

The Ecosystem of Open-Source Music Production Software – A Mining Study on the Development Practices of VST Plugins on GitHub

Bogdan Andrei
Informatics Institute
University Of Amsterdam
Amsterdam, the Netherlands
b.m.andrei@uva.nl

Mauricio Verano Merino
Computer Science Department
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
m.verano.merino@vu.nl

Ivano Malavolta
Computer Science Department
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
i.malavolta@vu.nl

Abstract—In this study we shed light on a unique and interdisciplinary domain, where music, technology, and human creativity intersect: music production software. Today *software* technologies are the predominant means of music production, with a vibrant ecosystem for commercial and open-source products.

In this work we target VST plugins, the de-facto standard for developing and prototyping music production software. We analyze 15,847 data points over 299 GitHub repositories containing VST plugins. Our results include a systematic quantification of the (i) characteristics of open-source VST projects in terms of, *e.g.*, duration, size, contributors, stars/watchers, licensing, (ii) most used technologies for developing VST plugins, and (iii) code quality and testing practices in VST projects.

Our findings provide a comprehensive understanding of the current state of the practice in VST plugins development, highlighting successful projects, opportunities for improvement, and future research directions for software engineering researchers.

I. INTRODUCTION

Software technologies are the prevalent tool in music production over the past decades [1]. Artists with millions of online streams and multiple Grammy nominations such as Depeche Mode, Daft Punk, Ed Sheeran, and many more are known to be using *Digital Audio Workstations (DAWs)* such as Ableton Live, Apple Logic Pro, and Image Line FL Studio [2], [3] in their music production process. The main reasons for the success of DAWs include, among many: (i) their integration with sophisticated tools for editing, mixing, and mastering, allowing artists to go deeper into sound design and music composition, (ii) the possibility of connecting real-time instruments to DAWs via standard interfaces, and (iii) the possibility for artists to have the freedom to explore ideas without the constraints of a traditional studio [2].

In 1996, the **Virtual Studio Technology (VST)** [4] has been introduced as an open software interface which allows third-party developers to create software instruments and effects in the form of plugins that can be integrated into existing DAWs [5]. The VST technology opened up new possibilities for music producers and creatives, enabling the usage of a wide range of audio processing tools within the DAW without the need of physical musical instruments [6]. Currently, many

of the major DAWs are compatible with VST and can act as VST host applications [5].

The technical success of VST is also confirmed by its florid ecosystem of developers, distributors, and publishers. The global audio plugins market, which includes VST plugins, is projected to reach approximately 3.25 billion dollars by 2030 and is growing at a compound annual growth rate (CAGR) of 15.2% from 2023 to 2030 [7]. Prestigious manufacturing companies – *e.g.*, Yamaha, Moog, Korg – which historically were specialized on hardware musical instruments only, are diversifying their offer now by developing their own pure software-based instruments via VST plugins, some of them even running in the Cloud (*e.g.*, Roland Cloud [8]). The open-source ecosystem of VST plugins is equally vibrant; for example, as of October 2024, the GitHub repository of JUCE, a C++ framework for developing VST plugins, has 6.6k stars, 1.7k forks, 14k commits, and more than 1.4k issues and pull requests [9]. The JUCE framework is actively used by organizations like Fender, Yamaha, Bose, Arturia and it is used in educational contexts as well, *e.g.*, in the Center for Computer Research in Music and Acoustics at Stanford [10].

Despite their heavy use in today’s music industry, an aspect that still remains unexplored is how practitioners *develop* VST plugins. Studying such practices is relevant since *the development of a VST plugin is a complex and highly interdisciplinary activity*, requiring knowledge in analog/audio and digital signal processing, music theory, sound synthesis, high-performance software development, and human-computer interaction [6]. Moreover, *developing VST plugins is still a craftsmanship activity with limited software engineering body of knowledge* supporting practitioners towards developing high-quality software in terms of, *e.g.*, performance, reliability, and maintainability.

The **goal** of this study is to characterize the ecosystem of open-source VST plugins and how practitioners develop VST plugins *in real contexts*. In this study we target open-source VST projects on GitHub since (i) the VST open-source community is primarily active on the GitHub platform.

In this work, we apply software repository mining tech-

niques [11] targeting the source code, documentation, commits, and pull requests in 299 GitHub repositories containing real open-source VST plugins. We extract a total of 15,847 individual data points, which we then analyze both quantitatively and qualitatively. Our analysis allows us to synthesize an objective characterization of the state of the practice in VST plugins development in terms of: (i) characteristics of the projects (e.g., duration, size, contributors, licensing, etc.), (ii) used technologies, and (iii) applied code quality practices.

