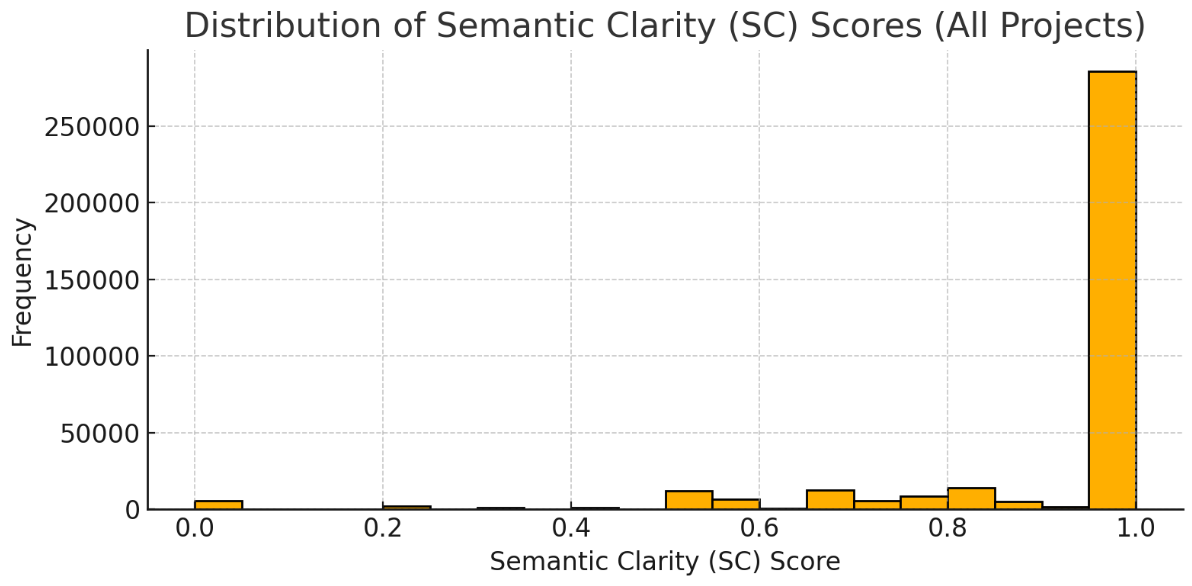
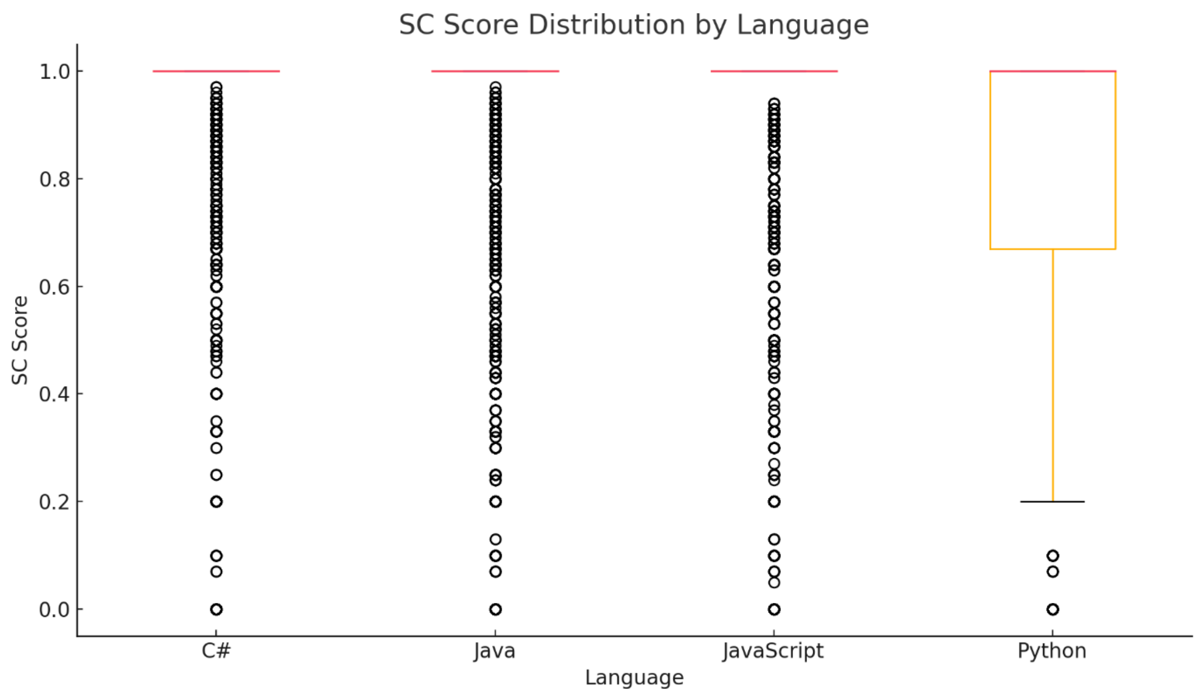
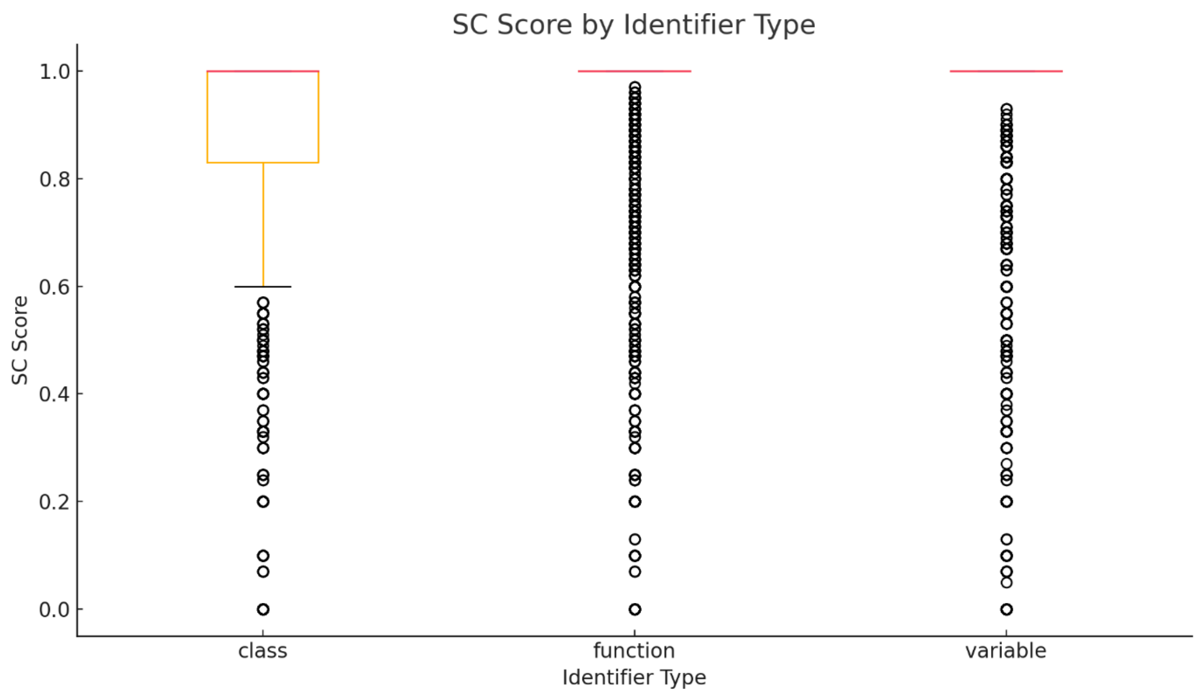
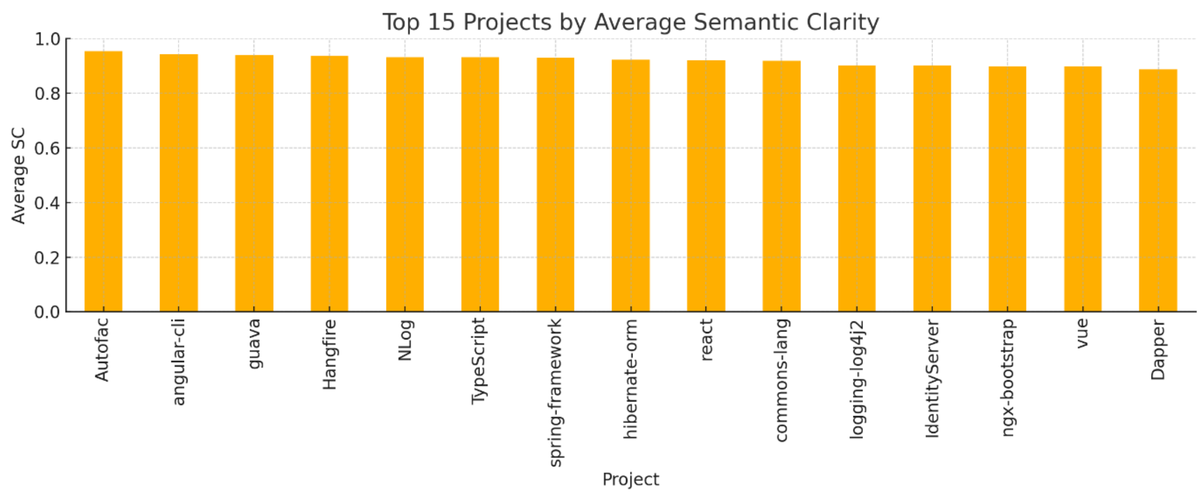
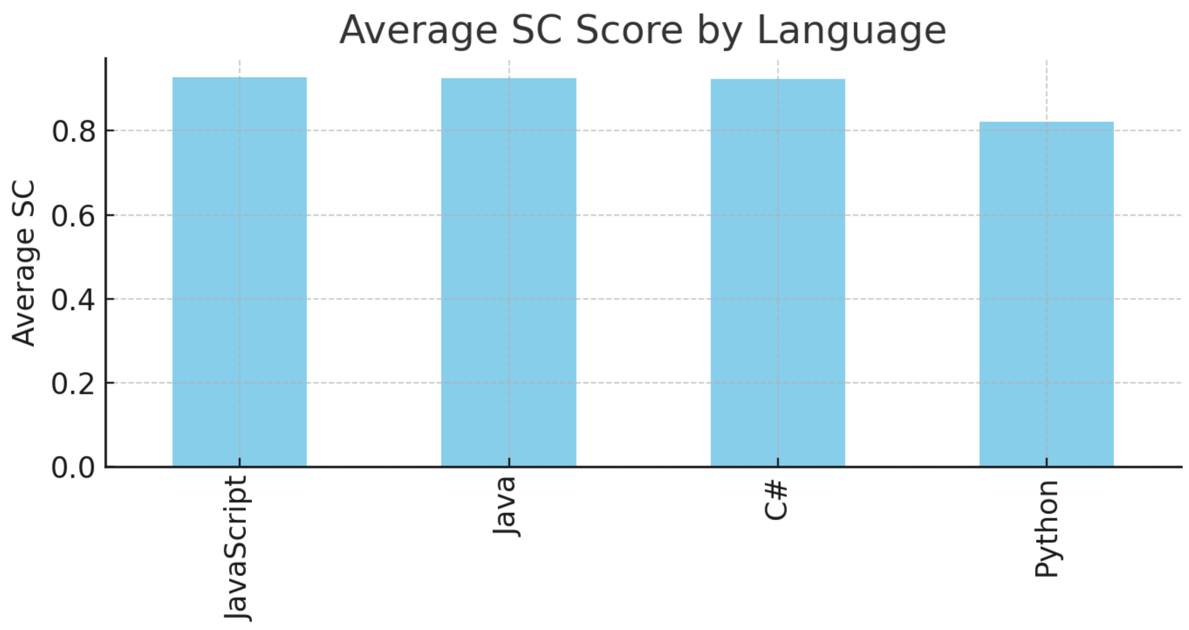

