Information and Software Technology 72 (2016) 110–124

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software metrics fluctuation: a property for assisting the metric selection process

Elvira-Maria Arvanitou[a], Apostolos Ampatzoglou[a,*], Alexander Chatzigeorgiou[b], Paris Avgeriou[a]

[a] Department of Mathematics and Computer Science, University of Groningen, Zernike Campus, Groningen, The Netherlands
[b] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 24 May 2015
Revised 22 December 2015
Accepted 22 December 2015
Available online 30 December 2015

Keywords:
Object-oriented metrics
Fluctuation
Case study
Software evolution

ABSTRACT

Context: Software quality attributes are assessed by employing appropriate metrics. However, the choice of such metrics is not always obvious and is further complicated by the multitude of available metrics. To assist metrics selection, several properties have been proposed. However, although metrics are often used to assess successive software versions, there is no property that assesses their ability to capture structural changes along evolution.

Objective: We introduce a property, *Software Metric Fluctuation* (SMF), which quantifies the degree to which a metric score varies, due to changes occurring between successive system's versions. Regarding SMF, metrics can be characterized as *sensitive* (changes induce high variation on the metric score) or *stable* (changes induce low variation on the metric score).

Method: SMF property has been evaluated by: (a) a case study on 20 OSS projects to assess the ability of SMF to differently characterize different metrics, and (b) a case study on 10 software engineers to assess SMF's usefulness in the metric selection process.

Results: The results of the first case study suggest that different metrics that quantify the same quality attributes present differences in their fluctuation. We also provide evidence that an additional factor that is related to metrics' fluctuation is the function that is used for aggregating metric from the micro to the macro level. In addition, the outcome of the second case study suggested that SMF is capable of helping practitioners in metric selection, since: (a) different practitioners have different perception of metric fluctuation, and (b) this perception is less accurate than the systematic approach that SMF offers.

Conclusions: SMF is a useful metric property that can improve the accuracy of metrics selection. Based on SMF, we can differentiate metrics, based on their degree of fluctuation. Such results can provide input to researchers and practitioners in their metric selection processes.

1. Introduction

Software measurement is one of the most prevalent ways of monitoring software quality [21]. In practice, software quality measurement activities are governed by a measurement plan (e.g., developed based on the IEEE/ISO/IEC-15939 Std. [1]), which, among others focuses in defining the measurement goals, and the metrics selection process. According to Fenton and Pfleeger [16], building a measurement plan involves answering three main questions, two on defining the measurement goals, and a third one, on selecting appropriate metrics:

- *What to measure*? This question has two levels: (a) what quality attributes to measure?—this is related to the identification of the most important concerns of the stakeholders, and (b) what parts of the system should be assessed?—this is related to whether quality measurement should be performed on the complete system (*measure-in-large*) or on a specific design "hot-spot" (*measure-in-small*) [16].
- *When to measure*? This question has two levels as well. The first level concerns the measurement frequency, where one can choose between two major options: (i) perform the measurement tasks once during the software lifecycle (*measure once*), or (ii) perform measurement tasks many times during the software lifecycle (*measure repeatedly*) [16]. The second level concerns the development phase(s) when measurement is to be performed. This decision sets some additional constraints to the metric selection process, in the sense that if one selects to

* Corresponding author. Tel.: +30 2310611090.
E-mail addresses: e.m.arvanitou@rug.nl (E.-M. Arvanitou), apostolos.ampatzoglou@gmail.com, a.ampatzoglou@rug.nl (A. Ampatzoglou), achat@uom.gr (A. Chatzigeorgiou), paris@cs.rug.nl (P. Avgeriou).

http://dx.doi.org/10.1016/j.infsof.2015.12.010
0950-5849/

perform measurement activities in an early development phase the available metric suites are different from those that are available at the implementation level. Usually, design-level metrics are less accurate than code-level metrics; however, they are considered equally important, because they provide early indications on parts of the system that are not well-structured. A detailed discussion on high-level metrics (design-level) and low-level metrics (code-level), can be found in [3].

- *How to measure*? While answering this question, one should select the most fitting measure from the vast collection of available software quality metrics.

All aforementioned questions are inter-dependent, so the order of answering them varies; for example one could start from metric selection (i.e. 'how') and then move on to answer 'when' and 'what' (i.e., measurement goal), or the other way around. When answering one of the questions, the available options for answering the subsequent questions are getting more limited (an elaborate discussion on the inter-connection among the answers to these questions is presented in Section 7). For example, if someone selects to measure cohesion at the design phase, the set of available metrics is limited to the high-level cohesion metrics, as discussed by Al Dallal and Briand [3]; whereas if one selects to measure cohesion at implementation level, the set of available metrics is broadened to the union of high- and low-level cohesion metrics [3]. Due to the high number of available metrics, the selection of metrics that quantify the target quality attributes is far from trivial. For example, concerning cohesion and coupling, recent studies describe more than 20 metrics for each one of them [3] and [19]. This selection process becomes even more complex by the option to choose among multiple *aggregation functions*. Such functions are used to aggregate metric scores from the micro-level of individual artifacts (e.g. classes), to the macro-level of entire systems [13] and [33], whose industrial relevance is discussed in detail by Mordal et al. [29]. In order to assist this metric selection process, researchers and practitioners have proposed several metric properties that can be used for metrics validation and characterization [1,9] and [10].

Metrics selection becomes very interesting in the context of software evolution. Along evolution metric scores change over time reflecting the changes of different characteristics of the underlying systems. For example a metric that concerns coupling, changes from one version of the system to the other, reflecting the changes in the dependencies among its classes. Therefore, a quality assurance team needs to decide on the accuracy with which they wish to capture small-scale changes from one version of the system to the other. This decision is influenced by the goals of the measurement (i.e., the answers to the first two aforementioned questions—"*what and when to measure?*"). In particular, both available options (i.e., capture small changes or neglect them) may be relevant in different contexts. For example, when trying to assess the overall software architecture, the quality team might not be interested in changes that are limited inside a specific component; on the contrary, when trying to assess the effect of applying a source code refactoring, e.g., extract a superclass [18], which is a local change, a small fluctuation of the metric score should be captured. Thus, a property that characterizes a metric's ability to capture such fluctuations would be useful in the metric selection processes. Nevertheless, to the best of our knowledge, there is no such property in the current state of the art for research or practice.

Therefore, in this paper, we define a new metric property, namely ***Software Metrics Fluctuation (SMF)***, as *the degree to which a metric score changes from one version of the system to the other* (for more details see Section 3). While assessing a metric with respect to its fluctuation, it can be characterized as **stable** (*low fluctuation*: the metric changes insignificantly over successive versions) or as **sensitive** (*high fluctuation*: the metric changes substantially over successive versions). Of course, the property is not binary but continuous: there is a wide range between metrics that are highly stable and those that are highly sensitive. Although the observed metric fluctuations depend strongly on the underlying changes in a system, the metrics calculation process also plays a significant role for the assessment of fluctuation. For example, "*What structural characteristics does it measure?*", "*How frequently/easily do these characteristics change?*" or "*What is the value range for a metric?*". In order for the fluctuation property to be useful in practice it should be able to distinguish between different metrics that quantify the same quality attribute (e.g., cohesion, coupling, complexity, etc.). This would support the metric selection process by guiding practitioners to select a metric that is either stable or sensitive according to their needs for a particular quality attribute. Additionally, several metrics work at the micro-level (e.g., method- or class-level), whereas practitioners might be interested in working at a different level (e.g., component- or system-level). The most frequent way of aggregating metrics from the micro- to the macro-level is the use of an aggregation function (e.g., average, maximum, sum, etc.). Therefore, we need to investigate if SMF is able to distinguish between different aggregation functions when used for the same metric. Such an ability would enable SMF to provide guidance to practitioners for choosing the appropriate combination of metric and aggregation function.

In this paper we empirically validate SMF by assessing: (a) the fluctuation of 19 existing object-oriented (OO) metrics, through a case study on open-source software (OSS) projects—see Section 5, and (b) its usefulness by conducting a second case study with 10 software engineers as subject—see Section 6. The *contribution* of the paper is comprised of both the ***introduction and validation of SMF as a property*** and the ***empirical evidence*** derived from both case studies. The organization of the rest of the paper is as follows: *Section 2* presents related work and Section 3 presents background information on the object-oriented metrics that are used in the case studies; *Section 4* discusses fluctuation and introduces the definition of a software fluctuation metric; *Section 5* describes the design and results of the case study performed so as to assess the fluctuation of different object-oriented metrics; *Section 6* presents the design and outcome of the case study conducted for empirically validating the usefulness of SMF; Section 7 discusses the main findings of this paper; *Section 8* presents potential threats to the validity; and *Section 9* concludes the paper.

2. Related work

Since the proposed Software Metrics Fluctuation property allows the evaluation of existing metrics, past research efforts related to desired metric properties will be presented in this section. Moreover, since the proposed property is of interest when someone aims at performing software evolution analysis, other metrics that have been used in order to quantify aspects of software evolution will be described as well.

2.1. Metric properties

According to Briand et al. [9,10] metrics should conform to various theoretical/mathematical properties. Specifically, Briand et al., have proposed several properties for cohesion and coupling metrics [9,10], namely: *Normalization* and *Non-Negativity, Null Value* and *Maximum Value, Monotonicity,* and *Merging of Unconnected Classes* [9,10]. The aforementioned metric properties are widely used in the literature to mathematically validate existing metrics of these categories (e.g., by Al Dallal et al. [2]). Additionally, in a similar context, IEEE introduced six criteria that can be used for assessing the validity of a metric in an empirical manner.

Concerning empirical metric validation, the 1061–1998™ IEEE Standard for Software Quality Metrics [1] discusses the following properties: *Correlation, Consistency, Tracking, Predictability, Discriminative power*, and *Reliability*. From the above metric properties, the only one related to evolution (i.e., takes into account multiple versions of a system) is the tracking criterion. Tracking differs from fluctuation in the sense that tracking is assessing the ability of the metric to co-change with the corresponding quality attribute, whereas the proposed fluctuation property is assessing the rate with which the metric score is changing along software evolution, due to changes in the underlying structure of the system.

2.2. Metrics for quantifying software evolution

Studying software evolution with the use of metrics is a broad research field that has attracted the attention of researchers during the last two decades. According to Mens and Demeyer [28] software metrics can be used for *predictive* and *retrospective reasons*. For example, metrics can be used to identify parts of the system that are critical, or evolution-prone (i.e., predictive); or for analysing the software per se, or the followed processes (retrospective). For example, Gîrba et al. [20] propose four metrics that can be used for forecasting software evolution, based on historical measurements, e.g., *Evolution of Number of Methods (ENOM)* and *Latest Evolution of Number of Methods (LENOM)*.