The **main contributions** of this study are: (i) a reusable dataset with 15,847 data points about 299 real VST projects on GitHub, (ii) a systematic characterization of the state of the practice with respect to music production software, (iii) a discussion of the obtained results, including their main implications, and (iv) the full replication package of the study. This study is the first empirical investigation on (open-source) VST plugins for music production.

The **target audience** of this study includes: (i) *VST practitioners* for getting an objective characterization of the state of the practice of VST plugins development and position themselves in such ecosystem; (ii) *musicians and creatives* in general who can use our results as a map for better understanding the VST plugins ecosystem and possibly select candidate open-source VST plugins they can use in their productions; (iii) *software engineering researchers* who can use the results of this study and the research lines we suggest as inspiration for future research contributions.

II. BACKGROUND AND RELATED WORK

A. Virtual Studio Technology (VST)

Audio processing plugins provide specialized audio processing capabilities to be used within a DAW [12] or other compatible software for music production and audio editing. As of today, several technologies exist for developing audio processing plugins, such as CLAP [13], AAX [14], and LV2 [15]. One of the most commonly-used technologies is VST [4], introduced by Steinberg in 1996. Most of VST plugins are either instruments (i.e., VSTi) or sound effects (i.e., VSTfx). VSTis generate audio by imitating real-life instruments (e.g., synthesizers, guitars, drums), while VSTfxs process audio by performing the same functions as hardware audio processors (e.g., delay, reverb, distortion, phaser) [6].

Figure 1 illustrates the anatomy of a typical VST plugin. The two main subsystems of a VST plugin are (i) the Graphical User Interface (GUI) and (ii) the Digital Signal Processing (DSP) [6]. The *GUI* provides a custom user interface allowing users to control various *VST Parameters*, such as volume or pitch. These parameters are then sent to the *DSP subsystem*, where the *Digital Processor* processes the data values set by the user (e.g., set the volume to 5dB). The DSP subsystem manipulates the digital signals coming from the VST host (e.g., the DAW) according to user-defined VST parameters. VST plugins often also include support for the Musical Instrument Digital Interface (MIDI [17]), allowing the plugin to send, manipulate, and receive *MIDI events* representing musical instructions, such as pitch, timing, and loudness of

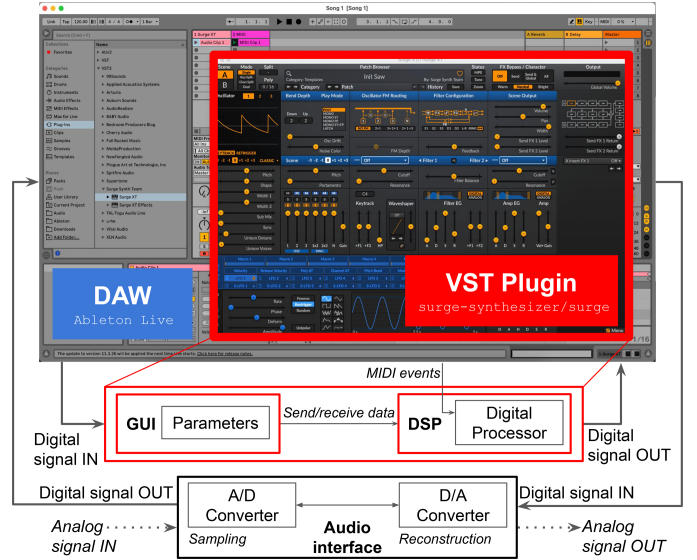


Fig. 1: Anatomy of a VST plugin based – example based on the `surge-synthesizer/surge` VST plugin deployed in the Ableton Live [16] DAW.

notes. The role of the audio interface is to (i) convert the analog signal, collected from sound sources (e.g., instruments and microphones, or other audio lines), into a digital signal via the *Analog-to-Digital (A/D) Converter* and (ii) send the digital signal to the DAW for manipulation according to the loaded VST plugins. Once the digital signal is processed, it is sent to the audio interface again and converted back into an analog signal by the *Digital-to-Analog (D/A) Converter*. The reconstructed analog signal is then sent back to monitors (e.g., headphones or speakers).

