A Method for Assessing Class Change Proneness

Elvira-Maria Arvanitou
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
e.m.arvanitou@rug.nl

Apostolos Ampatzoglou
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
a.ampatzoglou@rug.nl

Alexander Chatzigeorgiou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
achat@uom.gr

Paris Avgeriou
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
paris@cs.rug.nl

ABSTRACT

Change proneness is a quality characteristic of software artifacts that represents their probability to change in the future due to: (a) evolving requirements, (b) bug fixing, or (c) ripple effects. In the literature, change proneness has been associated with many negative consequences along software evolution. For example, artifacts that are change-prone tend to produce more defects, and accumulate more technical debt. Therefore, identifying and monitoring modules of the system that are change-prone is of paramount importance. Assessing change proneness requires information from two sources: (a) the history of changes in the artifact as a proxy of how frequently the artifact itself is changing, and (b) the source code structure that affects the probability of a change being propagated among artifacts. In this paper, we propose a method for assessing the change proneness of classes based on the two aforementioned information sources. To validate the proposed approach, we performed a case study on five open-source projects. Specifically, we compared the accuracy of the proposed approach to the use of other software metrics and change history to assess change proneness, based on the 1061-1998 IEEE Standard on Software Measurement. The results of the case study suggest that the proposed method is the most accurate and reliable assessor of change proneness. The high accuracy of the method suggests that the method and accompanying tool can effectively aid practitioners during software maintenance and evolution.

CCS CONCEPTS

• **Software and its engineering** → Software creation and management → {Software development techniques → *Object-oriented development*, Software verification and validation → *Empirical software validation}*

EASE'17, June 15-16, 2017, Karlskrona, Sweden
© 2017 Association for Computing Machinery
ACM ISBN 978-1-4503-4804-1/17/06…$15.00
http://dx.doi.org/10.1145/3084226.3084239

KEYWORDS

Change proneness; Software quality; Metrics; Case study

1. INTRODUCTION

Change proneness is the susceptibility of software artifacts to change, without differentiating between types of change (e.g., new requirements, debugging activities, and changes that propagate from changes in other classes) [17]. In the literature, change proneness has been extensively studied as a software quality characteristic from various perspectives. We present three such perspectives as examples of using change proneness. First, in the ***software maintenance and evolution literature***, change proneness has been associated with many undesired consequences. For instance, a class that changes very frequently is more error/defect-prone and is more difficult to maintain along software evolution [16]. Second, in the ***technical debt literature,*** change proneness is considered as a factor in calculating interest probability. In other words, it is claimed that change-prone classes are more probable to accumulate interest than less change-prone ones, since interest manifests only during maintenance activities [5]. Third, in the ***design pattern literature,*** change proneness has been examined as an indicator for the necessity of applying patterns. More specifically, the pattern community claims that placing a pattern in a design spot that is not changing frequently (i.e., within a group of classes that are not change-prone) might lead to unnecessary design complexity (i.e., a simpler solution would more preferable than using the pattern) [9].

Based on the above, it becomes evident that change proneness is a useful indicator for various use cases. Therefore, it is of paramount importance to measure change proneness of software systems, as accurately as possible, and further monitor it, since it changes over time. According to a recent mapping study on software design-time quality attributes, change proneness is assessed either by considering: (a) the ***history of changes*** of artifacts, which can be captured by measures such as: frequency of changes along evolution, and extent of change (such as number of lines added / deleted / modified, etc.); or (b) the ***structural characteristics*** of software, such as coupling, size and complexity [7]. However, existing methods in the state-of-the-art are not very accurate. A possible explanation for their low accuracy is the fact that they do not combine the two aforementioned information sources. To improve the accuracy of assessing change proneness, we propose investigating: (a) the probability of an artifact per se to undergo changes, which needs to be considered as one source of change (e.g., a modification of the requirements, a bug fixing request, etc.), and (b) the dependencies to other artifacts as an additional one, in the sense that changes can be propagated from other artifacts, as well. The former aspect can be estimated by analyzing the source code change history, whereas the latter can be estimated by structural analysis.

In this study, we propose a method for assessing change proneness of software artifacts, based on the two aforementioned aspects. In particular, we build upon an existing method introduced by the third author [24] and enhance it with additional parameters. The original method calculates class change proneness by synthesizing: (a) the internal probability of the class to undergo changes (***internal change probability***); and (b) the probability of the class to receive changes due to ripple effects—i.e., changes that propagate from one class to the other due to structural dependencies (***propagation factor***). However, the original method does not support calculating these two factors, leading to the use of constants as internal change probabilities and propagation factors to all system classes. Nevertheless, these parameters are not expected to be uniform for all classes. Thus, the contribution of this study is the enhancement of the method by efficiently calculating these two factors (more details are provided in Section 3). In particular we propose the use of:

- ***Ripple Effect Measure (REM)*** as a proxy of the propagation factor. We had previously introduced REM and validated it both theoretically and empirically as a valid instability measure [6]; and
- ***Percentage of Commits in which a Class has Changed*** (PCCC) as a proxy of internal change probability. This metric has been adapted from the *Commits per File* metric that has been introduced by Zhang et al. as an indicator of complexity, in the sense that a frequently changed file is expected to be more complex [29].

As an outcome, the proposed method calculates a metric, namely ***Change Proneness Measure (CPM)***. To evaluate the validity of the proposed method, and particularly the proposed CPM as an assessor of change proneness, we compare its accuracy with the accuracy of using: (a) existing coupling metrics, (b) only historical data, and (c) the original method, as assessors of change proneness. The reasons for selecting to compare the proposed metric (CPM) with the aforementioned alternative assessors are discussed in detail in the case study design section (see Section 4). The evaluation is performed in an empirical manner, based on the guidelines provided by the IEEE Standard on Software Measurement [1].

The rest of the paper is organized as follows: In Section 2 we discuss related work, whereas in Section 3 we present the proposed method for quantifying change proneness. Section 4 presents the design of the case study, whereas its results are presented in Section 5. In Section 6 we discuss the main findings of the validation. Finally, Sections 7 and 8 present threats to validity and conclude the paper, respectively.

