Reusing My Own Code: Preliminary Results for Competitive Coding in Jupyter Notebooks

Natanon Ritta, Tasha Settewong
Chaiyong Ragkhitwetsagul, Thanwadee Sunetnanta
Faculty of ICT, Mahidol University
Nakhon Pathom, Thailand

Raula Gaikovina Kula, Kenichi Matsumoto
Nara Institute of Science and Technology (NAIST)
Nara, Japan

***Abstract*—The reuse of already existing code is widely considered a popular software development practice, that provides both benefits and drawbacks for all stakeholders involved. Prior work reports on how code reuse is a common practice in software development projects and data science projects such as machine learning pipelines. Recently, there has been much code reuse work in the context of competitive programming. Although there is work such as detecting plagiarism, there is no work that studies how a competitor will reuse their own code. In this paper, we present a preliminary study on the code reuse behavior of three grandmasters' Jupyter notebooks in the Kaggle Competitions, an online competition platform for data scientists, and report the types of code they often reuse. Grandmasters are the highest level reached in competitions (novice, expert, master, and grandmaster). We find that Grandmasters are less likely to reuse specialized code, but instead, tend to reuse common functions like importing packages (importing the pandas library). They are most likely to reuse common abstractions like importing packages, configurations, file IO operations, show data, plotting graphs, defining functions, and exploring files. The work opens up new research potential into recommending how developers can reuse their own code.**

***Index Terms*—Jupyter notebook, code reuse, code similarity**

I. Introduction

Nowadays software development does not only limit to writing programs by developers but also programs (i.e., models) that are created by learning from data using machine learning (ML) techniques. This type of ML-based software can have both similarities and differences from traditional software. In terms of similarities, the models are still trained by having the developers write code. The code is used to specify what data to learn and what machine learning algorithm, along with its parameter configurations, to be used. Nonetheless, the software's behavior is no longer specified by the developers, but based on the data that it learns.

In software development, code reuse is a common practice. Developers reuse code from their existing projects, other people's projects, or online sources [1], [2]. The significant advantage is that code reusing might help to reduce time and effort in development. Code is written once and reused many times. Additionally, code reusing is able to improve reliability if the original code is well-written and tested and the developers are familiar with it. Nevertheless, code reuse provides some drawbacks too. Code reusing makes redundancy happen in software development, which affects the maintainability of the software. Moreover, it can propagate bugs.

Kaggle[1] is an online community for data scientists and people interested in machine learning. Kaggle allows users to collaborate, find and publish their code and datasets. On the Kaggle platform, the users can use the provided Jupyter notebooks[2] integrated with GPU to join a competition or solve challenges about data science and machine learning. The Kaggle Competitions is a section that made Kaggle popular. The competitions listed in Kaggle can be both from a company, like Google or American Express, aiming to solve its business problems or from the community that shares challenging problems to practice. Some competitions, mostly by companies, have a time limit and come with prizes for the highest scorers as an incentive for joining and producing results. Kaggle users can compete in a competition by forming a team (with at least one person). In this paper, we call a Kaggle user who joins a competition as a "competitor".

Nonetheless, most of the studies in code reuse have been performed in software projects or their related artifacts [3]–[10], but only a few on machine learning or data science that use Jupyter notebooks [11]–[13]. No study of code reuse has been done in machine learning competitions like Kaggle before. Thus, we fill the gap by performing an empirical study to understand how Kaggle competitors reuse code in their competitions. The result from this preliminary study can shed light on understanding code reuse in the context of machine learning coding competitions and also suggest a way to aid the developers to manage the reused code or recommend related code for reuse. In this study, we aim to answer the following research questions:

- *RQ1: What are the most re-used code and their types for Kaggle competitors?* First, we want to explore to what extent the Kaggle competitors reuse code in their submissions to competitions, and also investigate the types of such reused code.
- *RQ2: What are the common re-used or similar code and their types across Kaggle competitors?* Second, we want to study whether the Kaggle competitors also reuse code from (or share similar code with) other competitors.