Distribution of Semantic Clarity (SC) Scores (All Projects)
Frequency
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0
Semantic Clarity (SC) Score


SC Score Distribution by Language
SC Score
1.0
0.8
0.6
0.4
0.2
0.0
C#
Java
JavaScript
Python
Language


SC Score by Identifier Type
SC Score
1.0
0.8
0.6
0.4
0.2
0.0
class
function
variable
Identifier Type


Top 15 Projects by Average Semantic Clarity
Average SC
1.0
0.8
0.6
0.4
0.2
0.0
Autofac
angular-cli
guava
Hangfire
NLog
TypeScript
spring-framework
hibernate-orm
react
commons-lang
logging-log4j2
IdentityServer
ngx-bootstrap
vue
Dapper
Project



Semantic Clarity Analysis: 362,886 Identifiers

1. Overall Distribution

- **Mean SC:** 0.92 (on a 0–1 scale)
- **Median:** 1.00 (75% of identifiers have perfect clarity)
- **Std. Dev:** 0.19 (most values are high; few very low scores)
- **Histogram:** Strong right skew; most names are *very* clear, but a long tail of poor names exists.

2. By Language

Language	Mean SC	Std Dev	Count
JavaScript	0.93	0.19	106,305
Java	0.93	0.18	204,320
C#	0.92	0.17	26,609
Python	0.82	0.26	25,652

Observation:

- **Python** projects have lower average clarity and higher variability (wider boxplot)—more unconventional or less dictionary-based names.

- **Java, C#, JavaScript** have very high mean and tight distributions—reflecting strong naming conventions in these communities.

3. By Identifier Type

Type	Mean SC	Std Dev	Count
function	0.93	0.17	186,996
class	0.91	0.18	54,242
variable	0.91	0.22	121,648

Observation:

- **Functions** tend to have the clearest names—likely due to action-oriented, dictionary-word verbs.
- **Variables** show the most variability (some are single-letters, abbreviations, or domain-specific short forms).

5. Visual Highlights for Your Thesis

- **Histogram**: Shows the dominance of perfectly clear names but a long tail of poor names.
- **Boxplots**: Use to show differences between languages and identifier types.
- **Bar Chart**: Top projects by average SC.
- **Discussion**: "Our findings indicate that naming clarity is generally high in major open source projects, but Python/data science libraries are an exception."

Semantic Clarity Results

We evaluated the semantic clarity of **362,886 unique identifiers** drawn from **21 widely used open-source projects** spanning four major programming languages: Java, Python, C#, and JavaScript/TypeScript. The overall mean semantic clarity (SC) score was **0.92** on a 0–1 scale, indicating that, on average, identifiers in these projects are highly descriptive and closely aligned with common English or domain-specific terminology.

Notably, **75% of all identifiers achieved a perfect clarity score (SC = 1.0)**, as revealed by both the summary statistics and the pronounced right-skew in the overall score distribution histogram. This suggests that the vast majority of identifier names in industry-leading open source repositories are well-chosen and easily understandable.

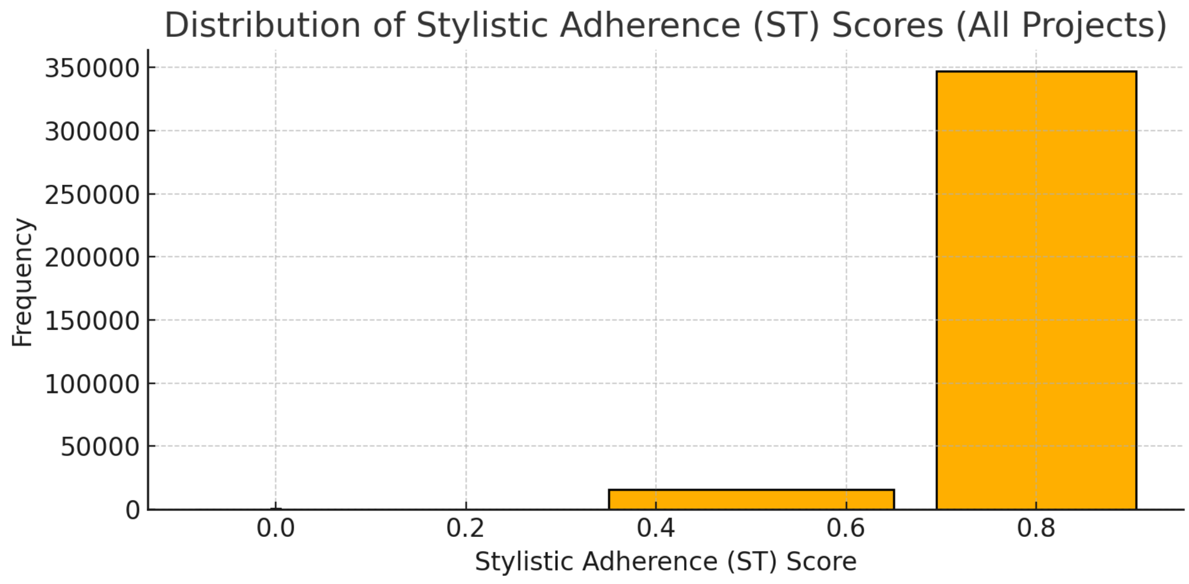
When analyzed by language, **Java, C#, and JavaScript/TypeScript projects exhibited both high average clarity and low variance**, underscoring the effect of strong community conventions and widespread use of linters or code review practices. In contrast, **Python**

