PyGress: Tool for Analyzing the Progression of Code Proficiency in Python OSS Projects

Rujiphart Charatvaraphan*, Bunradar Chatchaiyadech*, Thitirat Sukijprasert*, Chaiyong Ragkhitwetsagul*, Morakot Choetkiertikul*, Raula Gaikovina Kula†, Thanwadee Sunetnanta*, Kenichi Matsumoto‡
*Faculty of Information and Communication Technology, Mahidol University, Thailand
†Graduate School of Information Science and Technology, The University of Osaka, Japan
‡Graduate School of Information Science, Nara Institute of Science and Technology, Japan

***Abstract*—Assessing developer proficiency in open-source software (OSS) projects is essential for understanding project dynamics, especially for expertise. This paper presents "PyGress", a web-based tool designed to automatically evaluate and visualize Python code proficiency using pycefr, a Python code proficiency analyzer. By submitting a GitHub repository link, the system extracts commit histories, analyzes source code proficiency across CEFR-aligned levels (A1–C2), and generates visual summaries of individual and project-wide proficiency. The PyGress tool visualizes per-contributor proficiency distribution and tracks project code proficiency progression over time. PyGress offers an interactive way to explore contributor coding levels in Python OSS repositories. The video demonstration of the PyGress tool can be found at https://youtu.be/hxoeK-ggcWk, and the source code of the tool is publicly available at https://github.com/MUICT-SERU/PyGress.**

***Index Terms*—Code proficiency, Python, OSS**

I. Introduction

Python, known for its ease of use and adaptability to various domains, stands as a versatile and most popular language [1] with a plethora of popular open-source software (OSS) projects. Python's simplicity in coding empowers developers to swiftly prototype and construct applications. What sets Python apart is its rich ecosystem, encompassing a multitude of OSS libraries that cater to diverse needs. Libraries such as NumPy, Pandas, and Scikit-learn are pivotal in the fields of data science and machine learning. Matplotlib, Plotly, and Seaborn are used for static plots and data representations.

OSS projects thrive on contributions from developers with diverse backgrounds and varying levels of expertise. Understanding contributors' proficiency can help uncover patterns of project success, contributor retention, and growth. A distinctive characteristic of OSS is its emphasis on transparency. In contrast to commercial projects, open-source initiatives embrace a collaborative development model, distributed control, and a diverse contributor base. The survival of open-source projects hinges on factors like robust community support, dedicated maintainer commitment, adequate funding, and ongoing relevance. The bus factor [2], representing a project's vulnerability to knowledge loss caused by losing key project contributors, is also a concern in OSS, urging practices such as documentation, diverse contributions, and shared responsibilities. Thus, OSS maintainers must focus on fostering a welcoming community for new and existing contributors, and ensuring the long-term sustainability of the project, often within a volunteer-based environment with flexible deadlines [3].

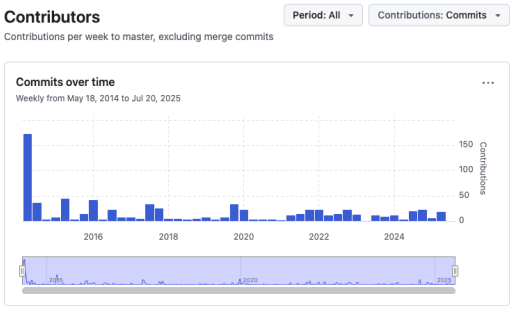

Fig. 1: Current state-of-the-art only depicts contributions over time. This example is from the Django-silk Project

To effectively ensure the expertise required to maintain an OSS project, our key idea lies in assessing developers' proficiency, i.e., an individual's programming skills. This encompasses the capability to not only write code but also understand and effectively debug it. By knowing the contributor's proficiency, maintainers can guide and mentor new contributors, which includes delegating suitable coding tasks for them [3].

Currently, there is a noticeable gap in the Python OSS ecosystem—*there is no automated way available for measuring and visualizing proficiency levels over time, which is crucial for tracking contributions in OSS projects.* OSS project maintainers lack a reliable method to differentiate contributors based on their code proficiencies, hindering the ability to understand the skill distribution within the teams. For example, Figure 1 shows the contributors and their code contributions to the django-silk[1] project, a live profiling and inspection tool for the Django framework, on GitHub. With this information, the maintainers of the project only know the amount of code committed by each contributor over time. However, this contribution summary merely shows the frequency of the

[1]https://github.com/jazzband/django-silk

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00359

commits, and the maintainers do not gain any insights into the proficiency levels of such committed code.