[1]https://www.kaggle.com/
[2]https://jupyter.org

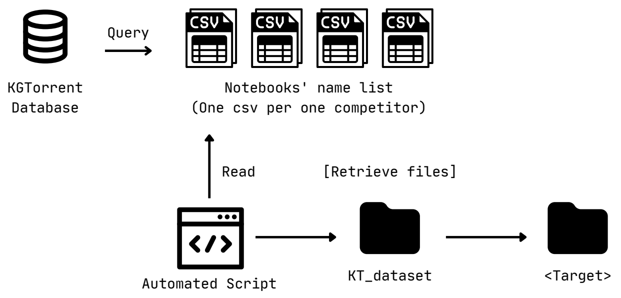


Fig. 1: Overview of Data Collection

The main contributions of the paper are as follows.

1) We are the first to study code reuse in machine learning competitions using the Kaggle platform.
2) We present the different types of reused code cells by Kaggle competitors.

All code and scripts are available at https://github.com/NAIST-SE/ReuseJupyterNotebook.

II. Study Design

In this section, we explain our study design, which includes the data preparation and tools employed to answer our research questions.

A. Data Collection

a) Target Dataset: We choose the dataset called KGTorrent [14] for this study. KGTorrent is a dataset of Python Jupyter notebooks derived from Kaggle. Currently, there are more than 2,910 competitions [14] and 167,397 datasets available on Kaggle[3]. The dataset comprises of a companion database for keeping metadata such as users, competitions, teams, and a collection of Jupyter notebooks (called kernels). The database is derived from the Meta Kaggle[4], which is available freely on the Kaggle website. The dataset contains 248,761 Jupyter notebooks of 175 GB in size.

b) Data Collection: Figure 1 shows an overview of how we extract Python code snippets from the KGTorrent dataset. The KGTorrent dataset is in the form of a single directory that contains all Jupyter notebooks.

In order to retrieve the Jupyter notebook files from the dataset, we had to set up the provided corresponding database and query the list of Jupyter notebooks that belong to each competitor in Kaggle. We wrote an automated shell script to perform the task of querying the Jupyter notebook filenames and copying the notebooks from the dataset to disk. Next, we selected all competitors whose notebook tier is Grandmaster to be analyzed in this study. KGTorrent has 5,598,921 users in total, but only 16 competitors were Grandmaster tier of notebooks and participated in at least 20 competitions at the time of analysis. After that, we selected 3 of 16 competitions to be a sample for the experiment.

[3]Data as of 3 September 2022.

[4]https://www.kaggle.com/datasets/kaggle/meta-kaggle

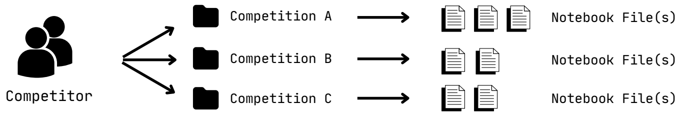


Fig. 2: Notebook Files Extraction

TABLE I: Summary Statistics of Competitors

Competitor	#Competitions	#Notebook Files
Competitor 1	15	156
Competitor 2	25	815
Competitor 3	12	406

c) Dataset Preparation: We then designed and created the directory structure for the collected data, as shown in Figure 2. For each competitor, we created directories containing the notebook files that they created for competitions that they joined, one directory per competition. A competitor may have joined multiple competitions, and one competition can have more than one Jupyter notebook.

Due to the inconsistency between the metadata of the companion database and the notebooks dataset, all queried notebook names might not exist on the notebooks dataset. Consequently, we run an automated script to check how many competitions remain before processing the data. For the data to be processed, we selected three competitions because they have at least 10 competition files after verification by an automated script, all of which are notebook grandmasters. Table I is the summary statistics of the prepared data for the three competitors.

