Opening the Valve on Pure-Data: Usage Patterns and Programming Practices of a Data-Flow Based Visual Programming Language

Anisha Islam
University of Alberta
Edmonton, Canada
aislam4@ualberta.ca

Kalvin Eng
University of Alberta
Edmonton, Canada
kalvin.eng@ualberta.ca

Abram Hindle
University of Alberta
Edmonton, Canada
hindle1@ualberta.ca

ABSTRACT

Pure Data (PD), a data-flow based visual programming language utilized for music and sound synthesis, remains underexplored in software engineering research. Existing literature fails to address the nuanced programming practices within PD, prompting the need to investigate how end-users manipulate nodes and edges in this visual language. This paper systematically extracts and analyzes 6,534 publicly available PD projects from GitHub. Employing source code parsing, pattern matching, and statistical analysis, we unveil usage patterns of PD by the end-user programmers. We found that most revisions of the PD files are small and simple, with fewer than 64 nodes, 51 connections, and 3 revisions. Most PD projects have less than 17 PD files, 31 commits, and only 1 author working on the PD files. The median differences in the number of nodes and edges between each commit and its parents, modifying the same file, are 3 and 0, respectively, implying small changes across various revisions of a PD file. Our findings contribute a valuable dataset for future studies, addressing the dearth of research in PD. By unraveling usage patterns, we provide insights that empower scholars and practitioners to optimize the programming experience for end-users in the realm of visual programming languages.

CCS CONCEPTS

• **Software and its engineering → Visual languages**.

KEYWORDS

Visual Programming Language, Pure Data, End-User Programmers

ACM Reference Format:
Anisha Islam, Kalvin Eng, and Abram Hindle. 2024. Opening the Valve on Pure-Data: Usage Patterns and Programming Practices of a Data-Flow Based Visual Programming Language. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643991.3644865

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04
https://doi.org/10.1145/3643991.3644865

1 INTRODUCTION

Pure Data (PD) [1, 2], one of the most popular visual programming languages for computer musicians, can be used for various musical applications, such as creating smart tuners [3], musical multi-agent systems [4], and real-time applications for speech processing, sound synthesis, and music transcription [5–8]. In PD, these diverse functionalities are achieved by placing multiple *objects* or functions on a canvas linked by *connections*, allowing the transfer of outputs from one object to the inputs of others. The arrangement and nature of these connections dictate the flow of data, resembling the structure of *nodes* and *edges* in graph theory. Figure 1 shows an example of a PD program with multiple nodes and edges that generates a triangle wave.

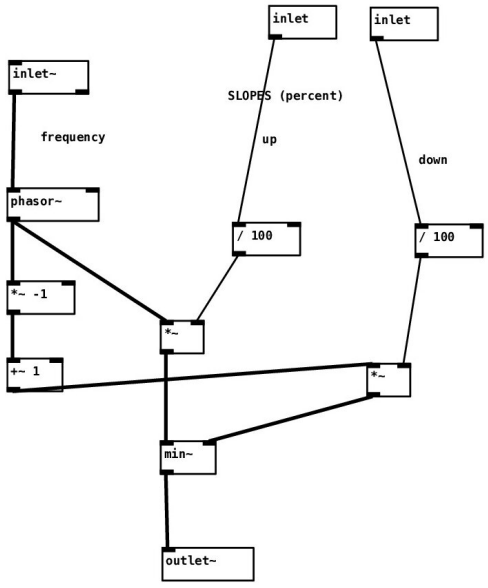


Figure 1: PD program for generating a triangle wave

A survey of 175 computer musicians, usually categorized as end-user programmers, by Burlet *et al.* showed that PD was the preferred language for 47.4% of them [9]. End-user programmers surpass the number of professional programmers [10, 11]. However, there is a lack of research on the specific needs of these end-user programmers and the challenges they face in their disciplines [12].

Despite the popularity of PD among computer musicians, it lacks publicly available datasets that can reveal usage patterns and help

address the domain-specific needs of end-user programmers. In contrast, other visual programming languages like Scratch have public datasets that can be explored and analyzed by researchers [13]. Therefore, creating a similar dataset for PD would enhance the programming experience of end-users in visual programming languages. It would also be valuable for researchers and practitioners who want to study how computer musicians use this visual programming language and what challenges they face.