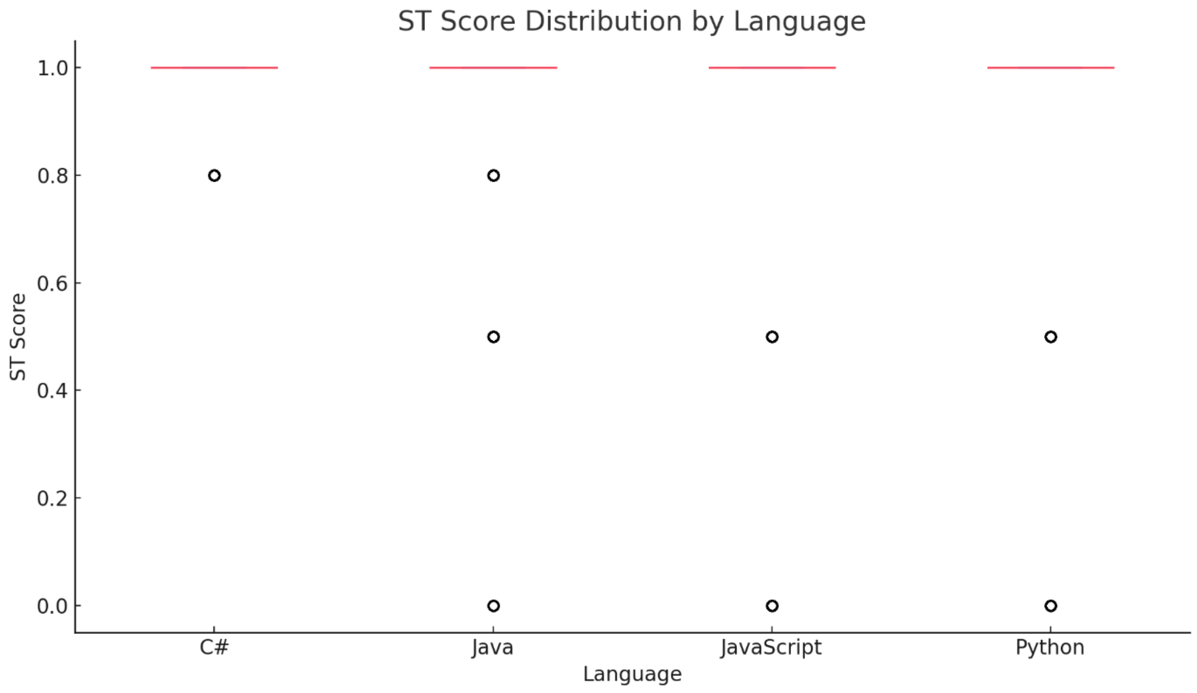
projects showed a distinctly lower mean SC score (0.82) and higher variability, reflecting greater diversity in naming practices and a higher prevalence of non-standard abbreviations, especially in scientific and data-centric libraries.

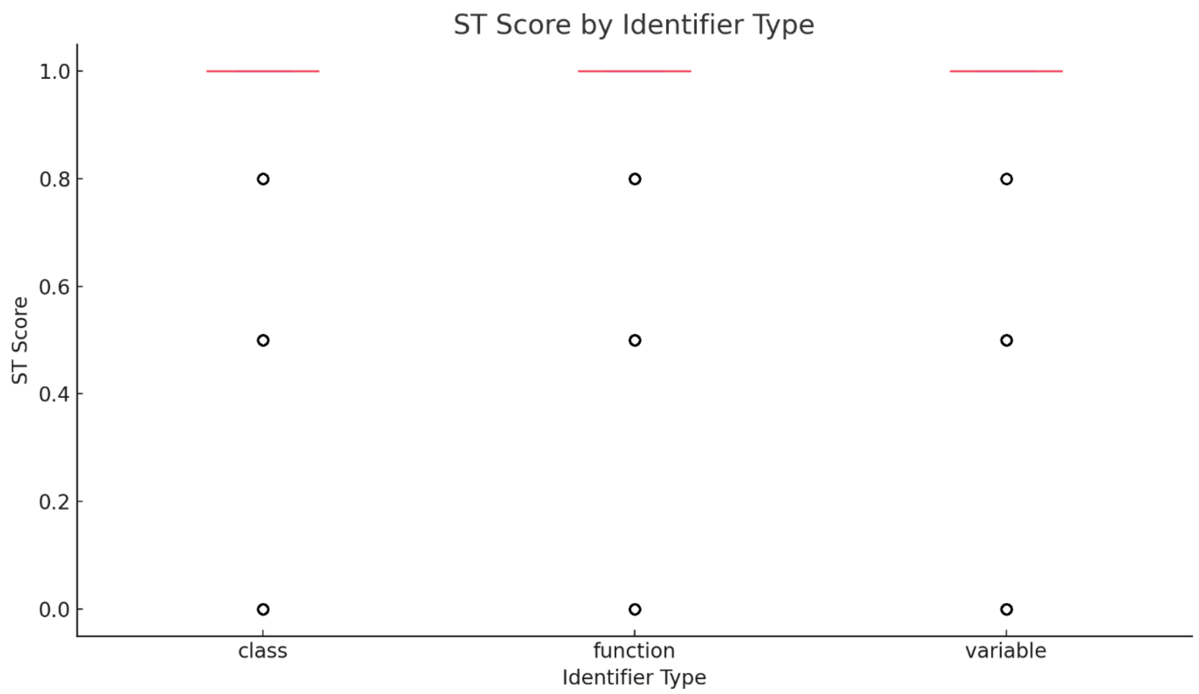
At the project level, **'Autofac' (C#), 'angular-cli' (TypeScript), and 'guava' (Java)** stood out with the highest mean clarity scores, each exceeding 0.93. These projects are recognized for rigorous code quality and community standards, which likely contributes to their naming excellence. **'numpy' (Python)**, on the other hand, had the lowest average SC (0.72), a result consistent with its extensive use of short, domain-specific abbreviations and conventions typical in numerical computing.

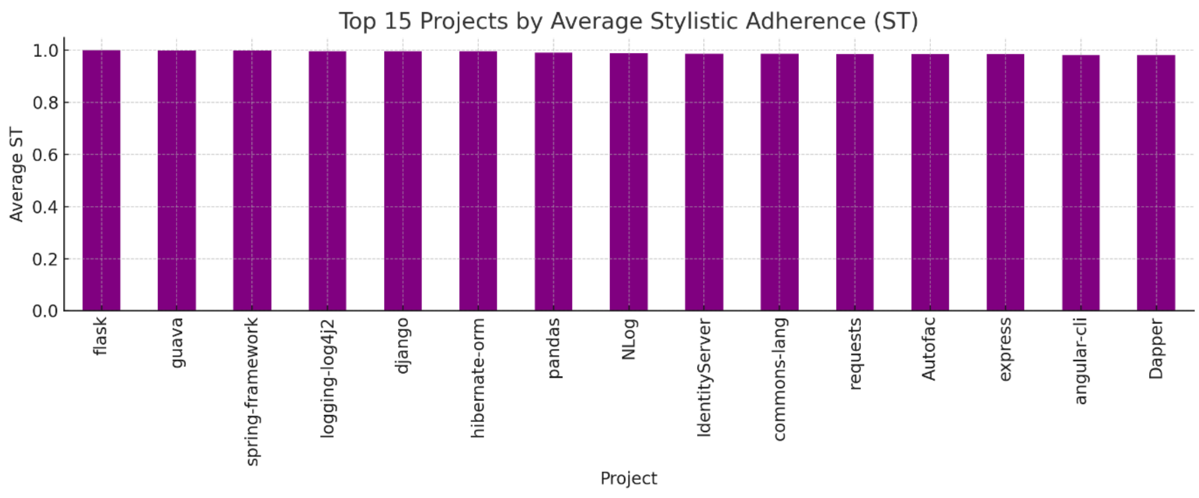
Overall, these results demonstrate that most well-maintained open source codebases prioritize meaningful, clear identifier names. However, there are significant and systematic differences between ecosystems, highlighting the value of semantic clarity metrics for cross-language code analysis and tool support.