2. RELATED WORK

In this section, we present studies that are related to the quantification of change proneness. Han et al. have proposed a metric for assessing change proneness of classes, based on UML class diagrams. The approach was based on studying the behavioral dependencies of classes [14]. The proposed measure is different from our work in the sense that it is based solely on structural information and completely disregards the change history of classes. In a similar context, Lu et al. performed a meta-analysis to investigate the ability of object-oriented metrics to assess change proneness [20]. The results of the study suggested that size metrics are the optimum assessors of change proneness, followed by cohesion and coupling metrics [20]. The outcome of this case study can be considered as expected in the sense that larger classes are by nature more probable to change in a next version of the systems, since they are probably related to more requirements and are probably receiving more ripple effects from other classes.

Furthermore, Koru and Tian [18] focused only on highly change-prone classes and classes with high values of size, coupling, cohesion, and complexity measures. The results of their study pointed out that the most change-prone classes were not those with the highest metric scores (although they have been highly ranked) [18]. This outcome verifies our intuition that structural metrics alone cannot form an accurate assessor of change proneness. Finally, Schwanke et al. focused only on bug-related change frequency (i.e., fault proneness), and tried to identify assessors for it [23]. The results suggested that fan-out (i.e., other artifacts to which a module depends on) is a fairly good assessor of change proneness [23], further highlighting the appropriateness of coupling metrics as assessor of change proneness.

Finally, in the early '80s Yau and Collofello proposed some measures for design and source code stability. Both measures were considering the probability of an actual change to occur, the complexity of the changed module, the scope of the used variables, and the relationships between modules [27][28]. However, the specific studies (they are among the first ones that discuss software instability as a quality attribute) are kept in a rather abstract level, without proposing specific metrics or tools for quantifying them. In a more recent study (2007), Black proposed an approach for calculating a complexity-weighted total variable propagation definition for a module, based on the model proposed by Yau and Collofello. The approach calculates complexity metrics, coupling metrics, and control flow metrics, and their combination provides an estimate of change proneness [10].

As a summary of related work we have identified the following limitations: (a) rely on a single source of information (i.e. structural or historical data), (b) the accuracy of the metric-based approaches is rather low, and (c) most of existing approaches lack applicability in the sense that they do not provide tools.. To this end, in this work, we propose a method that achieves higher accuracy than metric-based approaches and we accompany our method with a tool, so as to enhance its applicability.

3. PROPOSED METHOD

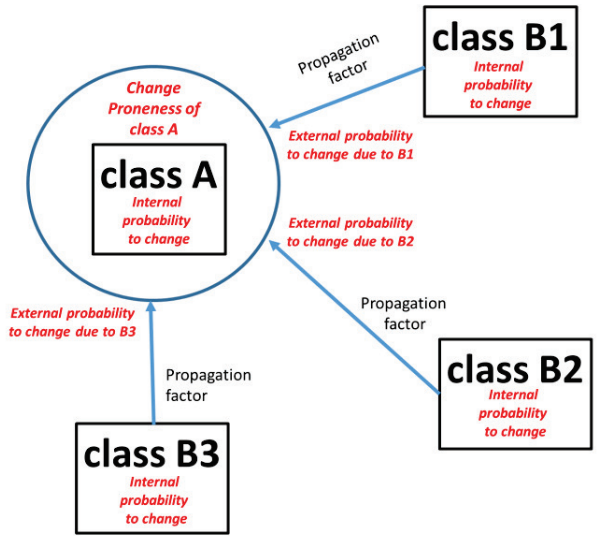
Assessing whether a given software module will change in a future version is an ambitious goal, because any actual decision to perform changes to a class is subject to numerous factors. The probability that a certain class will change in the future is affected not only by the likelihood of modifying the class itself but also by possible changes in other classes that might propagate to it. These so-called ripple effects [15] causing change propagation are the result of dependencies [24]) among classes through which a change in a class (such as the change in a method signature – i.e., method name, types of parameters and return type) can affect other classes enforcing them to be modified.

The method that we employ in this study [24] analyzes the dependencies in which each class is involved and calculates class change proneness. The calculation of change proneness is based on two main factors: the ***internal probability to change*** (i.e., the probability of a class to change due to changes in requirements, bug fixing, etc.) and the ***external probability to change***, which corresponds to the probability of a class to change due to ripple effects (i.e., changes propagating from other classes). Each dependency carries a different probability of propagating changes (***propagation factor***), which is used in the calculation of the corresponding external probability to change: if class `A` has a dependency to another class `B`, the external probability of `A` to change due to `B` is obtained as:

$$P(A{:}external_{B}) = P(A|B) \bullet P(B)$$

*P(A|B) is the **propagation factor** between classes B and A (i.e., the probability that a change made in class B is emitted to class A). P(B) refers to the **internal probability** of changing class B.*

To illustrate the application of our method, let's consider the example of Figure 1.



class B1
Internal probability to change
Propagation factor
Change Proneness of class A
class A
Internal probability to change
External probability to change due to B1
External probability to change due to B2
Propagation factor
class B2
Internal probability to change
External probability to change due to B3
Propagation factor
class B3
Internal probability to change

Fig. 1. *Rationale of calculation of Change Proneness Metric (CPM)*

The calculation of change proneness for class A (see Figure 1) should take into account:

- *Internal probability* to change of A — P(A)
- *External probability* to change due to ripple effects from B1 — $P(A{:}external_{B1})$. This value represents the probability of A to change because of its dependency to B1. Thus it depends both on the internal probability of B1 to change (as a trigger to the ripple effect) and the possibility of these changes to propagate along the B1→A dependency (as a proxy of the probability that the change will be emitted from `B1` to `A`).
- *External probability* to change due to ripple effects from B2 — $P(A{:}external_{B2})$.
- *External probability* to change due to ripple effects from B3 — $P(A{:}external_{B3})$.

Since a class might be involved in several dependencies (e.g., class `A` in Figure 1) and because even one change in dependent classes (in either `B1` or `B2`, or `B3` for the example of Figure 1) will be a reason for changing that class, ***the change proneness measure (CPM) is calculated as the joint probability of all events that can cause a change to a class***. Thus, in the aforementioned example (see Fig. 1), class `A` might change due to the following events (whose probabilities to occur we join): (a) change in `A itself`, (b) a ripple effect from `B1`, (c) a ripple effect from `B2`, or (d) a ripple effect from `B3`:

$$\textbf{CPM(A)} = \textit{Joint Probability}\ \{P(A), P(A{:}external_{B1}), P(A{:}external_{B2}), P(A{:}external_{B3})\}$$

The accuracy in estimating CPM depends on the precision of the estimates of the internal probability of change for each class and the propagation factor for each dependency.