In this paper, we provide a dataset [14, 15] for PD comprising of mirrors of the original git repositories, and an SQLite [16] database of extracted metadata containing: revisions of the PD files; parsed source code of PD files; and the commit information of the project repositories, including author and committer information, commit messages, and commit parents. We obtained PD project names from the World of Code (WoC) [17] using maps, pattern matching, and filtering. Subsequently, we recorded and parsed the revision history of all the revisions of each PD file across the collected projects. In addition, we retrieved details on authors, committers, commit messages, and commit parents for all commits in our project repositories. We recursively traversed the parent history of all commits that altered a PD file in order to identify the commits that directly influenced the file's current state in the current commit. Using the extracted data, we constructed a database, analyzed our data, and addressed the following research questions.

RQ1: Are node changes in the revisions of a PD file typically limited to a small scale, involving only a single node?

RQ2: Are edge changes in the revisions of a PD file typically limited to a small scale, involving only a single edge?

We observed that most revisions of the PD files contain few nodes and connections, with an average of approximately 3 revisions per PD file. Most PD projects typically contain fewer than 17 PD files, less than 31 commits, and involve only 1 author for the PD files. The median differences in nodes and edges between each commit and its parents, which modified the same file, are 3 and 0, respectively.

Our extracted data can help analyze the evolution of PD files in a repository and give insights into the typical usage patterns of the end-user programmers. Our data can also facilitate future research that aims to improve the end-user programmer experiences using visual programming languages like PD. The code and data used to create the dataset, the mirrored PD repositories, and the dataset itself are publicly accessible for future research [14, 15].

2 METHODOLOGY

We can describe the methodology of our paper in five stages: project name collection, revision history extraction, parsing the contents of the revisions of the PD files, commit information retrieval, and dataset construction.

2.1 Project Name Collection

We used the WoC (version U) maps and datasets to get the names of the PD projects. We found 309,964 blob IDs for files with `.pd` extension using the `b2f` map. We filtered out the false positive blob IDs by searching for the `#N` syntax of PD in the base64 decoded contents of the files. After the first level of filtering, we got 273,166 blob IDs for the PD files and used the `b2P` map to get the fork normalized project names from these blob IDs. We retrieved 8,454 project names and cloned the GitHub repositories linked to them. Finally, we checked for PD files in the currently checked-out branch in the repositories and got 6,534 PD projects for our dataset.

We selected the publicly available PD projects on GitHub because of the widespread use of Git [18] as a version control system among computer musicians. A survey by Burlet *et al.* showed that 54% of 175 computer musicians use version control systems, and Git is the most favored one among them [9]. Our collection of projects reflects the PD projects on GitHub within a specific time range, including academic, personal, professional, and experimental projects.

2.2 Revision History Extraction

In the second stage, we extracted the revision history of the PD files on the *default branch*, where the mirror version of the GitHub repository is currently checked out, for all the cloned projects. Initially, we mirrored the git repositories in May 2023. In a September-October 2023 update of the mirrors, 28 deletions were identified since the last mirroring. For the deletions, we processed and analyzed the May 2023 mirrors.

We obtained the commit SHAs of the default branch of the PD projects and used them to identify the PD file names with the `.pd` extension. Then, we retrieved a detailed log of changes for each PD file and the commit SHAs that changed the file. We linked the commits with the revisions that modified the file, which gave us the revision history of the PD files.

2.3 Parsing the Contents

Next, we accessed the contents of the revisions of a PD file linked to a specific commit and preprocessed the generated contents of the PD files, such as inserting a new line, when needed, before `#N` or `#X` to make them compatible with our parser. Then, we parsed the contents of the PD files into an Abstract Syntax Tree (AST) that captures the essential information from the PD source code. The AST contains information about the nodes or objects, such as their ID, attributes, counts, and types. It also has information about the edges or connections, such as their count, source, and destination nodes, and, when serialized as JSON, has a content-based SHA-256 [19] identifier.

2.4 Commit Information Retrieval

We extracted the authors' and committers' names and emails linked to the commit SHAs using git commands. For privacy reasons, we stored the author information as SHA-256 values of their identifiers. We also extracted the commit messages from our PD projects, which can explain the reasons behind specific changes and identify the defect-fixing changes in visual code, as explained by Eng *et al.* [20, 21]. Furthermore, we extracted the parents of all commits of our projects so that we can generate the *content parents* of the commits, which is a term we use to describe a commit that changes a PD file in our repository and is part of the direct ancestry of another commit that also changes the same PD file. For instance, commit `c0` has two parent commits, `c1` and `c2`; `c1` has a parent commit: `c3`, and `c2` has a parent commit: `c4`. If `c0`, `c3`, and `c4` are the only commits affecting a PD file, then `c3` and `c4` are the content parents of `c0` for that file. We obtained the content parents of all

commits that modified the PD files in our projects by recursively traversing the commit parents that we stored in our database.