Additionally, Ó Cinnéide et al. [30] characterize metrics as *volatile* or *inert* by investigating if the values of specific metrics are changed due to the application of a refactoring (i.e., binary value). Despite the fact that the notions used by Ó Cinnéide et al. [30] are similar to ours (characterize a metrics as sensitive or stable), they are only able to capture if the application of a refactoring changes a metric value or not. As a result, volatility can be calculated only if a detailed change record is available, whereas in our study fluctuation can be calculated simply by using a time series of metric values. In addition to that, volatility, as described by Ó Cinnéide et al. [30] is a binary property, whereas SMF is a continuous property which captures the degree of change. As a consequence the discriminative power of SMF is substantially higher. For example, two metrics might have changed during a system transition, but the first might have been modified by 5% and the other by 90%. SMF will be able to capture such differences between metrics, whereas the approach by Ó Cinnéide et al. [30] would characterize them both as equally volatile.

3. Quality attributes and object-oriented metrics

As already discussed in Section 1, different software quality metrics can be calculated for assessing the same quality attribute (QA). In this study we focus on metrics from two well-known metric suites [6,34]. The employed suites contain metrics that can be calculated at the detailed-design and the source-code level (related to the *when to measure* question, discussed in Section 1), and can be used to assess well-known internal quality attributes, such as complexity, coupling, cohesion, inheritance, and size. We note that although all detailed-design metrics can be calculated at source code level, as well, we categorize them in the earliest possible phase (i.e., detailed-design), in which they can be calculated[1]. The aforementioned quality attributes have been selected in accordance to [14], where the authors, using metrics quantifying these QAs, performed an exploratory analysis of empirical data concerning productivity. The selected metric suites are described as follows:

- **Source-code-level metrics:** Riaz et al. [31] presented a systematic literature review (SLR) that aimed at summarizing software metrics that can be used as maintainability predictors. In addition to that, the authors have ranked the identified studies, and suggested that the work of van Koten and Gray [34], and Zhou and Leung [36] were the most solid ones [31]. Both studies (i.e., [34,36]) have been based on two metric suites proposed by Li and Henry [26] and Chidamber et al. [14], i.e., two well-known object-oriented set of metrics. The majority of the van Koten and Gray metrics are calculated at the implementation phase.
- **Detailed-design-level metrics:** On the other hand, a well-known object-oriented metric suite that can be calculated at the detailed-design phase is the quality metrics for object-oriented design (QMOOD) suite, proposed by Bansiya and Davis [6]. The QMOOD metric suite introduces 11 software metrics that are used to assess internal quality attributes that are similar to those of the Li and Henry suite [26]. The validity of the QMOOD suite has been evaluated by performing a case study with professional software engineers [6].

The two selected metric suites are presented in Table 1. From the 21 metrics described in Table 1, for the purpose of our study we ended up with 19 by:

- Excluding the direct access measure (DAM) metric from the QMOOD suite [6], because the Li and Henry [26] metric suite does not offer metrics for the encapsulation quality attribute; and
- Considering the number of methods (NOM) metric from both metrics suite as one metric in our results, since it is defined identically in both studies.

The metrics that are presented in Table 1 are accompanied by the quality attribute that they quantify and the development phase in which they can be calculated. Concerning quality attributes, we have tried to group metrics together, when possible. For example, DAC (Data Abstraction Coupling) could be classified both as an abstraction metric and as a coupling metric. However, since no other metric from our list was related to abstraction, we preferred to classify it as a coupling metric. Similarly, NOP (Number of Polymorphic Methods) is originally introduced as a polymorphism metric. However, polymorphism is often associated with the elimination of cascaded-if statements (e.g., the Strategy design pattern), which in turn is associated with complexity measures (e.g., WMPC). Thus, instead of eliminating it (similarly to DAM), we preferred to treat it as a complexity measure, calculated at the design level.

4. Software metrics fluctuation

In this section we present a measure for quantifying the metric fluctuation property. One of the first tasks that we have performed while designing this study was to research the literature in order to identify if an existing measure could be able to quantify the metric fluctuation property, based on the following high-level requirements:

(a) Based on the definition of SMF (i.e., *the degree to which a metric score changes from one version of the system to the other*), the identified metric should take into account the ***order of measurements in a metric time series***. This is the main characteristic that a fluctuation property should hold, in the sense that it should quantify the extent to which a score changes between two subsequent time points.
(b) As a border case from the aforementioned requirement, the identified metrics should be able to ***reflect changes***

[1] For example, number of methods (NOM) can be calculated from both UML class diagram and source code, but it is mapped to detailed-design since the class diagrams are usually produced before the source code. On the other hand, message passing coupling (MPC) can only be calculated from the source code since it needs the number of methods called.

Table 1
Object-oriented metrics.

Suite	Metric	Description	Develop. phase	Quality attribute
van Koten and Gray	DIT	*Depth of Inheritence Tree:* Inheritence level number, 0 for the root class.	Design	Inheritance
	NOCC	*Number of Children Classes*: Number of direct sub-classes that the class has.	Design	Inheritance
	MPC	*Message Passing Coupling:* Number of send statements defined in the class.	Source code	Coupling
	RFC	*Response For a Class*: Number of local methods plus the number of methods called by local methods in the class.	Source code	Coupling
	LCOM	*Lack of Cohesion of Methods*: Number of disjoint sets of methods (number of sets of methods that do not interact with each other), in the class.	Source code	Cohesion
	DAC	*Data Abstraction Coupling:* Number of abstract types defined in the class.	Design	Coupling
	WMPC	*Weighted Method per Class*: Average cyclomatic complexity of all methods in the class.	Source code	Complexity
	NOM	*Number of Methods*: Number of methods in the class.	Design	Size
	SIZE1	*Lines of Code*: Number of semicolons in the class.	Source code	Size
	SIZE2	*Number of Properties*: Number of attributes and methods in the class	Design	Size
QMOOD	DSC	*Design Size in Classes:* Number of classes in the design.	Design	Size
	NOH	*Number of Hierarchies:* Number of class hierarchies in the design.	Design	Inheritance[a]
	ANA	*Average Number of Ancestors:* Average number of classes from which a class inherits information.	Design	Inheritance[a]
	DAM	*Data Access Metric:* Ratio of the number of private (protected) attributes to the total number of attributes.	Design	Encapsulation
	DCC	*Direct Class Coupling:* Number of other classes that the class is directly related to (by attribute declarations and message passing).	Source code	Coupling
	CAM	*Cohesion Among Methods:* Sum of the intersection of method parameters with the maximum independent set of all parameter types in the class.	Design	Cohesion
	MOA	*Measure of Aggregation:* Number of data declarations whose types are user defined classes.	Design	Coupling[b]
	MFA	*Measure of Functional Abstraction:* Ratio of the number of methods inherited by a class to the total number of methods accessible by methods.	Design	Inheritance[a]
	CIS	*Class Interface Size:* Number of public methods	Design	Size
	NOP	*Number of Polymorphic Methods:* Number of methods that can exhibit polymorphic behavior	Design	Complexity
	NOM	*Number of Methods:* Number of methods in the class.	Design	Size

[a] All metrics whose calculation is based on inheritance trees are marked as associated to inheritance.
[b] Since aggregation is a specific type of coupling, we classified MOA as a coupling metric.

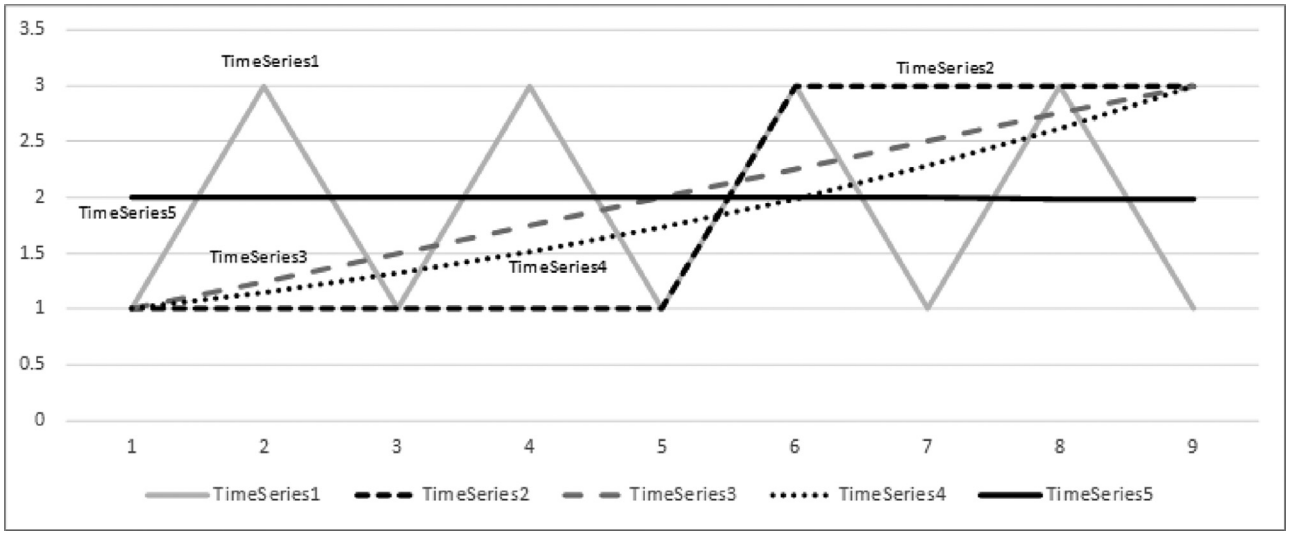

Fig. 1. Fluctuation example.

between individual successive versions and not by treating the complete software evolution as a single change. In other words, the property should be able to discriminate between time series that range within the same upper and lower value, but with a different change frequency (e.g., see *TimeSeries1* and *TimeSeries2* in the following example—Fig. 1) between subsequent points in time.

(c) The proposed fluctuation property should ***produce values that can be intuitively interpreted***, especially for border cases. Therefore, if a score does not change in the examined time period, the fluctuation metric should be evaluated to zero. Any other change pattern should result in a non-zero fluctuation value. Finally, the metric should produce its highest value for time series that constantly change over time and fluctuate between the one end of their range to the other end, for every pair of successive versions of the software.

