Introducing a Ripple Effect Measure: A Theoretical and Empirical Validation

Elvira-Maria Arvanitou[1], Apostolos Ampatzoglou[1], Alexander Chatzigeorgiou[2], Paris Avgeriou[1]
[1] Department of Computer Science, University of Groningen, Netherlands
[2] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
e.m.arvanitou@rug.nl, a.ampatzoglou@rug.nl, achat@uom.gr, paris@cs.rug.nl

Context: Change impact analysis investigates the negative consequence of system changes, i.e., the propagation of changes to other parts of the system (also known as the ripple effect). Identifying modules of the system that will be affected by the ripple effect is an important activity, before and after the application of any change.

Goal: However, in the literature, there is only a limited set of studies that investigate the probability of a random change occurring in one class, to propagate to another**.** In this paper we discuss and evaluate the Ripple Effect Measure (in short REM), a metric that can be used to assess the aforementioned probability.

Method: To evaluate the capacity of REM as an assessor of the probability of a class to change due to the ripple effect, we: (a) mathematically validate it against established metric properties (e.g., non-negativity, monotonicity, etc.), proposed by Briand et al., and (b) empirically investigate its validity as an assessor of class proneness to the ripple effect, based on the 1061-1998 IEEE Standard on Software Measurement (e.g., correlation, predictive power, etc.). To apply the empirical validation process, we conducted a holistic multiple-case study on java open-source classes.

Results: The results of REM validation (both mathematical and empirical) suggest that REM is a theoretically sound measure that is the most valid assessor of the probability of a class to change due to the ripple effect, compared to other existing metrics.

***Keywords*—**ripple effect; software metrics; case study

I. INTRODUCTION

Change impact analysis is related to assessing and identifying the consequences, caused by modifications in one part of a system, on other parts of the same system, known as the *ripple effect* [20]. Studying and quantifying the ripple effect can provide benefits both before and after the application of a change:

- **before the actual application of changes:** for program comprehension and effort estimation [10], [19]; and
- **after changes have been applied:** for test case prioritization and the identification of relationships among software components [22].

Despite these benefits, to the best of our knowledge, there is only a limited set of metrics for assessing whether a class is prone to the ripple effect (i.e., its probability to change due to a modification in another class of the system). When exploring the proneness of a class to the ripple effect, there are two aspects that need to be assessed: (a) the probability of the source class to undergo changes, i.e., the event that will trigger the ripple effect phenomenon, and (b) the dependencies that will potentially propagate the change to dependent classes. Although the former can only be estimated by analyzing the source code change history, the latter can be estimated by structural analysis, which will be the focus of this study. In order to come up with potential metrics for structurally assessing class proneness to the ripple effect, we need to consider: a) the number of dependencies, and b) a parameter named *propagation factor*, i.e., the probability of a change to propagate from one class to the other through a dependency [24]. The only type of metrics that might be able to assess these two aspects are coupling metrics, since class coupling is defined *as the degree to which one class is connected to other classes of the system*. However, the ability of coupling metrics to assess[1] if a class would change due to the ripple effect has not been empirically validated.

By inspecting the definitions of the most popular coupling metrics (see Section VI), we observed that none of them can capture both the number of dependencies of a class and their propagation factor. Thus, in this paper we describe a structural coupling metric, namely Ripple Effect Measure (REM)[2], and validate its capacity as an assessor of the probability of a class to change due to the ripple effect. The calculation of REM is based on the identification of efferent class dependencies, the quantification of the propagation factor for each one of them, and the aggregation from the dependency level to the class level. The validation process of REM is two-fold:

- a theoretical validation aiming at mathematically proving that REM holds basic properties of software measurement (e.g., non-negativity, normalization, etc.) [11];
- an empirical one, aiming at investigating the validity [1] of REM as an assessor of a class proneness to the ripple effect, by comparing it to existing coupling metrics.

The rest of the paper is organized as follows: In Section II we discuss *related work*, whereas in Section III we present the *REM* and its *calculation* process. Section IV discusses the *me-*

[1] The term *assess* is used as defined in the *1061 IEEE Standard* for Software Quality Metrics [1], i.e., as the ability to substitute, track the changes of, and predict the levels of a quality factor; and to discriminate [1] between low and high values of the quality factor.

[2] REM has been introduced by Ampatzoglou et al. [5], as part of a case study on the instability of pattern-participating classes. In this paper, we discuss its foundations to enable the self-containment of this manuscript, but our main focus is on its validation.

978-1-4673-7899-4/15/$31.00 ©2015 IEEE

tric validation processes that will be used during REM validation. Section V presents the *outcome of the theoretical validation* of REM. Section VI presents the *design* and the *results of the empirical validation* of REM. Section VII *discusses the main findings* of the validation; finally, Sections VIII and IX present *threats to validity* and *conclude* the paper, respectively.

II. Related Work

In this section, we present studies that are related to the quantification of the ripple effect. As indirect related work, we present studies that attempt to quantify stability, i.e., the ability of the software to remain unchanged, regardless of the changes, occurring to other parts of the system. The connection between stability and ripple effect has been discussed from the earliest papers on ripple effect by Yau and Collofello [27], [28], [29].

Specifically, 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 [28], [29]. The metrics proposed by Yau and Collofello are similar to REM, in the sense that they both take into account the dependencies between modules, and the global variables that are part of the global interface of a module. The main points of differentiation of these metrics, compared to REM, is: (a) the inclusion of the probability of the change to occur in the class that emits the ripple effect – leading to a metric that is not structural, (b) the inclusion of a complexity metric that is used for quantifying the effort needed to apply the change – which is irrelevant to the probability of a change to propagate, but only related to the effort that will be needed to apply the change.