2.5 Dataset Construction

In the final stage of our methodology, we used the revision metadata, the parsed contents of all the revisions of the PD files, the author information, and the commit history obtained from the previous stages to construct our dataset [14, 15]. We created a relational database using SQLite, a portable and self-contained database system that does not require a separate server [16]. We opted for SQLite as our database because it can execute complex queries involving multiple tables, utilizing operations like filtering and joining, and expedite data retrieval using indexes.

Our dataset can reveal the usage pattern of PD by end-user programmers by utilizing the revision metadata to track the development of PD files over time, the parsed source code to compare the differences, and the information about the nodes and edges to investigate the distribution and variety of objects. The commit messages and author information provide valuable insights for the researchers since they can use them to understand the rationale behind each commit and the degree of participation in PD projects.

We can also use SQL queries to access the content parent information and compare the changes in the same file across different commits, which can help us understand how much PD files vary from one commit to another. Moreover, we provide mirrored repositories of the PD projects, which can serve as a reference and assist researchers in examining the project in its original context.

Figure 2 illustrates the schema of our database. Our dataset consists of seven tables, which are explained below.

Projects: This table houses 6,534 PD project names, the default branch names, and the number of total commits in these projects.

Revisions: The Revisions table contains the metadata for each revision of the PD files. It stores the project names, PD file names, revision names, commit SHAs that modified the PD files, commit dates, SHA-256 hash values of the parsed contents of the revisions, and the number of nodes and edges in the revisions of the PD files. The total number of rows in this table is 1,113,345.

Commit_Messages: This table stores the commit SHA and commit messages of unique commit and commit message pairs (505,871).

Authors: The Authors table contains the commit SHAs and the hashed author and committer names and emails for the 505,748 unique commit and author committer pairs.

Commit_Parents: The Commit_Parents table preserves information about all commits and their parent commits in our projects. There are 532,006 rows in this table.

Content_Parents: The Content_Parents table allows us to trace the content parents of the commits that modified a PD file. This table consists of the names of the PD projects and files, the commit SHAs that modified the PD files, and the content parents of the commits. This table has 1,154,438 rows, each representing one of the commit and content parent pairs.

Contents: This table stores the actual parsed contents of the revisions of the PD files and their hashed values. There are 203,324 unique hash values and parsed PD contents present in this table.

3 DATA ANALYSIS

In this section, we demonstrate the versatility of our constructed dataset [14, 15] by illustrating how we can use our dataset to gain deep insights into various aspects of PD projects. We queried our database using SQL and analyzed our stored data, such as revisions, nodes, edges, authors, commits, and files, to get a comprehensive view of the development dynamics of PD projects. The summary statistics of our extracted data are presented in Table 1.

Distribution of Nodes and Edges: We used the revision metadata stored in our dataset to analyze the distribution of nodes and edges in the revisions of the PD files. Figures 3a and 3b depict the node and edge counts in the revisions of the PD files in our dataset. We observe that most revisions of a PD file have few nodes and connections, with 647,157 files having a node count between 1 and 64 and 579,482 files having an edge count between 1 and 51.

In contrast, only six files have a high node count ($\geq$ 30,000), and twelve files have a high edge count ($\geq$ 30,000). Interestingly, the files with a high node count are the revisions of the same PD file in one project. Additionally, 256,703 files lack edges, exceeding those with zero nodes (189,480), suggesting that some PD files may contain nodes but lack connections.

Distribution of Authors: The author distribution in Figure 3c reveals that 80.39% of projects (5,253) involve a single author working on the PD files. In contrast, the project *jptrkz_pd-macambira* stands out, with 58 authors and the highest PD file count (15,198) in our dataset, with 10,131 commits. Based on this observation, we can infer that most PD projects are likely not collaborative in nature, with only a limited number of authors contributing to the PD files in each project.

Distribution of PD Files: Figure 3d demonstrates the distribution of PD files (not including revisions) in our dataset. The majority (75.43%) of projects (4,929) have fewer than 17 PD files. In contrast, three projects titled *jptrkz_pd-macambira*, *aidanreilly_pd_patches*, and *collaborative-music-lab_NIME* have a large number of PD files: 15,198; 12,531; and 11,875, respectively. These projects also have high numbers of commits: 10,131; 265; and 340, indicating active, long-lasting, and ongoing development in these repositories.

