Characterizing the Evolution of Size and Inheritance in Object-Oriented Software

Bruno L. Sousa
Department of Computer Science - UFMG
Belo Horizonte, Minas Gerais, Brazil
bruno.luan.sousa@dcc.ufmg.br

Mariza A. S. Bigonha
Department of Computer Science - UFMG
Belo Horizonte, Minas Gerais, Brazil
mariza@dcc.ufmg.br

Kecia A. M. Ferreira
Department of Computing - CEFET-MG
Belo Horizonte, Minas Gerais, Brazil
kecia@decom.cefetmg.br

Glaura C. Franco
Department of Statistics - UFMG
Belo Horizonte, Minas Gerais, Brazil
glaura@est.ufmg.br

ABSTRACT

Software evolution consists of adapting, correcting, and updating a system. It is widely known that the systems became increasingly complex and challenging to be maintained over their evolution. However, the way the systems' internal structure evolves is not known in detail. Understanding how the internal software structure evolves is essential to help developers better plan, manage, and perform software maintenance tasks. In this work, we present an empirical analysis to investigate how the internal structure of object-oriented systems evolves from the perspective of inheritance hierarchy and size. Besides, we analyzed the set of classes within the systems that affect these dimensions' evolution and how such classes evolve. We used a methodology based on time series, linear regression techniques, and trend tests to analyze the evolution of object-oriented systems. Using this methodology, we identified the function that better explains how the size and the inheritance tree evolve. This study revealed eight software evolution properties, among them: inheritance hierarchy tends to increase in depth and decrease in breadth; inheritance hierarchy depth and the size of classes grow according to a linear model. The empirical evidence found in this work provides a fine-grained knowledge of how object-oriented software systems' internal structure evolves from internal dimensions.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Software Evolution, Software Metrics, Inheritance, Size, Time Series

SBQS '21, November 08–11, 2021, Vitória, ES

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Bruno L. Sousa, Mariza A. S. Bigonha, Kecia A. M. Ferreira, and Glaura C. Franco. 2018. Characterizing the Evolution of Size and Inheritance in Object-Oriented Software. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Software evolution consists of developing, maintaining, and updating software systems [34]. If a software system does not evolve, it risks losing market share to competitors [39]. Nevertheless, software maintenance is a challenging, complex, and time-consuming task. During a software life cycle, the internal software structure usually suffers several changes to accommodate the modifications and meet the users' demands, resulting in high costs. Thus, it is essential to understand how the systems' internal structure evolves to control and reduce the software costs and efforts.

To provide empirical evidence about how software evolution occurs, Lehman et al. [33–39] analyzed the evolution of a system and identified eight characteristics of software evolution. Such characteristics are known as Lehman's laws and became a landmark on this topic. Researchers have investigated how the software systems evolve from the internal aspects, e.g., inheritance hierarchy [9, 13, 24, 41, 44, 50] and size [5–8, 20, 21, 25–28, 32, 47]. Although the literature has investigated how software systems evolve, there is still no exact characterization of how this evolution occurs. For instance, the works have investigated inheritance and its impact on the internal software quality [9, 13, 24, 41, 44, 50]. However, they have not detailed how the evolution of this aspect occurs or which pattern it follows. The evolution pattern of size in software systems is open in the literature since there is no well-defined pattern on how this aspect evolves. The size's pattern has already been defined as super-linear [20, 21, 27, 32], sub-linear [5, 8, 28], linear [47], and following the Pareto distribution [26]. Considering that all these gaps still existing in the literature have motivated us to investigate and detail how software systems' internal dimensions evolve.

This work inspects accurately how object-oriented software evolves considering two dimensions: inheritance hierarchy and size. We aim to identify the evolution pattern of these dimensions. To do that, we investigate the following research questions:

RQ1. Which model better describes the evolution pattern of the dimensions in software systems?

RQ2. What set of classes within the software affects the dimensions growth and decrease, and how do these classes evolve?

We use time series with data from four software metrics: NOA (Number of Attributes) and NOM (Number of Methods), for size, and DIT (Depth of Inheritance Tree) and NOC (Number of Children), for characterizing inheritance hierarchy. NOA and NOM are the numbers of attributes and methods of a class [40]. DIT indicates the depth of a class in its inheritance hierarchy, and NOC is the number of immediate subclasses of a given class [10]. We chose them because they are the size and inheritance software metrics that the dataset used in this study provides. The dataset is available in the literature, and it was used to analyze causality among internal object-oriented dimensions [11]. The data analyzed in this study are from 10 Java-based open-source projects. We did not consider data of LOC (Lines of Code) to measure class size because many previous studies have performed an analysis of software evolution with this metric [5–7, 15, 20, 21, 25–28, 32, 47].

This work contributes with eight properties of object-oriented software evolution regarding inheritance hierarchy and size. The properties extracted in this work are:

(1) the inheritance hierarchy increases in depth and decreases in breadth;
(2) inheritance hierarchy depth grows according to a linear model;
(3) inheritance hierarchy breadth decreases according to a quadratic model;
(4) a small part of the system influences the evolution of the inheritance hierarchy;
(5) depth and breadth evolution are not associated;
(6) size of classes grows according to a linear model;
(7) a small group of classes affects the evolution of system size;
(8) the evolution of NOA and NOM are associated.

The organization of this paper comprises: Section 2 presents the used dataset. In Section 3, we describe our methodology. Section 4 reports the main results of inheritance hierarchy and size evolution. Section 5 summarizes the evolution properties. Section 6 discusses the threats to validity and our decision to mitigate them. Section 7 presents some related works, and Section 8 concludes this paper.

2 DATASET

The data analyzed in this work are from the public dataset COMETS (Code Metrics Time Series), which contains time series regarding 17 software metrics extracted from 10 open-source Java systems. The metrics provided by COMETS are (i) Number of Attributes (NOA), (ii) Number of Public Attributes (NOPA), (iii) Number of Private Attributes (NOPRA), (iv) Number of Attributes Inherited (NOAI), (v) Number of Lines of Code (LOC), (vi) Number of Methods (NOM), (vii) Number of Public Methods (NOPM), (viii) Number of Private Methods (NOPRM), (ix) Number of Methods Inherited (NOMI), (x) Fan-in, (xi) Fan-out, (xii) Weighted Methods per Class (WMC), (xiii) Depth of Inheritance Tree (DIT), (xiv) Number of Children (NOC), (xv) Coupling Between Objects (CBO), (xvi) Response For a Class (RFC), and (xvii) Lack of Cohesion in Methods (LCOM).

The time series in COMETS comprehend an interval of bi-weeks, i.e., 14 days between the observations [11]. Table 1 summarizes the central COMETS' characteristics.

We identified the other three datasets with metric data: D'Ambros dataset [14], Helix [51], and Qualitas Corpus [49]. However, Qualitas Corpus does not provide time series for object-oriented metrics, and Helix does not include the time series of all metrics used in this study. D'Ambros' dataset provides a time series of inheritance and size metrics, comprising between 90 and 99 observations for five systems. In contrast, COMETS comprises a more significant number of systems and versions, as shown in Table 1. Therefore, we decided to use COMETS because it is the largest dataset we have identified with time series regarding inheritance hierarchy and size metrics.