B. Related work

Music Software Development: The study of software development for music is a multidisciplinary research field that brings together individuals from diverse backgrounds, including music, signal processing, computer science, and electronics [18]. For example, in 2010, Damnjanovic *et al.* [19] conducted a survey to explore the demographics of the UK audio and music research community. This study highlights the diversity within this community, examining the tools employed in music software development and the challenges of ensuring software reproducibility. Similarly, Cannam *et al.* [18] introduced the *Sound Software*, aimed at supporting audio and music researchers in the UK. This work identified critical software engineering challenges faced by researchers, such as lack of confidence in programming, insufficient knowledge of code management tools and practices, and difficulties with code reusability due to platform incompatibilities.

In many cases, music software development is carried out by end-users – individuals without formal technical backgrounds. Researchers have investigated the processes and tools used by these end-user developers. Burlet *et al.* [20] analyzed the software development practices of *computer musicians*, a

community of end-user programmers who often use visual music-oriented languages like Max/MSP and Pure Data. They compared software development practices in Max/MSP or Pure Data repositories versus general software repositories. Similarly, Islam *et al.* [21] conducted a systematic analysis of over 6,000 repositories containing Pure Data programs, offering insights into the programming habits and trends within this community.

VST Plugin Development: Despite the vibrant ecosystem of the music industry, there is relatively little research on VST plugin development and music production software in general. Reuter [2] studied the music production practices of hip-hop and EDM artists and their potential impact on pop music production. Among many, one of the most important results of this study is that hip-hop and EDM artists tend to use DAWs that are less and less based on the metaphor of the recording studio (*e.g.*, think about the concept of track and instrument) and more on a game-like trial-and-error compositional combination of third-party loops, presets, and other music production services that are typically provided via the Cloud. Stickland *et al.* proposed an online DAW collaboration framework for improving the studio mixing experience of artists working remotely [22]. The framework allows up to 30 artists to synchronously communicate with each other, while performing remote audio mixing in real-time on a shared DAW project and receiving updates of remote collaborators' actions. A recent study proposed the integration of generative musical systems directly in the DAW [23]; the proposed plugin allows creatives to execute Python expressions at runtime directly in the timeline of the DAW, promoting Python code to a fully-fledged (virtual) instrument playing together with all the other audio/MIDI tracks. As of today, those studies acknowledge to some extent the presence of VST plugins, but none of them targeted *how* VST plugins are developed in practice.

Observational studies: Several observational studies applied mining software repositories techniques similar to ours to study the development practices in open-source communities. For example, Choetkiertikul *et al.* [24] focuses on understanding the characteristics of Jupyter Notebooks hosted on Kaggle and GitHub that are used in data science projects. Malavolta *et al.* [25] conducted a mixed-method observational study about the state-of-the-practice of robotics software development in terms of targeted quality attributes and architecture documentation in the context of 399 real open-source projects.

III. STUDY DESIGN

We designed and conducted this study according to well-established guidelines for mining software repositories on GitHub [11] and empirical software engineering [26], [27].

To ensure independent verification of this study, we provide a complete replication package [28] containing both raw and aggregated data and all mining, analysis, and plotting scripts.

A. Goal and Research Questions

Following the template by Basili *et al.* [29], the **goal** of this study is to analyse *the VST plugins ecosystem* for the

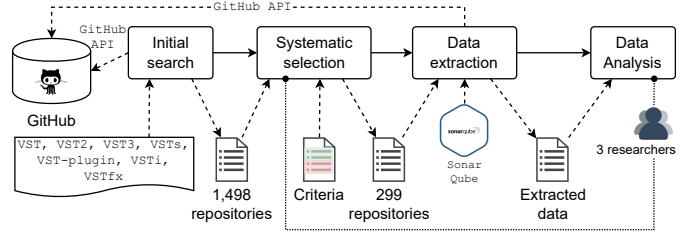


Fig. 2: Overview of the Study.

purpose of *characterizing their main development practices* with respect to *project characteristics, technologies, and code quality* from the point of view of *VST plugin developers, musicians and creatives, and researchers* in the context of *open-source VST projects on GitHub*. This goal is achieved by answering the following research questions (RQs).

RQ1 – What are the main project characteristics of VST projects on GitHub? This RQ quantitatively assesses (i) the spread of VST projects on GitHub, (ii) their main characteristics in terms of repository metadata (*e.g.*, repository age, number of commits, open/closed pull requests, stars, watchers, and licenses), and (iii) the categories of the projects (*i.e.*, instrument or audio effect). *Audience:* VST developers, musicians and creatives, and researchers.