Distribution of Commits. The number of commits per project can be extracted from our database to reveal the average activity level of PD projects. Our analysis shows that 75.23% PD projects (4,916) have at most 31 commits, indicating that the PD projects are small and do not go through active development.

Distribution of Revisions. We extracted the number of revisions per PD file from our dataset. We noticed that, on average, a PD file has approximately 3 revisions, indicating that most PD files are stable and do not frequently go through significant changes.

4 RESULTS

To answer our research questions, we used our dataset [14, 15] to measure how the nodes and edges in each revision of a PD file

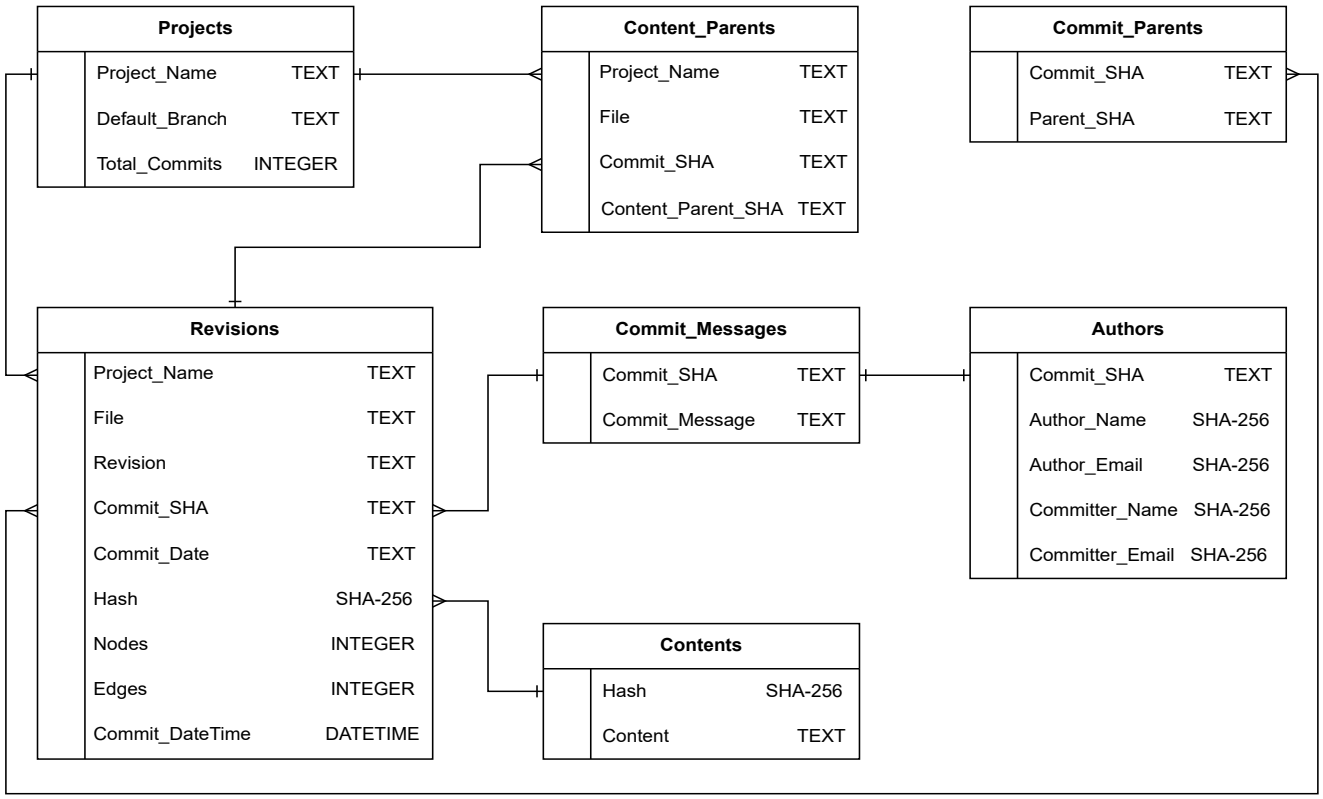


Figure 2: Schema of the SQLite dataset

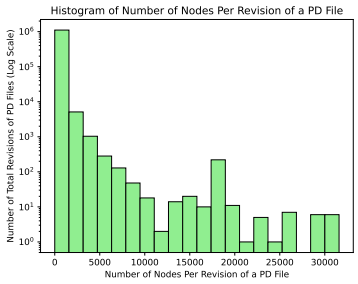
Table 1: Summary statistics of our extracted data

