Retrieval; Entropy.**

I. Introduction

The identification of faulty source code entities is generally based on product metrics, such as size, cohesion, or coupling (*e.g.*, [1], [2], [3]), as well as process-oriented metrics, such as number of file changes (*e.g.*, [4]). However, we believe that fault proneness is a complex phenomenon hardly captured solely by structural characteristics of code entities. Indeed, several studies showed that identifiers impact program comprehension (*e.g.*, [5], [6], [7], [8]) and code quality (*e.g.*, [3], [9], [10]). We concur with Deißenböck and Pizka's observation that proper identifiers improve quality and that identifiers should be used consistently [6]. Source code with high quality identifiers, carefully chosen and consistently used in their contexts, likely ease program comprehension and support developers in building consistent and coherent conceptual models.

In this paper, we present, to the best of our knowledge, the first empirical study on the relation between the terms composing identifiers and fault proneness. Terms are identifier components (*e.g.*, *get* and *String* are the terms composing *getString*). We conjecture that a term should carry a single meaning in the context where it is used. Terms referring to different concepts or used inconsistently in different contexts may increase the program comprehension burden by creating a mismatch between the developers' cognitive model and the intended meaning of the term, thus ultimately increasing the risk of fault proneness.

Context definition is left intentionally blurred as it may stand for a code region (*e.g.*, method or attribute, class or component) as well as for the developers' knowledge and mental models of any given code region or artifact. Misunderstanding impacts the cognitive process and is difficult to quantify. We use linguistic information extracted from a given code region as a surrogate of the developers' mental models. More precisely, linguistic information extracted from methods and attributes is used to quantify *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 exploits Information Retrieval (IR) [11] techniques to measures the "conceptual" dispersion of the entities in which the term appears. The higher the context coverage of a term, the more unrelated are the linguistic information and, possibly, the concepts of the corresponding entities.

To support our conjecture that terms with high entropy and high context coverage may help to locate fault-prone methods and attributes we report a preliminary case study on two open-source programs: ArgoUML and Rhino. We show that there is a statistically significant relation between *term entropy*, *context coverage*, and odds ratio that an entity is fault prone. Thus, the contributions of this paper can be summarized as follows:

- a novel measure characterizing the "physical" (entropy) and "conceptual" (context) dispersions of terms;
- a preliminary empirical study showing the relation between entropy and context coverage with fault proneness.

Structure of the paper. 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 IR methods, fault proneness, and the quality of source code identifiers. In the following sections, we discuss the related literature.

A. Entropy and IR-based Metrics

Several metrics based on entropy exist. Olague *et al.* [12] 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.* [13] combined entropy with component-dependency graphs to measure component cohesion. Entropy was also used by Snider [14] in an empirical study to measure the structural quality of C code.

IR methods have also been used to define new measures of source code quality. Etzkorn *et al.* [15] presented a new measure for object-oriented programs that examines the implementation domain content of a class to measure its complexity. The content of methods in a class has also been exploited to measure the conceptual cohesion of a class [2], [3], [16]. In particular, IR methods were used 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 [9] defined new coupling metrics based on semantic similarity. Binkley *et al.* [17] 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. A case study also 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 [18], and fault proneness. For example, Gyimóthy *et al.* [1] 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.* [4] 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.* [19] showed that the previous correlations between object-oriented metrics and fault-proneness are mostly due to the correlations between the metrics and size. Hassan [20] 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 fault predictor than other previous predictors.

C. Identifiers and Program Comprehension

The problem of word choice variability in human system communication was studied in [21]; in collaborative environments, the probability of having two developers use the same identifiers for different entities is between 7% and 18% [21]. Thus, naming conventions are crucial for improving the source code understandability. Butler *et al.* [10] 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.* [5], 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 by Lawrie *et al.* [22]. Recently, Binkley *et al.* [8] performed an empirical study of the impact of identifier style on code readability and showed that Camel-case identifiers allow more accurate answers.

III. Entropy and Contexts

To calculate term entropy and context coverage, we extract identifiers found in class attributes and methods (*e.g.*, names of variables and methods, method parameters). We split identifiers using a Camel-case splitter to build the term dictionary (*e.g.*, *getText* is split into *get* and *text*). We filter terms whose length is less than two characters and summarize the linguistic information in a *term-by-entity* matrix. 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.

A. Term Entropy

Entropy of a discrete random variable measures the amount of uncertainty [23]. To compute the term entropy, we consider terms as random variables with some associated probability distributions. Given a term, its entries in the term-by-entity matrix are the counts of term occurrences and thus by normalizing over the sum of its row entries a probability distribution for each term is obtained. 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.