To make the aforementioned requirements more understandable, let us assume the time series of Fig. 1. For the series of the example, we would expect that a valid fluctuation property would rank *TimeSeries*1 as the most sensitive, and *TimeSeries*5, as the most stable. From the literature [11,12,17,23], we identified three

measures that we considered as possible quantification of the fluctuation property, namely:

- *Volatility* [30], which traditionally has been used as a measure of the variation of price of a financial instrument over time, derived from time series of past market prices. Volatility is calculated as the standard deviation of returns (i.e., the ratio of the score in one time point, over the score in the previous time point).
- *Coefficient of Variance—CoV* [17] is a standardized measure of dispersion, which is defined as the ratio of the standard deviation over the mean.
- *Auto-correlation of lag one* [11] is the similarity between observations as a function of the time lag between them. It is calculated as the correlation co-efficient between each score with the score in the previous time point.

However, none of these metrics was able to conform to the aforementioned requirements, and the intuitive interpretation of Fig. 1. Specifically:

- *Volatility* ranks *TimeSeries4* as the most stable series (because the returns remain the same throughout the evolution). However, this result is not intuitively correct. The reason for this, is that volatility is calculated by using the standard deviation of returns (i.e., scorei-1/scorei). In *TimeSeries4*, the returns are stable, although it is clearly evident that the fluctuation is limited and with no ripples at all.
- *Coefficient of Variance*, ranks *TimeSeries1* and *TimeSeries2*, as having exactly the same fluctuation. However, this interpretation is not intuitively correct, in the sense that *TimeSeries2* changes only once in the given timespan. The reason for this is that *CoV* is calculated based on *standard deviation* and *average value*, which are the same for both series.
- *Auto-correlation of lag one*, ranks *TimeSeries3* and *TimeSeries4*, as the most stable series. However, this result is also not intuitive. The reason for the inability of the auto-correlation of lag one to adequately act as a fluctuation measure, is the fact that it explores if a series of numbers follow a specific pattern, in which one value is a function of the previous one. This is the case of *TimeSeries3* which is an arithmetic progression and of *TimeSeries4*, which is an exponential progression.

Therefore, none of the examined existing measures is able to quantify the SMF property. We thus estimate the Software Metrics Fluctuation property, as the "*average deviation from zero of the difference ratio between every pair of successive versions*". The mathematical formulation of metric fluctuation (***mf***) is as follows:

$$mf = \sqrt{\frac{\sum_{i=2}^{i=n}\left(\frac{\text{score}_i - \text{score}_{i-1}}{\text{score}_{i-1}}\right)^2}{n-1}}$$

$\boldsymbol{\Sigma}$:sum,
$\boldsymbol{n}$:total number of versions,
$\boldsymbol{score_i}$: metric score at version i,
$\boldsymbol{score_{i-1}}$: metric score at version $i-1$

For calculating the deviation from zero, we used the squared root of the second power of the ratio of the difference, in a way similar to the one of standard deviation. Based on the aforementioned definition, the closer to zero *mf* is the more stable the metric is; the higher the value of *mf* is, the more sensitive the metric becomes. Using *mf*, the ranking of the time series of Fig. 1 is as follows (listed from most sensitive to most stable): *TimeSeries1* > *TimeSeries2* > *TimeSeries3* ≈ *TimeSeries4* > *TimeSeries5*, which is intuitive.

5. Case study on assessing the fluctuation of object-oriented metrics

In this section we present the design and the results obtained by a case study on 20 open-source software (OSS) projects, in order to assess the ability of SMF to differentiate between metrics that quantify the same quality attribute and investigate possible differences due to the used aggregation function. In Section 5.1, we present the case study design, whereas in Section 5.2 the obtained results.

5.1. Study design

Case study is an observational empirical method that is used for monitoring projects and activities in a real-life context [32]. The main reason for selecting to perform this study on OSS systems is the vast amount of data that is available in OSS repositories, in terms of versions and projects. The case study of this paper has been designed and is presented according to the guidelines of Runeson et al. [32].

5.1.1. Objectives and research questions

The goal of this study, stated here using the Goal-Question-Metrics (GQM) approach [7], is to "***analyze*** *object-oriented metrics* ***for the purpose of*** *characterization* ***with respect to*** *their fluctuation,* ***from the point of view of*** *researchers* ***in the context of*** *software metric comparison*". The evaluation of the fluctuation of metrics is further focused on two specific directions:

RQ$_1$: *Are there differences in the fluctuation of metrics that quantify the same quality attribute?*
RQ$_2$: *Are there differences in the fluctuation of metrics when using different functions to aggregate them from class level to system level?*

The first question aims at comparing the fluctuation of metrics that quantify the same quality attribute. For example, concerning complexity, we have compared the fluctuation of WMPC [26], and NOP [6] metrics. In this sense a quality assurance team can select a specific quality attribute, and subsequently compare all available metrics that quantify this attribute in order to select one or more metrics based on their fluctuation. We have examined coupling, cohesion, complexity, inheritance and size from both metric suites (i.e., [6] and [26]).

The second question deals with comparing different functions that aggregate metrics from class to system level, with respect to metric fluctuation. We have examined the most common aggregation functions, i.e. average (AVG), sum (SUM), and maximum (MAX) [8]. The decision to use these three aggregation functions is based on their frequent use and applicability for ratio scale measures [25]. Specifically, from the available aggregation functions in the study by Letouzey and Coq [25], we have preferred to use:

- MAX over MIN, because in many software metrics the minimum value is expected to be zero, and therefore, no variation would be detected;
- AVG over MEDIAN, because in many software metrics the median value is expected to be either zero or one, and therefore, no variation would be detected.

Although we acknowledge the fact that other more sophisticated aggregation functions exist, we have preferred to employ the most common and easy to use ones, in order to increase the applicability and generality of our research results.

5.1.2. Case selection and unit analysis

The case study of this paper is characterized as embedded [32], in which the context is the OSS domain, the subjects are the OSS

Table 2
Subjects and units of analysis.

Case	Category	# Classes	#Versions	AVG(LoC)
Art of Illusion	Games	749	32	9,306
Azureus Vuze	Communication	3,888	25	2,160,
Checkstyle	Development	1,186	36	9,627
Dr Java	Development	3,464	58	15,282
File Bot	Audio & Video	7,466	25	92,079
FreeCol	Games	794	41	6,593
FreeMind	Graphics	443	42	6,106
Hibernate	Database	3,821	51	23,753
Home Player	Audio & Video	457	32	4,913
Html Unit	Development	920	29	3,389
iText	Text Processing	645	23	54,857
LightweightJava Game Library	Development	654	42	8,485
ZDF MediaThek	Audio & Video	617	41	1,742
Mondrian	Databases	1,471	33	8,339
Open Rocket	Games	3,018	27	19,720
Pixelator	Graphics	827	33	3,392
Subsonic	Audio & Video	4,688	42	62,369
Sweet Home 3D	Graphics	341	25	6,382
Tux Guitar	Audio & Video	745	20	3,645
Universal Media Server	Communication	5,499	51	58,115

projects and the units of analysis are their classes, across different versions. In order to retrieve data from only high quality projects that evolve over a period of time, we have selected to investigate well-known and established OSS projects (see Table 2) based on the following criteria, aiming at selecting 20 OSS projects[2]:

c1 *The software is a popular OSS project in* ***Sourceforge.net***. This criterion ensures that the investigated projects are recognized as important by the OSS community, i.e. there is substantial system functionality and adequate development activity in terms of bug-fixing and adding requirements. To sort OSS projects by popularity, we have used the built-in sorting algorithm of sourceforge.net.

c2 *The software has more than* ***20 versions*** (official releases). We have included this criterion for similar reasons to c1. Although the selected number of versions is ad-hoc, it is set to a relatively high value, in order to guarantee high activity and evolution of the project. Also, this number of versions provides an adequate set of repeated measures as input to the statistical analysis phase.

c3 *The software contains more than* ***300 classes***. This criterion ensures that we will not include "*toy examples*" in our dataset. After data collection, a manual inspection of the selected projects has been performed so as to guarantee that the classes per se are not trivial.

c4 *The software is written in* ***java***. We include this criterion because the employed metric calculation tools analyze Java bytecode.

Building on the aforementioned criteria, we have developed the following selection process:

1. Sort Sourceforge.net projects according to their popularity (c1)—step performed on January 2014.
2. Filter java projects (c2).
3. For the next project, check the number of versions in the repository (c3).
4. If number of versions $>$ 20, download the most recent version of the project, and check the number of classes (c4).
5. If number of classes $>$ 300, then pick the project as a case for our study (c4).
6. If the number of projects $<$ 20, go back to step 3, if not, the case selection phase is completed.

[2] We aimed at selecting data for 20 OSS projects, to ensure the existence of enough cases for an adequate statistical analysis.

In order to more comprehensively describe the context in which our study has been performed, we have analyzed our dataset through various perspectives, and provide various demographics and descriptive statistics. First, concerning the actual changes that systems undergo, we test if the selected subjects (i.e., OSS projects) conform to the Lehman's law of continuous growth [24], i.e., increase in number of methods. The results of our analysis suggest that in approximately 75% of transitions from one version to the other the number of methods has increased, whereas it remained stable in about 13%. Second, in Fig. 2, we present a visualization of various demographic data on our sample. Specifically, in Fig. 2a, we present a pie chart on the distribution of LoC, in Fig. 2b a pie chart on the distribution of developers, in Fig. 2c a pie chart on the distribution of years of development, and in Fig. 2d a pie chart on the distribution of downloads.

5.1.3. Data collection and pre-processing

As discussed in Section 3, we have selected two metric suites: Li a Henry [26] and QMOOD [6]. To automatically extract these metric scores we have used *Percerons Client* (retrieved from: www.percerons.com), a tool developed in our research group, which calculates them from Java bytecode. Percerons is a software engineering platform [4] created by one of the authors with the aim of facilitating empirical research in software engineering, by providing: (a) indications of componentizable parts of source code, (b) quality assessment, and (c) design pattern instances. The platform has been used for similar reasons in [5] and [22]. On the completion of data collection, each class (unit of analysis) was characterized by 19 variables. Each variable corresponds to one metric, and is a vector of the metric values for the 20 examined project versions.

We note that *Percerons Client* calculates metric values, even detailed-design metrics, from the source code of applications, whereas normally, such metrics would be calculated on design artifacts (e.g., class diagrams). Therefore, for the needs of this case study, we assume that: (a) design artifacts are produced with as many details as required in order to proceed with the implementation phase, and (b) source code implementation follows the intended design (i.e., there is no design drift). Supposing that these two assumptions hold, metrics calculated at source code level and detailed-design level will be equivalent. For example, the values for DIT, NOM and CIS would be the same regardless of the phase that they are calculated. A threat to validity originating from these assumptions is discussed in Section 8.