	Total	Mean	Min	Q1	Q2	Q3	Max
Nodes Per File	1,113,345	93.78	0	9	25	64	31,586
Edges Per File	1,113,345	84.78	0	3	16	51	35,932
Revisions Per PD File	484,555	2.29	1	1	2	3	246
Diff Nodes Per Revision of a PD File	1,113,345	23.95	-35,264	0	3	27	29,592
Diff Edges Per Revision of a PD File	1,113,345	21.17	-37,180	0	0	17	35,932
Authors Per Project	6,533	1.32	1	1	1	1	58
PD File Per Project	6,534	74.15	1	1	4	17	15,198
Commits Per Project	6,534	78.18	1	3	9	31	24,256

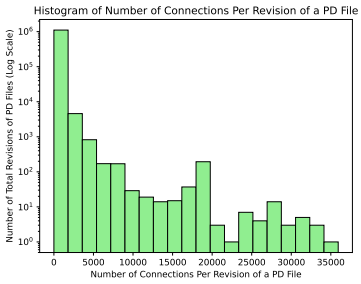
changed compared to that of the content parents. We can get the content parents, revision metadata, and parsed content of each commit that changed the PD file from our database, and the extracted data can then be used to find the difference in nodes and edges between commits that changed the PD file.

RQ1: The Change of Nodes in the Revisions of a PD File: To quantify the structural evolution of a PD file in terms of nodes, we calculated the difference between the node count of each revision of a PD file and the node count of the content parent commits that modified the file using our dataset. Figure 3e shows that this difference is less than 27 for the majority (75.19%) of the revisions of the PD files (837,149), with a median node count change of 3. The positive median indicates that there are more nodes in a new revision of a PD file than its content parents, and most revisions involve more than 1 node count change.

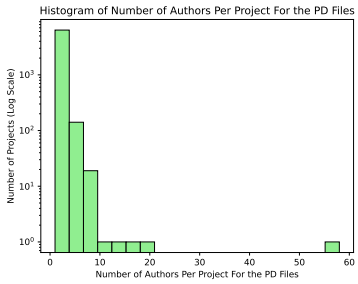
RQ2: The Change of Edges in the Revisions of a PD File: A similar trend is observed while comparing the differences in edges or connections in different versions of a PD file. Most revisions (75.06%) have less than 17 connection count changes from their content parent versions. Additionally, the median value of edge count difference is 0, indicating infrequent and generally minor changes in the number of connections within the revisions of the



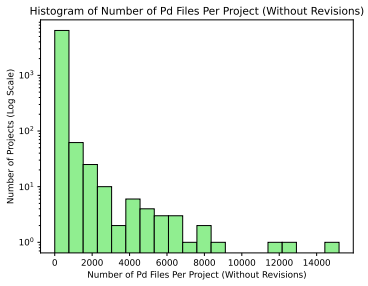
(a) Distribution of number of nodes per revision of a PD file



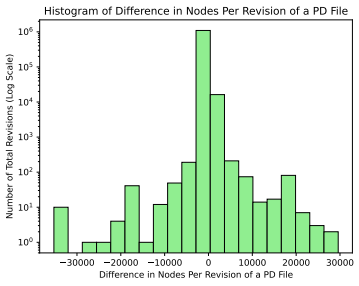
(b) Distribution of number of connections per revision of a PD file



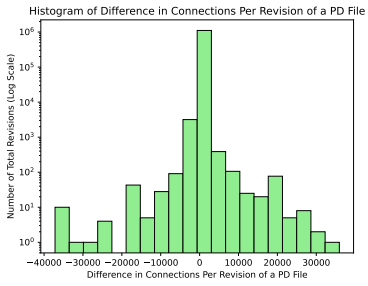
(c) Distribution of number of authors per project



(d) Distribution of PD files per project



(e) Difference in node count of a commit and its content parents



(f) Difference in edge count of a commit and its content parents

Figure 3: Distribution of nodes, edges, difference in nodes and edges, authors, and PD files (The Y-axis is on a base 10 log scale)

PD files. Figure 3f demonstrates the connection count differences between a commit and its content parents.