Regarding the ***internal probability of change*** we use the percentage of commits in which a class has changed. In particular, we study all commits between two successive versions of a system and count in how many of those, each class has changed. This percentage is calculated for all past pairs of versions, and the obtained average is used as the internal probability of change. We note that we preferred to use an average of change frequency among all pairs of versions, instead of the change frequency in the whole lifecycle. The benefit of this decision is that the internal probability of change is calculated over a number of commits that are in the same level of magnitude as the predicted variable (i.e., the probability to change from one version to the next one).

Concerning the ***propagation factor*** of changes among dependent classes we use the Ripple Effect Measure (REM) [6], which quantifies the probability of a change occurring in class `B` to be propagated to a dependent class `A`. REM essentially quantifies the percentage of the public interface of a class that is being accessed by a dependent class. The calculation of REM is based on dependency analysis. Such change propagations [13], are the result of certain types of changes in one class (e.g., a change in the method signature—i.e., method name, types of parameters and return type—that is invoked inside another method) that potentially emit changes to other classes. Such types of changes vary across different types of dependencies. According to van Vliet [25], there are three types of class dependencies, namely: generalization, containment, and association. We note that the aforementioned way of change propagation through class dependencies refers to designs that follow basic object-oriented design principles, i.e. encapsulation (classes do not hold public attributes). In cases that classes hold public attributes, these public attributes are also considered as a reason for change propagation, in the sense that they belong to the class public interface. REM has been empirically and theoretically evaluated as a valid assessor of the existence of the ripple effect through a case study on open-source projects [6].

4. CASE STUDY DESIGN

To empirically investigate the validity of CPM to change proneness, we performed a case study on five open source projects. We compare CPM to: (a) ***coupling metrics***, (b) ***history of changes***, and (c) the ***original method*** [24].

Coupling metrics have been considered in this study for two reasons: (a) *they represent the existence / strength of dependencies among modules*. Therefore, they are the structural metrics that can be considered as a proxy of change propagation probability; and (b) *they are reported* in four related studies ([14], [18], [20], and [23] see Section 2) *as fair assessors of change proneness*. In order to be as inclusive as possible, we selected metrics from three different metric suites (Chidamber and Kemerer [11], Li and Henry [19], and QMOOD [8]), which are well-known and tool-supported. Also, the aforementioned metric suites include both code- and design-level coupling metrics. We note that all metrics described in related work have been investigated in our study. However, in some cases there are naming mismatches (e.g., fan-out has the same definition as DCC that we use) due to the use of different quality model/metric suite for the definition of the metric. The metrics that have been used as control variables are:

- ***Coupling Between Objects (CBO)***: Number of classes to which a class is coupled [11].
- ***Response For a Class (RFC)***: Number of local methods, plus the number of methods called by local methods in the class [11].
- ***Message Passing Coupling (MPC)***: Number of send statements defined in the class [19].
- ***Data Abstraction Coupling (DAC)***: Number of abstract data types defined in the class [19].
- ***Measure of Aggregation (MOA)***: Number of data declarations of a user defined type [8].

In addition to existing coupling metrics, we have also decided to compare CPM to the estimate that is offered by only using ***change history data*** so as to judge their assessing power. Thus, we will be able to verify whether the combination of historical and structural data (as performed in CPM) works better than they do in isolation. Finally, we compare CPM to the ***original method*** (i.e. as it has been proposed in [24]) so as to demonstrate the benefit of enhancing the original method with the parameters described in Section 3.

The case study has been designed and reported according to the guidelines of Runeson et al. [22]. In this section we present: (a) the goal of the case study and the derived research questions, (b) the description of cases and units of analysis, (c) the data collection procedure, and (d) the data analysis process. Additionally, in this Section, we present the metric validation criteria.

4.1 Metric Validation Criteria

To investigate the validity of CPM and compare it with the other three assessors, we employ the properties described in the 1061 IEEE Standard for Software Quality Metrics [1]. The standard defines six metric validation criteria suggesting the statistical test that shall be used for evaluating every criterion:

- ***Correlation*** assesses the association between a quality factor and the metric under study to warrant using the metric as a substitute for the factor. The criterion is quantified by using a correlation coefficient [1].
- ***Tracking*** assesses if the values of the metric under study can follow the changes in product quality that occur during their lifecycle. The criterion is quantified by using the coefficient of rank correlation for a set of project versions [1].
- ***Consistency*** assesses whether there is consistency between the ranks of the quality factor values (over a set of software components) and the ranks of the corresponding metric values. Consistency determines whether a metric can accurately rank a set of artifacts in terms of quality. The criterion is quantified by using the coefficient of rank correlation [1].
- ***Predictability*** assesses the accuracy with which the metric under study applied at a time point t_1 is able to predict the levels of the quality factor at a time point t_2. The criterion is quantified through the standard estimation error for a regression model using as predictor the studied metric [1].
- ***Discriminative Power*** assesses if the metric under study is capable of discriminating between high-quality and low-quality components. Discriminative power can be quantified through a contingency table (see [1]) and employing Mann-Whitney and the Chi-square test for differences in probabilities. However, this type of quantification was not applicable for coupling metrics, because they cannot be recoded as categorical variables, without setting arbitrary thresholds. Therefore, we use an equivalent test for assessing discriminative power, i.e., Kruskall-Wallis test [12].
- ***Reliability*** assesses if the metric under study can fulfill all five aforementioned validation criteria, over a sufficient number of applications. This criterion can offer evidence that a metric can perform its intended function consistently. This criterion can be assessed by replicating the previously discussed tests (for each of the aforementioned criteria) to various software systems [1].

4.2 Research Objectives and Research Questions

The aim of this study, expressed through a GQM formulation, is: *to* ***analyze*** *CPM and other metrics (i.e., coupling, historical data, and the original method)* ***for the purpose of*** *comparison* ***with respect to*** *their validity to assess if a class is prone to change in the upcoming system version,* ***from the point of view of*** *researchers* ***in the context of*** *software maintenance and evolution*.

Driven by this goal and the validity criteria discussed in 1061-1998 IEEE Std. [1], two relevant research questions have been set: The first research question aims to investigate the validity of the proposed Change Proneness Measure in comparison to the other three existing metrics, with respect to the first five validity criteria (i.e. correlation, consistency, tracking, predictability and discriminative power). For the first research question, we employ a single dataset comprising all examined projects of the case study.

