ClassName Distribution Visualization: detecting inconsistencies in class names

Nour Jihene Agouf*[1], Stéphane Ducasse[2], and Anne Etien[3]

[1]Inria Lille - Nord Europe – Institut National de Recherche en Informatique et en Automatique – France
[2]Inria Lille - Nord Europe (Inria Lille - Nord Europe) – Institut National de Recherche en Informatique et en Automatique – Parc Scientifique de la Haute Borne 40, avenue Halley Bât.A, Park Plaza 59650 Villeneuve dÁscq, France
[3]Inria Lille - Nord Europe – Institut National de Recherche en Informatique et en Automatique – France

Résumé

Understanding software starts with understanding the functionalities of its classes. These functionalities are summarized in a simple descriptive class name. Class names should be both correct in the sense that they refer to the exact functionality of the class and, consistent with the system's naming convention. However, not all class names fulfill these criteria, which makes it hard for developers to understand the logic behind thousands of lines of source code. The lack of software understanding leads to an unorganized software evolution. Luckily, with the use of simple techniques such as visualizations, large data can be transformed from an abstract form into visual shapes familiar to the human brain which makes it easy for developers to assess and memorize the logic behind the lines of source code. Indeed, our approach is based on a visualization called 'ClassNames Distribution' which gives an overview of the distribution of classes over packages and helps in detecting inconsistencies in class names from an inheritance perspective. The idea behind inheritance consistent naming is that a hierarchy represents a family of classes having common behavior. This behavior is usually described at the end of the class name when programming in English, other behavior might also emerge therefore a new vocabulaire is used. 'ClassNames Distribution' visualization does not only give the opportunity for developers to have an overview of the system's architecture and the vocabulary used to avoid future naming violations but it is firstly intended to help in detecting suspicious misnaming cases in order to correct them for a better software evolution.

*Intervenant