In a more recent study (2007), Black, proposed an approach for calculating a *complexity-weighted total variable definition propagation 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 the proposed ripple effect measure [9]. The main points of similarity and differentiation are the same, as those of Yau and Collofello. An *empirical stud*y conducted by Elsih and Rine [15], investigated the *ability of existing metrics* to assess the design stability of classes. The results suggested that coupling metrics (CBO and RFC) are the optimum assessors of class stability, i.e., the reciprocal of the ripple effect. However, the used stability measure was not the actual one, but an estimation based on dependencies and attribute sharing. Finally, in a research effort on a different direction, by Alshayeb and Li [4], the authors propose a system *design instability measure* that quantifies the actual changes that have been performed from one version of the system to the other. Nevertheless, this approach is not comparable to REM, in the sense that this study is an after-the-fact analysis, whereas REM is an estimator of the proneness of a class to the change due to the ripple effect.

III. Ripple Effect Measure

Dependency analysis is at the core of algorithms that explore the ripple effect caused by class change (see e.g., [9], [13], [14], [28], [29]), in the sense that changes propagate, across system classes, through their dependencies. Such change propagations (i.e., the most common ripple effect) [18], 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[3]. 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. Next, we describe the abovementioned types of dependencies, along with the types of changes that can propagate through them:

- **Generalization** is used to represent "is-a" relationships. In a generalization, there are three possible reasons for change propagation: (a) super method invocation (use of super), (b) access of protected fields, and (c) override or implementation of abstract methods by the subclass.
- **Containment** is used to represent "has-a" or "part-whole" relationships. In a containment relationship, changes can propagate due to method calls of the container class to the public interface of the containee class.
- **Association** is used to represent relationships due to the declaration of a local variable inside a method, or due to the use of a class object as a parameter/return type in a method of another class. In an association relationship, changes can propagate due to method calls of a class to the public interface of another class.

We note that the aforementioned way that changes are propagated through class dependencies are described for 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.

Let us consider the design of Fig. 1, in which two relations exists (`A1` depends on `A`, and `A1` depends on `B`). Are these relationships equally strong? Is it equally probable for a change occurring in `A` and a change occurring in `B` to ripple into `A1`? In order to answer such questions, a metric that quantifies the probability of a ripple effect between classes is necessary. To this end, we defined the Ripple Effect Measure (REM), a metric that quantifies the probability of a random change occurring in the public interface of a *source class* (`A` or `B`) to be propagated to a *dependent class* (`A1`) that uses it, by assuming that all elements of the source classes (i.e., attributes and methods) have the same probability to change[4]. To calculate REM, we need to consider the count of:

[3] We note that this work, does not handle conceptual dependencies among classes, as they would be enforced by class contracts. Our work, aims at providing a purely structural metric for assessing the probability of a class to change due to the ripple effect.

[4] As a static measure REM discards changes that occur in other parts of the source code (private parts or method bodies), in the sense that changes in the public interface affect only the class that changes (other classes do not have access to it) and co-change of method bodies can only be tracked by historical analysis (for more details on the differences between class change proneness and instability see [5]).

- all members of the source classes (A or B) that are accessed by the dependent class (A1), which (if changed) will emit one or more changes to the dependent class (A1); and
- all members of the public interface of source classes (A or B).

The ratio of the two aforementioned counts is an estimate of the probability that a random change in the public interface of source class will occur in a member that will emit this change to the dependent class. In other words, as the number of the members of the source class that emit changes to another dependent class, approaches the total number of members that can change in the source class, it becomes more probable for changes to propagate from the source class to the dependent class. Thus, and by taking into account the three types of dependencies and the way changes propagate across them, REM for a dependency (i.e., the propagation factor) between the dependent and the source class, can be calculated as follows:

$$\text{REM}_{\text{dependency}} = \frac{NDMC+NOP+NPrA}{NOM+NA} \qquad (1)$$

NDMC: Number of distinct method calls from the dependent class to the source class (plus, super class method invocations for the case of generalization)

NOP: Number of polymorphic methods in the source class (valid only for generalization)

NPrA: Number of protected attributes in the source class (valid only for generalization or friend classes)

NOM: Number of methods in the source class

NA: Number of attributes in the source class

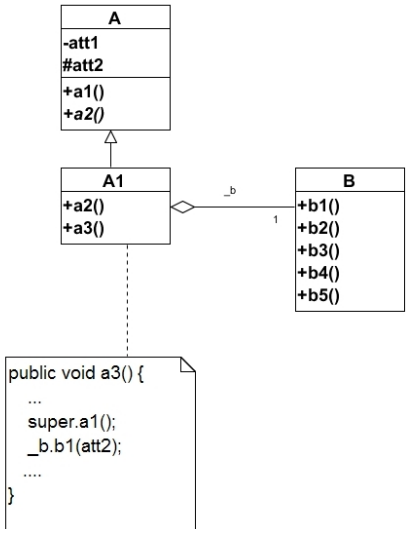

Fig. 1. Coupling Intensity Example

Based on the above, in the example of Fig. 1, there are three changes which would ripple from superclass A to subclass A1:

- **change in the signature of a1**. Changing the signature of a1(), would lead to a compile error in the corresponding invocation (see body of method a3).
- **change in the signature of a2**. Changing the signature of a2() would lead to a compile error in the place where a2() is overridden, since a2() is declared as abstract in the superclass.
- **change in the name of att2.** Changing the name of attribute att2, would lead to a compile error in the body of a3.