To address this, we introduce ***PyGress**, a web-based prototype tool for automated analysis and visualization of developer proficiency in Python OSS projects.* PyGress assesses Python source code against the Common European Framework of Reference (CEFR) [4] proficiency levels, ranging from A1 (basic) to C2 (proficient). Our automated tool accepts a GitHub repository URL and evaluates Python code from each commit, and generates visualizations that support both project-wide and contributor-specific insights.

II. Background

A. The Common European Framework of Reference for Languages (CEFR)

CEFR [4] is a globally recognized standard for evaluating language proficiency, which categorizes language skills using a six-level scale ranging from A1 (breakthrough or beginner), A2 (waystage or elementary), B1 (threshold or intermediate), B2 (vantage or upper intermediate), C1 (effective operational proficiency or advanced), and C2 (mastery or proficiency). This system simplifies the assessment of language competence for educators, students, and others involved in language education and testing. In addition, the CEFR also facilitates the comparison of qualifications with other national examinations, benefiting employers and academic institutions.

B. pycefr: Python Proficiency Level through Code Analysis

pycefr [5] is an automated tool to assess proficiency in Python projects. It analyzes a given Python project and classifies Python constructs into one of the six levels mirroring the CEFR framework. For example, an `if` and nested list statements are classified as A1 and A2 (basic), respectively. `break` and list comprehension are classified as B1 and B2 (intermediate). Lastly, the generator function and metaclass are classified as C1 and C2 (proficient), respectively. We adopt pycefr as a key component in PyGress to analyze the proficiency of OSS developers and projects over multiple commits.

III. PyGress: Evolution of Code Proficiency

PyGress enables the analysis of Python code proficiency evolution within open-source projects. This is achieved by examining the complete commit history of a given GitHub repository. For each commit, the tool extracts the Python source files and analyzes them using pycefr to determine proficiency levels aligned with the six CEFR levels.

The steps PyGress uses to extract and aggregate the contributors' proficiency scores of any Python projects are described in Algorithm 1. First, PyGress collects all the commits in each project and groups them by the committers (lines 4–8). Then, it iterates through each committer, retrieves the commits, and extracts all the changed Python files in that commit (lines 9–11). Next, PyGress extracts the file *before* and *after* the commit (lines 12–13). The complete Python files before and after the commit are required because pycefr works at the file level. Then, pycefr is executed by giving the before and after versions of the same Python file, and the proficiency of the code introduced in the commit is collected by computing the differences between the proficiency scores of the after and before versions (lines 14–16). Lastly, the derived proficiency scores are aggregated into the set of project proficiencies over all the commits (line 17).

Algorithm 1 Extracting and Aggregating Contributor's Proficiency Scores

```
Input: project
Output: proficiencies
Procedure: ExtractContributorProficiencies
 1: proficiencies ← {}
 2: committers ← {}
 3: commits ← extract_commits(project)
 4: for c_i ∈ commits do
 5:     committer ← get_committer(c_i)
 6:     add_commits(committer, c_i)
 7:     committers ∪ committer
 8: end for
 9: for committer ∈ committers do
10:     for c_i ∈ get_all_commits(committer) do
11:         for f_i ∈ changed_files(c_i) do
12:             before ← get_version(f_i, c_{i-1})
13:             after ← f_i
14:             p_{f_i,before} ← pycefr(before)
15:             p_{f_i,after} ← pycefr(after)
16:             proficiency_{committer,c_i} ← p_{f_i,after} − p_{f_i,before}
17:             proficiencies ∪ proficiency_{committer,c_i}
18:         end for
19:     end for
20: end for
```

In Figure 2, we illustrate further the process of analyzing the proficiency of the code introduced in the commit, i.e., *added proficiency scores*. After getting the Python code before and after the commit, PyGress subtracts the frequency of code proficiency before the commit from the score after the commit, in each level. For example, if the proficiency score ({*A1, A2, B1, B2, C1, C2*}) of a Python file before a commit is {*46, 41, 25, 14, 12, 3*} and after the commit is {*57, 49, 31, 12, 13, 8*}. Then, this commit introduces 38 new code constructs classified into the six-CEFR levels as {*11, 8, 6, 7, 1, 5*} accordingly. We discarded negative scores caused by code deletion, i.e., replaced them with 0, as we focus only on the added code. This score is added to the pool of proficiency scores for that contributor.

The concept of PyGress can be applied to other programming languages by switching the code proficiency level from pycefr to others (e.g., jscefr [6] for JavaScript).