B. Code Reuse Detection

There are myriad code similarity tools available both for commercial and research. Each tool has a different technique to measure the similarity of code and support different programming languages [15].

a) Similarity Detection of Code Cells: In this work, we have chosen NCDSearch [16] as the code similarity tool. The tool performs the search by receiving a code snippet (a file) as a query and locating similar code snippets within another file or a folder of files. The code similarity is calculated using Lempel-Ziv Jaccard Distance, which is an approximation of Normalized Compression Distance. Output from NCDSearch is in the form of the query, the similar code fragment found from the search, and their distance, which is in the range of 0 to 1. The distance value of zero means that the similar code fragment is exactly the same as the query. On the contrary, the

TABLE II: Number of Code Reuse Per Competitor

Competitor	#Competitions	#Reused Code Snippets
Competitor 1	15	48
Competitor 2	25	52
Competitor 3	12	28
Total	-	128

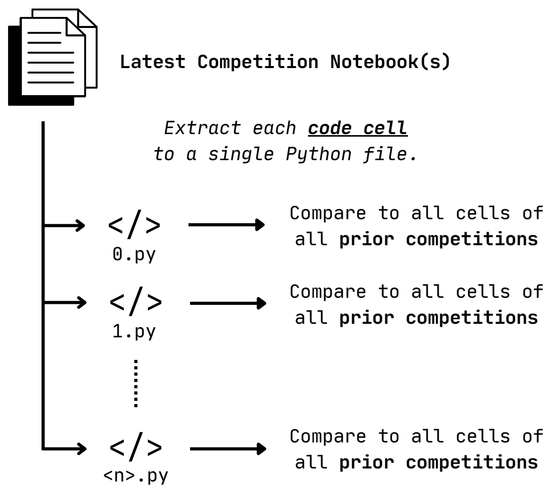

Latest Competition Notebook(s)
Extract each code cell to a single Python file.
0.py
Compare to all cells of all prior competitions
1.py
Compare to all cells of all prior competitions
<n>.py
Compare to all cells of all prior competitions

Fig. 3: Overview of the Code Similarity Analysis

distance value of one means they are entirely different. The tool supports several languages including Python.

The code cells that are considered reused in our study have a distance in the range of 0 to 0.5. A distance of similarity that is more than 0.5 is not considered as matched based on the default configuration of NCDSearch. According to this approach, if a competitor reuses the code from a prior competition by splitting the original code of one cell into multiple smaller code cells, the split code cells will be detected as reused. This is because the smaller code cells will be used as a query and would match with a very close or zero distance to the bigger original code that contains them.

As shown in Table II, 128 code cells are reported by NCDSearch as reused from the three selected competitors. For competitor 1, we analyzed 15 competitions and found 48 reused code snippets. For competitor 2, we analyzed 15 competitions with 52 reused code snippets found. Lastly, for competitor 3, we analyzed 12 competitions and found 28 reused code snippets.

b) Locating Similar Code Cells: To locate reuse code snippets among the competitions of each competitor, we ran NCDSearch with the default configuration between the notebook files in each competition. Figure 3 shows an overview of locating similar codes among competitions. First, we extracted each code cell within a Jupyter notebook to a Python file (with an extension `.py`). Next, we compared all Python files that are extracted from Jupyter notebooks of the latest competition, in which a competitor participated, to other extracted Python files from all prior competitions. We consider the latest competition of each competition from the date of Jupyter Notebook creation, and the created date comes from the metadata from the database. Only cells of the notebook(s) from the latest competition will be decided as reused code. This approach allows us to see which code cells are being reused among the competitions. The result of NCDSearch was saved for later analysis.

c) Classification of Similar Code Cells: To characterize the similar code cells, we then had to manually classify them into common groupings. Hence, to allow for a systematic process, the first author investigates similar code snippets and the related code comments and code surrounding the cells. Then with the two additional authors, we then abstracted the coding of the function of the code cell. If the code cell could be classified into more than one type, the first author chose a type of code cell that was mostly related. Table III shows our seven types of code cells in Kaggle notebooks including

1. Import packages: Code in the cell is mostly about importing packages to the kernel. This cell shows the code that imports external packages to be used in the notebook. One example is `import pandas as pd`. This import statement adds the Pandas library to the notebook, which allows data analysis and manipulation.

2. Configure: The code for configuration is used to adjust some configurations of the library and usually appears in the code cell that imports the packages. When writers have preferred configs, they instantly change the library configuration after importing packages.

3. File I/O: We consider both Python internal library command (i.e., with open) and the external library's command (i.e., `pd.read_csv`) as File I/O.