Thus, REM, for the dependency of A_1 on A, can be calculated as follows:

$$\text{REM}_{\text{A1(A)}}{}^{5} = \frac{NDMC+NOP+NPrA}{NOM+NA} = \frac{1+1+1}{2+2} = 0.75$$

Similarly, there is only one change which would propagate from containee class (B) to the container class (A1), i.e. change in the signature of b1. Changing the signature of b1, would lead to a compile error in the corresponding invocation. We note that if any other method of B is changed, the change will not propagate to A1, as long as the method is not called in the implementation of A1 (in the example no other method of B is invoked from A1, except b1). Thus, REM, for the dependency of A_1 on B, can be calculated as:

$$\text{REM}_{\text{A1(B)}} = \frac{NDMC+NOP+NPrA}{NOM+NA} = \frac{1+0+0}{5+0} = 0.20$$

Therefore, based on REM, for the example of Fig. 1, we can claim that a change occurring in A is more probable to propagate in A1, than a change occurring in B.

Until this point, the calculation of REM is performed at the class dependency level, i.e., from a class to one other class. In order to aggregate REM to a higher level of granularity, i.e., at class level, one should take into account the dependencies of a class on all other classes. In the example of Fig. 1, class A1 will have to change if a change propagates from either class B or A or both. Therefore, for one class having several dependencies, we propose the use of the *joint probability of all events* (i.e., change in any dependency) [2], for aggregating the score of REM from dependency to class level. For example, in the case of Fig. 1, the joint probability is calculated as follows:

$$\mathbf{P(A \cup B)} = P(A) + P(B) - P(A) \cdot P(B)$$

$$= 0.75 + 0.20 - 075*0.20 = 0.8$$

P(A∪B): $\text{REM}_{\text{A1(A, B)}}$

P(A): $\text{REM}_{\text{A1(A)}}$

P(B): $\text{REM}_{\text{A1(B)}}$

IV. Validation Process

According to Briand et al. [11], metric validation should be performed as a two-step process, i.e., providing a theoretical and empirical validation. The *theoretical validation* aims at mathematically proving that a metric satisfies certain criteria, whereas the *empirical validation* aims at investigating the accuracy with which a specific metric quantifies the correspond-

[5] This refers to the REM of class A1, due to its dependency to A

ing quality factor. The validation of the REM metric will be performed using both types of validation. Concerning the theoretical validation of REM, we will use the properties for coupling metrics proposed by *Briand et al.* [11][6]. The used properties are:

- **Normalization and Non-Negativity**: The metric score should have a pre-defined lower and upper bound, and not be negative [11];
- **Null Value and Maximum Value**: The metric score should be null or take the maximum value, only under very specific circumstances [11];
- **Monotonicity**: The metric score for a system *S* should be higher than the metric score for a system *S*' (which is identical to *S*, except from one class, i.e., *C* and *C*' respectively), if the metric for class *C* is higher than the metric score for class *C*' [11]; and
- **Merging of Classes**: If a class *C*' is the union of two classes *C*1 and *C*2, then the metric score for *C*' should be less or equal to the sum of the scores of *C*1 and *C*2 [11]. For the special case of unconnected classes *C*1 and *C*2, the property suggests that the merged class should have a metric score that is equal to the sum of the individual scores.

Regarding the empirical validation of REM we will use the properties described in the *1061 IEEE Standard* for Software Quality Metrics [1], for comparing REM to existing coupling metrics (for details on these coupling metrics see Section VI). In this Standard, six metric validation criteria are introduced, accompanied by the statistical test that shall be used for evaluating every criterion, as follows:

- **Correlation** assesses the association between a quality factor and the metric under study. The criterion is desirable, to ensure that the use of the metric can substitute the quality factor. The criterion is quantified by using a correlation coefficient [1].
- **Consistency** assesses if the metric under study is consistently correlated with the quality factor, by using their ranks. The criterion is desirable to ensure that the metric under study can accurately rank, by quality, a set of products or processes. The criterion is quantified by using the coefficient of rank correlation [1].
- **Tracking** assesses if the metric under study is capable of tracking changes in product quality over their lifecycle. The criterion is quantified by using coefficient of rank correlation for a set of project versions [1].
- **Predictability** assesses the accuracy with which the metric under study is able to predict the levels of the quality factor. The criterion is quantified through the standard estimation error for a regression model using as predictor the metric under study [1].
- **Discriminative Power** assesses if the metric under study is capable of separating groups of high-quality and low-quality components. Although the criterion is proposed to be quantified through a contingency table (see [1]), the proposed quantification was not applicable for coupling metrics, because they cannot be recoded to categorical variables, without setting arbitrary thresholds. Therefore, we use an equivalent test for assessing discriminative power, i.e., independent t-test [16].
- **Reliability** assesses if the metric under study can fulfill all five aforementioned validation criteria, over multiple systems. This criterion can be assessed by replicating the previously discussed tests (for each of the aforementioned criteria) to various software systems [1].

[6] We acknowledge the existence of various other metric properties (e.g., the ones proposed by Weyuker in 1998 [26]), we preferred to use the ones proposed by Briand et al., because they are coupling-specific.

V. Theoretical Validation

Based on the definition of REM (see Section III), we validate it against all coupling properties in the next paragraphs.

A. Normalization and Non-Negativity

REM at both dependency and class levels is normalized and non-negative. Concerning normalization, the **lower bound of REM is zero** (since the lower bound of the numerator is zero – i.e., classes that *do not call any method* from other classes, and *do not override polymorphic methods,* and *do not access any protected attributes* of a superclass); whereas the **upper bound of REM is 1** (since $NDMC + NOP \leq NOM$ and $NPrA \leq NA$). Regarding non-negativity, **REM is always assigned to a positive value,** because it involves only additions and divisions of positives numbers.