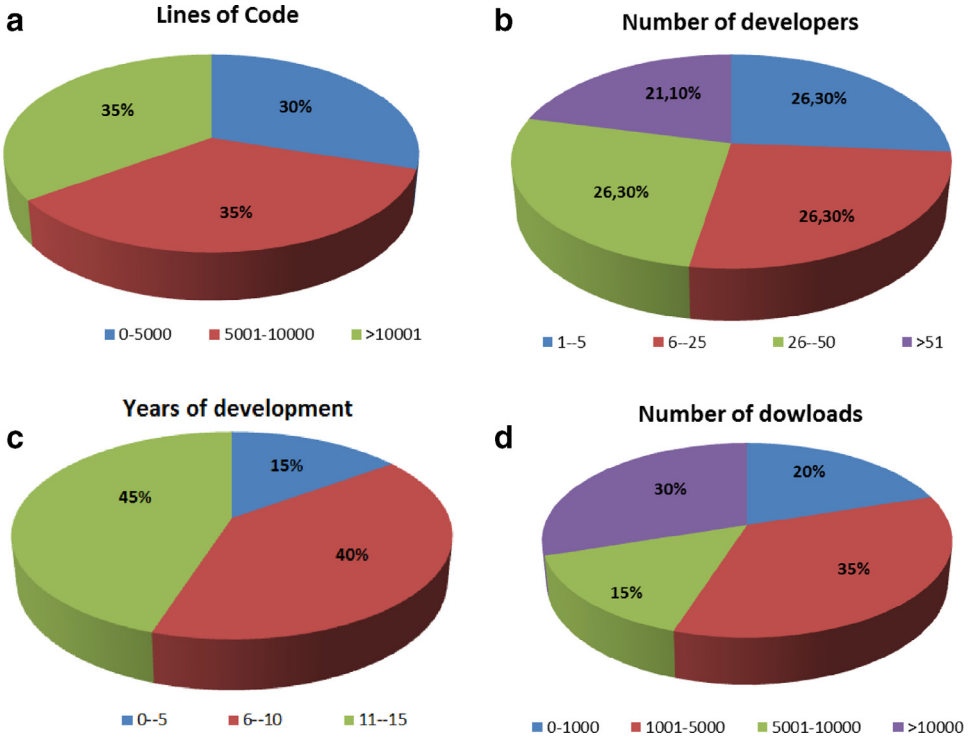

Fig. 2. Sample demographics.

Additionally, in order to be able to perform the employed statistical analysis (Software Metric Fluctuation—see Section 3), we had to explore an equal number of versions for each subject (OSS project). Therefore, since the smallest number of versions explored for a single project was 20, we had to omit several versions from all other projects (that had more than 20 versions). In order for our dataset to be as up-to-date as possible, for OSS projects with larger evolution history, in our final dataset we have used the 20 most recent versions. Finally, to answer RQ_2 we have created three datasets (one for each aggregation function—*MAX*, *SUM* and *AVG*), in which each one of the 20 cases was characterized by the same set of metrics. We note that for the DSC metric, only the *SUM* function is applicable, since both the use of *AVG* or *MAX* function at class level, would result to a system score of 1.00. Similarly, results on the NOH metrics could be explored only through the *SUM* and *AVG* aggregation functions.

5.1.4. Data analysis

In order to investigate the fluctuation of the considered metrics, we have used the *mf* measure (see Section 3), and hypothesis testing, as follows:

- We have employed *mf* for quantifying the fluctuation of metric scores retrieved from successive versions of the same project. On the completion of the data collection phase, we have recorded 20 cases (OSS software projects) that have been analyzed by calculating *mf* (across their 20 successive versions). In particular we have calculated *mf* for each metric score at system level, three times, one for each different aggregation function—MAX, SUM and AVG;
- We have performed paired sample *t*-tests [17] for investigating if there is a difference between the mean *mf* of different metrics (aggregated at system level with the same function) that quantify the same quality attribute;
- We have performed Friedman chi-square (x^2) ANOVA [17] for investigating if there is a difference between the mean *mf* of the same metric, using different aggregation functions. For identifying the differences between specific cases we have performed post hoc testing, based on the Bonferroni correction [17].

5.2. Results

In order to assess the fluctuation metrics and aggregation functions, in Table 3 we present the results of the mean *mf*, calculated over all projects and all versions with all three aggregation functions. The mean *mf* is accompanied by basic descriptive statistics like *min*, *max*, and *variance* [17]. For each quality attribute, we present the corresponding metrics, and the corresponding *mf*. We preferred not to set an *mf* threshold for characterizing a metric as stable or sensitive, but rather use a comparative approach. To this end, we consider comparable:

- Metrics that quantify the same quality attribute and have been aggregated with the same function, e.g. Compare avg(wmpc) versus Avg(nop); and
- The same metrics aggregated with different functions, e.g., avg versus Max.

In addition, in order to enable the reader to more easily extract information regarding each research question, we used two notations in Table 3:

- *The color of the cell* (applicable for metrics), represents if the specific metric is considered the most stable or the most sensitive within its group, based on the mean score. On the one hand, as most sensitive (see light grey cell shading), we characterize metrics that present the maximum *mf* value, regardless of the aggregation function—e.g. NOP. On the other hand, as most stable (see dark cell shading) we characterize those that present the minimum *mf*, regardless of the aggregation function—e.g. WMPC. We note that these characterizations are only based on descriptive statistics, and therefore are influenced by extreme values, corresponding to specific systems. A final assessment of the sensitivity of metrics will be provided

Table 3
Object-oriented metric fluctuation.

QA	Metric	Aggr. func.	Mean	Min	Max	Variance
Complexity	WMPC	**AVG**	**0.063**	0.005	0.315	0.005
		SUM	0.214	0.005	1.206	0.064
		MAX	*0.224*	0.000	0.608	0.041
	NOP	AVG	0.306	0.002	2.775	0.363
		SUM	*0.633*	0.004	5.057	1.566
		MAX	**0.227**	0.000	0.922	0.050
Cohesion	LCOM	**AVG**	**0.256**	0.006	0.994	0.085
		SUM	0.402	0.009	1.669	0.264
		MAX	*0.791*	0.000	4.725	1.517
	CAM	**AVG**	**0.109**	0.007	1.339	0.085
		SUM	*0.233*	0.006	1.249	0.071
		MAX	0.211	0.000	4.129	0.851
Inheritance	NOCC	**AVG**	**0.122**	0.005	0.551	0.019
		SUM	0.243	0.009	1.074	0.060
		MAX	*0.543*	0.000	5.965	1.739
	DIT	**AVG**	**0.072**	0.003	0.187	0.004
		SUM	*0.207*	0.005	0.469	0.018
		MAX	0.113	0.000	0.308	0.011
	NOH	**AVG**	**0.097**	0.006	0.345	0.012
		SUM	*0.172*	0.017	0.738	0.024
		MAX	N/A	N/A	N/A	N/A
	ANA	**AVG**	**0.149**	0.005	0.484	0.023
		SUM	*0.283*	0.008	0.962	0.068
		MAX	0.161	0.000	0.484	0.030
	MFA	AVG	0.289	0.000	1.255	0.139
		SUM	*0.390*	0.000	1.471	0.210
		MAX	**0.076**	0.000	0.967	0.049
Coupling	DAC	**AVG**	**0.459**	0.007	5.736	1.583
		SUM	0.491	0.015	5.335	1.356
		MAX	*0.550*	0.000	4.781	1.106
	RFC	**AVG**	**0.070**	0.003	0.301	0.006
		SUM	*0.181*	0.009	0.642	0.022
		MAX	0.114	0.000	0.487	0.017
	MPC	**AVG**	**0.125**	0.005	0.480	0.018
		SUM	0.214	0.008	0.740	0.031
		MAX	*0.345*	0.000	4.129	0.819
	DCC	**AVG**	**0.097**	0.006	0.229	0.007
		SUM	*0.217*	0.010	0.681	0.026
		MAX	0.126	0.000	0.375	0.015
	MOA	**AVG**	**0.113**	0.002	0.319	0.011
		SUM	*0.204*	0.004	0.693	0.033
		MAX	0.113	0.000	0.408	0.016
Size	NOM	*AVG*	*0.230*	0.190	0.292	0.001
		SUM	**0.180**	0.007	0.642	0.020
		MAX	0.207	0.000	1.005	0.081
	CIS	**AVG**	**0.092**	0.003	0.219	0.005
		SUM	0.201	0.006	0.601	0.018
		MAX	*0.231*	0.000	0.939	0.073
	DSC	AVG	N/A	N/A	N/A	N/A
		SUM	0.974	0.004	36.351	65.484
		MAX	N/A	N/A	N/A	N/A
	SIZE1	**AVG**	**0.079**	0.004	0.325	0.006
		SUM	0.169	0.009	0.450	0.012
		MAX	*0.182*	0.000	0.841	0.051
	SIZE2	**AVG**	**0.072**	0.005	0.227	0.004
		SUM	0.179	0.008	0.516	0.014
		MAX	*0.231*	0.000	1.151	0.087

after we examine the existence of statistically significant differences (see Table 4—Section 5.2.1).

- *Font style* (applicable for aggregation functions), emphasizes the combination of metrics and aggregation functions that produces the most stable/sensitive versions of the specific metric. For example, concerning WMPC, the MAX function is annotated with italic fonts, since it provides the highest *mf* value—most sensitive, whereas the AVG function (annotated with bold) provides the lowest *mf*—most stable.

The observations that can be made, based on Table 3, are discussed in Sections 5.2, after the presentation of hypotheses testing. Specifically, in Section 5.2.1 we further investigate the differences among metrics assessing the same quality attribute, whereas in Section 5.2.2, we explore the differences among different functions aggregating the scores of the same metric.

Table 4
Differences by quality attribute.