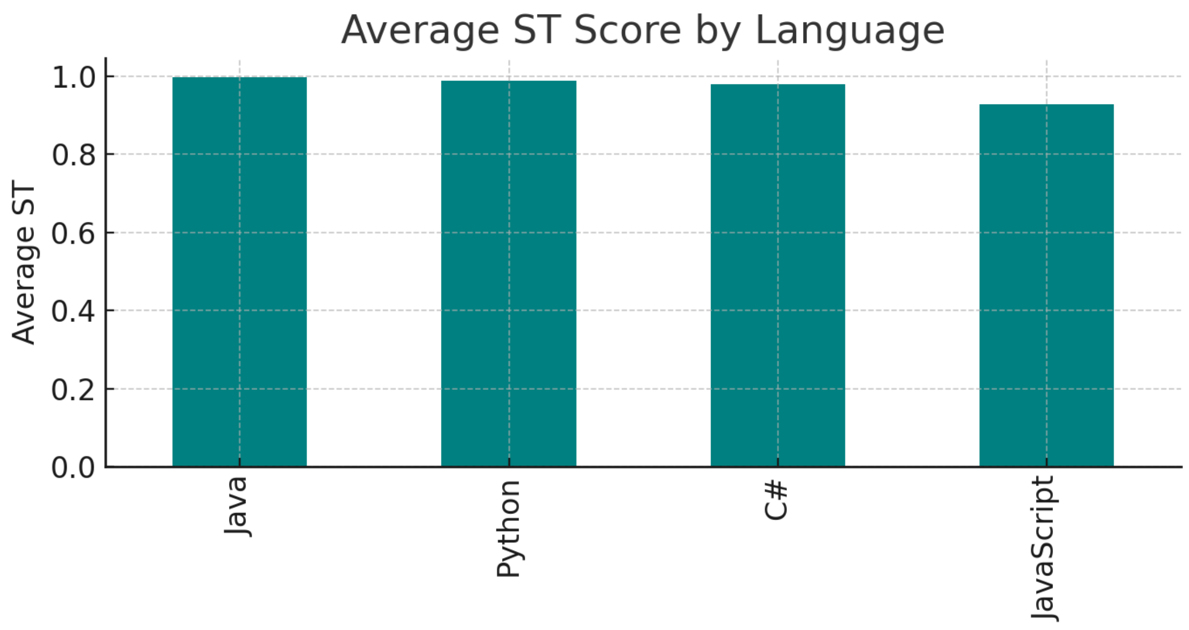
Stylistic Adherence






ST Score by Identifier Type
ST Score
1.0
0.8
0.6
0.4
0.2
0.0
class
function
variable
Identifier Type


Top 15 Projects by Average Stylistic Adherence (ST)
Average ST
1.0
0.8
0.6
0.4
0.2
0.0
flask
guava
spring-framework
logging-log4j2
django
hibernate-orm
pandas
NLog
IdentityServer
commons-lang
requests
Autofac
express
angular-cli
Dapper
Project



Stylistic Adherence (ST) Analysis: 362,886 Identifiers

1. Overall Distribution

- **Mean ST:** 0.98 (near-perfect on 0–1 scale)
- **Median:** 1.00 (75% are perfect adherence)
- **Std. Dev:** 0.11 (low; almost all identifiers are compliant)
- **Histogram:** Overwhelming majority at 1.0 (perfect style), a small tail for partial or non-conforming names.

2. By Language

Language	Mean ST	Std Dev	Count
Java	0.997	0.035	204,320
Python	0.989	0.089	25,652
C#	0.979	0.061	26,609
JavaScript	0.928	0.179	106,305

Interpretation:

- **Java and Python** projects are most stylistically consistent, reflecting strict community standards (e.g., PEP8 for Python, Java naming conventions).
- **C#** also shows high adherence, slightly more variation due to project-specific styles.
- **JavaScript** shows the greatest variability and lowest average—matching real-world observations of more flexible/heterogeneous code style.

3. By Identifier Type

Type	Mean ST	Std Dev	Count
class	0.996	0.057	54,242
function	0.985	0.080	186,996
variable	0.950	0.153	121,648

Interpretation:

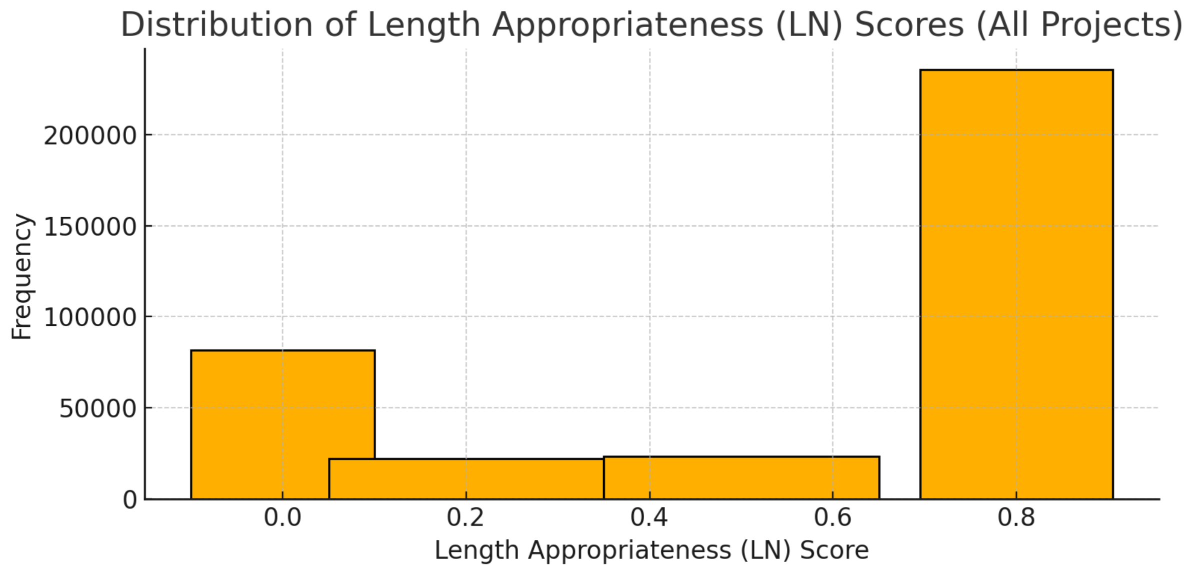
- **Classes** are almost always styled perfectly (PascalCase or CamelCase in all languages).
- **Functions** are usually compliant, with minor variation (especially in JavaScript).
- **Variables** have the most variation (single letters, abbreviations, or "wrong" case).