4. Show data: Code cell that contains built-in or external function(s) used for printing data (e.g., `print()` and `pd.head()`) is considered as a Show Data type. Writers mostly use it to preview both a specific part and general of data.

5. Plot graph: Code cell is about performing graph plotting with the imported library. Calling a graph plotting function can be the form of multiple lines of code or only a single line of code. The keyword that made us consider which code cell is Plot Graph is a graph displaying commands like `.hist()` or `.plot()`.

6. Define function: This code type means code cells that are used for defining function(s). We classify all the code that has `def` to be this type because it is a sign to define the function in Python.

7. Explore files: This code type is a code cell with OS command(s) to list files directory and show them as an output of a cell. Calling OS command with `!command` syntax (i.e., `!ls`) and commands from OS package (i.e., `os.listdir(.)`) for file exploration both are considered as this type.

III. Findings

We now return to answering our two research questions.

A. *RQ1: What are the most re-used code and its type for each competitor?*

To answer this research question, we need to know the type of reused code, so performed the code type classification manually using the predefined code types (see Table III). We performed this manual analysis on the results from 3 randomly selected competitors out of the total 16 Grandmaster competitors.

TABLE III: Code Cell classification

Type of Code	Description
Import packages	Code in the cell Is mostly about importing packages to the kernel.
Configure	Adjusting a configuration for a package or anything.
File I/O	File I/O function(s) or command(s) are called whether `read` or `write`.
Show data	The function that is used to print out the data is called e.g., `print()`, `pd.head()`.
Plot graph	Performing a graph plotting with any library
Define function	The code cell is used for defining the function.
Explore files	Calling the OS command to list the file and show it as an output of a cell.

As shown in Figure 4, the x-axis represents all reused code cells of each competitor grouped by code type. The y-axis shows a distance of similarity between reused code and the original one. Each of the 3 analyzed competitors has different reused codes and types. The first competitor (Figure 4a) reused the code cells for importing the package the most, followed by the code cells for plotting graphs. The second competitor (Figure 4b) reused the code cells for showing data the most, followed by the code cells that call the OS command for file exploration. The last competitor (Figure 4c) reused Import Package code cells the most, followed by File I/O & Show Data.

Additionally, we added the suffix number to instances of reused code that share the same type so that we could distinguish them. Moreover, we can see that the results in Figure 4 can have more than one instance of one code type (e.g., Import package_1 and Import package_2).

B. RQ2: What are the common re-used code and its type across competitors?

Although the most reused code cells and their type might be different for all three competitors, according to the finding shown in Figure 4, they have the same common reused code type, which is the code for importing the package(s). Therefore, we can answer RQ2 that *Import Package is the most common type that is reused among the three competitors.*

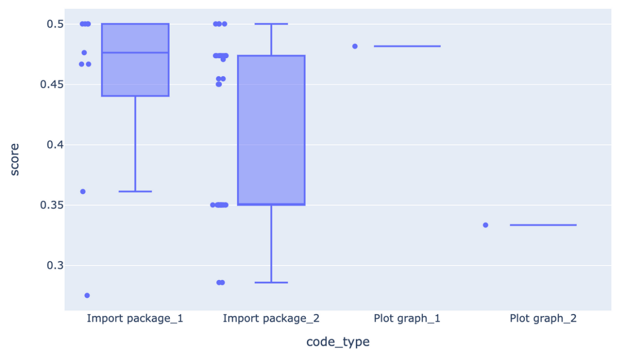
The example of the detected reuse code cells for the Import Package type is `import pandas as pd`. For plotting graph, an example is `targets.sum(axis=1).hist()`. For showing the data, an example is `ss.head()`. Lastly, an example of the reused code of File I/O type is shown below.

```
File I/O and Show data:
# LOAD TEST META DATA
test = pd.read_csv
('../input/siim-isic-melanoma-
classification/test.csv')
test.head()}
```