The second research question aims to investigate the validity in terms of reliability. Reliability is examined separately since, according to its definition, each of the other five validation criteria should be tested on different projects. In particular, for this research question we consider each software project as a different dataset and then results are cross-checked to assess the metric's reliability. The two research questions are formulated as follows:

RQ$_1$: *How does CPM compare to the other metrics, based on the criteria of the 1061-1998 IEEE Std?*

- **RQ$_{1.1}$**: How does CPM compare to the other metrics, w.r.t. correlation?
- **RQ$_{1.2}$**: How does CPM compare to the other metrics, w.r.t. consistency?
- **RQ$_{1.3}$**: How does CPM compare to the other metrics, w.r.t. tracking?
- **RQ$_{1.4}$**: How does CPM compare to the other metrics, w.r.t. their predictive power?
- **RQ$_{1.5}$**: How does CPM compare to the other metrics, w.r.t. their discriminative power?

RQ$_2$: *How does CPM compare to the other metrics, w.r.t. their reliability*?

4.3 Case and Units of Analysis

This study is an embedded multiple-case study, i.e. it studies multiple cases where each case is comprised of many units of analysis. Specifically, the cases of the study are open source projects, where classes are units of analysis. We note that an alternative to the aforementioned scenario would be the consideration of classes as cases and units of analysis and the compilation of a single dataset. However, this decision would hurt the validity of the dataset in the sense that projects with different characteristics (e.g., number of commits per versions, number of classes, etc.) would be merged in one dataset. Therefore, any reporting of results is performed at the project level. The results are aggregated to the complete dataset by using the mean function and a percentage of projects in which the results are statistically significant.

As subjects for our study, we selected to use the last five versions of five open source projects written in Java. A short description of the goals of these projects is provided below, whereas some demographics are provided in Table I. **jFlex** is a lexical analyzer generator (also known as scanner generator) for Java.. ***jUnit*** is a simple framework to write repeatable tests. ***Apache-commons-io*** is a utility library that assists developing IO functionality. ***Apache-commons-validator*** provides the building blocks for both client- and server-side data validation. A***pache-velocity-engine*** is a general-purpose template engine.. The main motivation for selecting ***jFlex*** was the intention to reuse an existing dataset, which has been developed and used for a research effort with similar goals (see [24]). The rest of the projects have been selected as representative projects of good quality, since the ***Apache foundation*** is well-known for producing high-quality projects, whereas ***jUnit*** is a very well-reputed project that is very frequently used / reused in software development. All classes of these systems have been used as cases for this study. Therefore, our study was performed on approximately 650 java classes (on average: approx.130 classes per project).

Table I. OSS Project Demographics

Project	#classes	Avg #commits in training transitions	#commits in predicted transition	Training Versions	Predicting Versions
jFlex	47	83	85	1.4.1 1.6.0	1.6.0 1.6.1
jUnit	164	142	674	4.8.1 4.11	4.11 4.12
commons-io	62	167	413	1.4 2.4	2.4 2.5
commons-validator	145	186	41	1.3.1 1.5.0	1.5.0 1.5.1
velocity-engine	229	86	82	1.6.1 1.6.4	1.6.4 1.7.0

4.4 Data Collection

For each unit of analysis (i.e., class), we recorded twelve variables, as follows:

- **Demographics**: 3 variables (i.e., project, version, class name).
- **Assessors**: 8 variables (i.e., CPM, CBO, RFC, MPC, DAC, MOA, PCCC, and CPM_old[1]). These variables are going to be used as the independent variables for testing correlation, predictability and discriminative power. We note that although the calculation of CPM takes as input PCCC, we use both of them as independent variables since we want to isolate the power of using historical data of class changes as an assessor of change proneness (see introduction of Section 5). All metrics are calculated in the last training version.
- **Actual changes**: We use PCCC for the transition between the last two versions of a class (i.e., those that we want to predict—see last column of Table I), as the variable that captures the actual changes. This variable is going to be used as the dependent variable in all tests.

The metrics evaluated as assessor of change proneness have been calculated by using three tools. ***PCCC*** is calculated by a tool that has been developed by the first and the second author. The tool is using the GitHub API to calculate in how many commits each class has been modified. The tool receives as input a starting and an ending commit hash tag. The tool is freely available for download in the web[2]. ***CPM*** has been calculated by modifying the tool of Tsantalis et al. [24]. The tool in its original version was used to calculate ***CPM_old***. In the updated version [2], which is freely available for download in the web[3], REM has substituted the

[1] The assessment of change proneness as performed by the original method

[2] http://www.cs.rug.nl/search/uploads/Resources/CommitChangeCalc.rar

[3] http://www.cs.rug.nl/search/uploads/Resources/InstabilityCalculator.rar

propagation factor, so as to increase the realism of the calculated change probability. We note that concerning internal probability of change we feed the tool with PCCC calculated by the aforementioned tool. ***The rest of the coupling metrics***, have been calculated using the Percerons Client tool. Percerons is an online platform [4] created to facilitate empirical research in software engineering, by providing, among others, source code quality assessment [3].

4.5 Data Analysis

The collected variables (see previous section) will be analyzed against the six criteria of the 1061 IEEE Standard (see Section 4.1) as imposed by the guidelines of the standard (see Table II).

Table II. Measure Validation Analysis

Criterion	Test	Variables
Correlation	Pearson correlation	Assessors Actual changes (*last version of the projects*)
Consistency	Spearman correlation	Assessors Actual changes (*last version of the projects*)
Tracking	Spearman correlation	Assessors Actual changes (*across all versions*)
Predictability	Linear Regression	Independent: Assessors Dependent: Actual changes (*last version of the projects*)
Discriminative Power	Kruskal Wallis Test	Testing: Assessors Grouping: Actual changes (*last version of the projects*)
Reliability	all the aforementioned tests (*seperately for each project –across all versions*)	

For presenting the results on ***Correlation and Consistency***, we use the correlation coefficients (coeff.) and the levels of statistical significance (sig.). The value of the coefficient denotes the degree to which the value of the actual changes is in analogy to the value of the assessor. To represent the ***Tracking*** property of the evaluated metrics, we report on the consistency (i.e. the coefficient of rank correlation between the quality factor and metric values) for multiple project versions. In particular, we report the mean correlation coefficient and the percentage of versions, in which the correlation was statistically significant. For reporting on ***Predictability***, with a regression model, we present the level of statistical significance of the effect (sig.) of the independent variable on the dependent (how important is the predictor in the model), and the accuracy of the model (i.e., mean standard error). While investigating predictability, we produced a separate linear regression model for each predictor (univariate analysis), because our intention was not to investigate the cumulative predictive power of all metrics, but of each metric individually.

