1 INDEPENDENT VARIABLES

Previous work indicates that structural, evolution-based, and code smells are state-of-the-art features to represent change-prone classes. Following these findings, these code features will be used as the independent variables in this work. The metrics to be used as independent variables are depicted in Tables 1 to 3. Table 1 contains the structural metrics to be used. These metrics are grouped in cohesion, coupling, complexity, inheritance and size.

Table 1: Structural metrics.

Metric	Definition	Source
Cohesion metrics		
LCOM	**Lack of cohesion in methods**. The number of pairs of methods in the class using no common attributes minus the number of pairs of methods that do.	(Chidamber and Kemerer, 1994) [3]
TCC	**Tight class cohesion**. The percentage of pairs of public methods of the class which are directly connected methods. Two methods are called connected, if they use common attributes, directly or indirectly.	(Bieman and Kang, 1995) [2]
LCC	**Loose class cohesion**. The percentage of pairs of public methods of the class which are directly or indirectly connected. If there are methods $m_1, \ldots, m_n$, such that m_i and m_{i+1} are connected for $i = 1, \ldots, n-1$, then m_1 and m_n are indirectly connected.	(Bieman and Kang, 1995) [2]
Coupling metrics		
CBO	**Coupling Between Object Classes**. A count of the number of other classes to which it is coupled.	(Chidamber and Kemerer, 1991) [3]
RFC	**Response For a Class**. A set of methods that can potentially be executed in response to a message received by an object of that class.	(Chidamber and Kemerer, 1991) [3]
FANIN	**Fan-in**. The number of external classes that invoke methods from the analyzed class.	(Henry and Kafura, 1981) [5]
FANOUT	**Fan-out**. The number of external method invocations made by the analyzed class	(Henry and Kafura, 1981) [5]
Complexity metrics		
WMC	**Weighted Methods Per Class**. The sum of the cyclomatic complexity of the methods of a class.	(Chidamber and Kemerer, 1991) [3]
AvgCyclomatic	**Average Cyclomatic Complexity**. Average cyclomatic complexity for all nested functions or methods.	(McCabe, 1976) [8]
SumCyclomatic	**Sum Cyclomatic Complexity**. Sum of cyclomatic complexity of all nested functions or methods.	(McCabe, 1976) [8]
MaxCyclomatic	**Max. Cyclomatic Complexity**. Maximum cyclomatic complexity of all nested functions or methods.	(McCabe, 1976) [8]
Inheritance metrics		
DIT	**Depth of Inheritance Tree**. The number of nodes between the root of the inheritance tree and the analyzed class.	(Chidamber and Kemerer, 1991) [3]
Size metrics		
LOC	**Total lines of code**. Count all lines of executable code within the system, class, or method.	(Lorenz and Kidd, 1994) [7]
NOSI	**Number of static invocations**. Counts the number of invocations to static methods.	(Aniche, 2015) [1]
totalFieldsQty	**Total Fields**. The number of Fields.	(Lanza et al., 2006) [6]
privateFieldsQty	**Private Fields**. The number of private fields.	(Lanza et al., 2006) [6]
protectedFieldsQty	**Protected Fields**. The number of protected fields.	(Lanza et al., 2006) [6]
publicFieldsQty	**Public Fields**. The number of Public Fields.	(Lanza et al., 2006) [6]
staticFieldsQty	**Static Fields**. The number of static fields.	(Lanza et al., 2006) [6]
defaultFieldsQty	**Default Fields**. The number of default fields.	(Lanza et al., 2006) [6]
visibleFieldsQty	**Visible Fields**. The number of visible fields.	(Lanza et al., 2006) [6]
finalFieldsQty	**Final Fields**. The number of final fields.	(Lanza et al., 2006) [6]
logStatementsQty	**Statements**. The number of statements in a class.	(Lorenz and Kidd, 1994) [7]
staticFieldsQty	**Static Fields**. The number of static fields.	(Lanza et al., 2006) [6]
returnQty	**Returns**. The number of return instructions.	(Aniche, 2015) [1]
loopQty	**Loops**. The number of loops (i.e., for, while, do while, enhanced for).	(Aniche, 2015) [1]
comparisonsQty	**Quantity of comparisons**. The number of comparisons (i.e., == and !=).	(Aniche, 2015) [1]
tryCatchQty	**Try/catches**.The number of try/catches.	(Aniche, 2015) [1]
parenthesizedExpsQty	**Parenthesized expressions**. The number of expressions inside parenthesis.	(Aniche, 2015) [1]
stringLiteralsQty	**String literals**. The number of string literals (e.g., "John Doe"). Repeated strings count as many times as they appear.	(Aniche, 2015) [1]
numbersQty	**Quantity of Number**. The number of numbers (i.e., int, long, double, float) literals.	(Aniche, 2015) [1]