RQ2 – What are the most used technologies in VST projects on GitHub? This RQ aims at building a map of the most used programming languages, development frameworks, and libraries in VST plugin development. *Audience:* VST developers, researchers.

RQ3 – What are the main code quality practices applied in VST projects on GitHub? This RQ aims at building a map of the level of quality of the source code of VST projects in terms of presence of bugs, code smells, vulnerabilities, cyclomatic and cognitive complexity, presence of comments, and adoption of testing practices. *Audience:* VST developers, researchers.

B. Initial Search

As shown in Figure 2, we first query via the GitHub REST API for all repositories whose `topic` is strictly related to VST development; Figure 2 reports the specific keywords of our search; all keywords represent the abbreviation of Virtual Studio Technology Plugin [4], alongside its category (*i.e.*, VSTi, VSTfx), versions (*i.e.*, VST2, VST3), and their syntactical variations. We decided to scope our search on GitHub topics since (i) we are interested in a very specific technology (*i.e.*, VST) with no strong syntactical variations in its name, and (ii) a preliminary search of the most popular open-source VST plugins [30] revealed that they tend to be consistently tagged as VST projects on GitHub. After duplicates removal, this step leads to 1,498 potentially-relevant repositories.

C. Systematic Selection

Inspired by the systematic literature review method [26], we *manually analyze* each of the 1,498 repositories and select those fulfilling the inclusion and exclusion criteria (see

Table I). A repository is included into the dataset if it satisfies *all* the inclusion criteria and *none* of the exclusion criteria.

TABLE I: Selection criteria for VST projects on GitHub.




ID	Description
I1	Repositories containing VST plugins, either standalone (<i>i.e.</i> , providing standalone VST hosts) or DAW-compatible ones.
I2	Repositories containing the source code of a VST plugin that can be (compiled and) executed on a physical machine.
E1	Repositories containing DAWs/hosts, instead of VST plugins.
E2	Repositories whose README file is not written in English.
E3	Repositories containing only a README file, without source code.
E4	Repositories containing standalone applications that use VST's hosts.
E5	Repositories containing frameworks/libraries for developing VST plugins.
E6	Repositories without README file.
E7	Repositories containing tutorials, demos, or template projects.
E8	Repositories containing audio samples for already existing VST plugins.
E9	Repositories consisting of a replication package for a scientific study (<i>e.g.</i> , a study on distortion effects, resonance, etc.).

Three researchers are involved in this phase, and emerging conflicts are resolved collaboratively. The selection is done in four rounds: in the first three rounds, we use 200 repositories that are classified independently by the three researchers. Then, their agreement level is statistically assessed via the Cohen Kappa statistics, and a discussion on the differences between researchers takes place. In the first round, we obtained an average value of 0.58 (moderate agreement), 0.67 (substantial agreement) in round 2, and 0.83 (almost perfect agreement) in round 3. Given the high level of agreement reached in the third round, the remaining 898 repositories are classified by one researcher. This phase led to the final set of 299 GitHub repositories containing open-source VST plugins usable in real-world contexts.

In order to provide context to our quantitative results in RQ1, we also build a random sample of 299 GitHub projects that are not containing VST plugins. This sample will act as a baseline for the main findings in RQ1¹. The sample is constructed in such a way that (i) selected GitHub repositories have the same distribution of the 299 mined VST projects in terms of used programming languages, age, and freshness and (ii) all selected GitHub repositories satisfy non-VST-related selection criteria (*i.e.*, E2, E3, E6, E7, E9 in **Table I**). Due to the peculiarity of some programming languages (*e.g.*, CSound Document for `maxkling/patch-discord`), we obtained a final set of 286 baseline GitHub projects.





























D. Data extraction


In this phase we locally collect all data required to properly answer the RQs of the study. **Table II** presents the metrics that we collect for each repository on GitHub, grouped by RQ. We used the following complementary sources and tools:

-  The full text of the README file in the main branch of the repository (299 data points).
-  GitHub metadata about the analysed repository, mined via the official GitHub REST API (6,877 data points).
-  The full text of all source code files contained in the main branch of the repository (416,642 files in total).

¹Due to available resources, we collect only automatically-computable characteristics from non-VST projects (*e.g.*, number of commits, size, etc.).

TABLE II: Extracted metrics for each GitHub repository.