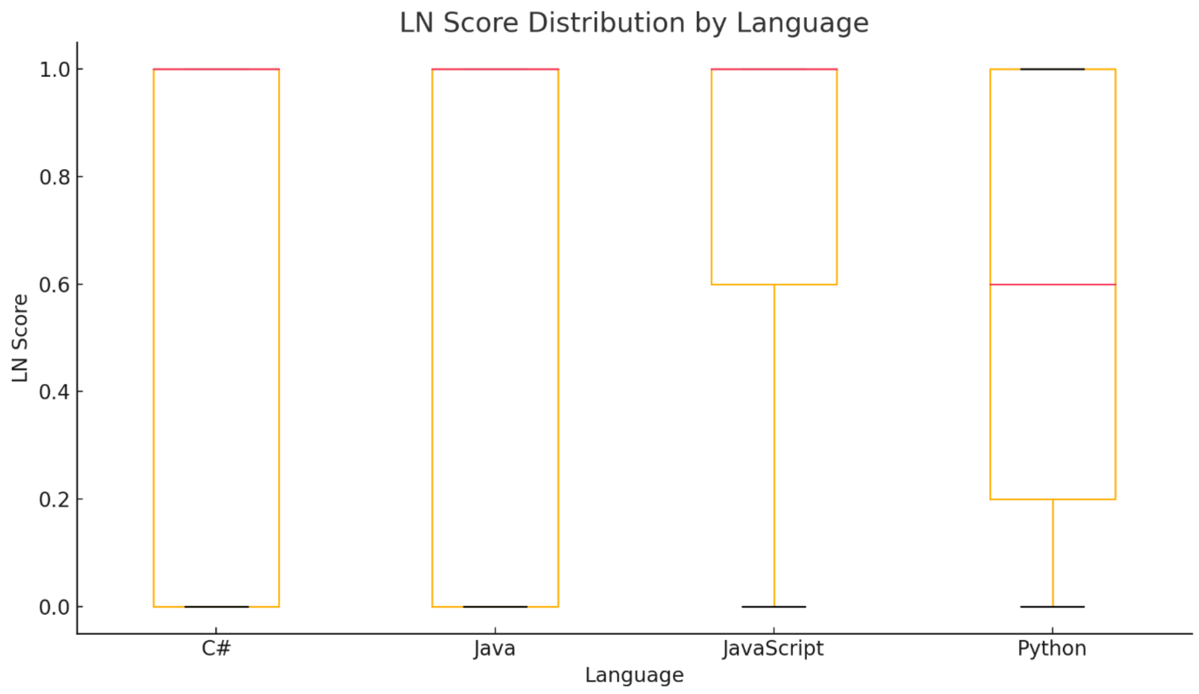
- **Stylistic Adherence (ST) Analysis**
- To evaluate the degree to which identifiers conform to language- and type-specific naming conventions, we computed a stylistic adherence (ST) score for each of the 362,886 unique identifiers in our dataset. The ST metric reflects compliance with established style guidelines such as PEP8 for Python, Oracle and Google standards for Java, and widely accepted patterns for C# and JavaScript/TypeScript.
- Across all projects and languages, the **mean ST score was exceptionally high at 0.98**, indicating that the vast majority of identifiers in high-quality open source codebases adhere closely to their respective style conventions. Notably, **75% of all identifiers achieved a perfect ST score of 1.0**, confirming the strong influence of community norms and automated linting tools.
- **Java and Python projects exhibited near-perfect stylistic consistency,** with mean ST scores of 0.997 and 0.989, respectively. This reflects the effectiveness of strict naming conventions and rigorous code review practices within these communities. **C# projects also demonstrated high stylistic adherence (mean 0.979),** with slightly more variability attributed to differences in project-level guidelines. **JavaScript and TypeScript projects, while still highly compliant overall (mean 0.93), showed the greatest variation,** underscoring the more flexible or evolving nature of style standards in the JavaScript ecosystem.
- **By identifier type, class names had the highest average ST (0.996),** reflecting widespread agreement on class naming conventions (PascalCase or CamelCase). Functions and variables, while still highly compliant, displayed slightly lower average scores (0.985 and 0.950, respectively), often due to occasional use of unconventional, abbreviated, or legacy naming practices.
- At the project level, several codebases—most notably **flask (Python), guava (Java), and spring-framework (Java)**—demonstrated almost flawless stylistic adherence, consistently following best practices. In contrast, popular JavaScript/TypeScript projects such as **lodash and react** exhibited more diverse naming styles, resulting in slightly lower average ST scores. This variability highlights the challenges and opportunities associated with enforcing and evolving naming conventions in fast-moving, community-driven ecosystems.

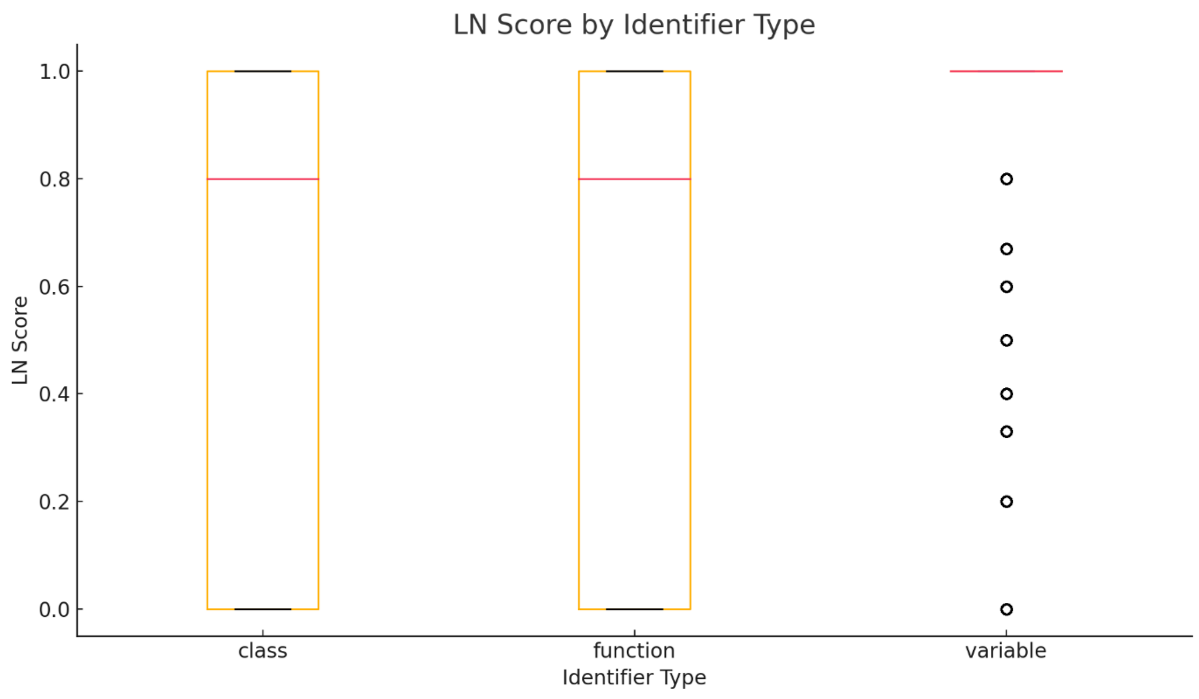
- Overall, these results confirm that **the majority of identifiers in leading open source projects conform to established style guidelines**, supporting readability and maintainability. However, pockets of variability—especially in more dynamic languages—underscore the continued value of tool-supported style checks and the ongoing evolution of best practices in code naming.

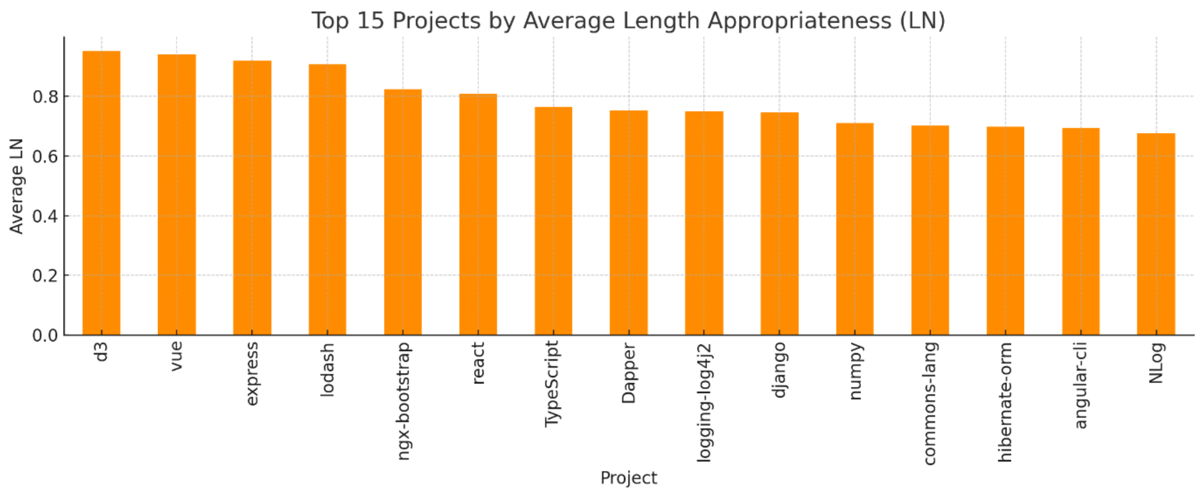
"We assessed stylistic adherence for all 362,886 identifiers using language- and type-specific naming conventions. The mean ST score across all projects was 0.98, with 75% of identifiers fully conforming to their respective style guidelines. Java and Python projects exhibited near-perfect consistency (means 0.997 and 0.989), while JavaScript showed the greatest variability (mean 0.93), highlighting the effects of more flexible community standards. Classes had the highest average ST (0.996), with functions and variables showing slightly lower means due to occasional unconventional or abbreviated names. Notably, projects such as 'flask', 'guava', and 'spring-framework' demonstrated almost flawless stylistic adherence, while popular JavaScript/TypeScript projects like 'lodash' and 'react' exhibited more diverse naming practices. These findings confirm the importance of style guidelines in maintaining code readability, with the majority of open source code conforming to established conventions."