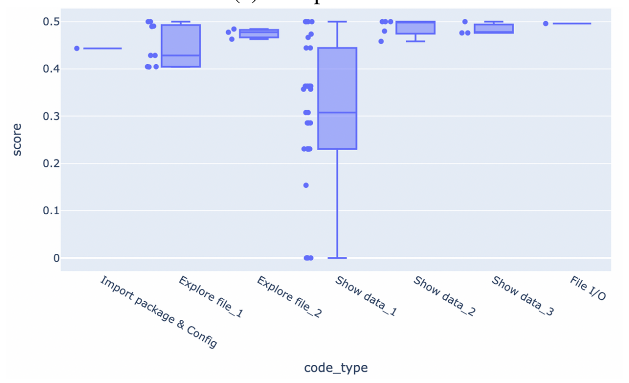
IV. Threats to Validity

Internal Validity: First, we encountered a problem while analyzing the KGTorrent dataset. There was data inconsistency between metadata in the database and the raw files in the dataset. Occasionally, we could not find a Jupyter notebook in the dataset even though its name was available in the database. Thus, this may affect the number of notebook files we retrieved. Second, we defined the code cell types by ourselves by having the first author perform a random check of the existing Kaggle notebooks. This may lead to incomplete code types. We mitigated this threat by having the second author to validated the code types. Lastly, the manual code type classification may be subject to human errors.

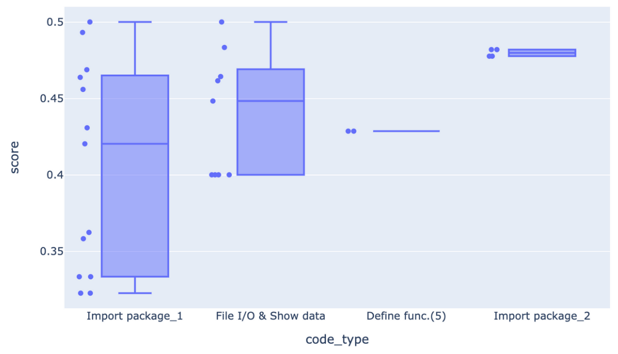
External Validity: We analyzed only the Jupyter notebooks of the Grandmaster tier. The finding may not be generalized to other notebook tiers.



(a) Competitor 1



(b) Competitor 2



(c) Competitor 3

Fig. 4: Types of Reused Code by Each Competitor

V. Related Work

Källén et al. [11] performed a large-scale study of code cloning in Jupyter notebooks with an analysis of 2.7 million Jupyter notebooks hosted on GitHub. They found that 50% of all the analyzed notebooks do not have unique code snippets at all. Moreover, for notebooks written in Python, 80% of all the code snippets are clones with some modifications to other locations. We similarly found that the Kaggle competitors created clones of their existing competitions. According to the study of Koenzen et al. [12], snippets of code for visualization are among the ones that are duplicated the most. Some developers or data scientists spend time browsing for code examples online, which usually are API examples. Moreover, they found that many codes are cloned from the code examples more than open-source code in the version control system like GitHub. We similarly observed that code cells for plotting graphs, i.e., visualizations, are among the mostly reused ones. Sigvardsson [13] reports that code reuse in Jupyter notebooks mostly has a strong correlation with `numpy`, `matplotlib`, and `pandas` libraries. The work also states that code reused has the benefit of saving time and resources. We also report in our work that code cells for importing packages are the one that is reused the most.

Code reuse (i.e., code duplication or code clones) is a normal practice adopted in modern software development. Developers reuse the code that has been written by themselves in existing projects and also from other sources such as Stack Overflow [1] or GitHub [2]. Reusing code has both benefits and drawbacks. It can save the developer's time by not writing the same code again and also by making good use of well-implemented code. It also may result in the propagation of defects or security vulnerabilities and increase maintenance efforts [17]. Currently, the focus is to inform the developers about the amount of reused code in their software so that they can make an informed decision on how to manage them [18]–[20].

VI. Challenges and Future Work

In this paper, we investigated the code reuse in Kaggle competitions by the competitors. We selected 16 competitors who are Grandmaster tier of notebooks in Kaggle. All notebooks of all competitors have been compared by using NCDSearch to find similar code, which implies code reuse. We randomly selected 3 of the 16 competitors to do a manual code type classification and report the result.