B. Null Value and Maximum Value

REM at both dependency and class levels is defined for all possible cases of dependencies and classes, and has a specific maximum value. REM would not be defined (assigned a null value) only in the case that the denominator would be zero, i.e., classes depending on other **classes with no attributes and no methods**. However, such classes are not expected to exist, in the sense that they would not offer any functionality to the system. As mentioned in Section V.A, the **maximum value of REM is 1**.

C. Monotonicity

In the original definition of monotonicity, two levels of granularity are involved, i.e., the system level and the class level, where the values from class level are aggregated to system level. In the case of REM, the two levels of granularity are different, i.e., the REM at dependency level, is aggregated to class level as discussed in Section III. Therefore, in order to explore the monotonicity of REM, we apply the definition, by mapping class to system (original definition) and dependency to class (original definition).

For simplicity, assume a class C_1 with two dependencies (D_{11} and D_{12}), and a class C_2 with one dependency having exactly the same REM as the first dependency of C_1 and a second dependency that is different (D_{21} and D_{22}, where $REM_{D11}=REM_{D21}$ and $REM_{D12} > REM_{D22}$). Based on the aforementioned assumptions and the definition of monotonici-

ty, in order for REM to be monotonic, we need to prove that $REM_{C1(D11,D12)} > REM_{C2(D21,D22)}$. We mathematically prove the previous relationship, as follows:

$$REM_{C1(D11,D12)} > REM_{C2(D21,D22)} \quad =>$$

$$\cancel{REM_{D11}} + REM_{D12} - REM_{D11} * REM_{D12} >$$

$$\cancel{REM_{D21}} + REM_{D22} - REM_{D21} * REM_{D22} \quad =>$$

$$REM_{D12} * \cancel{(1 - REM_{D11})} > REM_{D22} * \cancel{(1 - REM_{D21})} \quad =>$$

$$REM_{D12} > REM_{D22} \ ,$$

which holds, based on the original assumption ($REM_{D12} > REM_{D22}$). Therefore, REM is a **monotonic coupling** metric. A proof of monotonicity for classes with n dependencies, could be provided, but it is omitted due to space limitations.

D. Merging of Classes

For similar reasons to monotonicity, we will argue that REM holds for the merging of classes property, through an example with two dependencies. Let class C_1 be a class with one dependency (D_1) and class C_2 be another class with a different dependency (D_2). The merged class C' will have two dependencies (D_1 and D_2). In order for the 'merging of classes' property to hold for REM, we need to prove that $REM_{C'(D1,D2)} \leq REM_{C1(D1)} + REM_{C2(D2)}$. The mathematical proof of the previous relationship is as follows[7]:

$$REM_{C'(D1,D2)} \leq REM_{C1(D1)} + REM_{c2(D2)} \quad =>$$

$$\cancel{REM_{D1}} + \cancel{REM_{D2}} - REM_{D1} * REM_{D2}$$

$$\leq \cancel{REM_{D1}} + \cancel{REM_{D2}} \quad =>$$

$$-REM_{D1} * REM_{D2} \leq 0,$$

which holds, because both REM_{D1} and REM_{D2} are non-negative numbers (based on the explanations provided in Section V.A). Thus, REM is able to handle **merging of classes** as implied by the corresponding property. We note that the property of the special case of *merging unconnected classes is not satisfied* by the REM, due to its probabilistic nature.

VI. Empirical Validation

In order to empirically investigate the validity of REM to assess if a class will change due to the ripple effect, we performed a case study on two open source projects. We compare REM to existing coupling metrics with respect to all the criteria described in Section IV [1]. The coupling metrics that have been used as control variables are:

- **Coupling Between Objects (CBO)**: Number of classes to which a class is coupled [12]
- **Response For a Class (RFC)**: Number of local methods, plus the number of methods called by local methods in the class [12].
- **Message Passing Coupling (MPC)**: Number of send statements defined in the class [21].
- **Data Abstraction Coupling (DAC)**: Number of abstract data types defined in the class [21].
- **Measure of Aggregation (MOA)**: Number of data declarations of a user defined type [8].

In order to be as inclusive as possible, we selected metrics from three different metric suites (Chidamber and Kemerer [12], Li and Henry [21], and QMOOD [8]), which are well-known and tool-supported. Also, the aforementioned list of metrics includes both code- and design-level coupling metrics.

A. Case Study Design

The case study has been designed and reported according to the guidelines of Runeson et al. [23]. Therefore, 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 case selection criteria, (d) the data collection procedure, and the (e) data analysis process.

***Research Objectives and Research Questions*:** The aim of this case study, expressed through a GQM formulation, is*:* to *analyze* coupling metrics *for the purpose of* evaluation *with respect to* their validity to assess if a class will change due to the ripple effect, *from the point of view of* software engineers *in the context of* change impact analysis.

Based on this goal we set two research questions: the first, on the first five validity criteria (i.e., correlation, consistency, tracking, predictability and discriminative power), in which no distinction between projects is necessary (i.e., all projects are handled as one dataset); and the second on reliability. Reliability is investigated through a separate research question, because it entails performing the analysis for the aforementioned five validity criteria separately for each project (i.e., each project is considered as a different dataset, and those datasets are cross-checked in order to assess metrics' reliability). Therefore, we ask the following research questions:

RQ_1: Can REM assess if a class will change due to the ripple effect, based on the criteria of the 1061-1998 IEEE Std., compared to other existing metrics?