QA	Metric-1	Metric-2	AVG	SUM	MAX
Complexity	WMPC	NOP	−1.866	−1.697	−0.049
			0.07	0.10	0.96
Cohesion	LCOM	CAM	1.527	1.371	1.580
			0.14	0.18	0.13
Inheritance	NOCC	DIT	1.814	0.842	1.481
			0.08	0.41	0.15
		NOH	0.833	1.258	N/A
			0.41	0.22	
		ANA	−1.251	−1.905	1.266
			0.22	0.07	0.22
		MFA	−2.035	−1.537	1.604
			0.05	0.14	0.12
	DIT	NOH	−1.165	1.297	N/A
			0.29	0.21	
		ANA	−2.600	−1.847	−1.528
			0.02	0.08	0.14
		MFA	−2.631	−2.088	0.882
			0.02	0.05	0.39
	NOH	ANA	−2.098	−1.909	N/A
			0.05	0.07	
		MFA	−2.637	−2.479	N/A
			0.01	0.02	
	ANA	MFA	−1.834	−1.187	1.545
			0.08	0.25	0.14
Coupling	DAC	RFC	1.405	1.177	1.805
			0.18	0.25	0.09
		MPC	1.202	1.050	0.634
			0.24	0.31	0.53
		DCC	1.310	1.038	1.798
			0.21	0.31	0.09
		MOA	1.237	1.081	1.852
			0.23	0.29	0.08
	RFC	MPC	−2.538	−1.578	−1.118
			0.02	0.13	0.29
		DCC	−2.023	−3.034	−0.820
			0.06	0.00	0.42
		MOA	−2.574	−1.017	0.035
			0.02	0.32	0.97
	MPC	DCC	1.539	−0.204	1.054
			0.14	0.84	0.30
		MOA	0.523	0.411	1.121
			0.61	0.69	0.28
	DCC	MOA	−0.965	0.617	0.506
			0.35	0.54	0.62
Size	NOM	CIS	10.467	−1.611	−0.419
			0.00	0.12	0.68
		DSC	N/A	−0.987	N/A
				0.34	
		SIZE1	9.486	1.006	0.304
			0.00	0.33	0.76
		SIZE2	13.680	0.069	−0.481
			0.00	0.95	0.64
Size (cont.)	CIS	DSC	N/A	−0.980	N/A
				0.34	
		SIZE1	0.878	2.116	1.059
			0.39	0.05	0.30
		SIZE2	2.472	2.009	−0.004
			0.02	0.06	0.99
	DSC	SIZE1	N/A	0.993	N/A
				0.33	
		SIZE2	N/A	0.992	N/A
				0.33	
	SIZE1	SIZE2	0.575	−0.964	−0.634
			0.57	0.35	0.53

5.2.1. Differences in the fluctuation of metrics assessing the same quality attribute

To investigate if the results of Table 3 are statistically significant, we have performed paired sample *t*-tests for all possible comparable combinations of metrics (see Table 4). For each comparable cell (i.e., both metrics can be aggregated with the same aggregation function), we provide the *t*- and the sig value of the test. In order for a difference to be statistically significant, sig. should be less than 0.05 (see light grey cells). The sign of *t*-value, represents, which metric has a higher *mf*. Specifically, a negative sign suggests that the metric of the second column has a higher *mf* (i.e., is more sensitive) than the first. For example, concerning NOCC and MFA, the signs suggest that MFA is more sensitive when using the AVG function.

The main findings concerning RQ_1, are summarized in this section organized by quality attribute. From this discussion we have deliberately excluded metrics that cannot be characterized as most stable or sensitive w.r.t. the examined quality attribute (e.g., ANA—inheritance).

- ***Complexity***: Concerning complexity, our dataset includes two types of metrics: (a) one metric calculated at source code level (Weighted Methods per Class (*WMPC*)—based on number of control statements), and (b) one metric calculated at design level (Number of polymorphic methods (*NOP*)—based on a count of polymorphic methods. The results of the study suggest that number of polymorphic methods (*NOP*) is the most sensitive complexity measure, whereas weighted methods per class (*WMPC*) are the most stable one. However, this difference is not statistically significant.
- ***Cohesion***: Regarding cohesion, the cohesion among methods of a class (*CAM*) metric, which can be calculated on the detailed design, (defined in the QMOOD suite) is more stable than the lack of cohesion of methods (*LCOM*) metric that is calculated at source code level (defined in the Li and Henry suite). Similar to complexity, this result is not statistically significant.
- ***Inheritance***: The metrics that are used to assess inheritance are all calculated from design level artifacts. The most sensitive metrics related to inheritance trees are number of children classes (*NOCC*) and measure of functional abstraction (*MFA*), whereas the most stable are number of hierarchies (NOH) and depth of inheritance tree (DIT). The fact that DIT is the most stable inheritance metric is statistically significant, only when the AVG function is used.
- ***Coupling***: Coupling metrics are calculated at both levels of granularity. Specifically, data abstraction coupling (DAC) and measure of aggregation (MOA) are calculated at design-level, whereas message passing coupling (MPC), direct class coupling (DCC) and response for a class (RFC) are calculated at source code level. The most sensitive coupling metric is data abstraction coupling (DAC), whereas response for a class (RFC) and direct class coupling (DCC) are the most stable ones. The result on the stability of RFC is statistically significant, only with the use of AVG aggregation function.
- ***Size***: Concerning size we have explored five metrics, one on code level—lines of code (*SIZE1*), and four on design level—design size in classes (*DSC*), number of properties (*SIZE2*), class interface size (*CIS*) and number of methods (*NOM*). The number of properties (*SIZE2*) metric is the most stable size measure; whereas the most sensitive are number of methods (NOM) and class interface size (CIS).The results reported on the sensitivity of NOM are statistically significant concerning the AVG aggregation function.

5.2.2. Differences in metrics' fluctuation by employing a different aggregation function

Similarly to Section 5.2.1, in this section we provide the results of investigating the statistical significance of differences among the *mf* for the same metric, when using a different aggregation function. In Table 5, we present the results of an analysis of variance, and the corresponding post-hoc tests. Concerning the ANOVA we report the *F*-value and its level of significance (sig.), whereas for each post-hoc test only its level of significance. When the level of significance for the *F*-value is lower than 0.05, the statistical analysis implies that there is a difference between aggregation functions (without specifying in which pairs). To identify the pairs of aggregation functions that exhibit statistically significant differences, post hoc tests are applied. Statistically significant differences are highlighted by light grey shading on the corresponding cells.

The results of Table 5 suggest that the use of different aggregation functions can yield different fluctuations for the selected metrics, at a statistically significant level. Therefore, to provide an overview of the impact of the aggregation functions on metrics' sensitivity, we visualize the information through two pie charts, representing the frequency with which software metrics are found to be the most stable or the most sensitive (see Fig. 3).

For example, if someone aims at sensitive metrics, preferable choices is aggregation by *MAX* or *SUM* (50% and 44%, respectively), whereas *AVG* rarely produces sensitive results. On the other hand, AVG should be selected if someone is interested in stable metrics, since it yields the stable results for 83.3% of the cases. From these observations we can conclude that different aggregation functions can be applied to the same metric and change the fluctuation property of the specific metric.

5.3. Interpretation of results

Concerning the reported differences in the fluctuation of metrics assessing the same quality attribute, we can provide the following interpretations, organized by quality attribute:

- ***Complexity***: *NOP* is more sensitive than *WMPC*. This result can be interpreted from the fact that the calculation of *WMPC* includes an additional level of aggregation (from method to class), and the function that is used for this aggregation is the AVG. Based on the findings of this study, the AVG function provides relatively stable results, in the sense that in order to have a change of one unit in the aggregated *WMPC*, one control statement should be added in all methods of a class. Therefore, the change rate of *WMPC* value is relatively low.
- ***Cohesion***: *LCOM* is more sensitive than *CAM*. This result can be explained by the fact that the addition of a method in a class during evolution is highly likely to join some disjoint clusters of the cohesion graph[3], and therefore decrease the value of *LCOM*. Consequently, *LCOM* value is expected not to be stable during evolution.
- ***Inheritance***: The fact that *NOCC* is the most sensitive among the inheritance metrics is intuitively correct, since the addition of children is the most common extension scenario for a hierarchy. On the contrary, since only a few of these additions can lead to an increase of *DIT*, this metric is among the most stable ones. Similarly, *NOH* is not subject to many fluctuations, in the sense that adding or removing an entire hierarchy is expected to be a rather infrequent change.
- ***Coupling***: The observation that *MPC* is more sensitive coupling metric than *RFC* could be explained by the fact that MPC

[3] The calculation of the LCOM employed by van Koten and Gray [34] is based on the creation of a graph, in which nodes are methods, and edges are shared attributes. The number of disjoint graphs is LCOM.

Table 5
Differences by aggregation functions.

QA	Metric	x^2 test (sig.)	Post hoc tests		
			AVG-SUM	AVG-MAX	SUM-MAX
Complexity	WMPC	15.487 0.000	0.00	0.00	0.88
	NOP	8.380 0.015	0.03	0.90	0.08
Cohesion	LCOM	7.000 0.030	0.03	0.01	0.00
	CAM	28.900 0.000	0.00	0.01	0.00
Inheritance	NOCC	11.873 0.003	0.00	0.01	0.68
	DIT	24.025 0.000	0.00	0.06	0.00
	NOH	27.900 0.000	N/A	N/A	0.00
	ANA	18.405 0.000	0.00	0.94	0.01
	MFA	22.211 0.000	0.01	0.00	0.00
Coupling	DAC	1.848 0.397	0.16	0.70	0.71
	RFC	19.924 0.000	0.00	0.02	0.02
	MPC	9.139 0.010	0.00	0.10	0.13
	DCC	20.835 0.000	0.00	0.31	0.00
	MOA	15.718 0.000	0.00	0.43	0.00
Size	NOM	7.900 0.019	0.05	0.09	0.09
	CIS	18.231 0.000	0.00	0.00	0.45
	SIZE1	17.797 0.000	0.00	0.00	0.23
	SIZE2	15.823 0.000	0.00	0.01	0.68

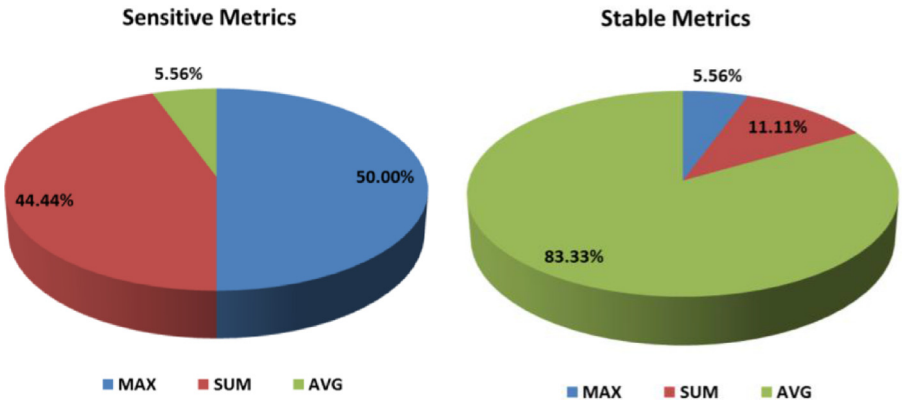

Fig. 3. Metrics sensitivity overview.

counts individual send messages, i.e., method invocations to other classes. This count can be affected even by calling an already called method. On the contrary, *RFC* (sum of method calls and local methods) is more stable, since it depends on the number of distinct method calls, and thus for its value to change a new method should be invoked.