3 RESEARCH METHOD

This section presents the research method we used to analyze how the inheritance hierarchy and size metrics evolve. All statistical analyses used in this work are considered significant on a 5% level.

3.1 Behavior Analysis

We consider in this work the time series computed at the class level. A system has many time series, which need to be normalized to extract a global serial measure for the systems. We use DIT and NOC metrics to characterize inheritance hierarchy and NOA and NOM to characterize size. We modeled the values of DIT and NOC by a statistical distribution in which the average is representative [17, 18]. In contrast, the average is not representative of the statistical distribution of the NOA and NOM values since these metrics follow a heavy-tailed distribution [17, 18]. Hence, we use the arithmetic average to normalize the DIT and NOC values and the sum to normalize the NOA and NOM values. A global time series is obtained by extracting the arithmetic average or sum from metric values of all classes for each version of a system.

We use linear regression methods [16] to model the global time series. We chose regression techniques instead of other techniques, e.g., ARIMA [3], due to its flexibility and adequacy to analyze time series in version scale. Considering that standard linear regression is not efficient with autocorrelated data, which is the case of time series, we also implemented some adjustments. We modeled the metrics of inheritance hierarchy – DIT and NOC –, and size – NOA and NOM – using the following types of model: (i) linear; (ii) quadratic (polynomial at degree 2); (iii) cubic (polynomial at degree 3); (iv) logarithmic at degree 1; (v) logarithmic at degree 2; and (vi) logarithmic at degree 3. The regression modeling in this work aims to characterize the evolution pattern of the inheritance hierarchy and size in the software systems and no provide a prediction model.

However, a system usually goes through several modifications and refactoring processes over its lifetime, and these activities may change the system's time series pattern or even affect its prediction. To deal with events that impact changes in a time series, we carry out an intervention analysis. *Intervention analysis* is a technique that evaluates and measures the effects these external factors cause in the time series [52]. This analysis evaluates the breakpoints that change the time series behavior and incorporates these critical points in the model aiming to adjust it to the new time series pattern. The intervention analysis allows us to improve the representation quality of the models.

Table 1: Systems of the COMETS dataset.

#	System Name	Description	Time Frame	# Versions
1	Eclipse JDT Core	Compiler and other tools for Java	07/01/2001 - 06/14/2008	183
2	Eclipse PDE UI	Set of tools to create, develop, test, debug and deploy Eclipse plug-ins, fragments, features, update sites and RCP products	06/01/2001 - 09/06/2008	191
3	Equinox Framework	OSGi application implementor	01/01/2005 - 06/14/2008	91
4	Hibernate Core	Database persistence framework	06/13/2007 - 03/02/2011	98
5	JabRef	Bibliography reference manager	10/14/2003 - 11/11/2011	212
6	Lucene	Search software and document indexing API	01/01/2005 - 10/04/2008	99
7	Pentaho Console	Software for business intelligence	04/01/2008 - 12/07/2010	72
8	PMD	Source code analyzer	06/22/2002 - 12/11/2011	248
9	Spring Framework	Java application development framework	12/17/2003 - 11/25/2009	156
10	TV-Browser	Electronic TV guide	04/23/2003 - 08/27/2011	221

Time series may also contain autocorrelation, i.e., a serial correlation between their values [12]. When we model autocorrelated data using linear regression, we need to incorporate the autocorrelation in the models detected in their residual to ensure a good representation of the data [2]. The inclusion of the autocorrelation in the models is performed via autoregression, using observations from previous time steps to model the value at the next time [12]. Hence, we evaluate the models' errors and incorporate the autocorrelation of our data in the generated models. After applying the autoregression in the residuals of the models, we include the autoregressive error coefficient into its respective model.

Finally, we compute the adjusted determination coefficient ($\overline{R}^2$) to assess the adequacy of the models. $\overline{R}^2$ is a measure of adjustment of a model, which allows us to understand to which extent the model explains the variability of analyzed data [42]. We chose $\overline{R}^2$ because it considers the number of parameters introduced in the model and penalizes the inclusion of less critical parameters. We also compare the $\overline{R}^2$ values and select the type that better describes the evolution of the metrics using an evaluation protocol composed of three stages:

(1) **Relevance:** select models with $\overline{R}^2$ higher than or equal to 90%.
(2) **Coverage:** among the models chosen in the previous stage, coverage selects the ones that cover the most significant number of systems.
(3) **Simplicity:** if we pick more than one model in Stage 2, we opt for the simplest model considering the order: (i) linear, (ii) quadratic, (iii) cubic, (iv) logarithmic at degree 1, (v) logarithmic at degree 2, and (vi) logarithmic at degree 3.

3.2 Trend Analysis

In the COMETS dataset, the -1 value indicates that a particular component is not present in a particular system version. These values are not representative of our analysis. Hence, we remove them and reorganize the time series observations considering only the values greater than -1. Besides, classes may be included and removed at any moment over the evolution process of object-oriented software. These changes may introduce a phenomenon in some classes that we named ghost class, that should be identified in the systems time series. Ghost classes are composed of breaks in the observations that divide the time series into several small sub-series. This break in the middle of the time series makes the trend analysis unfeasible. Therefore we identify and remove classes with this phenomenon from our analysis not to introduce bias in our study.

After organizing the data, we apply trend tests in the time series to identify whether it has a growth or a decrease trend. We used three tests based on hypothesis analysis to define the presence of trends in time series: (i)] Mann-Kendall [29]; (ii) Cox-Stuart [43]; and (iii) Wald-Wolfowitz [43]. We chose them because they are useful and efficient [29, 43]. We consider the following hypotheses:

- **H_0:** there is no trend in the time series.
- **H_1:** there is a trend in the time series.

Statistical tests may be prone to errors. Then, we defined the following criteria to consider the results of the three tests to determine the presence of trend: *"time series has a trend if, and only if, the null hypothesis is rejected at least in two of the three tests"*. When removing -1 values, some time series may substantially reduce their number of observations. To avoid applying trend tests in tiny time series, we analyze the trend in time series with ten or more continuous observations.

The Mann-Kendall test is sensitive to the presence of autocorrelation, and it may generate false-positives or false-negatives [23]. We developed an automatic checking approach to identify time series with autocorrelation to mitigate this problem. It analyzes the autocorrelation (ACF) and partial autocorrelation (PACF) plots of the time series to find the autocorrelated ones. ACF consists of a correlation of any series with its lagged values plotted along with the confidence band [3]. It describes how well a given value is related to its past observations. PACF consists of a plot of the partial correlation of the series with its own lagged values regressed at shorter lags. Non-stationary series were properly made stationary by taking successive differences in the original series. In the time series with autocorrelation, we use a modified Mann-Kendall test to evaluate trends instead of the original test. Hamed and Rao [23] modified the value of the variance from the original test and proposed a modified approach more suitable and powerful for autocorrelated data.