Continued on next page

Table 1 – continued from previous page

Metric	Definition	Source
assignmentsQty	**Assignments**. The number of assignments.	(Aniche, 2015) [1]
mathOperationsQty	**Math Operations**. The number of math operations (times, divide, remainder, plus, minus, left shit, right shift).	(Aniche, 2015) [1]
variablesQty	**Variables**. Number of declared variables.	(Aniche, 2015) [1]
maxNestedBlocksQty	**Max nested blocks**. The highest number of blocks nested together.	(Aniche, 2015) [1]
anonymousClassesQty	**Anonymous Classes**. The number of Anonymous classes.	(Aniche, 2015) [1]
innerClassesQty	**Inner Classes**. The number of inner classes.	(Aniche, 2015) [1]
lambdasQty	**Lambda Classes**. The number of lambda classes.	(Aniche, 2015) [1]
uniqueWordsQty	**Unique Words**. Number of unique words in the source code. At the method level, it only uses the method body as input. At the class level, it uses the entire body of the class as metric.	(Aniche, 2015) [1]
modifiers	**Modifiers**. The number of modifiers public/abstract/private/protected/native modifiers of classes/methods.	(Aniche, 2015) [1]
logStatementsQty	**Log Statements**. The number of log statements in the source code.	(Aniche, 2015) [1]
RatioCommentToCode	**Comment to Code Ratio**. The ratio of comment lines to code lines in a class.	(SciTools, 2021) [10]
AvgLine	**Average Number of Lines**. The average number of physical lines between methods of a class.	(SciTools, 2021) [10]
AvgLineComment	**Average Number of Lines with Comments**. The average number of lines containing comments between methods of a class.	(SciTools, 2021) [10]
TotalClassMethod	**Class Methods**. The number of class methods in a class.	(SciTools, 2021) [10]
variablesQty	**Class Variables**. The number of class variables in a class.	(SciTools, 2021) [10]
totalMethodsQty	**Total Methods**. The number of local methods in a class.	(SciTools, 2021) [10]
privateMethodsQty	**Private Methods**. The number of local private methods in a class.	(SciTools, 2021) [10]
protectedMethodsQty	**Protected Methods**. The number of local protected methods in a class.	(SciTools, 2021) [10]
publicMethodsQty	**Public Methods**. The number of local public methods in a class.	(SciTools, 2021) [10]
staticMethodsQty	**Static Methods**. The number of static methods in a class.	(SciTools, 2021) [10]
abstractMethodsQty	**Abstract Methods**. The number of abstract methods in a class.	(SciTools, 2021) [10]
defaultMethodsQty	**Default Methods**. The number of default methods in a class.	(SciTools, 2021) [10]
finalMethodsQty	**Final Methods**. The number of final methods in a class.	(SciTools, 2021) [10]
synchronizedMethodsQty	**Synchronized Methods**. The number of synchronized methods in a class.	(SciTools, 2021) [10]
AvgLineBlank	**Blank lines**. The average number of blanks for all nested functions or methods.	(SciTools, 2021) [10]
AvgLineCode	**Code**. Average number of lines containing source code for all nested functions or methods.	(SciTools, 2021) [10]
AvgCyclomatic	**Average Cyclomatic Complexity**. Average cyclomatic complexity for all nested functions or methods.	(SciTools, 2021) [10]
AvgCyclomaticModified	**Average Modified Cyclomatic Complexity**. Average modified cyclomatic complexity for all nested functions or methods.	(SciTools, 2021) [10]
AvgCyclomaticStrict	**Strict Cyclomatic Complexity**. Average strict cyclomatic complexity for all nested functions or methods.	(SciTools, 2021) [10]
Cyclomatic	**Cyclomatic Complexity**. Cyclomatic Complexity.	(SciTools, 2021) [10]
CyclomaticModified	**Cyclomatic Modified**. Modified cyclomatic complexity.	(SciTools, 2021) [10]
CyclomaticStrict	**Strict Cyclomatic Complexity**. Strict Cyclomatic Complexity.	(SciTools, 2021) [10]
Essential	**Essential Complexity**. Essential complexity. [aka Ev(G)]	(SciTools, 2021) [10]
MaxCyclomatic	**Max Cyclomatic Complexity**. Maximum cyclomatic complexity of nested functions or methods.	(SciTools, 2021) [10]
MaxCyclomaticModified	**Max Modified Cyclomatic Complexity**. Maximum modified cyclomatic complexity of nested functions or methods.	(SciTools, 2021) [10]
MaxCyclomaticStrict	**Max Strict Cyclomatic Complexity**. Maximum strict cyclomatic complexity of nested functions or methods.	(SciTools, 2021) [10]
MaxEssential	**Max Essential Complexity**. Maximum essential complexity of all nested functions or methods.	(SciTools, 2021) [10]
MaxInheritanceTree	**Depth of Inheritance Tree**. Maximum depth of class in inheritance tree. [aka DIT]	(SciTools, 2021) [10]
MaxNesting	**Nesting**. Nesting	(SciTools, 2021) [10]
PercentLackOfCohesion	**Lack of Cohesion in Methods**. 100% minus the average cohesion for package entities. [aka LCOM, LOCM]	(SciTools, 2021) [10]
SumCyclomatic	**Sum Cyclomatic Complexity** . Sum of cyclomatic complexity of all nested functions or methods. [aka WMC]	(SciTools, 2021) [10]
SumCyclomaticModified	**Sum Modified Cyclomatic Complexity** . Sum of modified cyclomatic complexity of all nested functions or methods.	(SciTools, 2021) [10]