Additionally, for presenting the ***Discriminative Power*** of each metric, we investigate whether groups of classes differ with respect to the corresponding metric score. The groups of classes have been created using the equal frequency binning technique [26]. For reporting on the hypothesis testing, we present the level of statistical significance (sig.) of the Kruskall-Wallis test. We note that in order for a metric to adequately discriminate groups of cases, the significance value should be less than 0.05, or 0.01 for strict evaluations. In the case of our study, we preferred to use the 0.01 threshold since many differences were significant at the 0.05 level, leading to inconclusive results. Finally, for reporting on the ***Reliability*** of metrics while assessing if a class will change, we present the results of all the aforementioned tests, separately for the five explored OSS projects. The extent to which the results on the projects are in agreement (e.g., is the same metric the most valid assessor of class change proneness for all projects?) represents the reliability of the considered metric.

5. RESULTS

In this section, we present the results of the case study. Section 6.1 presents the results on the comparison of CPM to the other candidate change proneness assessors, with respect to five validity criteria (correlation, tracking, consistency, predictability and discriminative power). Section 6.2 presents the assessment of the reliability of CPM.

5.1 Correlation, Consistency, Tracking, Predictability and Discriminative Power (RQ_1)

In this section we present the results obtained for answering RQ_1. In Table III, we present the results of correlation analysis. In particular, each row of the table represents one project, whereas each column an assessor of change proneness. The cells of the table denote the Pearson correlation co-efficient. The italic fonts denote statistically significant correlations, whereas bold fonts the assessor that is the most highly correlated with the actual change proneness. The two final rows of Table III (grey-shaded cells) correspond to the percentage of projects in which the specific assessor is statistically significantly correlated to the actual value of change proneness (***sig.***) and the projects for which the metric is the optimal assessor (***best assessor***). Table IV follows the same formatting, but the presented results correspond to the Spearman correlation coefficients. Finally, Table V presents the obtained results for tracking: in each row we present the mean Spearman correlation coefficient obtained by assessing the change proneness of all versions from the previous ones.

Table III. Correlation Analysis

Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
Io	*.615*	*.368*	*.872*	***.878***	-.063	*.431*	-.190	*.653*	.160
Velocity	***.152***	.105	-.033	.033	-.079	.068	-.045	.042	.068
validator	***.763***	-.017	*.328*	*.341*	-.072	.053	.116	*.675*	.184
jFlex	*.600*	.115	.048	.129	-	.053	-.028	**.799**	.045
jUnit	***.591***	*.305*	*.455*	*.452*	.141	*.395*	.231	*.543*	*.185*
% sig.	100%	40%	60%	60%	0%	40%	0%	60%	20%
best assessor	**60%**	0%	0%	20%	0%	0%	0%	20%	0%

Table IV. Consistency Analysis

Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
io	*.339*	*.059*	*.492*	***.524***	-.065	*.211*	.132	*.221*	.100
velocity	***.234***	.167	.021	.030	-.101	*.097*	-.054	-.039	.064
validator	*.437*	.011	*.227*	*.297*	-.022	*.074*	-.035	***.454***	*.270*
jFlex	***.620***	.389	.241	.380	-	*.240*	.181	.619	.204
jUnit	*.346*	*.285*	*.462*	*.431*	.064	*.211*	.166	***.393***	.151
% sig.	100%	40%	60%	60%	0 %	100%	0%	60%	20%
best assessor	**40%**	0%	0%	20%	0%	0%	0%	**40%**	0%

Table V. Tracking Analysis

Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
Io	.319	.071	.467	**.576**	-.055	.232	.099	.177	.085
velocity	.192	**.200**	.019	.026	-.079	.082	-.011	-.035	.054
validator	**.371**	.012	.216	.267	-.012	.070	-.007	.341	.257
jFlex	**.527**	.428	.265	.361	.000	.264	.199	.508	.224
jUnit	.311	.342	.416	**.474**	.058	.232	.183	.275	.136
% sig.	80%	40%	60%	80%	0 %	80%	0%	40%	20%
best assessor	**40%**	20%	0%	**40%**	0%	0%	0%	0%	0%

The results of Tables III and IV suggest that CPM is strongly correlated to the actual change proneness of a class [21]. In addition, CPM is the optimal assessor of change proneness, both in terms of actual value (see Table III) and ranking (see Table IV). Concerning correlation to the actual value of change proneness, the second most valid metric is PCCC. MPC is the second metric that can most accurately rank classes, based on their change proneness. Finally, the results of Table V imply that when considering the complete evolution of projects, the validity of CPM decreases to a moderate correlation [21]. Despite this decrease, CPM remains the most accurate assessor of class ranking, based on change proneness.

In Table VI we present the results of the Linear Regressions that have been performed to validate the predictive power of each assessor. The cells in Table VI represent the standard error of the regression model, whereas the rest of the notations remain unchanged. Similarly to the results presented in Table III, in Table VI we can observe that CPM and PCCC are the optimum predictors of class change proneness, followed by RFC and MPC. However, we need to note that CPM is significantly correlated with change proneness in all OSS projects that we have examined, whereas PCCC only in 60%.

Table VI. Predictability Analysis

Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
io	*.011*	*.013*	*.007*	***.006***	.014	*.013*	.014	*.010*	.014
velocity	*.006*	.006	**.001**	.006	.006	.006	.080	.006	.006
validator	***.021***	.033	*.031*	*.031*	.033	.033	.039	*.022*	.032
jFlex	*.010*	.013	.013	.013	-	.013	.013	**.007**	.013
jUnit	***.009***	*.011*	*.010*	.010	.011	*.010*	*.011*	***.009***	*.011*
% sig.	100%	40%	60%	40%	0%	40%	20%	60%	20%
best assessor	**40%**	0%	20%	20%	0%	0%	0%	**40%**	0%

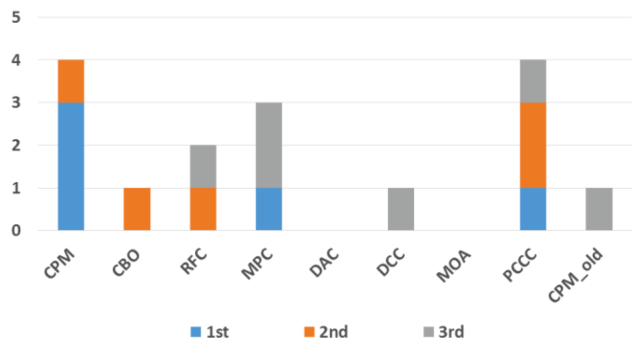
Finally, regarding discriminative power the results are presented in Table VII. All notations of Table VII remain the same, with the difference that cell values represent the level of significance in the differences of metric scores. The results of Table VI suggest that in all OSS projects CPM is able to discriminate groups of classes, based on their change proneness, i.e., classify them into groups with similar values of change proneness. The metrics that are ranked second, with respect to their discriminative power are CBO, MPC, and PCCC.