After running the statistical tests, we apply the trend criteria and identify the time series with a trend. We analyze the p-values resulting from Cox-Stuart, Wald-Wolfowitz, and Mann-Kendall - the original version for time series without autocorrelation or modified for time series with autocorrelation -. Finally, we evaluate the type

of trend in the time series by plotting each time series chart from the systems and visually analyzing their behavior. We manually classify the trends considering the following types:

- **Upward trend:** a pattern whose distance between the trend line and the x-axis increases over the x-axis.
- **Downward trend:** a pattern whose distance between the trend line and the x-axis decreases over the x-axis.
- **Undefined trends:** they are cases that do not follow a clear pattern. We also include here trends of times series whose values of the first and last observations are equal.

4 RESULTS

This section presents the results we found for inheritance hierarchy and size evolution in object-oriented systems.

4.1 Analysis of the Dimensions Evolution in the System Level

This section presents the results of the investigation of RQ1.

RQ1. *Which model better describes the evolution pattern of the dimensions in software systems?*

The analysis presented in this section investigates how inheritance hierarchy and size evolves and identifies the pattern that better describes this evolution.

We applied our method based on regression techniques to the global metrics time series from COMETS' systems. However, before modeling the time series, we analyzed these metrics' behavior over the evolution. We plotted the global time series extracted from the analyzed systems as line charts. In these charts, we evaluated if the metrics increase or decrease over time. Figure 1a exhibits the global time series charts regarding DIT and NOC, while Figure 1b shows the global time series charts from NOA and NOM.

4.1.1 Inheritance Analysis. Given the DIT evolution shown in Figure 1a, we observed that, for five systems, the global average of DIT increases over time. This phenomenon happens in `Eclipse PDE UI`, `JabRef`, `PMD`, `Spring Framework`, and `TV-Browser`. On the other hand, `Eclipse JDT Core` and `Hibernate Core` DIT decrease slowly and smoothly. `Pentaho Console` initially presents an increase of DIT, and then it decreases along with some releases, and then, such decrease becomes slow and smooth over time. We noted a high decrease of the global DIT in `Eclipse JDT Core` between the 100^{th} and 120^{th} versions, resulting from the refactoring or restructuring process in this system. However, before and after this event, DIT's global behavior in this software decreases very smoothly and is almost stable. Moreover, we observed that the global average of DIT in the `Equinox Framework` and `Lucene's` life cycle is practically constant, although there were some smooth variations in the systems' time series. We may then infer that DIT's global average tends to increase slightly over the software evolution in most systems.

Concerning the NOC evolution, we analyzed the charts in Figure 1a and identified that, in 60% of the systems, NOC decreases over time and, in some cases, such decrease is minimal. The systems presenting this pattern are `Eclipse JDT Core`, `Equinox Framework`, `Hibernate Core`, `Lucene`, `Pentaho Console`, and `PMD`. In `Eclipse JDT Core`, the series starts with a high value and drops shortly after that. NOC's global average follows a very smooth decreasing pattern, remaining almost stable over the whole system life cycle. On the other hand, in some systems, the global NOC has a growth behavior over their life cycle. The systems that presented this pattern are `Eclipse PDE UI`, `JabRef`, and `TV-Browser`. In `Spring Framework`, although the global NOC time series has smooth variations over time, it follows a stable pattern and remains practically constant over this system's life cycle. This analysis concludes that the global average of NOC decreases over the software evolution, tending to achieve zero. This result suggests that classes do not have many children in the software context, and they tend to reduce this number.

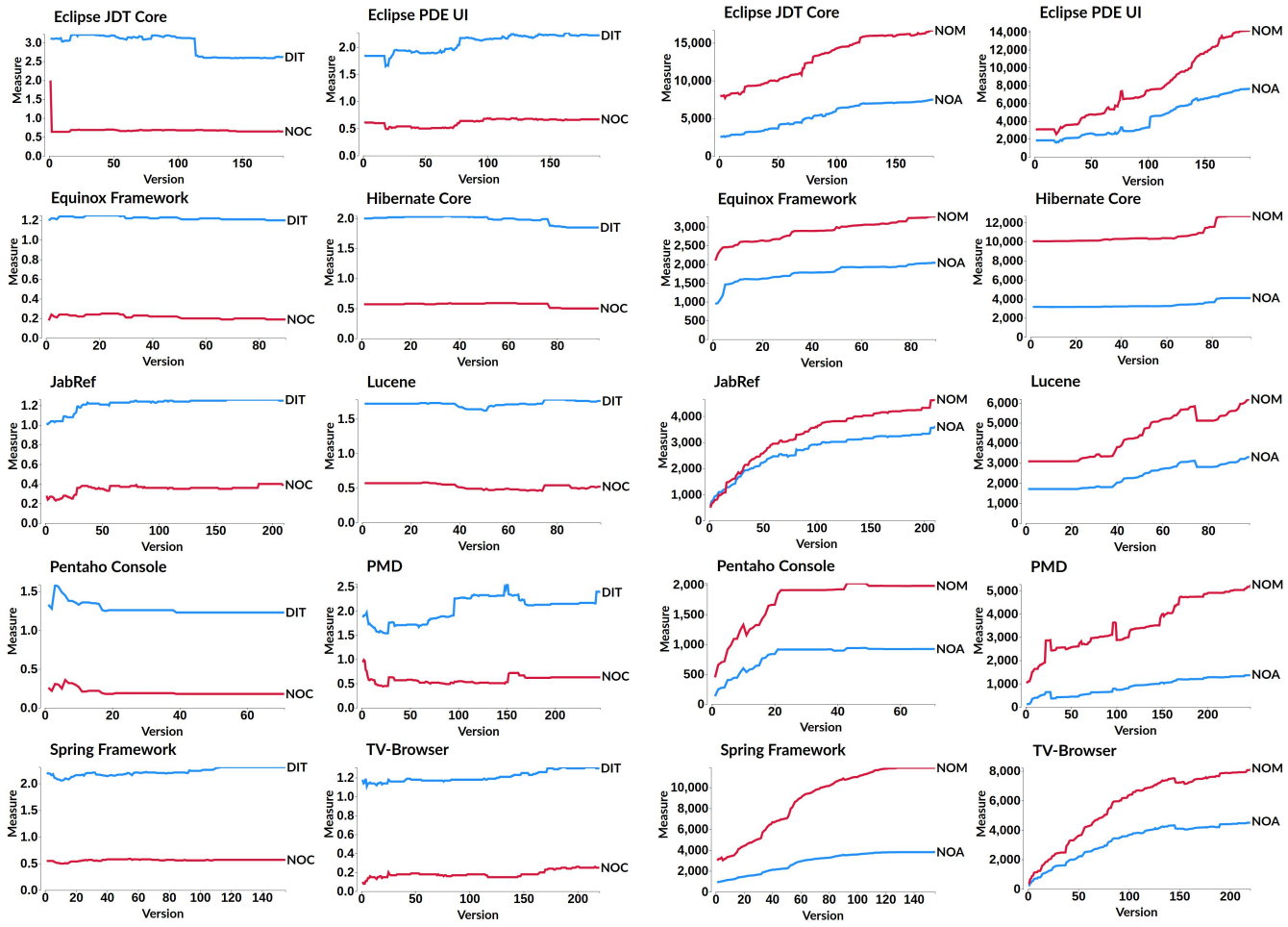
4.1.2 Size Analysis. Analyzing Figure 1b, we observed that NOA and NOM have a growth behavior in all the analyzed systems, i.e., the systems tend to increase their size in terms of attributes and methods over their life cycle. Although NOA and NOM grow over the software evolution, when comparing their global time series, we observed that NOM values are hugely higher than NOA in all analyzed systems, considering the absolute values in the time series. The exception is `JabRef` because NOA starts higher than NOM and follows this configuration up to the 17^{th} release. Then, NOM exceeds NOA and continues to grow faster than NOA over the `JabRef` evolution.