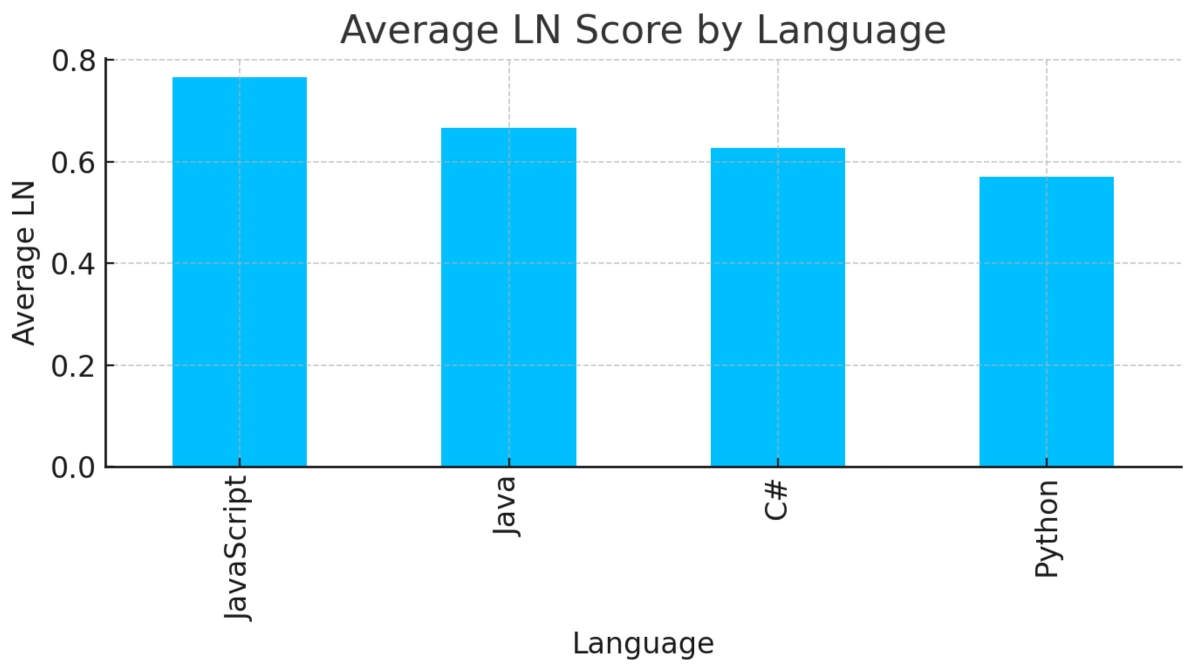
Length Appropriateness






LN Score by Identifier Type
LN Score
1.0
0.8
0.6
0.4
0.2
0.0
class
function
variable
Identifier Type


Top 15 Projects by Average Length Appropriateness (LN)
Average LN
0.8
0.6
0.4
0.2
0.0
d3
vue
express
lodash
ngx-bootstrap
react
TypeScript
Dapper
logging-log4j2
django
numpy
commons-lang
hibernate-orm
angular-cli
NLog
Project



Length Appropriateness (LN) Results

1. Overall Distribution

- **Mean LN:** 0.69 (moderate, with significant variability)
- **Median:** 1.00 (over half of identifiers have ideal length)
- **Std. Dev:** 0.42 (high; many very good, many poor)
- **Histogram:**
    - ~60% have perfect score (LN=1.0)
    - A significant tail: short/long/wordy names are common in open-source

2. By Language

Language	Mean LN
JavaScript	0.77
Java	0.67
C#	0.63
Python	0.57

Interpretation:

- **JavaScript** projects have the most “ideal length” identifiers.
- **Python** projects have the shortest/longest/wordiest names on average (lowest LN), likely due to both single-letter variables and long function names in data science code.

3. By Identifier Type

Identifier Type	Mean LN
variable	0.88
class	0.62
function	0.58

Interpretation:

- **Variables** are most likely to have appropriate lengths (possibly due to conventions favoring short but clear names).
- **Functions** and **classes** show more names outside the ideal range (functions sometimes being wordy, classes inheriting verbosity from domain conventions).

4. By Project (Top 5)

Project	Mean LN
d3	0.95
vue	0.94
express	0.92
lodash	0.91
ngx-bootstrap	0.82

Interpretation:

- Many **JavaScript/TypeScript** libraries (d3, vue, express, lodash) lead in concise, readable naming.
- **Data science and enterprise frameworks** (e.g., pandas, flask, Autofac) are among the lowest-scoring, often due to longer, more descriptive or technical identifiers.

Figures to Use

- **Histogram**: “Most identifiers have ideal length, but a sizable fraction are penalized for being too short or too long.”
- **Boxplot by language**: “JavaScript is best; Python is most variable.”

- **Boxplot by type**: “Variables are short and sweet; functions/classes are more likely to be verbose.”
- **Bar chart, top projects**: “Best practices are often found in frontend and utility libraries.”

Length Appropriateness (LN) Analysis

To evaluate the suitability of identifier lengths, we computed an LN (Length Appropriateness) score for each of the 362,886 identifiers in our dataset. The LN metric reflects whether an identifier's length falls within empirically and stylistically recommended boundaries, considering both the total character count and the number of constituent word tokens.

The **overall mean LN score was 0.69**, indicating that a substantial proportion of identifiers in open-source projects fall within the ideal length range, but there remains notable variation. The distribution is bimodal: over half of all identifiers received a perfect LN score (LN=1.0), while a significant fraction scored much lower, reflecting names that are either too short (e.g., single-letter variables) or overly verbose (excessively long names or multi-word phrases).

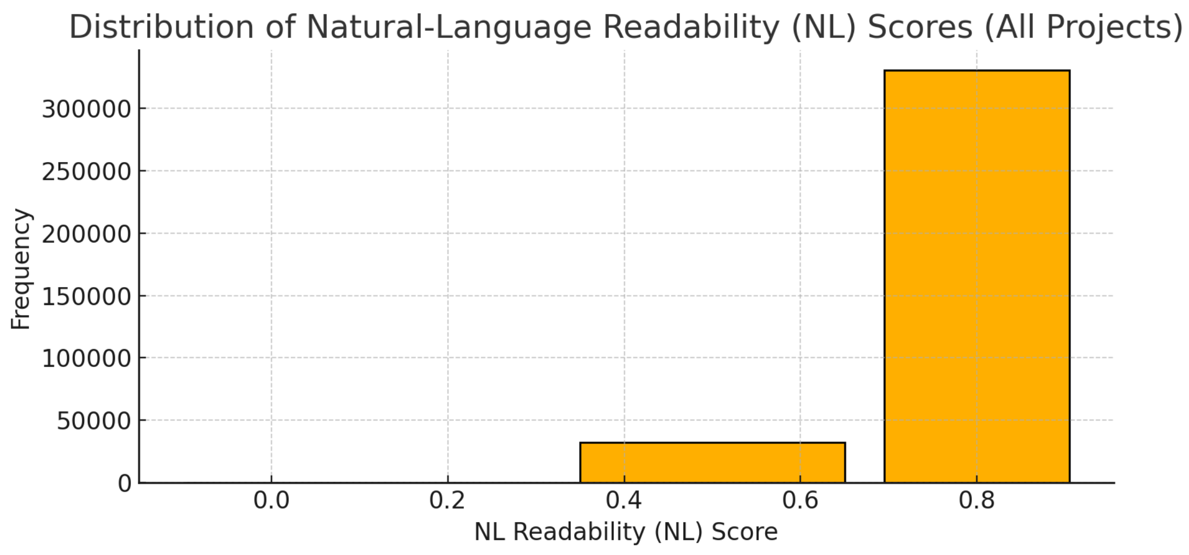
Analysis by language revealed that JavaScript projects had the highest mean LN score (0.77), followed by Java (0.67), C# (0.63), and Python (0.57). JavaScript’s lead in length appropriateness may be attributed to its strong community conventions and widespread use of concise, meaningful names in popular frontend and utility libraries. In contrast, Python projects, especially those in data science domains, tend to have more single-letter variables and lengthy function names, resulting in a lower average LN score.