5 CONCLUSION

In this paper, we examined the usage patterns of Pure Data (PD) by collecting and analyzing 6,534 public PD projects from GitHub. We provided a comprehensive open dataset with mirrored git repositories, including metadata for PD file revisions, parsed source code emphasizing nodes and edges information, and detailed commit history, including author and committer details, commit messages, commit parents, and content parents. This dataset serves as a valuable resource for analyzing various aspects of PD projects. Analyzing the data stored in our database, we discovered that most PD projects are small in size, typically containing fewer than 17 PD files with a limited number of nodes, connections, and revisions. We also noticed that projects generally have fewer than 31 commits and are associated with only 1 author who works on the PD files, implying an individual rather than collaborative development process for Pure Data. Moreover, we observed gradual and small changes in node and edge counts between each commit and its content parents. Researchers can use our dataset to query information about authors to identify collaboration within a project, extract commit messages for purposes such as identifying defect-fixing commits, and trace the change history of a file using the revision metadata, content parents, and their parsed contents.

In conclusion, our dataset helps researchers and practitioners to understand visual code development processes and facilitates future research in visual programming languages.

ACKNOWLEDGMENTS

We acknowledge the support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through their Discovery Grant, which facilitated this research.

REFERENCES

[1] Miller Puckette et al. Pure Data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.
[2] IEM. Pure Data. https://puredata.info/. Accessed: 2024-01-20.
[3] Hsin-Ming Lin and Chin-Ming Lin. Harmonic Intonation Trainer: An Open Implementation in Pure Data. In *NIME*, pages 38–39, 2015.
[4] Pedro Bruel and Marcelo Queiroz. A Protocol for creating Multiagent Systems in Ensemble with Pure Data. In *ICMC*, 2014.
[5] Roger K Moore. On the Use of the 'Pure Data' Programming Language for Teaching and Public Outreach in Speech Processing. In *INTERSPEECH*, pages 1498–1499, 2014.
[6] Gilberto Bernardes, Carlos Guedes, and Bruce Pennycook. EarGram: An Application for Interactive Exploration of Concatenative Sound Synthesis in Pure Data. In *From Sounds to Music and Emotions: 9th International Symposium, CMMR 2012, London, UK, June 19-22, 2012, Revised Selected Papers 9*, pages 110–129. Springer, 2013.
[7] Marcos Alonso, Günter Geiger, and Sergi Jorda. An Internet Browser Plug-in for Real-time Sound Synthesis using Pure Data. In *ICMC*, 2004.
[8] Marius Miron, Matthew EP Davies, and Fabien Gouyon. AN OPEN-SOURCE DRUM TRANSCRIPTION SYSTEM FOR PURE DATA AND MAX MSP. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 221–225. IEEE, 2013.
[9] Gregory Burlet and Abram Hindle. An Empirical Study of End-user Programmers in the Computer Music Community. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 292–302. IEEE, 2015.

[10] Kathryn T Stolee, Sebastian Elbaum, and Anita Sarma. End-User Programmers and their Communities: An Artifact-based Analysis. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 147–156. IEEE, 2011.

[11] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the Numbers of End Users and End User Programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 207–214. IEEE, 2005.

[12] Margaret Burnett. Software Engineering For Visual Programming Languages. In *Handbook of Software Engineering and Knowledge Engineering*, volume 2. World Scientific Publishing Company, 2001.

[13] Efthimia Aivaloglou, Felienne Hermans, Jesús Moreno-León, and Gregorio Robles. A Dataset of Scratch Programs: Scraped, Shaped and Scored. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 511–514. IEEE, 2017.

[14] Anisha Islam. Opening the Valve on Pure-Data Dataset. https://archive.org/details/Opening_the_Valve_on_Pure_Data, 2023. Accessed: 2023-12-07.

[15] Anisha Islam, Kalvin Eng, and Abram Hindle. Opening the Valve on Pure Data Dataset. https://doi.org/10.5281/zenodo.10576757, 2024.

[16] D. Richard Hipp. SQLite. https://www.sqlite.org/index.html, 2013. Accessed: 2023-11-12.

[17] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretzki, and Audris Mockus. World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data. *Empirical Software Engineering*, 26:1–42, 2021.

[18] Scott Chacon. Git SCM. https://git-scm.com, 2005. Accessed: 2024-01-26.

[19] FIPS Pub. Secure Hash Standard (SHS). *Fips pub*, 180(4), 2012.

[20] Kalvin Eng, Abram Hindle, and Alexander Senchenko. Identifying Defect-Inducing Changes in Visual Code. In *ICSME*. IEEE, 2023.

[21] Kalvin Eng, Abram Hindle, and Alexander Senchenko. Predicting Defective Visual Code Changes in a Multi-Language AAA Video Game Project. In *ICSME*. IEEE, 2023.