4.1.3 Modeling Analysis. After observing the metrics' evolution pattern, we modeled their global time series and assessed the generated models with our evaluation protocol (Section 3.1). Table 2 shows the results obtained for DIT and NOC, and Table 3 summarizes the results obtained for NOA and NOM. The "lin.", "quad.", "cub.", "log. 1", "log. 2", and "log. 3" columns indicate the $\overline{R}^2$ scores extracted for linear, quadratic, cubic, logarithmic at degree 1, logarithmic at degree 2, and logarithmic at degree 3 models, respectively.

To improve our discussion about the results, we adopted a color scheme highlighting the models selected in each stage. The green color indicates the models selected at Stage 1. Yellow shows the ones chosen in Stage 2, and red specifies the only model type selected at Stage 3, which is the one that better characterizes the pattern evolution of the metrics by following the parsimonious criteria.

Analyzing the DIT results in Table 2, we observe that most models have an $\overline{R}^2$ score higher than 90%. There is no model with $\overline{R}^2$ higher than or equal to 90% in `Equinox Framework`. Figure 1a shows that the `Equinox Framework's` evolution is very stable, with minimal variations in the DIT time series. This chart shows that the `Equinox Framework` contains a very smooth trend in the global DIT time series. Therefore, the models could not capture this trend and generate suitable adjustments to this particular time series. We identify that linear, quadratic, cubic, logarithmic at degree 1, and logarithmic at degree 2 attend Stage 2 of our evaluation protocol for the initially selected models. However, by applying the simplicity criteria, we conclude that the linear model is the one that better explains the growth evolution pattern of the DIT global average since it attends all aspects of our evaluation protocol.

The results of NOC reported in Table 2 shows that most of the generated models have $\overline{R}^2$ values higher than 90%. However, there is no model with $\overline{R}^2$ higher than or equal to 90% for the global



(a) DIT and NOC.

(b) NOA and NOM.

Figure 1: Global metrics time series of the analyzed systems.

Table 2: $\overline{R}^2$ values computed from the DIT and NOC models.

System	DIT						NOC					
	lin.	**quad.**	**cub.**	**log. 1**	**log. 2**	**log. 3**	**lin.**	**quad.**	**cub.**	**log. 1**	**log. 2**	**log. 3**
Eclipse JDT Core	98.00%	97.41%	98.28%	98.07%	97.48%	98.35%	88.99%	91.61%	91.99%	89.08%	91.72%	92.11%
Eclipse PDE UI	97.81%	97.87%	97.83%	97.45%	97.53%	97.52%	97.13%	96.69%	97.16%	96.92%	96.46%	96.95%
Equinox Framework	86.89%	85.01%	86.49%	86.95%	85.10%	86.54%	86.05%	67.85%	85.70%	86.68%	68.08%	86.28%
Hibernate Core	96.54%	96.67%	96.58%	96.58%	96.71%	96.59%	94.67%	94.93%	-	94.80%	95.12%	-
JabRef	98.69%	98.69%	98.79%	98.68%	98.67%	98.64%	94.81%	94.79%	95.05%	94.71%	94.70%	94.99%
Lucene	92.23%	92.21%	89.52%	92.13%	92.11%	89.42%	96.07%	92.03%	-	95.86%	91.43%	-
Pentaho Console	75.49%	83.43%	90.92%	77.50%	85.05%	91.62%	76.80%	76.83%	78.40%	79.45%	81.35%	82.52%
PMD	97.61%	97.64%	96.52%	97.79%	97.82%	96.84%	93.83%	93.83%	93.43%	93.86%	93.84%	93.83%
Spring Framework	97.21%	97.04%	97.08%	97.09%	96.98%	-	90.57%	90.67%	91.48%	90.76%	90.91%	91.63%
TV-Browser	98.24%	98.09%	97.94%	98.11%	97.97%	97.81%	96.57%	96.74%	96.74%	95.08%	94.79%	95.26%

NOC in the `Equinox Framework` and `Pentaho Console`. Analyzing the NOC global time series of these systems in Figure 1a, we observe many occurrences of interventions and change points in the `Pentaho Console` time series, which avoid defining a function to model it with a good $\overline{R}^2$ value. Concerning `Equinox Framework`, the global NOC time series trend is very smooth, almost constant,

Table 3: $\overline{R}^2$ values computed from the NOA and NOM models.

System	NOA						NOM					
	lin.	**quad.**	**cub.**	**log. 1**	**log. 2**	**log. 3**	**lin.**	**quad.**	**cub.**	**log. 1**	**log. 2**	**log. 3**
Eclipse JDT Core	99.82%	99.82%	99.83%	99.77%	99.79%	99.79%	99.80%	99.81%	99.81%	99.76%	99.77%	99.77%
Eclipse PDE UI	99.73%	99.73%	99.73%	99.61%	99.56%	99.51%	99.85%	99.85%	99.85%	99.78%	99.80%	-
Equinox Framework	98.25%	98.24%	98.22%	97.68%	97.86%	97.84%	99.29%	99.31%	99.31%	99.30%	99.33%	99.33%
Hibernate Core	98.75%	98.79%	-	98.86%	98.88%	-	98.61%	98.67%	-	98.71%	98.73%	-
JabRef	99.81%	99.81%	99.82%	99.70%	99.71%	99.77%	99.84%	99.84%	99.86%	99.67%	99.67%	99.70%
Lucene	99.30%	98.99%	99.29%	99.44%	99.03%	99.45%	99.09%	98.79%	99.08%	99.28%	98.90%	99.29%
Pentaho Console	98.11%	98.09%	98.25%	97.52%	97.49%	97.90%	98.41%	98.42%	98.52%	97.99%	97.98%	98.09%
PMD	99.51%	99.51%	99.53%	98.98%	98.92%	98.62%	99.23%	99.20%	99.16%	98.96%	98.96%	98.96%
Spring Framework	99.93%	99.94%	99.92%	99.91%	99.91%	99.91%	99.93%	99.94%	99.91%	99.88%	99.88%	99.89%
TV-Browser	99.90%	99.90%	99.90%	99.42%	99.58%	99.85%	99.92%	99.93%	99.93%	99.50%	99.69%	99.80%

and then the models could not capture this tendency to generate proper adjustments for this system. Two models, quadratic and logarithmic at degree 2, attend Stage 2 of our evaluation protocol. By applying the simplicity criteria, we conclude that the quadratic model better describes NOC's decrease pattern. Besides, considering that the global average of NOC does not decrease fast and there are several fluctuations between the system's versions, the quadratic function is the one that best adjusts to NOC decrease pattern.