Continued on next page

Table 1 – continued from previous page

Metric	Definition	Source
SumCyclomaticStrict	**Sum Strict Cyclomatic Complexity** . Sum of strict cyclomatic complexity of all nested functions or methods.	(SciTools, 2021) [10]
SumEssential	**Sum Essential Complexity** . Sum of essential complexity of all nested functions or methods.	(SciTools, 2021) [10]

Table 2 contains the process evolution-based metrics to be used. These metrics quantify the change history of the software between different releases considering the class scope. It considers newly created classes, changes in existing classes, frequency of changes, different weights to changes introduced to a class if it was changed in a far or in a recent past, the number of changes and the percentage of lines of code changed.

Table 2: Software evolution-based metrics (Source: Elish and Al-Khiaty, 2013 [4]).

Metric	Definition
BOC	**Birth of a Class**. The first time the class appears.
TACH	**Total Amount of Changes**. It is the sum of added lines, deleted lines, and twice changed lines between release $n-1$ and release n.
FCH	**First Change**. The first time the class has been exposed to changes.
LCH	**Last Change**. The last time the class has been exposed to changes.
CHO	**Change Occurred**. It is a binary metric that indicates whether or not the class has been exposed to changes from release $n-1$ to n.
FRCH	**Frequency of Changes**. The number of times (in term of releases) the class has been changed.
CHD	**Change Density**. Change density of a class C is its change size ($TACH(C)$) normalized by the size of the class (its total lines of code (LOC)).
WCD	**Weighted Change Density**. It is a cumulative frequency of change density (CHD) that favor the latest occurrence of changes over the old ones.
WFR	**Weighted Frequency of Changes**. Is a cumulative frequency of changes that favor the latest occurrence of changes over the old ones.
ATAF	**Aggregated Change Size Normalized by Frequency of Change**. This is obtained from accumulating size of changes of the class in the past and normalizing by frequency of changes.
LCA	**Last Change Amount**. It is defined as the last change size of the class when moving from release $i-1$ to release i.
LCD	**Last Change Density**. This metric is defined as its last change size (LCA) normalized by the size of the class.
CSB	**Changes since the Birth**. It is computed by comparing the size of the first version of a class with its current version.
CSBS	**Changes since the Birth Normalized by Size**. It is the CSB normalized by the size of the first version of the class
ACDF	**Aggregated Change Density Frequency**. It is obtained from cumulating density of changes introduced to the class in the past, and then this accumulated amount is normalized by the frequency of changes.