Interestingly, we find that grandmasters are less likely to reuse specialized code from their prior experiments. Instead, they are most likely to reuse common abstractions like importing packages, configurations, file IO operations, show data, plotting graphs, defining functions, and exploring files.

There are many potential future works. For example, we would like to explore what combinations of these abstract reused codes can be useful. Furthermore, different from grandmasters, we would like to also explore how newcomers or intermediate developers reuse their code.

Acknowledgment

This work has been supported by JSPS KAKENHI Grant Numbers JP8H04094, JP20K19774, and JP20H05706.

References

[1] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic Code Snippets on Stack Overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 560–581, mar 2021.

[2] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack Overflow in Github: Any Snippets There?" in *Proceedings of the International Conference on Mining Software Repositories (MSR '17)*, 2017.

[3] W. Amme, T. S. Heinze, and A. Schafer, "You Look so Different: Finding Structural Clones and Subclones in Java Source Code," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME '21)*. IEEE, 2021, pp. 70–80.

[4] "An Empirical Study on Accidental Cross-Project Code Clones," in *Proceedings of the IEEE 14th International Workshop on Software Clones (IWSC'20)*, 2020, pp. 33–37.

[5] W. Rahman, Y. Xu, F. Pu, J. Xuan, and X. Jia, "Clone Detection on Large Scala Codebases," in *14th International Workshop on Software Clones (IWSC'20), London, Canada, February 18, 2020*, 2020.

[6] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, "Method and implementation for investigating code clones in a software system," *Information and Software Technology*, vol. 49, no. 9-10, pp. 985–998, sep 2007.

[7] R. Tajima, M. Nagura, and S. Takada, "Detecting functionally similar code within the same project," in *Proceedings of the IEEE 12th International Workshop on Software Clones (IWSC '18)*, 2018, pp. 51–57.

[8] A. Capiluppi, D. Di Ruscio, J. Di Rocco, P. T. Nguyen, and N. Ajienka, "Detecting Java Software Similarities by using Different Clustering Techniques," *Information and Software Technology*, feb 2020.

[9] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," in *Proceedings of the International Conference on Software Engineering (ICSE '12)*, 2012, pp. 1289–1292.

[10] C. J. Kapser and M. W. Godfrey, "Supporting the analysis of clones in software systems," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 2, pp. 61–82, mar 2006.

[11] M. Källén and T. Wrigstad, "Jupyter Notebooks on GitHub: Characteristics and Code Clones," *The Art, Science, and Engineering of Programming*, vol. 5, no. 3, feb 2021.

[12] A. P. Koenzen, N. A. Ernst, and M.-A. D. Storey, "Code Duplication and Reuse in Jupyter Notebooks," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2020, pp. 1–9.

[13] U. Sigvardsson, "Code Cloning Habits Of The Jupyter Notebook Community," 2019.

[14] L. Quaranta, F. Calefato, and F. Lanubile, "KGTorrent: A Dataset of Python Jupyter Notebooks from Kaggle," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 550–554.

[15] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," *Empirical Software Engineering*, vol. 23, no. 4, pp. 2464–2519, aug 2018.

[16] T. ISHIO, N. MAEDA, K. SHIBUYA, K. IWAMOTO, and K. INOUE, "NCDSearch: Sliding Window-Based Code Clone Search Using Lempel-Ziv Jaccard Distance," *IEICE Transactions on Information and Systems*, vol. E105.D, no. 5, may 2022.

[17] C. J. Kapser and M. W. Godfrey, "Cloning considered harmful considered harmful: patterns of cloning in software," *Empirical Software Engineering*, vol. 13, no. 6, pp. 645–692, dec 2008.

[18] S. Tokui, N. Yoshida, E. Choi, and K. Inoue, "Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 642–646.

[19] E. Duala-Ekoko and M. P. Robillard, "CloneTracker: Tool Support for Code Clone Management," in *Proceedings of the 13th international conference on Software engineering - ICSE '08*, 2008, p. 843.

[20] H. Honda, S. Tokui, K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, "CCEvovis: A Clone Evolution Visualization System for Software Maintenance," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, vol. 2019-May, 2019, pp. 122–125.