Regarding NOA results in Table 3, almost all models presented $\overline{R}^2$ scores higher than or equal to 90%. We identify only two exceptions to which there is no relevant model, the cubic and logarithmic at Degree 3 models for the `Hibernate Core` global time series. Due to this, we did not highlight these cases with green. We observe that linear, quadratic, logarithmic at degree 1, and logarithmic at degree 2 described all the systems' global time series and attended Stage 2 of our protocol for the initially selected models. However, applying the simplicity criteria, we conclude that the linear model is the one that better explains the growth evolution pattern of NOA since it has attended all aspects of our evaluation protocol.

In the case of NOM, reported in Table 3, we also identify that most of the produced models had good $\overline{R}^2$ values. Three cases were selected at Stage 1 because our method could not find models representing them: cubic and logarithmic at degree 3 for `Hibernate Core` and logarithmic at degree 3 for `Eclipse PDE UI`. Due to this, we did not highlight these cases with green. Among the selected cases at Stage 1, we observe that linear, quadratic, logarithmic at degree 1, and logarithmic at degree 2 models represented all systems' global time series. However, following Stage 3 of our evaluation protocol, we conclude that the linear model better explains the global evolution pattern of NOM as well as for NOA.

Summary of RQ1. Inheritance hierarchy slightly grows in depth and decreases in breadth over the evolution. The linear model better explains the DIT growth pattern, and a quadratic-order model better describes the decrease pattern of NOC. Regarding size, the number of attributes and methods of classes increases over the software evolution following a linear model.

4.2 Analysis of Growth and Decrease of the Dimensions

This section describes the analysis of the inheritance hierarchy to answer RQ2.

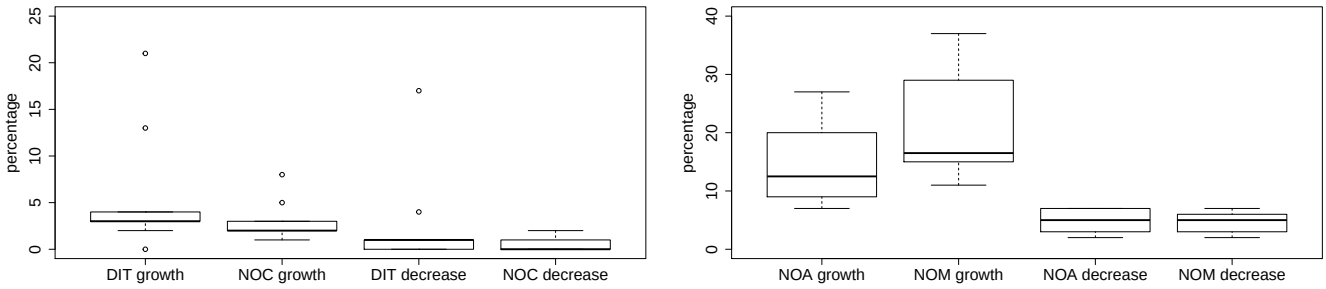
RQ2. *What set of classes within the software affects the dimensions growth and decrease, and how do these classes evolve?*

To answer this research question, we initially identified the classes in the analyzed systems responsible for increasing and decreasing their inheritance hierarchy's depth or breadth. Then, we carried out the trend analysis considering the time series regarding the systems' classes and extracted the percentage of classes that had growth and decrease in DIT and the classes with growth and decrease in NOC. Figure 2a presents the percentages obtained by DIT growth, DIT decrease, NOC growth, and NOC decrease. Table 4 details Figure 2a by presenting a descriptive analysis of the data. Following, we discuss the results regarding the growth and the decrease of the inheritance hierarchy separately.

This research question aims to identify the percentages of classes existing in the systems that directly affect the growth or decrease of the inheritance hierarchy and class size. We performed the trend analysis considering the time series from the systems' classes regarding the metrics. Besides, we computed the percentages of types responsible for increasing and decreasing these metrics values and summarized them in Figures 2a and 2b.

Tables 4 and 5 show the class percentages that interfere in the growth and decrease of the inheritance hierarchy and size. We discuss the results obtained in the sequel.

4.2.1 Classes Responsible for Growth. Figure 2a and Table 4 show that a tiny percentage of classes in the systems contribute to DIT and NOC growth. The median percentages of classes responsible for "DIT growth" and "NOC growth" are 3.00% and 2.00%, respectively. The maximum percentages are 21.00% and 8.00%, respectively. Therefore, the results indicate that a small group of classes in a system directly affects the inheritance tree's growth, in-depth, and breadth. Regarding DIT, which tends to increase over time, this group represents no more than 21.00% of the system's classes. For NOC, which tends to decrease over time, this group is even less and represents no more than 8.00% of the system's classes. Moreover, these values are outliers, as shown in the boxplot of Figure 2a.



(a) Inheritance hierarchy. **(b) Class size.**

Figure 2: Distribution of classes that affects the dimensions growth and decrease.

Table 4: Descriptive analysis of the distribution of classes that affects the inheritance hierarchy growth and decrease.

Event/Percentile	0%	25%	50%	75%	100%
DIT Growth	0.00	3.00	3.00	4.00	21.00
NOC Growth	1.00	2.00	2.00	2.75	8.00
DIT Decrease	0.00	0.00	1.00	1.00	17.00
NOC Decrease	0.00	0.00	0.00	1.00	2.00

Table 5: Descriptive analysis of the distribution of classes that affects size growth and decrease.

Event	0%	25%	50%	75%	100%
NOA Growth	7.00	9.00	12.50	19.25	27.00
NOM Growth	11.00	15.25	16.50	26.50	37.00
NOA Decrease	2.00	3.00	5.00	7.00	7.00
NOM Decrease	2.00	3.00	5.00	5.75	7.00

Figure 2b and Table 5 show that the median of the class percentage responsible for "NOA growth" and "NOM growth" is 12.50% and 16.50%, respectively. The maximum percentages indicate that 27.00% and 37.00% of the systems' classes contribute directly to increase NOA and NOM. These results show that just a small group of classes in a system have their number of attributes and methods increased over time. The classes with increasing NOM correspond to no more than 37% of the system. This group is even smaller, with no more than 27% of the systems' classes regarding NOA.

4.2.2 Classes Responsible for Decrease. Analyzing the decrease of DIT and NOC in Figure 2a and Table 4, we note that a small percentage of classes within a system directly influence these metrics to decrease over time. The median of the portion of classes within the systems with a decreasing DIT is 1.00%. For NOC, 50% of the analyzed systems do not present classes with decreasing trends. Besides, the maximum percentages for "DIT decrease" and "NOC decrease" are 17.00% and 2.00%, respectively.

These results show that just a tiny group of classes directly decreases their depth and breadth inheritance tree. As we identified that DIT tends to increase over time, we expected that the percentage of classes decreasing DIT would be less than the percentage of classes with increasing DIT patterns. In contrast, NOC has a decreasing trend, but the percentage of classes with a decreasing NOC in a system is less than that with a growing NOC. The possible explanation is: (1) if the number of classes in the system has not grown, decrease in NOC values of the classes within the system is more intense than the increase; (2) if the number of classes in the system has grown, most of the new classes will not have children. As the number of classes increases, the second hypothesis is more likely.