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

RQ_2: How does REM compare to the other coupling metrics, w.r.t. their reliability?

[7] We note that since class C1 has only one dependency (i.e., D1), $REM_{C1} = REM_{D1}$

***Case and Units of Analysis*:** This study is a holistic multiple-case study, i.e. it studies multiple cases where each case is comprised of a single unit of analysis. Specifically the subjects of the study are open source projects, where classes are cases and at the same time units of analysis.

***Case Selection*:** As subjects for our study, we selected to use two open source projects, i.e., JFlex (versions 1.4 and earlier - in total 14 versions) and JMol (versions 0.9 and earlier - in total 10 versions). The main motivation for selecting these projects was the intention to reuse an existing dataset, which has been developed and used for a research effort with similar goals (see [24]). Specifically, Tsantalis et al. [24] have explored the specific versions of these projects, to identify classes that have changed from the projects' previous versions. These changes have been manually inspected, so as to check whether they are due to reasons other than the ripple effect (main effect–referred as internal changes in the original publication [24]) or due to the ripple effect (referred as propagated changes in the original paper [24]). Consequently, we were able to reuse the extracted data that concerned changes due to ripple effect. All classes of these systems have been used as cases for this study. Therefore, our study was performed on 150 java classes.

We clarify that for assessing *correlation, consistency, predictability* and *discriminative power*, we used only the last version of JFlex and JMol as cases: according to the used IEEE standard, system evolution is not considered for these four validation criteria. We used the last versions of the projects, so as to examine the largest (in terms of lines of code and number of classes) versions of the systems. On the other hand, regarding *tracking* and *reliability* we used all examined versions of both systems.

***Data Collection*:** For each case (i.e., class), we primarily recorded ten variables, as follows:

- **Demographics**: 3 variables (i.e., project, version, class name).
- **Evaluated metrics**: 6 variables (i.e., REM, CBO, RFC, MPC, DAC, and MOA). These variables are going to be used as the independent variables for testing correlation, predictability and discriminative power.
- **Observed ripple effect**: For any transition between two successive versions of a class, this variable indicates if a class has changed due to the ripple effect (value equals 1), or whether it remained unaffected by the ripple effect (value equals 0). This variable is going to be used as the dependent variable in all tests.

The coupling metrics have been calculated by using two tools:

- **REM** has been calculated by modifying the tool of Tsantalis et al. [24]. The tool in its original version was created in order to calculate class change proneness, by using a constant value for the propagation factor. In the updated version [5], which is freely available for download in the web[8], REM has substituted the propagation factor, so as to increase the realism of the calculated change probability. From this tool, we use the REM calculation, and not the complete instability calculation.
- **the rest of the coupling metrics**, have been calculated using the *Percerons Client* tool[9]. Percerons is an online platform [7] created to facilitate empirical research in software engineering, by providing, among others, source code quality assessment [6]. The platform has been used for similar reasons in [3], [6] and [17].

***Data Analysis*:** For investigating both research questions, we used the statistical analysis that is advocated by the 1061-1998 IEEE Standard [1] (see Section IV), as summarized in Table I. We note that a coupling metric is expected to have a proportional relationship to the probability of a class to change due to the ripple effect: the larger the number of dependencies of a class and the stronger the dependencies, the more probable for this class to receive changes, from other classes.

TABLE I. MEASURE VALIDATION RESULTS

Criterion	Test	Variables
Correlation	Point bi-serial correlation	Coupling Metrics Observed ripple effect (*last version of the projects*)
Consistency	Rank bi-serial correlation	Coupling Metrics Observed ripple effect (*last version of the projects*)
Tracking	Rank bi-serial correlation	Coupling Metrics Observed ripple effect (*across all versions*)
Predictability	Logistic Regression	Independent: Coupling Metrics Dependent: Observed ripple effect (*last version of the projects*)
Discriminative Power	Independent Sample t-test	Testing: Coupling Metrics Grouping: Observed ripple effect (*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 correlation coefficient denotes the degree to which the value of the observed ripple effect is in analogy to the value of the predictor. We note that since the dependent variable (i.e., Observed ripple effect) is binary, we used the ***point bi-serial correlation*** and ***rank bi-serial correlation*** for assessing correlation and consistency, respectively. To represent the *Tracking* property of the evaluated metrics, we report on the consistency for multiple project versions, through 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., the total correctness value–percentage of correctly classified

[8] http://iwi.eldoc.ub.rug.nl/root/2014/ClassInstability/

[9] http://www.percerons.com

cases). While investigating predictability, we produced a separate logistic regression model for each predictor (univariate analysis), because our intention was not to investigate the cumulative predictive power of all coupling metrics, but of each metric individually. Similarly to correlation/consistency, we performed a logistic regression, since the dependent variable is binary, therefore the classification threshold was set to 0.5.

Additionally, for presenting the *Discriminative Power* of each metric, we investigate whether the two groups (actually propagated changes, and not propagated changes) differ with respect to the corresponding coupling metric score. For reporting on the independent sample t-tests, we present the mean difference (diff.) between the values of the testing variable of the compared groups, and the level of statistical significance (sig.). We note that in order for a metric to adequately discriminate groups of cases, the significance value should be less than 0.05.

Finally, for reporting on the *Reliability* of coupling metrics while assessing if a class will change due to the ripple effect, we present the results of all the aforementioned tests, separately for the two explored OSS projects. The extent to which the results on the two projects are in agreement (i.e., Are they statistically significant for both? Is the same metric the most valid assessor of class proneness to the ripple effect for both projects?) represents the reliability of the considered metric.