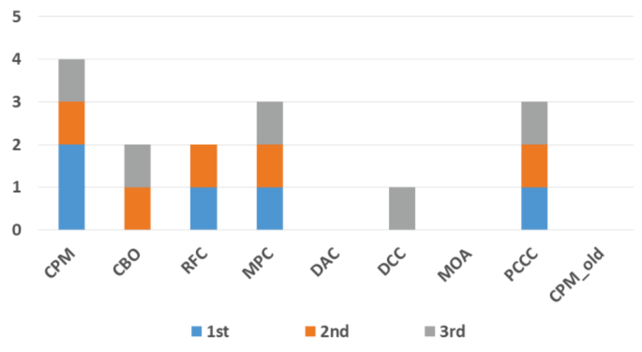
Table VII. Discriminative Power Analysis

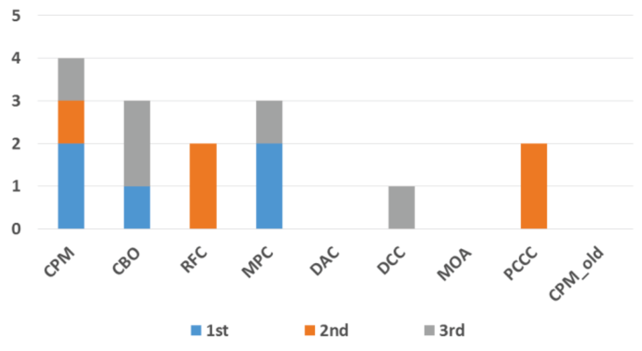
Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
io	**.000**	.494	**.000**	**.000**	.059	.060	.015	.146	.392
velocity	**.000**	.001	.064	.039	.291	.087	.725	.315	.053
validator	**.000**	.002	.021	.006	.181	.268	.926	**.000**	.105
jFlex	**.000**	**.000**	.056	.010	1.0	.016	.007	**.000**	.035
jUnit	**.000**	.024	**.000**	**.000**	.473	.119	.079	**.000**	.287
% sig. (<0.01)	100%	60%	40%	60%	0%	0%	20%	60%	0%

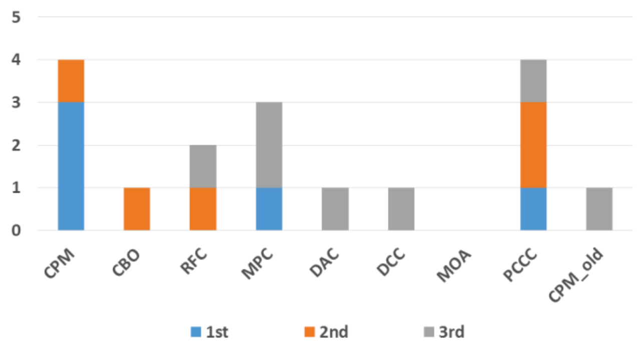
5.2 Reliability (RQ2)

Regarding RQ_2, we performed all the aforementioned tests separately for each one of the analyzed projects. In order for a metric to be considered a reliable assessor of change proneness, it should be consistently ranked among the top assessors for each criterion. To visualize this information, in Figures 2a - 2e we present a stacked bar chart for each validity criterion. In each chart, every bar corresponds to one change proneness assessor, whereas each stack represents the ranking of the assessor among the evaluated ones for each project. For example, in Figure 2a, we can observe that CPM is the top-1 assessor of change proneness, with respect to correlation in three projects and the top-2 assessor, for one other project. We need to clarify that in some Figures the count of 1st, 2nd and 3rd positions does not sum up to five, since in case of equal scores, all metrics have been scored with the highest rank.


(a) Correlation Analysis


(b) Consistency Analysis


(c) Tracking Analysis


(d) Predictability Analysis

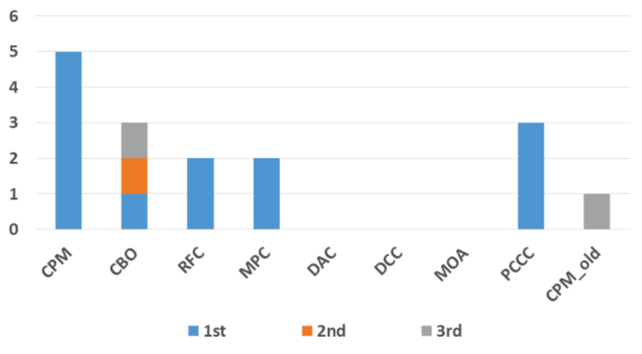

(e) Discriminative Power Analysis

Fig. 2. Reliability Assessment

To obtain a synthesized view of the aforementioned results in Table VIII we adopt a point system to evaluate the consistency with which each assessor is highly ranked among others in the multiple criteria. In particular, for every top-1 position we reward the assessor with three points, for every second position with two points, and for every top-3 position with one point. In Table VIII each row represents a criterion, whereas each column an assessor of change proneness. The cells represent the points that each assessor is gratified for each criterion. The last row, presents a sum of all criteria. Similarly, to before, bold fonts represent the most valid assessor.

Table VIII. Reliability Analysis

Project	CPM (Proposed)	CBO	RFC	MPC	DAC	DCC	MOA	PCCC	CPM_old
Corelation	**11**	2	3	5	0	1	0	8	1
Consistency	**9**	3	5	6	0	1	0	6	0
Tracking	**9**	5	4	7	0	1	0	4	0
Predictive	**11**	3	2	5	1	1	0	8	1
Discriminative	**15**	6	6	6	0	0	0	9	1
Total	**55**	19	19	29	1	4	0	35	3

The results of Figure 2 and of Table VIII suggest that CPM is the most reliable assessor of class change proneness, followed by PCCC and MPC. We also need to note that MOA and DAC are the least reliable assessors. Concerning specific validity criteria, CPM is more reliable concerning discriminative and predictive power, as well as correlation. Regarding the ability of CPM to rank classes, based on their change proneness, we can observe that the reliability of CPM is similar to MPC (although higher).