Figure 2b and Table 5 show that the median of the class percentages responsible for "NOA decreases" and "NOM decreases" is 5.00%, and the maximum percentage is 7.00% for both. These results show that a tiny group of classes within the systems, corresponding to no more than 7.00%, directly contributes to these metrics decreasing over time. Although the class groups responsible for the growth and decrease of these metrics are small, the high discrepancy between them is one reason for the growth trend in these metrics.

4.2.3 Growth versus Decrease. In this part of our study, we analyzed the intersection of trend results about the same dimension metrics to identify the percentage of classes. Regarding inheritance hierarchy, we analyzed the following behaviors: (i) both DIT and NOC grow, (ii) both DIT and NOC decrease, (iii) DIT grows and NOC decreases, and (iv) DIT decreases and NOC grows. In the case of size, we analyzed the following behaviors: (i) both NOA and NOM grow, (ii) both NOA and NOM decrease, (iii) NOA grows and NOM decreases, and (iv) NOA decreases and NOM grows. We investigated these behaviors from the perspective of system and trend classes. Tables 6 and 7 summarize the results obtained for these cases considering these two perspectives. Initially, we analyzed the behaviors considering the systems as a whole and computed the percentages by dividing the number of intersections by the total number of classes existing in the systems. After that, we analyzed the behaviors considering only trend classes and computed percentages by dividing the number of intersections by the number of classes presenting any trend.

Table 6: Intersection results for inheritance hierarchy.

System	System Perspective				Trend Class Perspective			
	i	**ii**	**iii**	**iv**	**i**	**ii**	**iii**	**iv**
Eclipse JDT Core	1%	0%	0%	3%	3%	1%	1%	15%
Eclipse PDE UI	0%	0%	0%	0%	4%	0%	0%	5%
Equinox Framework	0%	0%	0%	0%	0%	0%	0%	0%
Hibernate Core	0%	0%	0%	0%	0%	0%	0%	3%
JabRef	0%	0%	0%	0%	0%	0%	0%	0%
Lucene	0%	0%	0%	0%	4%	0%	0%	0%
Pentaho Console	0%	0%	0%	0%	0%	0%	0%	0%
PMD	0%	0%	0%	0%	1%	0%	0%	0%
Spring Framework	1%	0%	0%	0%	6%	0%	1%	0%
TV-Browser	0%	0%	0%	0%	0%	0%	1%	0%

Table 7: Intersection results for class size.

System	System Perspective				Trend Class Perspective			
	i	**ii**	**iii**	**iv**	**i**	**ii**	**iii**	**iv**
Eclipse JDT Core	22%	2%	1%	3%	43%	5%	2%	5%
Eclipse PDE UI	7%	3%	1%	2%	28%	11%	2%	7%
Equinox Framework	11%	1%	1%	0%	38%	4%	3%	2%
Hibernate Core	7%	1%	0%	0%	35%	6%	2%	1%
JabRef	10%	1%	0%	1%	39%	4%	2%	6%
Lucene	7%	1%	0%	1%	31%	3%	1%	3%
Pentaho Console	5%	1%	0%	1%	24%	6%	1%	4%
PMD	7%	1%	1%	0%	33%	5%	3%	1%
Spring Framework	18%	4%	1%	1%	42%	9%	1%	3%
TV-Browser	16%	3%	1%	1%	40%	9%	2%	3%

Observing the "System Perspective" results in Table 6, we do not identify high percentages for these cases. In cases *(ii)* and *(iii)*, the percentages of classes are so small that they did not even represent 1% of the classes within the systems. In cases *(i)* and *(iv)*, although some systems had values greater than 0, most have a minimal percentage of classes. Analyzing the "Trend Class Perspective" in Table 6, we observe that the percentages are too small. Despite some exceptions, such as Case *(iv)* in `Eclipse JDT CORE` and Case *(i)* in `Spring Framework`, the other cases did not present relevant percentages. These results bring pieces of evidence that DIT and NOC of a class evolve separately over the software life cycle and do not tend to follow a combined pattern or establish any relation over the software evolution.

As shown by the "System Perspective" in Table 7, cases *(iii)* and *(iv)* are too rare, and their maximum percentages in the analysis are 1% and 3%, respectively. Case *(i)* has the highest chance of occurring since its maximum percentage corresponds to 22% of the total systems' classes. In Case *(ii)*, the maximum percentage is 4%. On the other hand, observing the results in "Trend Perspective" in Table 7, we identify high percentages of Case *(i)*, which vary from ≈ 25% to ≈ 45%. These percentages had a significant increase compared to percentages of the system perspective. We also identified very low percentages for cases *(ii)* and *(iv)* similar to what happens in the system perspective. Such values show that these cases tend not to occur frequently during the software evolution. Therefore, the results suggest evidence of the relation between NOA and NOM over the software evolution. This evidence indicates that these software metrics grow together and follow a combined pattern over evolution.

Summary of RQ2. The evolution of depth and breadth from inheritance trees are directly affected by a small set of classes, and the DIT and NOC do not follow a related pattern over the evolution. A small but not irrelevant percentage of classes in a system has the attributes and methods increased over time in size. Decreasing methods or attributes are rare events. Analyzing the systems as a whole, on average, just ≈ 10% of the classes in a system have methods and attributes growing together. However, considering only trend classes, NOA and NOM grow together.

5 EVOLUTION PROPERTIES

This section compiles our results in eight software evolution properties related to inheritance hierarchy and size.

1st - Inheritance hierarchy tends to increase in depth and decrease in breadth over time. Our analysis indicates that the average of DIT (depth) of classes tends to increase. In contrast, the average of NOC (breadth) tends to decrease.

2nd - Inheritance hierarchy depth grows according to a linear model. An inheritance tree with many levels may introduce complexity in the system structure and make it hard to understand and maintain. The higher number of superclasses a class has, the more difficult it is to understand its objects' behavior. Due to this, Gamma et al. [19] define the principle *"favor composition over inheritance"* that recommends implementing reusable software using class composition rather than class inheritance.

3rd - Inheritance hierarchy breadth decreases according to a quadratic model. A quadratic function better models the global behavior of the breadth evolution in the inheritance hierarchy. Decreasing the mean number of children in a system may be due to two possible reasons: the new classes added to the system do not have children classes in general, or the inheritance trees are refactored over time.

4th - A small part of the system influences the growth and the decrease of the inheritance hierarchy. Regarding depth, no more than 21.00% of the system's classes have their DIT increased, and no more than 17.00% have their DIT decreased. No more than 8.00% and 2.00% of the systems' classes have their NOC increased or decreased regarding breadth. Although inheritance may introduce complexity in the system, it will occur with a small portion of the system.

5th - There is no association between the depth and the breadth of a class. A small percentage of classes within the systems presented an association between depth and breadth in growth and decrease. This finding shows that the number of children and the number of superclasses of a class evolve independently and do not follow a combined pattern.

6th - The size of classes grows according to the linear model. A linear function better models the evolution pattern of both the number of attributes and the number of methods of classes within the systems. Our results indicate how such growths occur in the structure of classes.