IV. Tool Implementation

A. System Architecture Design

From Figure 3, a user of PyGress gives a GitHub repository URL to the tool. Then, PyGress analysis starts by first cloning the given OSS project and passing it on to PyDriller [7], a Python library designed for exploring Git repositories and examining commit histories, to collect code change data based on each commit. PyDriller is configured to extract code changes in all Python files (`.py` extension) and commits data

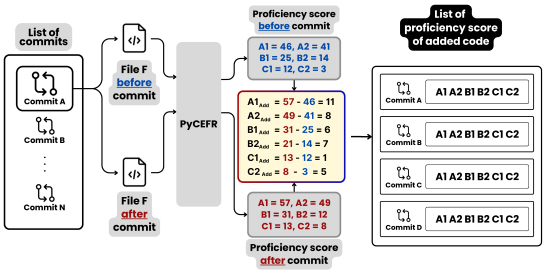

Fig. 2: Approach for analyzing code proficiency changes

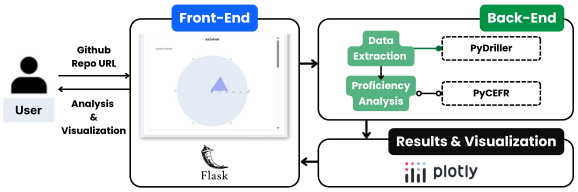

Fig. 3: System architecture of PyGress

Fig. 4: django-silk: Aggregated proficiencies–project level

from those projects, recording them into the code changes database. Next, the pycefr tool is activated to extract the proficiency scores. Lastly, PyGress returns the visualizations of the proficiency data back to the users.

B. Implementation

PyGress follows a modular architecture that separates data processing, visualization, and user interaction. Our design of the PyGress system consists of three main modules:

1) Back-end Module: The backend performs the core processing by employing PyDriller and pycefr to perform commit data extraction and Python proficiency analysis tasks.

2) Front-end Module: A Flask-based web application enables users to submit GitHub repository links and explore analysis results through a web browser. This module dynamically loads the generated visualizations and shows them to the users.

3) Visualization Module: The front-end includes a visualization module, which processes the analysis results to generate interactive charts using Plotly. Two primary visual representations are provided. First, spider charts, which depict the distribution of proficiency levels for all contributors and each individual contributor. Second, slider graphs, which illustrate code proficiency changes throughout the project timeline.

C. Examples of PyGress Visualizations

We chose the django-silk project as an example for the visualization of PyGress. From Figure 4, the first spider chart at the top shows the proficiency level of all the contributors in the project. We can observe that most of the Python code contributed to this project is at A1 and A2 proficiency levels, followed by B1, and some of B2, C1, and C2. Looking at the bottom chart of Figure 4, the duration where most of the development activities occurred was during May–September 2014, when the project started. Most of the code committed during that period was at the A1, A2, and B1 levels.

When investigating specifically at one contributor, `fbead467`[2] The result is as shown in Figure 5. We can see from the spider chart that this contributor mostly contributed code at levels A1 and A2. However, he or she also committed some B1 and, importantly, C2, as well. This contributor committed the code in March 2022. Thus, the part of the C2 code that they committed may need to be carefully maintained since it may not be easy to understand for other contributors.

V. Practical Applications

We applied PyGress to analyze the Python proficiency of three OSS projects, django-silk, pandas-profiling[3], and pytest-ansible[4]. The result is shown in Table I. We can see that the Python code in the three projects is mostly at the level A1 and A2. Looking only at the high proficiency code (i.e., C1 and C2) levels, we can observe that the django-silk contributors mostly committed high proficiency code more at the beginning of the project (2014–2017), potentially building the core logic of the project. In contrast, the contributors of pandas-profiling committed high proficiency code during the later years of the project (2020–2023). For pytest-ansible, there are not many high proficiency code constructs, and they are committed evenly across all the years.

We further analyzed the contributors to find the one that contributed the most proficient code (C1 and C2) to each of the three projects, i.e., the *most proficient contributor*. The result is

[2]An automatically generated anonymized ID for preserving privacy. The tool can be configured to show the actual contributor's name or account name.