- ***Size***: The fact that *NOM* and *CIS* are the most sensitive size metrics, was a rather expected result, in the sense that the addition/removal of methods (either public or not) is a very common change along software evolution. Therefore, the scores of these metrics are expected to highly fluctuate across versions. On the contrary, *SIZE1* (i.e., lines of code) has proven to be the most stable size metric, probably because of the large absolute values of this metric (we used only large projects), which hinder changes of a large percentage to occur frequently.

The results of the study that concern the differences in the fluctuation of metrics that are caused by switching among aggregation functions, have been summarized in Fig. 3, and can be interpreted, as follows:

- The fact that the *AVG* function provides the ***most stable*** results for 83% of the metrics (all except from *NOP, MFA* and *NOM*), can be explained by the fact that most of the projects were quite large, in terms of number of classes. Therefore changes in the

numerator (sum of changes in some classes) could not reflect a significant difference in the *AVG* metric scores at system level. Thus, replicating the case study on smaller systems might be useful for the generalizability of our results.
- The fact that both *MAX* and *SUM* functions provide the ***most sensitive*** versions for almost an equal number of metrics, suggests that these functions do not present important differences. However, specifically for source code metrics, it appears that the *MAX* function, provides more sensitive results. This result can be considered intuitive, in the sense that source code metrics are changing more easily, and produce larger variations, from version to version, compared to design level metrics. For example, changes in number of lines are more frequent and larger in absolute value than changes in the number of classes or methods. Thus, the likelihood of the maximum value of a metric change is higher in source code metrics, rather than design-level ones.

6. Case study on the usefulness of SMF in metrics selection

In order to validate the ability of SMF to aid software quality assurance teams in the metric selection process, we conducted a case study with 10 software engineers. In particular, we investigated if software engineers are able to intuitively assess metric fluctuation without using SMF, and how accurate this assessment is, compared to SMF. In Section 6.1, we present the case study design, whereas in Section 6.2 the obtained results.

6.1. Study design

With the term case study we refer to an empirical method that is used for monitoring processes in a real-life context [32]. For this reason, we have performed a case study simulating the process of metric selection. The case study of this paper has been designed and is presented according to the guidelines of Runeson et al. [32].

6.1.1. Objectives and research questions

The goal of this case study, stated here using the Goal-Question-Metrics (GQM) approach [7], is to *"**analyze** the SMF property **for the purpose of** evaluation **with respect to** its usefulness **in the context of** the software metrics selection process **from the point of view of** software engineers"*. The evaluation of the SMF property has been focused on two specific directions:

RQ$_1$: *Do software engineers have a uniform perception on the fluctuation of software metrics when not using the SMF property?*
RQ$_2$: *Does SMF provide a more accurate prediction of the actual metric fluctuation, compared to the intuition of software engineers?*

The first research question aims at investigating the need for introducing a well-defined property for quantifying metric fluctuation. In particular, if software engineers have diverse perception of what the fluctuation of a specific metric is, then there is a need for guidance that will enable them to have a uniform way of assessing metrics' fluctuation. The second research question deals with comparing: (a) the accuracy of software engineers' opinion when ranking specific combinations of metrics and aggregation functions subjectively, i.e. without using the SMF property, with (b) the accuracy of the ranking as provided objectively by the SMF property.

6.1.2. Case selection and data collection

To answer the aforementioned questions, we will compile a dataset in which rows will be the cases (i.e., combinations of metrics and aggregation functions) and columns will be: (a) how software engineers perceive metric fluctuation, (b) the metric fluctuation as quantified through SMF, and (c) the actual metric fluctuation. The case selection and data collection processes are outlined below.

6.1.2.1. Case selection. In order to keep the case study execution manageable we have preferred to focus on one quality attribute. Having included more than one quality attributes would increase the complexity of the metrics selection process, and would require more time for the execution of the case study. From the metrics described in Table 1, we have decided to focus only on the coupling quality attribute since that would offer:

- ***A variety of metrics***. We have selected a quality attribute that could be assessed with multiple metrics. Therefore, we have eliminated *complexity* and *cohesion* QAs.
- ***Metric calculation at both the source code and detailed-design level***. We have excluded the *inheritance* QA, since all related metrics can be calculated at the detailed-design phase. None of the metrics can be only calculated at the source code level.
- ***Metrics whose calculation is not trivial.*** To increase the realism of the metric selection process we have preferred to exclude from our case study the metrics quantifying the *size* QA, since their calculation is trivial.

Therefore, and by taking into account that we have used three aggregation functions (AVG, MAX, and SUM—as explained in Section 5.1.1) and five coupling metrics (DCC, MOA, DAC, MPC, RFC—as presented in Table 1), our dataset consists of 15 cases.

6.2.1.2. Data collection. For each one of the aforementioned 15 cases, we have collected 12 variables (i.e., columns in the dataset). The first 10 variables ($[V_1]$–$[V_{10}]$) represent the perception of software engineers on metrics fluctuations, whereas the other two: the fluctuation based on SMF ($[V_{11}]$) and the actual *mf*, which is going to be used as the basis for comparison ($[V_{12}]$).

Perception of software engineers on metrics fluctuation. To obtain these variables we have used a well-known example on software refactoring [18], which provides an initial system design (see Fig. 4a) and a final system design (see Fig. 4b), and explains the refactoring that have been used for this transformation. The aforementioned designs, accompanied with the corresponding code, have been provided to 10 software engineers (i.e., subjects). The case study participants possess at least an MSc degree in computer science and have a proven working experience as software engineers in industry (see Table 6).

The subjects have been asked to order the combinations of metrics and aggregation functions from 1st to 15th place, i.e. from the most stable (1st place) to the most sensitive (15th place), based on the influence of these changes to the metric scores. The 1–15 range has been used to discriminate between all possible metric/aggregation function combinations of the study. For example, an engineer who considers that metric *M* and aggregation function *F* captures most of the changes that have been induced on coupling, would assign the value 15 to that metric/function combination. For the most stable metric/function combination, he/she would assign the value 1. These rankings have been mapped to variables: $[V_1]$–$[V_{10}]$, one for each subject of the study. We note, that in order to increase the realism of the case study, we have not allowed the participants to make any calculation on paper, since this would not be feasible in large software systems. We note that in case of an equal value, fractional ranking has been performed: items that are equally ranked, receive the same ranking number, which is the mean of the ranking they would have received, under ordinal rankings. For example, if item *X* ranks ahead of items *Y* and *Z* (which compare equal), ordinal ranking would rank *X* as "1", *Y* as "2" and *Z* as "3" (*Y* and *Z* are arbitrarily ranked). Under fractional ranking, *Y* and *Z* would each get ranking number 2.5. Fractional ranking has the property that the sum of the ranking numbers is the same as under ordinal ranking. For this reason, it is used in statistical tests [15].

SMF ranking. The ranking by SMF (column $[V_{11}]$), is based on the empirical results obtained from our case study on 20 open

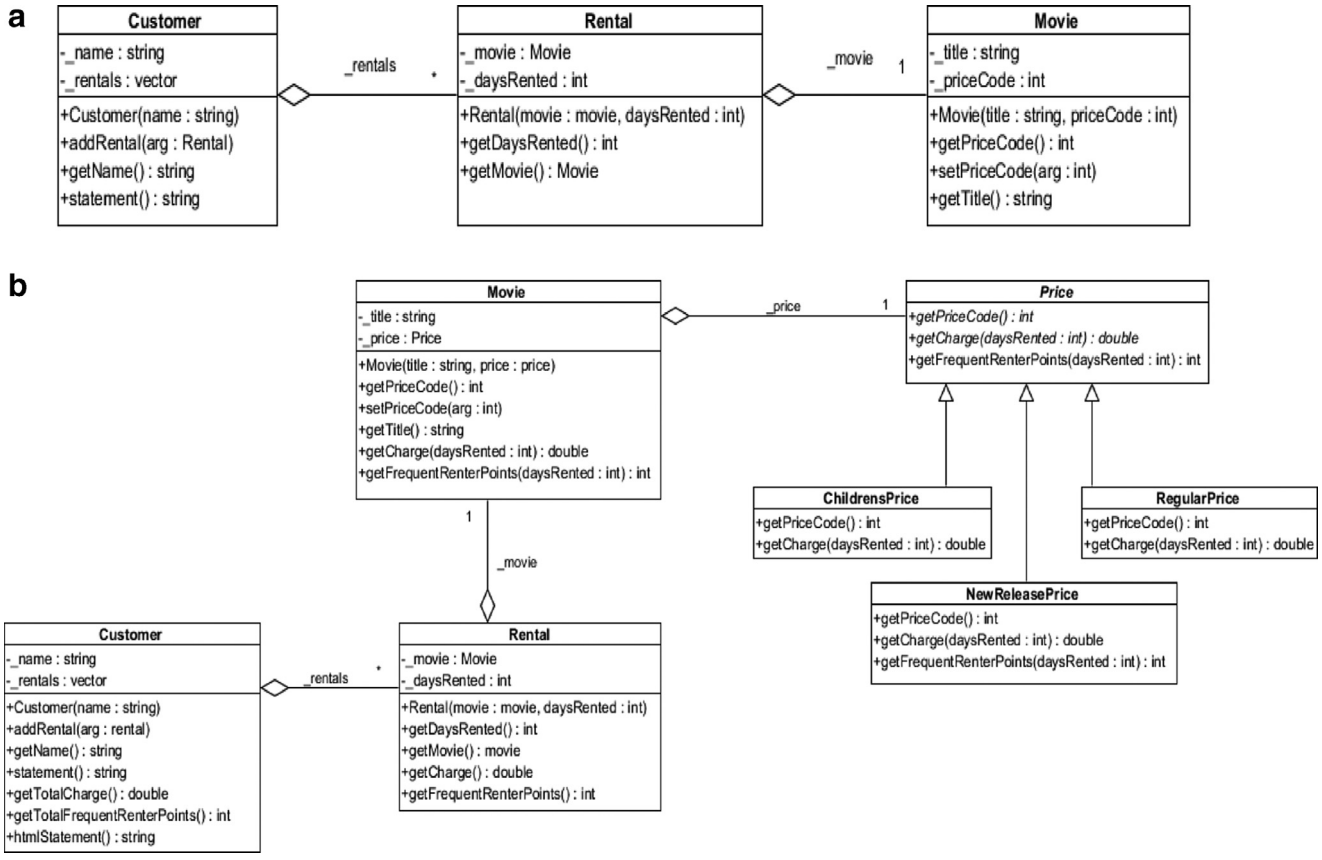

Fig. 4. Movie club (initial and final design) [18].

