A Quantitative Study of Code Complexity in OO Languages

Nick Cooper, Neelima Prasad, SriKrishna B.R.

September 20, 2024

1 Motivation

Object-Oriented programming languages are one of the most ubiquitous and powerful languages. Top industries utilize them. While they are similar at a very high level, OO langauges are drastically different and each one has certain strength and weaknesses. Learning about different concepts in class that object oriented languages posses, we wanted to understand the design of these languages and study their design in great detail

Our project aims to quantitatively measure the structural complexity of software applications across a range of OO-centric languages. Through this analysis, we hope to detect interesting relationships between language features and end-result programs. As an example, can we formally answer questions like: does multiple inheritance increase or decrease code complexity, on average?

2 Topic Content

Our project aims to quantitatively measure the structural complexity of software applications across three distinct OO-centric languages: Java, Ruby, and Python. We wanted to compare languages that came about during different time periods to better understand the trend of capabilities that OO languages started to harness. Our comparison would include both qualitiatve and quantitative components. Our qualitative analysis will include understanding how each language handles inheritance, polymorphism, abstraction/encapsulation and how this affects the complexity of the language. Our quanititave analysis will compare each language's complexity and compile/run heuristics. There will be two primary technical challenges associated with this project: data, and the quantification of complexity. The first is rather self explanatory; a large dataset of code written in OO languages is critical, as understanding large scale trends will be impossible without a decent sample size. Thankfully, many code datasets have emerged in recent years, thanks to the AI summer. See: Github Code Dataset for an example. The second challenge is far more interesting: how to measure complexity? Because this study will focus on OO languages, we inherit (pun fully intended) some nice theoretical properties right off the bat. Objects and their relations can be easily modeled with many mathematical constructs, such as graphs, categories, and combinatorial complexes, all of which bring their own set of analytical tools. As a simple example, an OO program could be modeled with a graph (possibly a tree), and various notions of complexity explored thereafter: diameter, size of the center, average degree, etc. Many of the aforementioned mathematical structures admit natural notions of "neighborhood", allowing topological summaries of complexity such as the persistent entropy (see: practical explanation) to be used as well. In summary, we hypothesize that interesting patterns can be extracted by studying the mathematical structure of OO software.