7th - A small group of classes affects the growth of system size. No more than 27.00% of the systems' classes increase the number of attributes, and no more than 37.00% increase the number of methods. Rarely, a class has its size decreased. In our study, 7.00%

is the higher percentage of classes with such behavior. Although the percentage of types having attributes or methods added to them is not very high, it is still relevant. Apart from refactoring, a class swelling means that more services were introduced in the class, and this may lead to non-focused and more complex classes and, therefore, to a more complex software structure.

8th - The evolution of the number of attributes and the number of methods are correlated. Our results suggest a positive correlation between NOA and NOM evolution regarding growth and decrease. Such a finding means that methods are also included when we include attributes in a class or vice-versa. When removing attributes, methods are also removed. This finding details the way a class grows or decreases.

6 THREATS TO VALIDITY

This section presents the threats to this work's validity and discusses the major decision we made to mitigate them.

We used statistical trend tests to identify classes that affect the software metrics' growth and decrease. However, statistical tests may represent a threat to validity since they may be prone to errors if not correctly applied. To mitigate this threat, besides choosing relevant and useful trend tests, we defined trend criteria that consider the results of all of them to point to the presence of trend in time series.

We analyzed the evolution of inheritance hierarchy and size in open-source Java systems. We considered a dataset composed of 10 different systems. Although our dataset has an appropriate amount of data and reflects the evolution of Java systems, we can not claim generalization of the results to other domains and contexts of development, such as proprietary software and systems written in any language other than Java.

We disregarded time series with "ghost" classes, i.e., classes with broken intervals, from our trend analysis. Removing "ghost" classes may be considered a threat to validity since they may contain information about the system. To mitigate this threat, we evaluated them separately to check if they do have relevant information. After assessing them, we concluded that they made up a tiny part of the systems and did not have significant trend patterns. Therefore, disregarding "ghost" classes would not introduce bias in our analysis.

We used linear regression techniques to model the pattern of our analyzed time series. Although regression techniques have been widely used to do this task [1, 4, 22, 30, 31, 45, 46, 48], the presence of interventions or autocorrelation in the time series may bias the generated models. To avoid this problem, after generating the models via regression techniques, we carried out intervention and residual analyses to incorporate the autocorrelation and ensure that the models have a reasonable adjustment.

7 RELATED WORK

Meyer [41] studied the use of inheritance mechanisms in object-oriented software and proposed a methodology to use it. His methodology documents 12 different forms to use inheritance and discusses the scenarios where each one must be applied. Some works analyzed the impact of depth of inheritance on maintenance [9, 13, 24]. However, while Daly et al. [13] found that inheritance hierarchy harms maintenance, Cartwright [9] identified a positive impact. Harrison et al. [24] compared systems with inheritance and systems without inheritance to measure this impact. They concluded that inheritance might make the systems harder to modify and affect their understandability. Tempero et al. [50] indicated that developers have often used inheritance in Java programs to define types. According to them, ≈ 75% of the types in Java systems are defined via inheritance. However, the types are usually defined at the first levels of the inheritance tree and do not have many children. Nasseri et al. [44] confirmed the finding of [50] about inheritance hierarchy be shallow in Java systems and indicate that the classes have a strong tendency to be added at levels 1 and 2 of the tree. They also pointed that the inheritance hierarchy tends to grow breadth-wise instead of depth-wise in open-source Java systems.

The literature has tried to characterize how the systems evolve in size, but it has drawn different conclusions. For instance, while some works have indicated that the size tends to increase super-linearly [20, 21, 27, 32], others have pointed that size grows sub-linearly [5, 8, 28], linearly [47], or following the Pareto distribution [26]. Besides, Capiluppi and collaborators [6, 7] observed that the size of the software systems increases over. It evolves similarly from three different aspects: size in KB, lines of code, and the number of files. Finally, Hatton et al. [25] aimed to quantify the growth rate of source code over time. They concluded that the open-source systems tend to double the average rate of produced source code every 42 months.

This paper presents an empirical study on the evolution inheritance hierarchy and size. It differs from the related works discussed here because we describe how the inheritance hierarchy evolves. None of the previous work described this evolution. The previous works have analyzed the evolution of size from other perspectives, e.g., number of files, number of classes, and number of lines of code. In the present work, we analyze the evolution of the size dimension from the perspective of data and features within classes.

8 CONCLUSION

This paper presents an empirical study on the evolution of the inheritance hierarchy and size in object-oriented systems. We investigated: (i) how these dimensions evolve and how their behavior pattern may be described, and (ii) the systems' portion that directly contributes to these dimensions evolving.

We used a software evolution dataset composed of software metrics time series regarding ten open-source Java systems. We used DIT and NOC to characterize the inheritance hierarchy and NOA and NOM to measure size. Our methodology considered linear regression and trend tests to model the global time series of the systems and identify the classes that tend to grow or decrease in terms of the investigated dimensions.

From the results of this study, we identified eight properties of software evolution in terms of class size and inheritance that bring new insights into how the internal structure of object-oriented software systems evolve. They are: (i) inheritance hierarchy tends to increase in depth and decrease in breadth over time; (ii) inheritance hierarchy depth grows according to a linear model; (iii) inheritance hierarchy breadth decreases according to a quadratic model; (iv) a

small part of the system influences the evolution of the inheritance hierarchy; (v) depth and breadth evolution are not associated; (vi) size of classes grows according to a linear model; (vii) a small group of classes affects the evolution of system size; (viii) the evolution of NOA and NOM are associated. The software evolution properties we identified may serve as a background predicting models to track these dimensions evolution and techniques to improve and control how these properties evolve.

As future work, it is essential to (i) build prediction models about inheritance and class size; (ii) investigate the evolution of other internal characteristics of software systems; and (iii) replicate these analyses for software systems developed in other contexts, e.g., proprietary systems, and different programming languages, e.g., C++ and C#.

ACKNOWLEDGMENTS

This work was supported by CAPES, CNPq, FAPEMIG, UFMG Programming Language Research Group, UFMG Graduate Program in Computer Science, CEFET-MG, and Department of Statistics of UFMG.