B. Results

In this section, we present the results of the empirical validation of REM, organized by research question. We first present the outcome of comparing REM to the other coupling metrics, w.r.t. their correlation, consistency, tracking, predictive and discriminative power as assessors of a class proneness to the ripple effect. Subsequently we present the results of comparing the reliability of REM to the reliability of the other coupling metrics.

RQ1: Correlation, Consistency, Tracking, Predictability and Descriminative Power

Table II presents the outcome of the statistical analysis related to RQ_1. Each row of Table II corresponds to one validity criterion, whereas each column to one metric. To enable the easy reading of Table II, we use visual aids to help the reader in focusing on the most important observations. Specifically, we denote statistically significant relationships at the 0.01 level with grey cell shading, and statistically significant relationships at the 0.05 level with light grey cell shading (when applicable). Finally, the optimal assessor of class proneness to the ripple effect, for each criterion, is denoted with bold fonts.

To answer RQ_1 we can suggest that *REM* and *CBO* are the only coupling metrics that can provide a statistically significant assessment (albeit weak) whether a class will change due to the ripple effect, w.r.t. the validity criteria described in IEEE 1061 Standard. The results can be summarized as follows:

- **Correlation, Predictability** and **Discriminative Power**: REM is the most valid metric while assessing if a class will change due to the ripple effect, followed by CBO. The results on REM validity are statistically significant at the 1% level, whereas for CBO at the 5% level. The result for other metrics are not statistically significant. Concerning the discriminative power of metrics, we note that, although the absolute value of the difference offered by CBO is higher than the one offered by REM, this difference is due to the range of values of the two metrics[10]. Therefore, we consider REM as the most valid assessor of class proneness to the ripple effect, based on its statistical significance.
- **Consistency**: REM is the most consistent assessor of class proneness to the ripple effect, followed by CBO and RFC. The results on the REM are statistically significant at the 1% level, whereas for CBO and RFC at the 5% level.
- **Tracking**: REM achieves the best results with respect to tracking class proneness to the ripple effect, followed by CBO. The results on REM are statistically significant for 26% of the examined cases.

TABLE II. MEASURE VALIDATION RESULTS

		REM	RFC	MPC	DAC	CBO	MOA
Correlation	*coef.*	**0.25**	0.11	0.06	0.07	0.18	0.13
	sig.	**0.00**	0.18	0.46	0.34	0.02	0.09
Consistency	*coef.*	**0.26**	0.15	0.11	0.11	0.18	0.01
	sig.	**0.00**	0.05	0.16	0.18	0.02	0.97
Tracking	*coef.*	**0.21**	0.16	0.18	0.14	0.19	0.17
	pct.	**26%**	17%	21%	13%	21%	17%
Predictability	*acc.*	**62.3%**	54.3%	54.3%	55.0%	57.6%	55.0%
	sig.	**0.01**	0.99	0.92	0.99	0.03	0.67
Descriminative Power	*diff*	**-0.16**	-5.21	-2.57	-0.02	-1.64	-0.08
	sig	**0.01**	0.49	0.93	0.27	0.02	0.68

RQ2: Reliability

Regarding RQ_2, we performed all the aforementioned tests separately for each one of the analyzed OSS projects, and compared them. The results for each project are summarized in Tables III and IV. The notation used in Tables III and IV is the same as that of Table II. Therefore, the shading of the cell corresponds to the level of statistical significance, whereas the bold fonts indicate the most valid assessor of class proneness to the ripple effect concerning a certain criterion.

[10] Range of REM is [0, 1], whereas range of CBO is [0, +∞).

TABLE III. MEASURE VALIDATION RESULTS (JFLEX)

		REM	RFC	MPC	DAC[11]	CBO	MOA
Correlation	*coef.*	**0.37**	0.24	0.00	N/A	0.27	0.18
	sig.	**0.02**	0.15	1.00	N/A	0.10	0.28
Consistency	*coef.*	0.37	0.41	0.39	N/A	**0.45**	0.18
	sig.	0.02	0.01	0.02	N/A	**0.00**	0.30
Tracking	*coef.*	**0.24**	0.18	0.18	N/A	**0.24**	0.17
	pct.	**28%**	14%	21%	N/A	**28%**	7%
Predictability	*acc.*	**67.6%**	58.9%	54.9%	N/A	62.2%	64.9%
	sig	**0.03**	0.16	0.99	N/A	0.08	0.29
Descriminative Power	*diff*	**-0.26**	-20.57	-0.02	N/A	-2.74	-0.46
	sig	**0.02**	0.15	1.00	N/A	0.10	0.28

TABLE IV. MEASURE VALIDATION RESULTS (JMOL)

		REM	RFC	MPC	DAC	CBO	MOA
Correlation	*coef.*	**0.24**	0.09	0.08	0.07	0.20	0.15
	sig.	**0.00**	0.33	0.37	0.45	0.03	0.10
Consistency	*coef.*	**0.24**	0.09	0.05	0.11	0.15	0.00
	sig.	**0.01**	0.36	0.56	0.25	0.11	0.98
Tracking	*coef.*	**0.19**	0.14	0.15	0.14	0.17	0.16
	pct.	**22%**	11%	11%	11%	22%	22%
Predictability	*acc.*	**60.5%**	45.0%	45.0%	51.8%	52.9%	50.9%
	sig	**0.03**	0.97	0.87	0.98	0.08	0.87
Descriminative Power	*diff*	**-0.13**	-0.38	-5.12	-0.02	-1.40	0.03
	sig	**0.04**	0.97	0.87	0.31	0.07	0.87

[11] We were unable to calculate the validity of DAC as an assessor of class proneness to the ripple effect in JFlex, due to the limited use of inheritance in the project: no class had any data abstraction coupling.