We considered sixteen code smell types that are described in Table 3 to be used as change-prone class predictors. The code smells are associated with symptoms of software maintainability problems. These types were chosen because they capture varied characteristics of code involving size, inheritance, complexity, coupling, and cohesion at the class and method levels. The smells described in Table 3 were used to calculate the smell diversity and density metrics.

Table 3: Types of code smells (Source: Rêgo, 2018 [9]).

Metric	Definition
Brain Class	Complex classes that centralize functionalities
Brain Method	Methods that centralize the intelligence of a class
Complex Class	Classes presenting a overly high cyclomatic complexity
Data Class	A class exposes its attributes, thus violating the information hiding principle
Dispersed Coupling	Occurs when a method calls methods from a large number of provider classes
Feature Envy	A method accesses the data of another object more than its own data
God Class	One class monopolizes the processing, and other classes primarily encapsulate data
Intensive Coupling	When a method calls many other methods from a few classes
Lazy Class	Understanding and maintaining classes always costs time and money. So if a class doesn't do enough to earn your attention, it should be deleted.
Long Method	A method contains too many lines of code. Generally, any method longer than ten lines should make you start asking questions
Long Parameter List	More than three or four parameters for a method

Continued on next page

Table 3 – continued from previous page

Metric	Definition
Message Chains	Message chains occur when a client requests another object, that object requests yet another one, and so on. These chains mean that the client is dependent on navigation along the class structure. Any changes in these relationships require modifying the client.
Refused Bequest	Subclass uses only some of the methods and properties inherited from its parents, the hierarchy is off-kilter. The unneeded methods may simply go unused or be redefined and give off exceptions.
Shotgun Surgery	Making any modifications requires that you make many small changes to many different classes.
Spaghetti Code	Declare a number of long methods without parameters
Speculative Generality	There's an unused class, method, field, or parameter.

REFERENCES

[1] Maurício Aniche. 2015. *Java code metrics calculator (CK)*. Available in https://github.com/mauricioaniche/ck/.

[2] James M Bieman and Byung-Kyoo Kang. 1995. Cohesion and reuse in an object-oriented system. *ACM SIGSOFT Software Engineering Notes* 20, SI (1995), 259–262.

[3] S.R. Chidamber and C.F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (1994), 476–493. https://doi.org/10.1007/s11219-020-09525-y

[4] Mahmoud O Elish and Mojeeb Al-Rahman Al-Khiaty. 2013. A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software. *Journal of Software: Evolution and Process* 25, 5 (2013), 407–437.

[5] Sallie Henry and Dennis Kafura. 1981. Software structure metrics based on information flow. *IEEE transactions on Software Engineering* 1, 5 (1981), 510–518.

[6] Michele Lanza and Radu Marinescu. 2010. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems.* Springer.

[7] M. Lorenz and J. Kidd. 1994. *Object-oriented software metrics - a practical guide.* Prentice-Hall, Inc.

[8] T.J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (1976), 308–320. https://doi.org/10.1109/TSE.1976.233837

[9] Diego Cedrim Gomes Rêgo. 2018. *Understanding and Improving Batch Refactoring in Software Systems.* Ph. D. Dissertation. Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

[10] Scitools. 2021. What Metrics Does Undertand Have? https://support.scitools.com/t/what-metrics-does-undertand-have/66.