Table 6
Subjects' demographics.

	AVG (SD)		
Age	31.3 (±8.42)		
Development experience in years	7.8 (±4.34)		
	Frequency		
	BSc	**MSc**	**PhD**
Degree	2	7	1
	Design	**Code**	**Research**
Type of experience	6	8	7
	Web/Mobile	**Scientific**	**Desktop Applications**
Application domain	5	9	5

source projects. In particular it has been extracted by sorting the mean metric fluctuation as presented in the 4th column of Table 3.

Actual ranking. Finally, in order to record $[V_{12}]$, we have calculated the actual metric fluctuation from the initial to the final system, based on the formula provided in Section 4. Although the values of $[V_{12}]$ have originally been numerical, we transformed them to ordinal ones (i.e., rankings), so as to be comparable to $[V_1]$–$[V_{11}]$. Similarly to $[V_1]$–$[V_{10}]$, equalities have been treated using the fractional ranking strategy [15]. The final dataset of this case study is presented in Table 7. It should be noted that SMF ranking does not perfectly match the actual ranking, because it has been derived by the metric fluctuation recorded in the case study presented in Section 5, i.e., in a different set of projects.

6.1.3. Data analysis

To answer RQ_1 we have performed correlation analysis by extracting the intra-class correlation (ICC) on variables $[V_1]$–$[V_{10}]$ to check the inter-rater agreement [17]. In particular, we have used the average ICC in order to get an estimate of the average correlation of rankings between the subjects. To answer research RQ_2, we have extracted Spearman correlation [17], between variables $[V_1]$–$[V_{11}]$ with $[V_{12}]$. The decision to apply a correlation analysis is based on the 1061 IEEE Standard for Software Quality Metrics Methodology [1], which suggests that a sufficiently strong correlation "determines whether a metric can accurately rank, by quality, a set of products or processes (in the case of this study: a set of metrics)".

In order to interpret the values obtained by the correlation analysis, we have used the thresholds provided by Marg et al. [27], which suggest that a correlation coefficient higher than 0.7 correspond to very strong relationships, correlations coefficients between 0.4 and 0.7 represent strong relationships, and correlation coefficients between 0.3 and 0.4 correspond to moderate relationships.

6.2. Results

By performing the aforementioned analysis on the data of Table 7, we have been able to answer the research questions stated

Table 7
Case study dataset.

Function	Metric	Actual fluctuation	Actual ranking [V_{12}]	SMF ranking [V_{11}]	[V_1]	[V_2]	[V_3]	[V_4]	[V_5]	[V_6]	[V_7]	[V_8]	[V_9]	[V_{10}]
AVG	MOA	0.00%	2	3	3	2	10	4	1	1	12	4	3.5	8.5
AVG	RFC	0.00%	2	1	8	4.5	1	1	4	4	1	7	1.5	10
MAX	RFC	0.00%	2	5	4	7	3	2	6	8	3	9	6.5	12
AVG	MPC	4.31%	4	6	6	8.5	4	10	10	5.5	7	8	5	4
AVG	DCC	8.33%	5	2	11	2	7	7	9	3	4	2	1.5	4
MAX	MOA	26.09%	6	4	1	2	12	5	3	2	10	1	3.5	6.5
MAX	MPC	33.33%	7	12	7	8.5	6	11	8	7	8	10	8	1.5
MAX	DCC	80.43%	8	7	10	4.5	9	8	7	5.5	6	3	6.5	1.5
SUM	RFC	91.38%	9	8	2	11.5	2	3	5	12	2	11	12	11
SUM	MPC	100.00%	10,5	10	9	11.5	5	12	12	10	9	12	10	6.5
SUM	MOA	100.00%	10,5	9	5	6	11	6	2	11	11	5	10	8.5
SUM	DCC	116.67%	12	11	12	10	8	9	11	9	5	6	10	4

Table 8
Metrics selection accuracy (Software engineers perception versus SMF-based).

Evaluator	Correlation coeff.	Sig.
#1	0.325	0.30
#2	0.563	0.05
#3	0.282	0.37
#4	0.543	0.06
#5	0.381	0.22
#6	0.689	0.01
#7	0.176	0.58
#8	0.173	0.59
#9	0.789	0.00
#10	−0.314	0.32
AVG (Evaluator)	0.361	N/A
SMF	0.794	0.00

in Section 6.1.1. First, concerning the correlation of the rankings provided by the software engineers, the analysis showed that the average ICC coefficient equals –0.227. The negative correlation coefficient suggests that there is no reliability in the way that software engineers intuitively assess the fluctuation of a metric. Therefore, there is a need for a property that objectively characterizes this metric property, and consequently guides software engineers in the metrics selection process.

In addition, in order to investigate if the guidance that SMF provides is more accurate than the intuition of a software engineer, we have extracted the Spearman rank correlation, and we present the results in Table 8. The results of Table 8 suggest that the ranking provided by SMF is more accurate in terms of correlation with the actual metric fluctuation (i.e., 79.4%), than the ranking provided by the intuition of software engineers.

Additionally, we can also observe that only three evaluators have been able to predict with a statistically significant accuracy the ranking of metrics with respect to their fluctuation. Although this result might suggest that these software engineers are not in great need of guidance, they represent only the 30% of the sample. The rest of the software engineers had performed a rather poor prediction, i.e., a correlation coefficient ranging from –0.314 to 0.543. For example, Evaluator #10, has characterized as the most sensitive metric MAX(RFC)—which in practice has not changed its value in the provided example, whereas as the most stable MAX(DCC)—in practice had an approximately 80% change.

Consequently, SMF can be characterized as a useful property in the metric selection process, for three reasons: (a) software engineers are in need of a property that can objectively characterize the degree of metrics fluctuation—since different software engineers perceive different metrics as stable or sensitive; (b) the use of SMF is leading to the most accurate prediction of metric fluctuation, compared to the intuition of software engineers; and (c) only 30% of our study's subjects (i.e., experienced software engineers) have been able to rank the metrics in a way that was statistically significant correlated with the actual metric ordering—the large majority of software engineers would perform inadequate metric selection, in terms of fluctuation, without the use of SMF.

7. Implications for researchers and practitioners

In order to provide an outcome that is directly exploitable by both researchers and practitioners, during the development of their measurement plans (see details in Section 1), we created a pivot table (see Table 9). The discussion of the results will be guided by two factors that the quality assurance teams should take into account when developing the measuring plan: (a) the decision to measure in-large or measure in-small ("*what to measure?*"), and (b) the development phase in which each metric is calculated ("*when to measure?*"). The characterization of any metric as *stable* or as *sensitive* can prove beneficial concerning the metrics selection process as follows:

- **Concerning the "what to measure" question:** On the one hand, metrics that have been characterized as sensitive are less fitting than stable metrics for measure-in-large [16] evaluations, because of the numerous and large fluctuations that hinder the ability to derive the overall trend of the corresponding metric. On the other hand, sensitive metrics are considered more fitting for measure-in-small evaluations, e.g., to evaluate the effect of a refactoring activity or the application of a design pattern, because they are more sensitive to code changes, than the more stable ones.
- **Concerning the "when to measure" question:** The answer to this question is influenced by both the decision on "what to measure?" and the available metrics' fluctuation. For example, suppose a case in which we want to select a metric that is sensitive (e.g., we are interested in performing a measure-in-small evaluation); if, in addition, for the specific quality attribute all design-level metrics are stable, then we should perform the evaluation at the implementation level.

The dimensions of Table 9 represent the main options concerning *when* and *what* to measure questions, whereas the content of cells present the optimal combination of metric and aggregation function with respect to fluctuation (*how to measure* question). From the pivot table (and specifically from the *when to measure* dimension), we deliberately excluded the measurement frequency factor, since fluctuation is relevant only for repeated measures.

The results of this paper, as summarized in Table 9, can be exploited by both researchers and practitioners, as follows:

- Researchers can perform metric selection, based on metrics fluctuation and the scope of their projects. More specifically, in

Table 9
Overview of suitable metrics/aggregation functions.

What to measure **(granularity/QA)?**		***When to measure?***	
		Design	**Implementation**
Measure in-small (*sensitive metrics are desired*)	*Complexity*	SUM(NOP)	MAX(WMC)
	Cohesion	SUM(CAM)	MAX(LCOM)
	Inheritance	MAX(NOCC)	N/A
	Coupling	MAX(DAC)	MAX(MPC)
	Size	MAX(CIS)	MAX(SIZE1)
Measure in-large (*stable metrics are desired*)	*Complexity*	MAX(NOP)	AVG(WMC)
	Cohesion	AVG(CAM)	AVG(LCOM)
	Inheritance	AVG(DIT)	N/A
	Coupling	AVG(MOA)	AVG(RFC)
	Size	AVG(SIZE2)	AVG(SIZE1)

cases that researchers wish to evaluate *design* or *implementation decisions* that have a ***local effect*** (e.g. refactorings or design patterns) and are interested in their effect on system level, they should prefer a ***sensitive*** metric. To this end, the results indicate that sensitive combinations of metrics and aggregation functions exist for all explored quality attributes.

- Researchers can use the proposed ***fluctuation evaluation***, while *introducing software quality metrics*, and accompany their empirical assessment with that evaluation. This can be done in addition to evaluating the validity of the metric, by using well-known international standards [1] or established SE guidelines [9,10].
- Practitioners can perform metric selection, based on metrics fluctuation and their ***software quality measuring plan***. After selecting: (a) the *quality attribute* they want to assess, and (b) if they want to *measure in-small*, or *measure in-large*, they can decide if they will use a sensitive or a stable metric. Then, based on the findings of this study, they can filter the available metrics from their quality dashboard. We note that metric selection cannot be blindly based on SMF, since the main properties of metrics should be considered as well. For example, if a software development team wishes to quantify a specific aspect of size, e.g., the number of classes in a system, they will use the DSC (Design Size in Classes) metric, regardless of its fluctuation. Nevertheless, if for example a software quality assurance team wants to quantify the change in coupling after a refactoring (without focusing on a specific type of coupling), the use of SMF can optimize the metric selection decision.
- When practitioners are ***interested in producing stable*** versions of *code-level metrics*, they should employ the *AVG* function, whereas, when they are ***interested in sensitive*** versions, they should use the *MAX* function. Similarly, regarding *design-level metrics*, stable versions of metrics are more frequently produced by using the *AVG* function. In the case of using the *SUM* or *MAX* aggregation function, a case-by-case examination is needed.
- Practitioners can include in their ***quality dashboards*** various *views of the same metric* (same metric with different aggregation functions), since they provide different information, and therefore can be exploited under different circumstances.