Metric	Source/tool	Analysis method
RQ1 – Main project characteristics		
VST category (<i>i.e.</i> , instrument, effect)		Card sorting
GitHub Topics		Summary statistics
Repository creation date		Summary statistics
Repository age		Summary statistics
Repository freshness		Summary statistics
Repository size		Summary statistics
Number of Commits		Summary statistics
Number of Forks		Summary statistics
Number of Stars		Summary statistics
Number of Watchers		Summary statistics
Number of Open PRs		Summary statistics
Number of Closed PRs		Summary statistics
Number of Contributors		Summary statistics
Contributors' experience in years		Summary statistics
Repository owner type (<i>i.e.</i> , organiz. or user)		Summary statistics
License type (<i>e.g.</i> , GPL, MIT)		Summary statistics
RQ2 – Most used technologies		
Programming languages		Card sorting
Frameworks	 	Card sorting
Libraries	 	Card sorting
RQ3 – Code quality practices		
Number of code smells		Summary statistics
Number of bugs		Summary statistics
Number of vulnerabilities		Summary statistics
Cyclomatic complexity		Summary statistics
Cognitive complexity		Summary statistics
Comment lines density		Summary statistics
Mentions of testing		Card sorting
Automated testing		Card sorting

-  The report produced by SonarQube v10.3, with quality gate set to `Sonar way` and the default quality profile for each used programming language (1,794 data points).

SonarQube [31] is a commonly-used static code analyzer, both in academia [32], [33] and in practice, with more than 400k organizations using it [31]. SonarQube is based on sets of language-specific rules, continuously kept up-to-date by their development team; for example, the standard C++ quality profile contains 672 rules, organized into four groups: bugs, vulnerabilities, code smells, and security hotspots. All together, we extract a total of 15,847 individual data points about the 299 analysed GitHub repositories containing VST plugins, to be further analyzed in the next phase.

E. Data analysis

We use both qualitative and quantitative techniques to analyze the extracted data and to minimize bias, all the data analysis activities involve three researchers.

Being this study purely observational and based primarily on metrics with a ratio scale [27, Ch. 6], the majority of the metrics are analyzed and interpreted by computing their **summary statistics** (see **Table II**), followed by a combination of data summarization and visualization techniques based on tables, histograms, boxplots, and violin plots.

Other metrics have a categorical scale (*e.g.*, used frameworks). In those cases, we apply a process inspired by the (open) **card sorting** technique [34]. For each metric, we

perform the card sorting in two phases: (i) we code each repository with its representative information about the data point (e.g., the specific audio effect supported by the VST plugin) and (ii) we group the extracted codes into meaningful groups with a descriptive title (e.g., whether the VST plugin is a virtual instrument or an audio effect). Subsequently, we use summary statistics to quantitatively summarize the categorized data about each metric. This mixed-method procedure enables us to understand the data and draw meaningful conclusions.

IV. MAIN VST PROJECT CHARACTERISTICS (RQ1)

A. VST category and used GitHub topics

VST plugins come in various forms. For this study, we classify them into three **categories**: (i) virtual instruments (VSTi) such as synthesizers and samplers, (ii) audio effects (VSTfx) such as reverb, distortion, and delay, and (iii) repositories containing both VSTi and VSTfx. Our analysis revealed that the majority of repositories, 219 (73.2%), focus on VSTfx, while 76 repositories (25.4%) contain VSTi. A small subset, 4 repositories (1.4%), consists of plugin packs containing both types of plugins (e.g., [zamaudio/zam-plugins](#)).

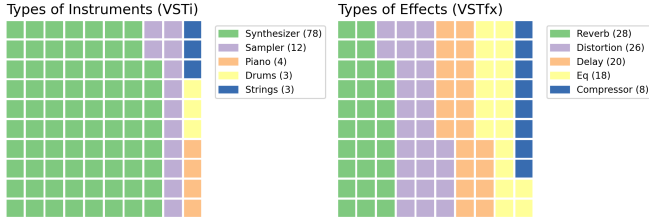


Fig. 3: Overview of the most common VST categories.