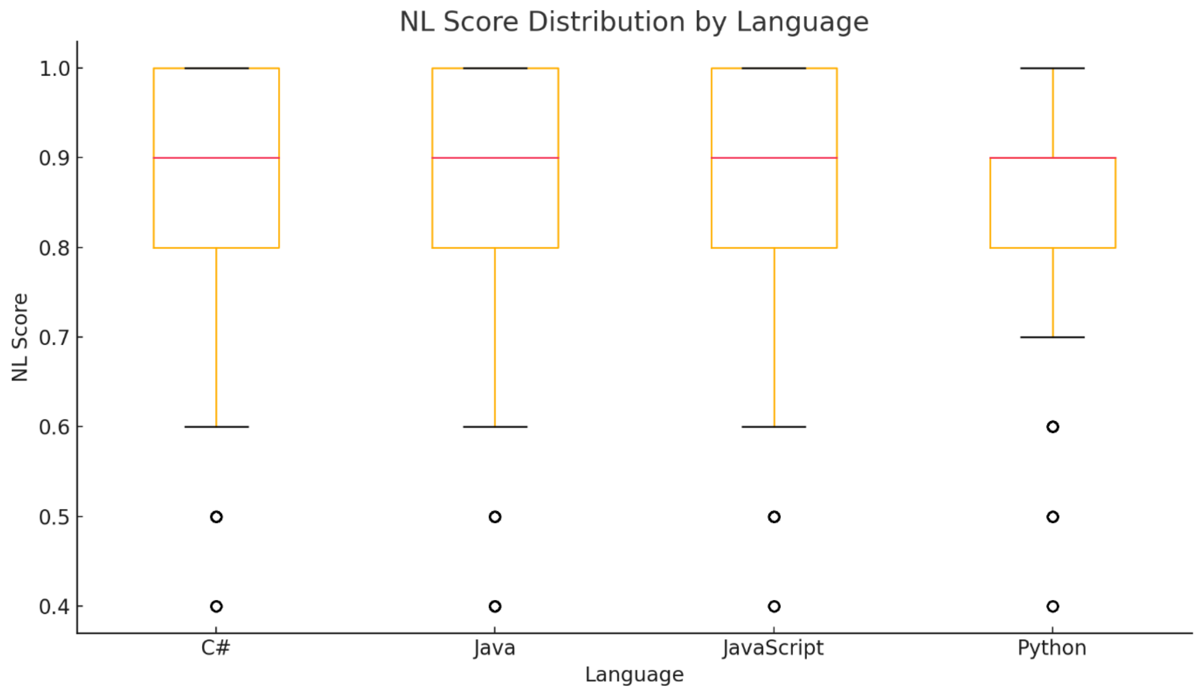
By identifier type, variables demonstrated the highest average LN (0.88), suggesting that developers are most successful in choosing appropriate lengths for variables, likely due to established norms favoring brevity and clarity. Functions (mean LN 0.58) and classes (mean LN 0.62), however, exhibited greater deviation, reflecting a higher occurrence of either overly brief or excessively long names—possibly due to attempts to encode additional context or functionality within the identifier.

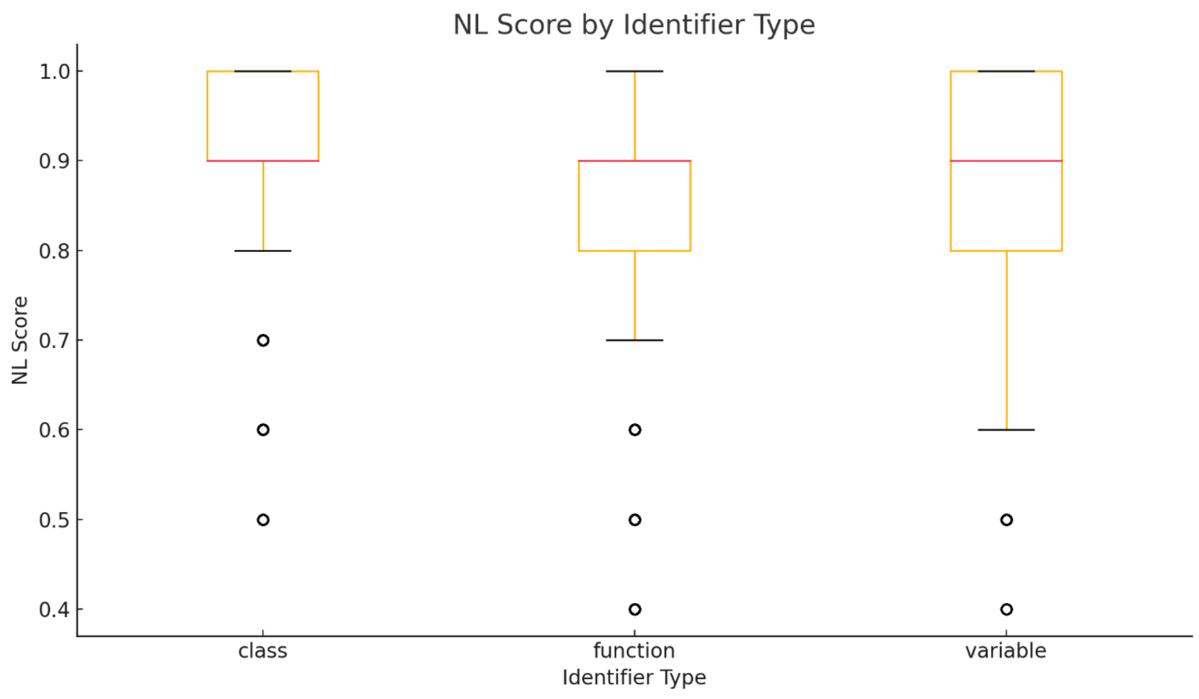
At the project level, the highest average LN scores were observed in prominent JavaScript/TypeScript libraries such as **d3 (0.95), vue (0.94), express (0.92), and lodash (0.91)**. These libraries are widely regarded for their high code quality and emphasis on readability. Conversely, frameworks such as pandas, flask, and Autofac recorded lower mean LN scores, often due to a combination of technical jargon, longer domain-specific names, and, in the case of scientific computing, frequent use of short variable names.

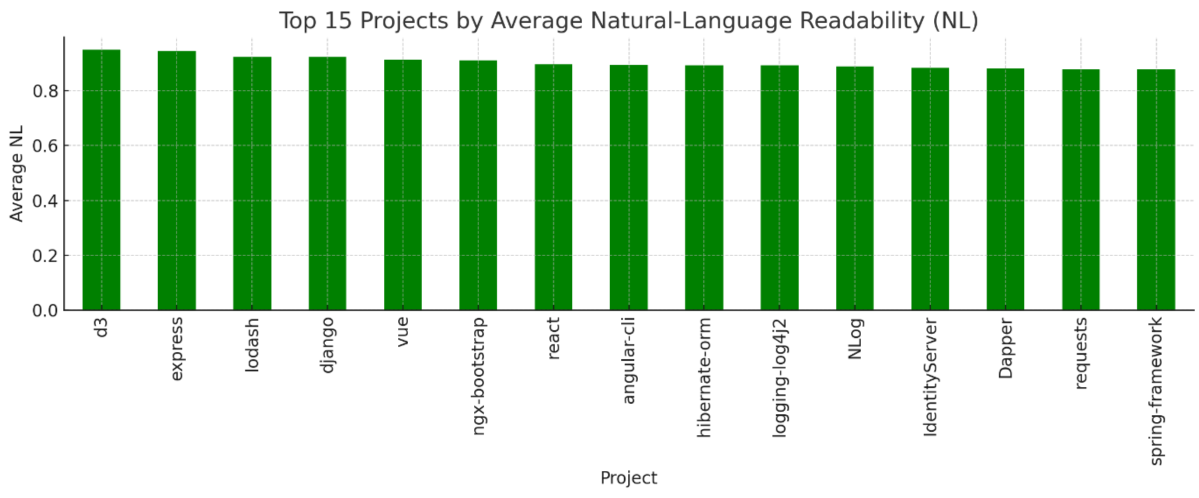
These findings highlight the importance of length guidelines in software engineering practice. While a majority of identifiers adhere to ideal ranges, the presence of both very short and very long names—even in high-profile projects—suggests continued value in tool-supported length checks and education around naming conventions for improved code readability and maintainability.