6. DISCUSSION

In this section, the outcome of this study is discussed from two different perspectives. First, we provide possible interpretations of the obtained results; and second, we present possible implications to researchers and practitioners.

6.1 Interpretation of the Results

The results of this study suggest that CPM, as a change proneness assessor, exceeds all other explored metrics on the considered validation criteria, followed by PCCC and MPC. It is expected that CPM outperforms other metrics, mostly because it combines the two aspects of change proneness (i.e., probability of the class itself to change due to changes in requirements, bug fixes, etc. and the probability of a class to change due to the ripple effects), whereas each other metric considers only one of the two. In particular, the assessment of internal probability of a class to change through past data provides an accurate proxy of changes through requirements/bug fixes/etc., similarly to the assessment of the ripple effect probability through the REM.

By further focusing on these two aspects of change proneness an interesting observation can be made, by examining the outcomes obtained by exploring each validation criterion. The ***actual value of change proneness*** is more related to the ***amount of changes*** that can be counted in the ***history*** of the project, rather than to project structure. This finding is implied by the fact that for criteria related to actual values (i.e., not rankings) PCCC is the second most accurate assessor of class change proneness. On the other hand, ***coupling metrics*** (e.g., MPC, RFC, etc.) ***perform better in assessing the ranking of classes with respect to change proneness***. This observation is intuitive since by nature PCCC is closer to change frequency in the sense that they are metrics of the same type, similar values, and range of values (esp. since we are exploring the same project). Regarding ranked metrics, where the aforementioned reasons (i.e., range of value) have been filtered out PCCC loses this advantage.

Another interesting observation is that the ***validity*** of all metrics in terms ***of tracking are lowered compared to consistency***. This outcome is expected since the training set for assigning the value of the internal class probability is getting smaller, while we are exploring earlier project versions, and therefore less accurate. This outcome implies that using a larger part of project history than five versions, might even more increase the validity of CPM and PCCC. However, this statement needs to be empirically evaluated by a follow-up study.

Furthermore, by ***comparing the coupling metrics of this study***, we can observe that MPC is the optimal assessor of class change proneness, followed by RFC and CBO. By contrasting RFC to MPC, we can conclude that the number of local methods that are used as a parameter in the calculation of RFC (and also consist the major difference in these two metrics) is not related to class change proneness. This finding complies with existing literature that suggests that class coupling is a better assessor of change proneness than complexity (a proxy of which is the number of local methods). By contrasting MPC to CBO it becomes apparent that the strength of a coupling relationship (offered by MPC) is more closely related to the notion of change proneness than the number of dependencies (counted by CBO).

Finally, by ***comparing the validity*** of ***CPM*** when calculated ***with*** the ***enhancements*** that we proposed, ***against*** its ***original implementation*** we can observe that its validity has been significantly improved. More specifically, the correlation, consistency, and tracking ability of the metrics have been improved by approximately 300%, whereas it's predictive power by 22%. Additionally, the discriminative power of the metrics has been increased by our enhancements by 100%. We note that in the original introduction of the metric, only its predictive power has been assessed.

6.2 Implications to Researchers and Practitioners

Based on the aforementioned results and discussions, we can provide implications for researchers and practitioners.

On the one hand, we encourage practitioners to ***use CPM*** in their ***quality monitoring processes***, in the sense that CPM is the optimal available assessor of the probability of a class to change. We expect that tool support[4] that automates the calculation process will ease its adoption. Based on the expected relations of change proneness to more high-level software characteristics (e.g., increased defect-proneness, more technical debt interest, etc.), it can be used as an assessor of future quality indicators. However, this claim needs to be verified through a follow-up study.

On the other hand, we encourage researchers to ***transform CPM so as to fit architecture evaluation purposes***, i.e., assess the probability of components to change. We believe that such a transformation would be of great interest for the architecture community. Also, such an attempt would increase the benefits of practitioners, in the sense that change impact analysis could scale into larger systems. Finally, we note that other claims that have already been stated in the manuscript that require further validation constitute interesting future work, i.e.:

- the increase in the assessing power of CPM when a larger portion of software history is considered as a training set for the method;
- the usefulness of the proposed metric in practice and its adoption by practitioners; and
- the validity of the CPM metric in other levels of granularity.

7. THREATS TO VALIDITY

In this section we present the threats to the validity of our case study. In this case study, we aimed at exploring if certain metrics are valid assessors of class change proneness. Therefore, possible threats to construct validity [22] deal with the way that these metrics and change proneness are quantified, including both the rationale of the calculation and tool support. However, concerning the rationale on how the metrics are calculated, it should be noted that their definition is clear and well documented (see Section 3), whereas the used tools have been thoroughly tested, before deployment. Additionally, in order to ensure the reliability [22] of this study, we: (a) thoroughly documented the study design in this document (see Section 4), so as to make the study replicable, and (b) all steps of data collection and data analysis have been performed by two researchers in order to ensure the validity of the process. Furthermore, concerning the external validity [22] of our results, we have identified two possible threats. First, we investigated only five OSS projects. The low number of subjects is a threat to generalization, in the sense that results on these projects cannot be generalized to the complete open source software projects population. However, since the units of analysis for this

[4] http://www.cs.rug.nl/search/uploads/Resources/InstabilityCalculator.rar

study are classes and not projects, we believe that this threat is mitigated. On the other hand, an actual threat concerns the reliability criterion, as it compares results from different projects and we only compare five OSS projects against each other. For this specific criterion, further investigation is required. Second, we investigated projects only written in Java due to the corresponding tool limitations. Therefore, the results cannot be generalized in other languages, e.g., C++. This threat becomes, even more important, because C++ projects are expected to also make of use the friend operator, which changes the scope of class attributes.

8. CONCLUSIONS

In this study, we presented and validated a new method that calculates the Change Proneness Metric (CPM), which can be used for assessing class change proneness. The method takes inputs from two sources: (a) class dependencies, which are used to calculate the portion of the accessible interface of a class that is used by other classes, and (b) class change history, which is used as a proxy of how frequently maintenance actions are performed (e.g., modify requirements, fix bugs, etc.). After quantifying these two parameters (for all classes and for all their dependencies), CPM can be calculated at the class level, by employing simple probability theory. In this work CPM has been empirically validated against various change proneness assessors, based on the criteria defined in the 1061-1998 IEEE Standard for a Software Quality Metrics [1]. The conducted case study was an embedded one, and was executed on five OSS projects with more than 650 classes.

