Table 3: The software metrics collected in the dataset. Complete descriptions can be found in the SonarQube documentation

Metric	Description
Size	
Number of classes	Number of classes (including nested classes, interfaces, enums and annotations).
Number of files	Number of files.
Lines	Number of physical lines (number of carriage returns).
Ncloc	Also known as Effective Lines of Code (eLOC). Number of physical lines that contain at least one character which is neither a whitespace nor a tabulation nor part of a comment.
Ncloc language distribution	Non Commenting Lines of Code Distributed By Language
Number of classes and interfaces	Number of Java classes and Java interfaces
Missing package info	Missing package-info.java file (used to generate package-level documentation)
Package	Number of packages
Statements	Number of statements.
Number of directories	Number of directories in the project, also including directories not containing code (e.g., images, other files...).
Number of functions	Number of functions. Depending on the language, a function is either a function or a method or a paragraph.
Number of comment lines	"Number of lines containing either comment or commented-out code. Non-significant comment lines (empty comment lines, comment lines containing only special characters, etc.) do not increase the number of comment lines."
Number of comment lines density	Density of comment lines = Comment lines / (Lines of code + Comment lines) * 100
Complexity	
Complexity	It is the Cyclomatic Complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. This calculation varies slightly by language because keywords and functionalities do.
Class complexity	Complexity average by class
Function complexity	Complexity average by method
Function complexity distribution	Distribution of method complexity
File complexity distribution	Distribution of complexity per class
Cognitive complexity	How hard it is to understand the code's control flow.
Package dependency cycles	Number of package dependency cycles
Test coverage	
Coverage	It is a mix of Line coverage and Condition coverage. Its goal is to provide an even more accurate answer to the following question: How much of the source code has been covered by the unit tests?
Lines to cover	Number of lines of code which could be covered by unit tests (for example, blank lines or full comments lines are not considered as lines to cover).
Line coverage	On a given line of code, Line coverage simply answers the following question: Has this line of code been executed during the execution of the unit tests?
Uncovered lines	Number of lines of code which are not covered by unit tests.
Duplication	
Duplicated lines	Number of lines involved in duplications
Duplicated blocks	Number of duplicated blocks of lines.
Duplicated files	Number of files involved in duplications.
Duplicated lines density	= (duplicated lines ÷ lines) * 100

Table 4: Code smells and anti-patterns detected in our projects. Descriptions are adapted from [10] and [1].

Name	Description
Duplicated code	The same code reused in different locations.
Blob	A big class (usually a singleton) that has dependencies with data contained in other data classes. It could monopolize several system operations.
Class data should be private	Class publicly exposing variables.
Cyclomatic complexity	Also referred to as McCabe's Cyclomatic Complexity. Refers to methods that should be simplified since they have too many independent execution paths.
Downcasting	A cast to a derived class.
Excessive use of literals	Too many literal variables embedded into the code instead of being declared as constants.
Feature envy	An object gets at the fields of another object to perform some sort of computation or make a decision, rather than asking the object to do the computation itself.
Functional decomposition	A class with too many functionalities, which needs to be broken down into smaller and simpler classes.
God Class	A huge class implementing different responsibilities.
Inappropriate intimacy	A method that has too much intimate knowledge of the inner workings, inner data... of another class or method.
Large class	A class that is too big.
Lazy class/Freeloader	A class with very limited functionalities.
Orphan variable or constant class	A class containing variables used in other classes.
Refused bequest	A children class that never uses the inherited methods.
Spaghetti code	A class with a very complex control flow.
Speculative generality	An abstract class with a very limited number of children who are not using its methods.
Swiss army knife	A class that is providing many services for different purposes, such as a Swiss army knife.
Tradition breaker	A class that, despite inheriting from another class, does not fully extend the parent class and has no subclasses.
Excessively long identifiers	Variable names that re too long and do not respect the naming conventions.
Excessively short identifiers	Variable names that are too short and do not allow understanding the purpose of the variable.
Excessive return of data	A method that returns more than what is needed from the calling methods.
Long method	A method that is too long.
Too many parameters	Methods with too many parameters, which usually become hard to read and to test.