REFERENCES

[1] E. Arisholm and L.C. Briand. 2006. Predicting fault-prone components in a java legacy system. In *ISESE'06*. ACM, 8–17.
[2] B.L. Bowerman and R.T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. Duxbury Press.
[3] G.E.P. Box and G.M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco.
[4] A. Capiluppi. 2003. Models for the evolution of OS projects. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE, 65–74.
[5] A. Capiluppi, J. Fernandez-Ramil, J. Higman, H.C. Sharp, and N. Smith. 2007. An empirical study of the evolution of an agile-developed software system. In *ICSE '07*. IEEE Computer Society, 511–518.
[6] A. Capiluppi, M. Morisio, and J.F. Ramil. 2004. The evolution of source folder structure in actively evolved open source systems. In *10th ISSM*. IEEE, 2–13.
[7] A. Capiluppi, M. Morisio, and J.F. Ramil. 2004. Structural evolution of an Open Source system: A case study. *Program Comprehension, Workshop Proceedings* 12 (2004), 172–182.
[8] A. Capiluppi and J.F. Ramil. 2004. Studying the evolution of open source systems at different levels of granularity: Two case studies. *International Workshop on Principles of Software Evolution (IWPSE)* (2004), 113–118.
[9] Michelle Cartwright. 1998. An empirical view of inheritance. *Information and Software Technology* 40, 14 (1998), 795–799.
[10] S. R. Chidamber and C. F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (June 1994), 476–493.
[11] C. Couto, C. Maffort, R. Garcia, and M. T. Valente. 2013. COMETS: a dataset for empirical research on software evolution using source code metrics and time series analysis. *ACM SIGSOFT Software Engineering Notes* 38, 1 (2013), 1–3.
[12] Paul S. P. Cowpertwait and Andrew V. Metcalfe. 2009. *Introductory Time Series with R* (1st ed.). Springer Publishing Company, Incorporated.
[13] John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1, 2 (1996), 109–132.
[14] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *MSR 2010*. IEEE, 31–41.
[15] D.P. Darcy, S.L. Daniel, and K.J. Stewart. 2010. Exploring complexity in open source software: Evolutionary patterns, antecedents, and outcomes. In *HICSS '10*. IEEE, 1–11.
[16] N.R. Draper and H. Smith. 1981. *Applied Regression Analysis*. Wiley.
[17] T.G.S. Filó. 2014. *Identification of Thresholds for Metrics of Oriented-Object Software*. Master's thesis. UFMG, Belo Horizonte, Minas Gerais,. (In Portuguese).
[18] T.G.S. Filó, M.A.S. Bigonha, and K.A.M. Ferreira. 2015. A Catalogue of Thresholds for Object-Oriented Software Metrics. *Proc. of the 1st SOFTENG* (2015), 48–55.
[19] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (1 ed.). Addison-Wesley Professional.
[20] M.W. Godfrey and Q. Tu. 2000. Evolution in open source software: A case study. In *Proceedings of ICSM*. IEEE, 131–142.
[21] J.M. Gonzalez-Barahona, G. Robles, M. Michlmayr, J.J. Amor, and D.M. German. 2009. Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering* 14, 3 (2009), 262–285.
[22] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy. 2000. Predicting fault incidence using software change history. *IEEE Transactions on software engineering* 26, 7 (2000), 653–661.
[23] K.H. Hamed and A.R. Rao. 1998. A modified Mann-Kendall trend test for autocorrelated data. *Journal of hydrology* 204, 1-4 (1998), 182–196.
[24] Rachel Harrison, Steve Counsell, and Reuben Nithi. 2000. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software* 52, 2-3 (2000), 173–179.
[25] L. Hatton, D. Spinellis, and M. van Genuchten. 2017. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process* 29, 5 (2017), e1847.
[26] I. Herraiz, J.M. Gonzalez-Barahona, and G. Robles. 2007. Towards a theoretical model for software growth. In *MSR'07: ICSE Workshops 2007*. IEEE, 21–21.
[27] I. Herraiz, G. Robles, J.M. González-Barahona, A. Capiluppi, and J.F. Ramil. 2006. Comparison between SLOCs and number of files as size metrics for software evolution analysis. In *CSMR'06*. IEEE, 8–pp.
[28] C. Izurieta and J. Bieman. 2006. The evolution of FreeBSD and Linux. In *Proceedings of the ESEM*. ACM, 204–211.
[29] Maurice George Kendall. 1975. *Rank correlation methods*. Charless Griffin, LDN.
[30] S. Kirbas, A. Sen, B. Caglayan, A. Bener, and R. Mahmutogullari. 2014. The Effect of Evolutionary Coupling on Software Defects: An Industrial Case Study on a Legacy System *(ESEM '14)*. ACM, Association for Computing Machinery, New York, NY, USA.
[31] S. Koch. 2005. Evolution of open source software systems–a large-scale investigation. In *Proceedings of the 1st ICOSS*. CiteSeer, 148–153.
[32] S. Koch. 2007. Software evolution in open source projects—a large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice* 19, 6 (2007), 361–382.
[33] M.M. Lehman. 1979. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software* 1 (1979), 213–221.
[34] M.M. Lehman. 1996. Laws of software evolution revisited. In *European Workshop on Software Process Technology*. Springer, 108–124.
[35] M. Lehman, D. Perry, and J. Ramil. 1998. Implications of Evolution Metrics on Software Maintenance. In *2013 IEEE International Conference on Software Maintenance*. IEEE Computer Society, Los Alamitos, CA, USA.
[36] M.M. Lehman and J.F. Ramil. 1999. The impact of feedback in the global software process. *Journal of Systems and Software* 46, 2 (1999), 123–134.
[37] M. M. Lehman. 1980. Programs, life cycles, and laws of software evolution. *Proc. IEEE* 68, 9 (1980), 1060–1076. https://doi.org/10.1109/PROC.1980.11805
[38] M. M. Lehman, D. E. Perry, and J. F. Ramil. 1998. On evidence supporting the FEAST hypothesis and the laws of software evolution. In *Proceedings Fifth International Software Metrics Symposium. Metrics*. 84–88. https://doi.org/10.1109/METRIC.1998.731229
[39] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. 1997. Metrics and laws of software evolution-the nineties view. In *Proceedings Fourth International Software Metrics Symposium*. IEEE, 20–32.
[40] Mark Lorenz and Jeff Kidd. 1994. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, Inc., USA.
[41] Bertrand Meyer. 1996. The many faces of inheritance: A taxonomy of taxonomy. *Computer* 29, 5 (1996), 105–108.
[42] Jeremy Miles. 2014. *R Squared, Adjusted R Squared*. American Cancer Society.
[43] P.A. Morettin and C.M.C. Toloi. 2006. *Time Serie Analysis*. Edgard Blucher. (In portuguese).
[44] E. Nasseri, S. Counsell, and M. Shepperd. 2008. An empirical study of evolution of inheritance in Java OSS. In *19th ASWEC*. IEEE, 269–278.
[45] J.F. Ramil and M.M. Lehman. 2000. Metrics of Software Evolution as Effort Predictors-A Case Study. In *icsm*. IEEE, 163–172.
[46] J. Ratzinger, M. Pinzger, and H. Gall. 2007. EQ-mine: Predicting short-term defects for software evolution. *Lecture Notes in Computer Science* 4422 LNCS, 12 – 26.
[47] G. Robles, J.J. Amor, J.M. Gonzalez-Barahona, and I. Herraiz. 2005. Evolution and growth in large libre software projects. In *IWPSE'05*. IEEE, 165–174.
[48] R. Shatnawi and W. Li. 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of systems and software* 81, 11 (2008), 1868–1882.
[49] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. 2010. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, APSEC (Ed.). IEEE, 336–345.
[50] Ewan Tempero, James Noble, and Hayden Melton. 2008. How do Java programs use inheritance? An empirical study of inheritance in Java software. In *European Conference on Object-Oriented Programming*. Springer, 667–691.
[51] Rajesh Vasa, Markus Lumpe, and Allan Jones. 2010. Helix - Software Evolution Data Set. http://www.ict.swin.edu.au/research/projects/helix.
[52] W.W.S. Wei. 2006. *Time Series Analysis: Univariate and Multivariate Methods*. Pearson Addison Wesley.