8. Threats to validity

This section discusses construct, conclusion, and external validity for this study. Internal validity is not applicable as the study does not examine causal relations. Furthermore, we study conclusion validity instead of reliability, as in purely quantitative case studies, the room for researcher bias is rather limited, if not zero, eliminating any threats to reliability [35].

A threat to construct validity is that the obtained *mf* and the conclusions regarding the fluctuation are dependent upon the actual changes that have been performed in each system. In other words, the fact that SMF is quantified and discussed based on the results of a case study implies that the fluctuation property of a specific metric is dependent upon the examined systems, as opposed to a property like monotonicity which can be assessed mathematically and independently from the systems on which the metric is applied. For example, a system with no changes at all would imply that a metric is stable, whereas stability is due to the lack of changes, and not a property of the metric itself. In order to mitigate this threat, during project selection we mined only active projects that were substantially different across successive releases. We manually validated these differences by inspecting the SIZE1 (Lines of Code) metric score. In addition, a possible threat to the validity of our results is the bias that might be caused by the time period between versions for different projects, and the differences in projects' development team size, which might affect the load of system changes. However, we believe that this threat is mitigated by the variance of these factors in our sample, as presented in Fig. 2.

Moreover, the current assessment of design-level metrics fluctuation has been performed on the source code of applications, and not on design artifacts. The metrics calculated on source code and design artifacts, can be considered as equivalent if the following two assumptions hold: (a) the design artifacts are fully detailed, and (b) there is no design drift. These assumptions do not usually hold in practice, as the level of detail of design artifacts and the degree of design drift varies across projects. For example, one could produce a class diagram only with class names and basic associations, while another could produce class diagrams with details in the level of even `getters` and `setters`. However, assessing metrics' fluctuation on these extreme cases would possibly provide different results, which would not be reliable, in the sense that they would be mostly related to the artifact level of detail, rather than the metrics' properties.

Furthermore, concerning conclusion validity, we need to note that the rather small sample size in the case study assessing the fluctuation of object-oriented metrics might to some extent influence the characterization of metrics as either sensitive or stable. However, we believe that the diversity of projects' characteristics as presented in Fig. 2, is sufficiently mitigating any effect from the sample size of investigated OSS projects. Finally, concerning threats to external validity, we have identified two issues. First, the results are heavily dependent on the size of the projects in terms of classes (see Section 4: RQ_2) and therefore cannot be generalized to smaller projects. However, this threat cannot be considered crucial, in the sense that quality assurance is more relevant to larger projects. Secondly, our dataset only included Java projects, and therefore results cannot be generalized to other programming languages, where different principles exist. For example, concerning the results on ANA, the results might be different for C++ projects, in which multiple inheritance is allowed.

9. Conclusions

Metric selection for long-term monitoring of software quality is a multi-criteria decision making process, in the sense that the software quality assurance team should agree on the important quality attributes, the frequency with which the measurement should be performed, and the scale to which it should be applied (system-wide or local). Apart from its inherent complexity, this process becomes even more challenging due to the plethora of existing software metrics attached to each quality attribute. To assist the metric selection process, in this paper we define a new metric property, namely fluctuation, as the degree to which a metric is able to capture changes in the underlying structure of the software system. Based on this definition, we investigated the fluctuation of 19 object-oriented metrics, through a case study on 20 open-source software projects. The results of the study indicated that source code metrics are in principle more sensitive than design level metrics, and that there are specific metrics that when used with different aggregation functions can provide both sensitive and stable measures of the investigated quality attributes. Finally, scenarios of use for the main results of this paper have been presented from both practitioners' and researchers' point of view.

As future work, we plan to replicate this study with different metrics (e.g., architecture ones) and compile a comprehensive list of sensitive and stable metrics that can be used in design and implementation phases. Additionally, interesting follow-up for this study would be to investigate the relationship of metrics fluctuation with the growth rate of the project. Therefore, we would be able to examine if the results are differentiated for smaller projects and gain interesting insight to metrics' fluctuation. Finally, we plan to evaluate the usability of taking into account metric fluctuation while producing measurement plans in industrial context.

References

[1] IEEE Std 1061-1998, IEEE standard for a software quality metrics methodology, IEEE Comput. Soc. 31 (December 1998).
[2] J. Al Dallal, Mathematical validation of object-oriented class cohesion metrics, Int. J. Comput. 4 (2) (2010) 45–52.
[3] J. Al Dallal, L. Briand, A precise method-method interaction-based cohesion metric for object-oriented classes, Trans. Softw. Eng. Methodol. ACM 21 (2) (March 2012).
[4] A. Ampatzoglou, O. Michou, I. Stamelos, Building and mining a repository of design pattern instances: Practical and research benefits, Entertainment Comput. Elsevier. 4 (2) (April 2013) 131–142.
[5] A. Ampatzoglou, A. Gkortzis, S. Charalampidou, P. Avgeriou, An embedded multiple-case study on OSS design quality assessment across domains, in: 7th International Symposium on Empirical Software Engineering and Measurement (ESEM' 13), ACM/IEEE Computer Society, Baltimore, USA, 10-11 October 2013, pp. 255–258.
[6] J. Bansiya, C.G. Davies, A hierarchical model for object-oriented design quality assessment, IEEE Comput. Soc. Trans. Softw. Eng. 28 (1) (January 2002) 4–17.
[7] V.R. Basili, G. Caldiera, H.D. Rombach, Goal question metric paradigm, Encyclopedia of Software Engineering, John Wiley & Sons, 1994, pp. 528–532.
[8] G. Beliakov, A. Pradera, T. Calvo, Aggregation Functions: A Guide for Practitioners, Springer, 2008.
[9] L. Briand, J.W. Daly, J.K. Wüst, A unified framework for coupling measurement in object-oriented systems, IEEE Comput. Soc. Trans. Softw. Eng. 25 (1) (January 1999) 91–121.
[10] L. Briand, J. Daly, J. Wüst, A unified framework for cohesion measurement in object-oriented systems, Empirical Softw. Eng., 3 (1) (1998) 65–117.
[11] P.M.T. Broersen, Automatic Autocorrelation and Spectral Analysis, Springer, 2006.
[12] M.J. Campbell, T.D.V. Swinscow, Statistics at Square One, Wiley-Blackwell, 2009.
[13] A. Chatzigeorgiou, E. Stiakakis, Combining metrics for software evolution assessment by means of data envelopment analysis, J. Softw. Maintenance Evol.: Res. Pract. 25 (3) (March 2013) 303–324.
[14] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, Managerial use of metrics for object oriented software: an exploratory analysis, IEEE Comput. Soc. Trans. Softw. Eng. 24 (8) (August 1998) 629–639.
[15] P. Cichosz, Data Mining Algorithms: Explained Using R, Wiley, 2015.
[16] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd Edition, Pws Pub Co, 1996.
[17] A. Field, Discovering Statistics using IBM SPSS Statistics, SAGE Publications, 2013.
[18] M. Fowler, Refactoring: Improving the Design of Existing Code, 1st Edition, Addison-Wesley, 1999.
[19] R. Geetika, P. Singh, Dynamic coupling metrics for object oriented software systems: A survey, SIGSOFT Softw. Eng. Notes, ACM 39 (2) (March 2014) 1–8.
[20] T. Gîrba, S. Ducasse, M. Lanza, Yesterday' s weather: guiding early reverse engineering efforts by summarizing the evolution of changes, in: 20th International Conference on Software Maintenance (ICSM' 04), IEEE Computer Society, Chicago, USA, 11-14 September 2004, pp. 40–49.
[21] O. Gómez, H. Oktaba, M. Piattini, F. García, A systematic review measurement in software engineering: state-of-the-art in measures, Communications in Computer and Information Science (CCIS), 10, Springer, 2008, pp. 165–176.
[22] I. Griffith, C. Izurieta, Design pattern decay: the case for class grime, in: 8th International Symposium on Empirical Software Engineering and Measurement (ESEM '14), ACM/IEEE Computer Society, Torino, Italy, 18-19 September 2014.
[23] J.C. Hull, Options, Futures and Other Derivatives, Prentice-Hall, New Jersey, USA, 1997.
[24] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, W.M. Turski, Metrics and laws of software evolution-the nineties view, in: 4th International Software Metrics Symposium (METRICS 1997), IEEE Computer Society, Albuquerque, New Mexico, 5-7 November 1997, pp. 20–32.
[25] J.L. Letouzey, T. Coq, The SQALE analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code, in: 2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID' 10), IEEE Computer Society, Nice, Paris, 22-27 August 2010, pp. 43–48.
[26] W. Li, S. Henry, Object-oriented metrics that predict maintainability, J. Syst. Softw., 23 (2) (November 1993) 111–122.
[27] L. Marg, L.C. Luri, E. O'Curra, A. Mallett, Rating evaluation methods through correlation, in: 1st Workshop on Automatic and Manual Metrics for Operational Translation Evaluation (MTE' 14), Reykjavik, Iceland, 26 May 2014.
[28] T. Mens, S. Demeyer, Future trends in software evolution metrics, in: 4th International Workshop on Principles of Software Evolution (IAWPSE '01), ACM, Vienna, Autria, 10-14 September 2001, pp. 83–86.
[29] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, S. Ducasse, Software quality metrics aggregation in industry, J. Softw. Evol. Proccess, 25 (10) (Octomber 2012) 1117–1135.
[30] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, I.H. Moghadam, Experimental assessment of software metrics using automated refactoring, in: 6th IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2012), ACM, Lund, Sweden, September 2012, pp. 49–58.
[31] M. Riaz, E. Mendes, E. Tempero, A systematic review on software maintainability prediction and metrics, in: 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09), IEEE Computer Society, Florida, USA, 15-16 October 2009, pp. 367–377.
[32] P. Runeson, M. Host, A. Rainer, B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, John Wiley & Sons, 2012.
[33] A. Serebrenik, M. van den Brand, Theil index for aggregation of software metrics values, in: 26th IEEE International Conference on Software Maintenance (ICSM' 10), IEEE Computer Society, Timisoara, Romania, 12–18 September 2010, pp. 1–9.
[34] A. van Koten, A.R. Gray, An application of Bayesian network for predicting object - oriented software maintainability, Inf. Softw. Technol. 48 (1) (January 2006) 59–67.
[35] C. Wohlin, M. Host, P. Runeson, M. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2012.
[36] Y. Zhou, H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, J. Syst. Softw. 80 (8) (2007) 1349–1361.