Figure 3 presents the most common VST instruments (VSTi) and VST effects (VSTfx) in our dataset. For readability purposes, we only include instruments that appear at least 3 times in different repositories and effects that occur in at least 4 repositories. We can observe that the set of VST instruments is less diverse than the set of VST effects; 9 repositories (11.8% of the total VSTi) contain unique VSTis, while 169 repositories (77.2% of the total VSTfx) contain unique VSTfxs. The reverb effect is the most common (e.g., [reillypascal/RAlgorithmicVerb](#)), followed by distortion (e.g., [MeijisIrlnd/Transfer](#)), delay (e.g., [Mg32/SymmetricPingPongDelay](#)), and equalizers (e.g., [Djaugo/EQ_Lite-Vst-Plugin-Win](#)). In contrast, virtual instruments (VSTi) show a more homogeneous distribution; synthesizers (e.g., [AnClark/Minaton-XT](#)) are the most popular ones, followed by samplers (e.g., [silver-var/ap_sampler](#)).

GitHub topics represent the tags associated with a repository on GitHub, and each repository can have multiple tags. The upper part of **Figure 4** displays a co-occurrence matrix of the top-20 topics in the dataset and its lower part shows a subset of the most common tags since 2011, and compares with the last five years. We can observe that `vst` is the most used topic (186 repositories – 62.2%), closely followed by `vst3` (152 repositories – 50.8%), and `juce` (95 repositories – 31.8%). However, if we look at the last 5 years data, it shows

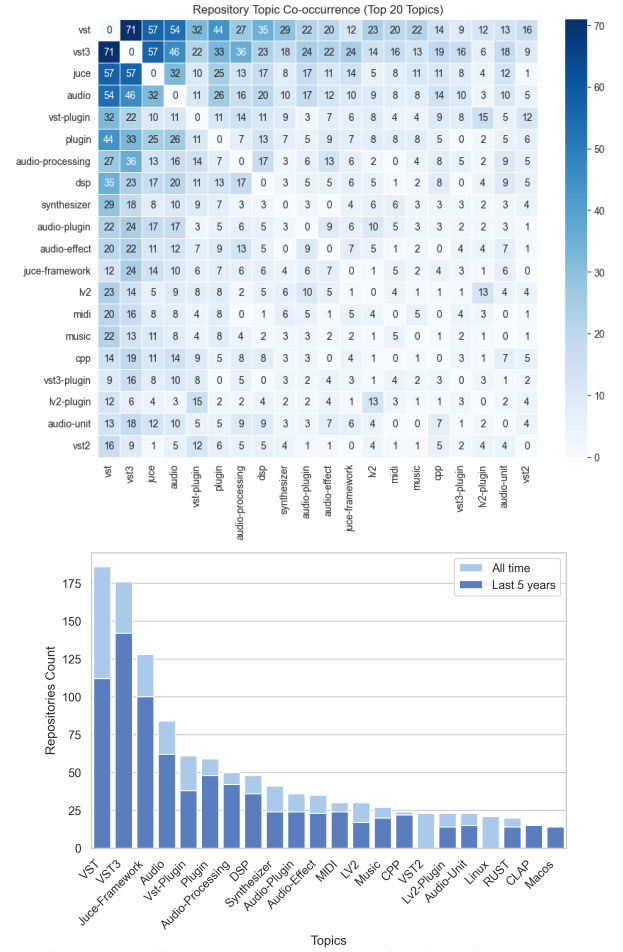


Fig. 4: Topics used across repositories since 2011.

a shift, particularly with the growth of `vst3` as a dominant topic across 136 repositories (45.5%). The third most popular topic is the JUCE [\[35\]](#) framework. This is not surprising since JUCE is one of the most actively used specialized frameworks for developing VST plugins.

B. Repository Characteristics

Figure 5 shows the **creation date** of the repositories in our dataset. [mzuther/K-Meter](#), the first open-source VST plugin in our dataset, was created in 2011.

We can also observe that the year with the most created repositories for VST plugins on GitHub is 2021, and after that, the creation of VST repositories seems to be slightly slowing down. The median **age** of selected repositories is 818 days (2.2 years), suggesting that VST plugin development is still relatively young. The median **freshness**, measured by the number of days since the last commit, is 431 days, indicating that most repositories are actively maintained with relatively recent updates. The median **size** of selected repositories is 2,095 KB, with sizes ranging from 1 KB (e.g., [alexeyr/digidist](#)) to 647,857 KB (e.g., [publicsamples/Peach](#)). Notably, the Peach plugin is the largest, as it includes images and preset files specific to the plugin.