[3]https://github.com/pandas-profiling/pandas-profiling

[4]https://github.com/ansible/pytest-ansible

TABLE I: Python proficiency of 3 OSS projects

	django-silk						pandas-profiling						pytest-ansible					
Year	A1	A2	B1	B2	C1	C2	A1	A2	B1	B2	C1	C2	A1	A2	B1	B2	C1	C2
2014	5,495	6,453	640	51	**137**	**131**												
2015	773	962	81	10	**14**	**31**							83	120	5	37	1	1
2016	1,136	1,474	142	10	**16**	**19**	2,484	3,791	253	0	84	3	440	692	89	102	4	4
2017	1,081	1,406	128	14	**40**	**38**	1,719	2,375	183	0	46	0	28	57	8	6	2	3
2018	151	200	11	0	2	1	1,183	1,706	131	0	25	3	201	378	28	22	3	2
2019	495	634	49	6	14	13	1,041	1,459	133	48	23	7	72	141	20	1	1	2
2020	136	213	19	2	7	4	6,111	7,277	800	246	**163**	**66**	28	36	1	0	0	0
2021	1,853	2,899	197	18	73	35	4,898	5,964	721	111	**203**	**37**	243	422	46	13	3	5
2022	154	300	28	2	3	6	1,604	2,040	183	69	**70**	**38**						
2023	81	126	5	0	1	2	1,410	1,957	150	25	**42**	**43**	326	511	64	20	6	2
2024	10	22	1	0	0	0	31	19	2	4	0	0	0	2	0	0	0	0
Total	11,365	14,689	1,301	113	307	280	20,481	26,588	2,556	503	656	197	1,421	2,359	261	201	20	19

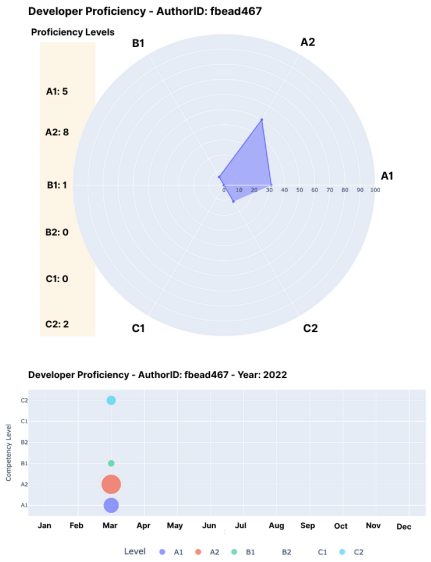

Fig. 5: django-silk: Individual contributor's proficiency

TABLE II: Contributions of the most proficient contributor of each project

Project	Contributor	Year	C1	C2
django-silk	e721d399	2014	133	127
		2015	12	29
pandas-profiling	edf72917	2019	20	7
		2020	92	42
		2021	77	19
pytest-ansible	5ac7cd15	2016	4	4
		2017	2	3
		2018	3	2

shown in Table II. For django-silk, the most proficient contributor (e721d399) committed most proficient code during 2014–2015 (highlighted in bold text). For pandas-profiling, the most proficient contributor (edf72917) contributed the most proficient code during 2019–2021. Lastly, for pytest-ansible, the most proficient contributor (5ac7cd15) contributed the most proficient code during 2016–2018. These contributors can potentially be considered as a bus factor of the project due to their highly proficient code that may be difficult to understand by others.

VI. Conclusion

We present an automated tool, PyGress, for evaluating Python code proficiency and its progression in OSS projects. The tool analyzes project commit history and visualizes the proficiencies for ease of understanding. The tool should be useful for the Python OSS project maintainers for getting insights into their codebase and managing the contributors accordingly for the sustainability of the project. For future work, we plan to improve the processing speed of the PyGress tool by modifying pycefr to analyze the diff data. We also plan to add more visualizations, e.g., heatmaps, to show the code proficiency in each project's modules. Lastly, we plan to validate the tool with OSS project maintainers.

VII. Acknowledgement

This research is partially supported by JSPS Kakenhi (A) JP24H00692 and the Faculty of ICT, Mahidol University.

References

[1] GitHub, "Octoverse," 2024. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-2024
[2] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, and V. Kovalenko, "Bus factor in practice," 2022, volume 32, Number 4, Pages 97–106. [Online]. Available: https://doi.org/10.1145/3510457.3513082
[3] E. Dias, P. Meirelles, F. Castor, I. Steinmacher, I. Wiese, and G. Pinto, "What makes a great maintainer of open source projects?" in *ICSE '21*, 2021, pp. 982–994.
[4] C. of Europe, "Common European Framework of Reference for Languages (CEFR)," 2025. [Online]. Available: https://www.coe.int/en/web/common-european-framework-reference-languages
[5] G. Robles, R. G. Kula, C. Ragkhitwetsagul, T. Sakulniwat, K. Matsumoto, and J. M. Gonzalez-Barahona, "pycefr: Python competency level through code analysis," in *ICPC '22*, 2022, p. 173–177.
[6] C. Ragkhitwetsagul, K. Kongwongsupak, T. Maneesawas, N. Puttiwarodom, R. Rojpaisarnkit, M. Choetkiertikul, R. G. Kula, and T. Sunetnanta, " jscefr: A Framework to Evaluate the Code Proficiency for JavaScript ," in *ICSME'24*, 2024, pp. 863–867.
[7] M. S. C. Shepard, "Pydriller: Python library for mining software repositories," 2018. [Online]. Available: https://github.com/ishepard/pydriller