The results of the validation suggested that CPM excels as an assessor of class change proneness compared to a variety of well-known metrics. In particular, the results implied that both the historical and the structural information are needed for an accurate assessment, since: (a) the historical data prove to more correlated to the actual values of change proneness, and (b) the structural dependencies data are more useful for ranking classes. In any case, the combined perspective that is provided by CPM has been evaluated as the optimal assessor of change proneness, with respect to all validation criteria. Based on these results, implications for researchers and practitioners have been provided.

REFERENCES

[1] 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, and 31 December 1998 (re-affirmed 9 December 2009).

[2] A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou and P. Avgeriou, "The Effect of GoF Design Patterns on Stability: A Case Study", Transactions on Software Engineering, IEEE Computer Society, 41 (8), pp. 781-802, August 2015.

[3] A. Ampatzoglou, A. Gkortzis, S. Charalampidou, and P. Avgeriou, "An Embedded Multiple-Case Study on OSS Design Quality Assessment across Domains", *7th International Symposium on Empirical Software Engineering and Measurement* (ESEM' 13), ACM/IEEE Computer Society, pp. 255-258, Octomber 2013, Baltimore, USA.

[4] A. Ampatzoglou, O. Michou, and I. Stamelos, "Building and mining a repository of design pattern instances: Practical and research benefits", Entertainment Computing, Elsevier, 4 (2), pp. 131-142, April 2013.

[5] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "A Financial Approach for Managing Interest in Technical Debt", LNBIP, Springer, vol. 257, pp. 117-133, 2016.

[6] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Introducing a ripple effect measure: a theoretical and empirical validation", *9th International Symposium on Empirical Software Engineering and Measurement* (ESEM 2015), Beijing, China, IEEE Computer Society, 22–23 October 2015.

[7] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, and P. Avgeriou, "A Mapping Study on Design-Time Quality Attributes and Metrics", Journal of Systems and Software, Elsevier, accepted for publication.

[8] J. Bansiya and C. G. Davies, "A hierarchical model for object-oriented design quality assessment", Transactions on Software Engineering, IEEE Computer Society, 28 (1), pp. 4-17, January 2002.

[9] J. M. Bieman, G. Straw, H. Wang, P. W. Munger, and R. T. Alexander, "Design patterns and change proneness: an examination of five evolving systems", 9th International Software Metrics Symposium (METRICS'03), IEEE Computer Society, pp. 40–49, 3-5 September 2003, Sydney, Australia.

[10] S. Black, "Deriving an approximation algorithm for automatic computation of ripple effect measures", Information and Software Technology, Elsevier, 50 (7–8), pp. 723-736, June 2008.

[11] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", Transactions on Software Engineering, IEEE Computer Society, 20 (6), pp. 476 - 493, June 1994.

[12] A. Field, "Discovering Statistics using IBM SPSS Statistics", SAGE Publications Ltd, 2013.

[13] M. Fowler, "Analysis patterns: Reusable object models", Addison-Wesley Professional, October 1996.

[14] Ah-R. Han, S. Jeon, D. Bae, and J. Hong, "Measuring behavioral dependency for improving change-proneness prediction in UML-based design models", Journal of Systems and Software, Elsevier, 83 (2), pp. 222-234, February 2010.

[15] F. M. Haney, "Module connection analysis: A tool for schedul-ing of software debugging activities," *Proceedings of the AFIPS Fall Joint Computer Conference*, pp. 173-179, 5-7 December 1972, USA.

[16] B. Isong, O. Ifeoma and M. Mbodila, "Supplementing Object-Oriented software change impact analysis with fault-proneness prediction", *15th International Conference on Computer and Information Science (ICIS' 16)*, IEEE Computer Society, pp. 1-8, Okayama, Japan, 26-29 June 2016.

[17] F. Jaafar, Y.-G. Guéhéneuc, S. Hammel, and G. Antoniol, "Detecting asynchrony and dephase change patterns by mining software repositories", Journal of Software: Evolution and Processes, Wiley & Sons, 26 (1), January 2014.

[18] A.G. Koru and J. Tian, "Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products", Transactions on Software Engineering, IEEE Computer Society, 31 (8), pp. 625-642, August 2005.

[19] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, Elsevier, 23 (2), pp. 111-122, November 1993.

[20] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis", Empirical Software Engineering, Springer, 17 (3), pp. 200-242, June 2012.

[21] L. Marg, L. C. Luri, E. O'Curran, and A. Mallett, "Rating Evaluation Methods through Correlation", *1st Workshop on Automatic and Manual Metrics for Operational Translation Evaluation* (MTE'14), Reykjavik, Iceland, 26 May 2014.

[22] P. Runeson, M. Host, A. Rainer and B. Regnell, "Case Study Research in Software Engineering: Guidelines and Examples", John Wiley & Sons, 2012.

[23] R. Schwanke, L. Xiao, and Y. Cai, "Measuring architecture quality by structure plus history analysis", 35th International Conference on Software Engineering (ICSE 2013), San Francisco, USA, ACM/IEEE Computer Society, pp. 891-900, 18-26 May 2013.

[24] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the Probability of Change in Object-Oriented Systems", Transactions on Software Engineering, IEEE Computer Society, 31 (7), pp. 601-614, July 2005.

[25] H. van Vliet, "Software Engineering: Principles and Practice", John Wiley & Sons, 2008.

[26] I. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques", Morgan Kaufmann, 2nd Edition, 2005.

[27] S. S. Yau and J. S. Collofello, "Some Stability Measures for Software Maintenance", Transactions on Software Engineering, IEEE Computer Society, 6 (6), pp.545-552, November 1980.

[28] S. S. Yau and J. S. Collofello, "Design Stability Measures for Software Maintenance", Transactions on Software Engineering, IEEE Computer Society, 11 (9), pp.849-856, Sept.

[29] J. Zhang, S. Sagar, an E. Shihab, "The Evolution of Mobile Apps: An Exploratory Study", International Workshop on Software Development Lifecycle for Mobile (DeMobile' 13), ACM, pp. 1-8, Saint Petersburg, Russia, 19 August 2013.