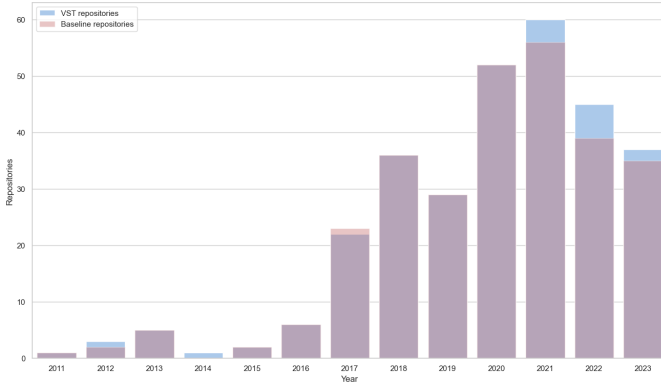


Fig. 5: Year of creation of the repositories.

C. Repository Activity

Table III and Figure 6 show the repositories activity descriptive statistics. For the sake of space, we describe only the most salient points we observe when looking at this data points.

TABLE III: Repositories and contributors descriptive statistics.

Repositories	Mean	STD	Min.	Q1 (25%)	Q2 (50%)	Q3(75%)	Max
Age (days)	996.04	882.55	0	265.5	818	1456	4356
Freshness (days)	644.87	666.01	13	119.50	431	950.50	3168
Size (Kb)	21559.41	66325.79	1	253.5	2081.00	10194.00	647857.00
Commits	58.79	355.51	1	1	1	1	4740
Forks	6.20	27.43	0	0	1	3	341
Stars	75.84	331.83	0	2	6	30	3847
Watchers	1.77	9.76	0	0	0	0	128
Open Pull Requests	0.16	0.70	0	0	0	0	7
Closed Pull Requests	17.03	250.67	0	0	0	0	4330
Maintainers' efficiency	0.07	0.25	0	0	0	0	0.99
Contributors	1.57	4.32	1	1	1	1	68
Contributors' experience	3281.29	1306.23	368	2189.5	3414.5	4278.25	6040

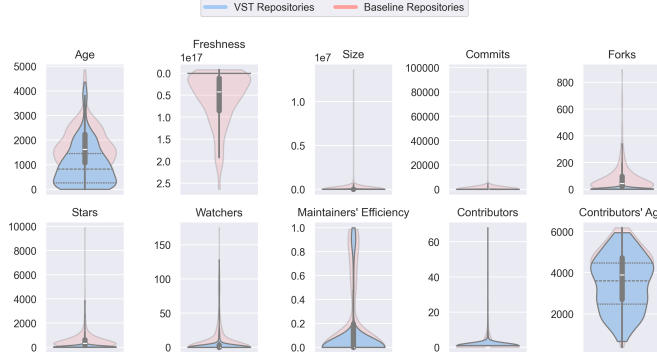


Fig. 6: Repositories and contributors descriptive statistics.

First of all, average **age** of selected VST plugins is ~ 996 days (about 2.7 years), with a relatively symmetrical distribution within Q1 and Q3 and a long tail of long-lived projects that are active since more than 3 years. The **freshness** metric has a similar distribution as that of age, with a median value of ~ 431 days (about 1.18 years); this result, together with the median, tells us that the majority of studies projects had their last commit more than one year ago. This observation finds support in other metrics as well and they represent a significant difference in the level of activity and engagement between a small number of highly successful projects and the majority. We elaborate further on these aspects in Section VII.

Moreover, we can observe that the distributions of the **number of commits, contributors, watchers, and open and**

closed pull requests (PRs) are positively skewed. Specifically, 75% or more of the data points for the number of watchers, open pull requests, and closed pull requests are zero, while the distributions of **number of commits and contributors** exhibit a similar pattern, with 75% of the values being one. The distributions of the **number of forks and stars** follow a comparable trend; however, the number of forks shows a slight increase in the second (Q2) and third quartiles (Q3). In contrast, the number of stars displays more variability across all quartiles, although the differences between them remain small. Upon closer examination of the **number of open and closed pull requests**, we observe a substantial disparity between the maximum values for each. A low number of open pull requests (e.g., 7) suggests that the repositories are likely well-maintained and up-to-date, with maintainers processing pull requests efficiently, closing most as they are reviewed and addressed, or indicating limited use of pull requests by contributors. Conversely, a high number of closed pull requests (e.g., 4,330) indicates a significant level of contribution and code changes over time. However, the distribution reveals that this is not representative of all repositories, but rather characteristic of a select few. **Maintainers' efficiency** refers to the ratio between closed PRs and all PRs. We can observe that for most of the repositories this is zero, which is aligned with the observed behavior in the open and closed PRs.