B. Term Context Coverage

The context coverage of term t_k (where $k = 1, 2, \ldots, m$) is computed as the average textual similarity of the contexts (here entities) containing t_k:

$$CC(t_k) = 1 - \frac{1}{N} \sum_{e_i, e_j \in C} sim(e_i, e_j)$$

where $C = \{e_l | a_{k,p} \neq 0\}$ is the set of all entities in which term t_k occurs, N is the number of compared entities, and

Table I
PROGRAMS USED IN THE EXPERIMENTATION.

Program	Version	LOC	# Files	# Entities	# Terms
ArgoUML	0.16	97,946	1,124	12,423	2,517
Rhino	1.4R3	18,163	75	1,624	949

We consider as entities both methods and attributes.

$sim(e_i, e_j)$ represents the textual similarity between entities e_i and e_j computed using Latent Semantic Indexing [24], an advanced IR method, with a subspace dimension fixed to 100. 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.

C. Aggregate Metric

In this preliminary investigation, we use the variable *numHEHCC*, number of high entropy and high context coverage, associated with all entities and defined as:

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

where a_{ij} is the frequency in the term-by-entity matrix of term t_i in entity E_j ($j = 1, 2, \ldots, n$) and $\psi()$ is a function returning one if the passed Boolean value is true; 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. This aggregate metric is used throughout the following case study to compute correlation, build linear, logistic models, and contingency tables.

IV. CASE STUDY

The context of the study are two open-source programs: Rhino, a JavaScript/ECMAScript interpreter and compiler, and ArgoUML, a UML modeling CASE tool. Table I reports some statistics on the two programs, *e.g.*, number of entities and identifier components (terms). We selected the version of ArgoUML that has the maximum number of faulty entities (v0.16) and one of the versions of Rhino with low number of bugs (v1.4R3). The choice of this two programs is twofold: they were previously used in other case studies (*e.g.*, [25]); and a mapping between faults and entities (attributes and methods) is available [25], [26].

A. Research Questions

It is well known that size is one of the best fault predictors [1]. Thus, we first verify that $numHEHCC$ is somehow at least partially complementary to size. Second, we believe that it is important to understand if entropy and context coverage help to locate entities likely to be fault prone when changed. Therefore, the case study is designed to answer the following research questions:

- **RQ$_1$ – Metric Relevance:** Does term entropy and context coverage capture characteristics different from size?
- **RQ$_2$ – Relation to Faults:** Do term entropy and context coverage help to explain the presence of faults in an entity?

In **RQ$_1$**, our goal is only to validate that the newly proposed metric brings information complementary to the one captured by LOC. However, the real focus of the paper is to answer **RQ$_2$**: evaluating whether the proposed metric, alone, can be used to explain fault proneness. To answer **RQ$_2$**, we assess a risk in terms of odds ratio, and thus, do not claim any causation.

B. Analysis Method

To statistically analyze **RQ$_1$**, we compute the correlation between the size measured in LOC and $numHEHCC$. Then, we estimate the linear regression models between LOC (independent variable) and $numHEHCC$ (dependent variable). Finally, as an alternative to the Analysis Of Variance (ANOVA) [27] for dichotomous variables, we build logistic regression models between fault proneness (explained variable) and LOC and the proposed new metric (explanatory variables). Thus, we formulate the null hypothesis:

H_{0_1}: $numHEHCC$ does not capture a dimension different from LOC.

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.

For **RQ$_2$**, we formulate the following null hypothesis:

H_{0_2}: There is no relation between $numHEHCC$ and fault proneness.

We use a prop-test (Pearson's chi-squared test) [27] to test the null hypothesis. If $numHEHCC$ is important to explain fault proneness, then the prop-test should reject the null hypothesis with a statistically significant p-value (95% significance level).

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) [27] indicating the likelihood of faulty entities to contain terms with high entropy and context coverage.

The term context coverage distribution is skewed toward high values. For this reason, we use the 10% highest values of term context coverage to define a threshold identifying the high context coverage property. 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). The thresholds values chosen for entropy and context coverage in our study are 4.17 and 0.79 for Rhino, and 4.9, and 0.83 for ArgoUML respectively.

C. Results

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

RQ$_1$ – Metric Relevance. Table II reports the results of Pearson's product-moment correlation for both Rhino and

Table II
CORRELATION TESTS.

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

Table III
LINEAR REGRESSION MODELS.

	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	$\prec 2e-16$

ArgoUML. As expected, some correlation exists between LOC and $numHEHCC$. 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 III). 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 approximation of the unexplained variance of the model. Thus, Table III supports the conjecture that LOC does not substantially explain $numHEHCC$. Correlation and linear regression models can be considered as further verification that LOC and $numHEHCC$ help to explain different dimensions of fault proneness.