From Tables III and IV, we can observe that REM is the only coupling metric that produces reliable results for all required tests (i.e., similar between the two software systems). Specifically, REM is the optimal assessor of class proneness to the ripple effect regarding *correlation*, *tracking*, *predictive* and *discriminative power*, for both projects. Concerning *consistency*, REM is the most valid assessor of class proneness to the ripple effect for JMol, but the 4th for JFlex. However, for both projects, the results on the consistency of REM are statistically significant, and therefore reliable. An interesting result that we can observe from Tables III and IV is that CBO, i.e., the second most valid assessor of class proneness to the ripple effect according to Table II, suffers from reliability issues. Specifically, CBO is not reliable w.r.t any criterion, since it provides statistically significant results, at most, only for one out of the two projects (i.e., consistency and tracking for JFlex and correlation for JMol).

VII. DISCUSSION

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

***Interpretation of the Results*:** The results of this study suggest that REM is a theoretically sound coupling metric and that exceeds all other metrics on the considered validation criteria, followed by CBO. REM outperforms other coupling metrics, mostly because it is specifically designed for assessing the probability of a class to change due to the ripple effect, and not coupling in general. To this end, REM has been designed so as to resolve the limitations of the other metrics and combine their strengths, as follows:

- REM takes into account **the number of dependencies**, similarly to CBO, DAC, and MOA. However, none of the aforementioned metrics considers the strength of these dependencies.
- REM is a **normalized measure** that is able to quantify the **strength of dependencies**. For similar metrics, which quantify the strength of coupling and at the same time are proportional to the number of dependencies, i.e. RFC and MPC, it is not clear to what extent their value is related to the two components they mix. For example, MPC is equal for two classes, for which the first has two dependencies and every dependency is used three times, and the second has six dependencies and each dependency is used once.
- REM is the only metric that takes into account the changes that can be propagated by using **protected attributes**, either through hierarchies or friend methods.

These characteristics of REM make it the most valid assessor of the probability of a class to change due to the ripple effect, as demonstrated by the obtained empirical evidence. Also, by comparing the rest of the coupling metrics, we can observe that CBO is the 2nd best assessor of class proneness to the ripple effect. This implies that the number of efferent couplings is indeed an important parameter, when exploring the ripple effect. The fact that CBO is a better assessor than MOA and

DAC can be explained, as they capture only a small portion of efferent couplings (i.e., number of aggregations and number of used abstract types respectively). Finally, by contrasting RFC to MPC, we can conclude that the distinct counting of called methods offered by RFC make it a better assessor than MPC, because regardless of how many times a method is called, a change will be propagated even with only one method call.

Additionally, although REM is the optimal assessor of a class proneness to the ripple effect, the results suggest that the strength of the correlation between them (see Table II) can only be considered as weak or moderate at best. This is an expected outcome, since in this study REM is validated against the actual class changes due to the ripple effect. However, as explained in Section I, the actual probability of change in any class, depends not only on the probability of changes to propagate, but also on the probability of changes to occur in the first place. Nevertheless, the quantification of the actual probability of change in a class, might demand the processing of data that are not always available (e.g., class change history, or subjective identification of design hot-spots, etc.). In such cases, the interested software engineer, would be forced to use a structural metric, among which REM is the optimal predictor of class proneness to the ripple effect, based on the results of this case study.

***Implications to Researchers and Practitioners*:** By taking into account the aforementioned results and discussions, we can provide various implications for researchers and practitioners. Specifically, we:

- Encourage **practitioners** to use REM in their *change impact analysis* activities, in the sense that REM is the optimal available assessor of the probability of a class to change due to the ripple effect. We expect that tool support that automates the calculation process (see Section VI.A) will ease its adoption. However, the applicability of REM in practice is still in need of further investigation.
- Encourage **researchers** to investigate options for improving the effect size of the metric. For example, by incorporating *the actual change proneness of classes in the calculation of REM*, so as to increase the validity of the proposed metric. Specifically, towards this direction, we plan to replicate this study, by using the tool presented in [5], feed it with historical data on class change proneness and use all the considered metrics as change propagation factors, and observe: (a) if the results in the fitness of metrics as assessors of class proneness to the ripple effect remain the same, and (b) if the validity of the calculated metrics increases (e.g., strength of correlation).
- Encourage **researchers** to investigate the possibility of *assigning weights to the various axes through which a class can receive changes* (i.e., protected attributes, called methods, and overridden methods) by an empirical study on class change history, and observe if such a change can increase the validity of REM.
- Encourage **researchers** to *transform REM* so as to fit *architecture evaluation* purposes, i.e., assess the probability of changes to propagate across components. We believe that such a transformation would be of great interest for the architecture community, which shows increased interest in metrics. Also, such an attempt would increase the benefits of practitioners, in the sense that change impact analysis could scale into larger systems.
- Encourage **researchers** to use *combinations of coupling metrics* to explore the possibility of increasing power of metric for predicting the ripple effect.

VIII. THREATS TO VALIDITY

In this section we present the threats to the validity of our case study[12]. In this case study, we aimed at exploring if certain coupling metrics are valid assessors of class proneness to the ripple effect. Therefore, possible threats to construct validity [23] deal with the way coupling and the ripple effect are quantified, including both rationale and tool support. On the one hand, the rationale on how the coupling metrics are calculated is not a threat, since their definition is clear and well documented, whereas the used tools have been thoroughly tested, before deployment. On the other hand, in order to assess the probability of a class to change due to the ripple effect, we used the number of actually propagated changes, which is exactly how the ripple effect should be calculated. The calculation of actually propagated changes, has been reused from an already published study [24], and their results have been manually validated by the first author.