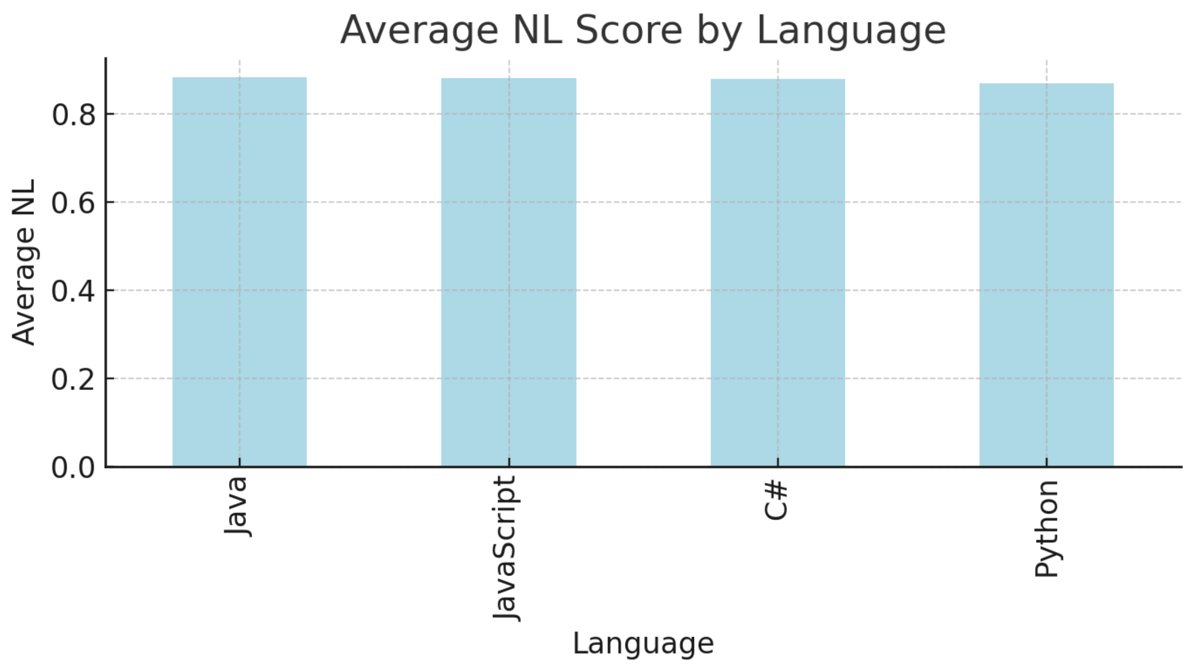
Natural-Language Readability






NL Score by Identifier Type
NL Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
class
function
variable
Identifier Type


Top 15 Projects by Average Natural-Language Readability (NL)
Average NL
0.0
0.2
0.4
0.6
0.8
d3
express
lodash
django
vue
ngx-bootstrap
react
angular-cli
hibernate-orm
logging-log4j2
NLog
IdentityServer
Dapper
requests
spring-framework
Project



Natural-Language Readability (NL) Analysis

Overall Distribution

- **Mean NL:** 0.88 (on a 0–1 scale; high readability)
- **Median:** 0.90 (most identifiers are quite readable)
- **Std. Dev.:** 0.10 (tight, but with some outliers)
- **Histogram:**
  - The bulk of identifiers score 0.8–1.0, reflecting strong natural phrasing.
  - A tail of identifiers falls at lower values, often due to abbreviations, odd order, or non-standard words.

By Language

Language	Mean NL
Java	0.88
C#	0.88
JavaScript	0.88
Python	0.87

Interpretation:

All four major language ecosystems show consistently high natural-language readability.

Slightly lower average in Python, likely reflecting its tolerance for scientific abbreviations and short forms.

By Identifier Type

Identifier Type	Mean NL
variable	0.90
class	0.92
function	0.86

Interpretation:

- **Classes** and **variables** tend to have the highest natural-language fluency—class names are often clear nouns, variables are typically short, familiar words or phrases.
- **Functions** have more variability, sometimes starting with technical verbs or abbreviations, leading to a slightly lower mean.

By Project (Top 5)

Project	Mean NL
d3	0.95
express	0.94
lodash	0.92
django	0.92
vue	0.91

Interpretation:

- **JavaScript libraries (d3, express, lodash, vue)** and **django** (Python) stand out for exceptionally natural, readable naming.
- At the lower end (not shown here), technical/data libraries and enterprise codebases have more abbreviations and less natural phrasing, reflecting domain-specific needs.

Key Takeaways for Your Thesis

- The **average identifier in leading open-source codebases is highly readable as a natural-language phrase**.
- **Most classes and variables** are as easy to read as dictionary English.

- **Functions** are more variable, likely due to more technical verbs or abbreviated action words.
- **JavaScript/TypeScript projects**, especially front-end and utility libraries, lead in natural-language fluency—aligning with their broad audience and culture of code clarity.
- **Python and Java projects** are nearly as good, with only minor penalties for scientific abbreviations or technical word use.

Natural-Language Readability (NL) Analysis

To systematically assess the readability of identifier names as natural-language phrases, we computed a Natural-Language Readability (NL) score for each of the 362,886 identifiers in the dataset. The NL metric quantifies how easily an identifier can be parsed and understood as an English phrase, incorporating linguistic factors such as pronounceability, common word usage, absence of filler or stopwords, and proper grammatical order.

The analysis revealed a **mean NL score of 0.88 and a median of 0.90**, with a standard deviation of 0.10. This tightly clustered distribution at the higher end of the scale indicates that most identifiers in these high-quality open-source projects are highly readable and closely resemble fluent English phrases. The histogram of NL scores shows a pronounced concentration between 0.8 and 1.0, but also a tail of lower-scoring identifiers, reflecting the occasional presence of abbreviations, technical jargon, or nonstandard constructions.

Across programming languages, **Java, C#, and JavaScript projects exhibited virtually identical mean NL scores (0.88 each)**, suggesting strong, shared conventions for naming across ecosystems. **Python projects**, while still highly readable overall (mean NL: 0.87), displayed a slightly lower average, likely due to more frequent use of scientific abbreviations and technical terms. This consistency across languages highlights the convergence of open-source communities on natural-language-inspired naming, regardless of syntax or idiom.

When broken down by identifier type, **class names had the highest average NL score (0.92),** followed by variables (0.90), and functions (0.86). This result aligns with the expectation that class and variable names are typically drawn from everyday nouns and noun phrases, whereas function names sometimes incorporate more technical verbs or unconventional constructs that may reduce their natural-language fluency.

At the project level, libraries such as **d3, express, lodash, vue, and django** stood out with the highest average NL scores, each exceeding 0.91. These projects are widely recognized for their strong commitment to code clarity and accessibility. In contrast, some enterprise and scientific projects exhibited slightly lower NL scores, reflecting domain-specific jargon, longer multi-word names, or abbreviations required by context.

These findings demonstrate that, across the open-source landscape, developers increasingly prioritize identifier names that are not only semantically precise and stylistically correct, but also easy to read and understand as natural language. The prevalence of high NL scores across languages and project types confirms the central role of linguistic readability in modern software engineering, and underscores the value of automated, linguistically informed analysis tools in supporting best practices for code comprehensibility.