The relevance of $numHEHCC$ in explaining faults is further supported by logistic regression models. Table IV 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 coefficient is not statistically significant as well as the interaction term, $LOC : numHEHCC$. This lack of significance is limited to Rhino: for ArgoUML, both LOC and the interaction term are statistically significant. In both models, $M_{ArgoUML}$ and M_{Rhino}, the LOC coefficient is, at least, one order of magnitude smaller than the $numHEHCC$ coefficient. This difference 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, conservatively, we can say that both LOC and $numHEHCC$ have the same impact in terms of probability. In other words, the models in Table IV support the conjecture that $numHEHCC$ helps to explain fault proneness. Based on the reported results, we can conclude that although some correlation exists between LOC and $numHEHCC$, statistical evidence allows us to reject, on ArgoUML and Rhino, the null hypothesis H_{0_1}.

RQ$_2$ – Relation to Faults. To answer **RQ$_2$**, we perform prop-tests and test the null hypothesis H_{0_2}. Indeed, (i) if prop-tests reveal that $numHEHCC$ divides 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

Table IV
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 V
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			

with positive $numHEHCC$, it will be possible to identify those terms leading to high entropy and high context coverage, identifying also their contexts and performing refactoring actions to reduce entropy and high context coverage.

Tables V and VI 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 the odds ratio of entities containing two or more terms with high entropy and high context coverage with those entities which only contain one such term. The results are not statistically significant (with an OR close to one). These results suggest that the difference between fault-prone entities and others is between not containing high entropy and high context coverage terms and containing one or more such terms.

We also analyzed the odds change for LOC and $numHEHCC$. For example, in the case of ArgoUML for a fixed LOC, 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.

D. Validity evaluation

Threats to validity of the results can be summarized as follows: we use manually-validated faults that have been used in previous studies but we cannot claim that all fault prone entities have been correctly tagged or that no fault prone entity has been missed. We used a threshold to identify terms with high entropy and context coverage and compute $numHEHCC$, which could influence the results. As future work, we plan to compute numHEHCC using different thresholds to corroborate our findings. The study is limited to two programs, ArgoUML 0.16 and Rhino 1.4R3. Results are encouraging but replications are needed to increase the generalizability of the results achieved.

Table VI
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			

V. CONCLUSION

In this paper we introduced *term entropy* and *context coverage* to measure, respectively, how scattered terms are across program entities and how unrelated are the methods and attributes containing these terms. We provided mathematical definitions of term entropy and context coverage and reported a preliminary case study on the effect of terms with high entropy and high context coverage on fault proneness of methods and attributes in ArgoUML and Rhino.

The results achieved support the conjecture that term entropy and context coverage only partially correlate with size. We also showed that the number of high entropy and high context coverage terms contained in a method or attribute helps to explain the probability of it being faulty. Furthermore, the odds ratio of being faulty for a method (or attribute) containing one or more terms with high entropy and high context coverage is six and two for Rhino and ArgoUML, respectively: if a Rhino (ArgoUML) method contains an identifier with a term having high entropy and high context its probability of being faulty is six (two) times higher.

Future work includes replicating the same study at class-level to relate *numHEHCC* with a larger suite of object-oriented metrics (such as CK metric suite) and studying terms that should be considered for refactoring. We also plan to analyze how the evolution of source code lexicon [28] impacts the term entropy and context coverage.

ACKNOWLEDGEMENTS

This work was partly supported by the NSERC Research Chairs in Software Change and Evolution and in Software Patterns and Patterns of Software; Antoniol's and Guéhéneuc's NSERC Discovery Grants; and, Arnaoudova's FQRNT Ph.D. scholarship.

REFERENCES

[1] 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.

[2] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides, "Modelling class cohesion as mixtures of latent topics," in *Proceedings of 25th IEEE International Conference on Software Maintenance*. Edmonton, Canada: IEEE CS Press, 2009, pp. 233–242.

[3] 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.

[4] 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.

[5] 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.

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

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

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

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

[10] 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*. IEEE CS Press, 2009, pp. 31–35.

[11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[12] 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.

[13] 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*. IEEE Computer Society, 2008, pp. 697–700.

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

[15] 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.

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

[17] 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*. IEEE Computer Society, 2007, pp. 99–110.

[18] 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.

[19] 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.

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

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

[22] 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*. IEEE CS Press, 2006, pp. 3–12.

[23] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 1991.

[24] 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.

[25] 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.

[26] 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.

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

[28] S. L. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the evolution of the source code vocabulary," in *Proceedings of 12th European Conference on Software Maintenance and Reengineering*. Kaiserslauter, Germany: IEEE CS Press, 2008, pp. 189–198.