Additionally, in order to ensure the reliability [23] of this study, we: (a) thoroughly documented the study design in this document (see Section VI.A), 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 [23] of our results, we have identified two possible threats:

- *we investigated only two OSS projects*. The low number of subjects is a threat to generalization, in the sense that results on these two projects cannot be generalized to the complete OSS projects population. However, since the units of analysis for this study are classes and not projects, we believe that this threat is mitigated. The only real threat concerns the reliability criterion, as it compares results from different projects and we only compare two OSS projects against each other. For this specific criterion, further investigation is required.
- *we investigated project only written in Java*. Due to tool limitations, we could only analyze projects written in Java. 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.

[12] The mathematical results presented in Section V, by definition, cannot suffer from threats to validity.

IX. Conclusions

In this study, we presented and validated a new coupling metric, named Ripple Effect Measure (REM), which can be used for assessing a class proneness to the ripple effect. The metric is defined at dependency level by calculating the portion of the accessible interface of a class that is used by other classes. After calculating REM for all dependencies, REM can be aggregated to class level, by employing simple probability theory. The validation of REM has been performed in a two-step process: first, we theoretically validated it against well-known coupling metric properties [11]; and second, we empirically validated against the metric validation criteria defined in the 1061-1998 IEEE Standard for a Software Quality Metrics [1]. To empirically investigate the validity of REM as an assessor of class proneness to the ripple effect, we performed a holistic multiple-case study on over 150 OSS project classes.

The results of the theoretical validation suggested that REM is mathematically sound, in the sense that all coupling metric properties can be proven, based on its definition. Additionally, the results of our case study suggested that REM excels as an assessor of a class proneness to the ripple effect metric compared to a variety of well-known coupling metrics. 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, 31 December 1998 (re-affirmed 9 December 2009).

[2] A.V. Aho and J.D. Ullman, "Foundations of Computer Science", *CS Press*, 3rd Ediion, 1995.

[3] S. Alhusain, S. Coupland, R. John, and M. Kavanagh, "Towards machine learning based design pattern recognition", *13th UK Workshop on Computational Intelligence* (UKCI '13), IEEE Computer Society, pp. 244–251, September 2013, Guildford, United Kingdom.

[4] M. Alshayeb and W. Li, "An empirical study of system design instability metric and design evolution in an agile software process", *Journal of Systems and Software*, Elsevier, 74 (3), pp. 269-274, 2005.

[5] 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, accepted for publication, 2015.

[6] 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.

[7] 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.

[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] 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.

[10] S. A. Bohner, "Impact analysis in the software change process: A year 2000 perspective", *4th International Conference on Software Maintenance* (ICSM' 96), IEEE Computer Society, pp. 42-51, 4-8 November 1996, Monterey, USA.

[11] L. C. Briand, J. W. Daly and J. K. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *Transactions on Software Engineering*, IEEE Computer Society, 25 (1), pp. 91-121, January 1999.

[12] 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.

[13] J. M. Conejero, E. Figueiredo, A. Garcia, J. Hernández, E. Jurado, "On the relationship of concern metrics and requirements maintainability", *Information and Software Technology*, Elsevier, 54 (2), pp. 212-238, February 2012.

[14] F. Dantas, A. Garcia, and J. Whittle, "On the role of composition code properties on evolving programs", 6th *International Symposium on Empirical Software Engineering and Measurement* (ESEM '12), ACM/IEEE Computer Society, September 2012, Lund, Sweden.

[15] M. O. Elish, D. Rine, "Investigation of metrics for object-oriented design logical stability", *7th European Conference Software Maintenance and Reengineering (CSMR' 03)*, IEEE Computer Society, pp.193-200, 26-28 March 2003.

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

[17] I. Griffith and C. Izurieta, "Design pattern decay: the case for class grime", *8th International Symposium on Empirical Software Engineering and Measurement* (ESEM' 14), ACM/IEEE Computer Society, pp. 1-4, September 2014, Torino, Italy.

[18] F. M. Haney, "Module connection analysis: A tool for scheduling of software debugging activities", *Fall Joint Computer Conference*, IEEE Computer Society, pp. 173-179, 5-7 December 1972, Anaheim, USA.

[19] S. Hassaine, F. Boughanmi, Y.-G. Guéhéneuc, S. Hamel, and G. Antoniol, "A seismology-inspired approach to study change propagation", *27th International Conference on Software Maintenance* (ICSM' 11), IEEE Computer Society, 25 - 30 September 2011, USA.

[20] E. Horowitz and R. C. W illiamson, "SODOS: a software documentation support environment—Its definition", *Transactions on Software Engineering*, IEEE Computer Society, 12 (8), pp. 849–859, August 1986

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

[22] P. Rovegard, L. Angelis, and C. Wohlin, "An empirical study on views of importance of change impact analysis issues", *Transactions on Software Engineering*, IEEE Computer Society, 34 (4), pp. 516–530, April 2008.

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

[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] E. J. Weyuker, "Evaluating software complexity measure", *Transactions on Software Complexity Measure*, IEEE Computer Society, 14 (9), pp. 1357–1365, 1988.

[27] S. S. Yau, J. S. Collofello and T. M. MacGregor, "Ripple effect analysis of software maintenance", 2nd *International Computer Software and Applications Conference* (COMPSAC' 78), IEEE Computer Society, pp. 60-65, 1978.

[28] 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.

[29] 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. 1985.