When comparing the repository activity of VST projects against non-VST projects (our baseline in Figure 6), we observe the following: (i) the size of VST projects (21559.41 Kb) is on average smaller than the size of non-VST projects (79358 Kb), but not statistically (Mann-Whitney test with $p=0.5003$), (ii) non-VST projects have a statistically-significant higher number of commits, forks, and stars, (iii) similarly, non-VST projects have statistically-significant higher number of watchers, open/closed issues and pull requests, but their median is equal to 0 for both VST and non-VST projects, and (iv) non-VST projects have a statistically-significant higher number of contributors, but their median is 1 for both VST and non-VST projects.

D. Contributors, ownership, and licensing

The **contributors' experience** represents the "age" of a user's GitHub profile in days. As shown in Table III, the distribution is slightly negatively-skewed, with the mean (3,281.29 days, ≈ 9 years) slightly lower than the median (3,414.5 days, ≈ 9.3 years). Moderate variability is evident from the standard deviation (368 days, ≈ 1 year) and a broad interquartile range ($IQR = 2,088.75$ days, ≈ 5.7 years), reflecting substantial spread in the middle 50% of the data.

Repositories Ownership refers to whether the repository is created by individuals or organizations. Individual developers dominate VST plugin development, being involved in 257 repositories (85.9%), while organizations are responsible for 42 repositories (14.1%). We observe that *Pongasoft* [36] leads with 4 VST plugins, followed by *Wolf-plugins* [37] with 3 plugins. Other organizations typically maintain 1 or 2 plugins.

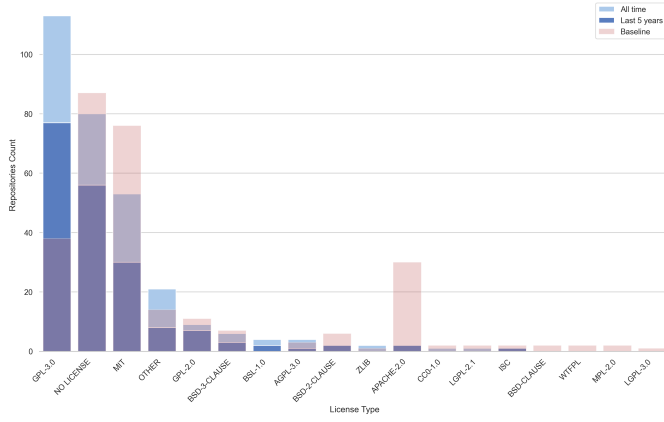


Fig. 7: Distribution of used licenses.

Figure 7 shows the 15 different types of **licenses** that we observed in our dataset. The two most popular licenses are GPL-3.0 [38] and MIT [39]. The first one is the most prevalent, used in 127 repositories (42.5%). The MIT license is the third most popular, featured in 53 repositories (17.7%). No license is specified in 79 repositories (26.4%), *i.e.*, the default copyright law applies, based on GitHub policies. Indeed, despite being hosted in a public open-source environment, GitHub states clearly that if a repository is not licensed, the developers retain all rights to their source code and no one may reproduce, distribute, or create derivative works from their work [40].

When comparing these results with non-VST projects, we observe that (i) non-VST projects exhibit the same trends about ownership, with 222 repositories owned by individual users and 64 repositories owned by organizations and (ii) in line with the results of VST projects, the most used license in our sample of non-VST projects is MIT (76), followed by GPL-3.0 (38) and Apache 2.0 (30), and (iii) a large number of non-VST projects does not specify any license as well (85).

V. MOST USED TECHNOLOGIES (RQ2)

A. Programming Languages

Figure 8 displays the top 15 programming languages (PLs) used across repositories each year, as reported on GitHub. It is important to note that a single repository may use multiple PLs. The most common language is C++, featured in 255 repositories (85.3% of the dataset) (*e.g.*, [xivilay/scale-remapper](#), [nathanjhood/Biquads](#)), followed by C, which is used in 174 repositories (58.2%) (*e.g.*, [vertver/Dynation](#)). We can categorize these top 15 languages into two main groups: configuration languages and utils (*CMake*, *MakeFile*, and *BatchFile*, and *Inno Setup* [41]), and general-purpose languages (GPLs) like *Python*, *R*, or *Rust*. Most of these are well-known languages for programmers, however, it is interesting to observe the appearance of Inno Setup, which is a free installer for Windows programs.

In Figure 8 we also observe that C++ has been used consistently since the early days of VST plugin development

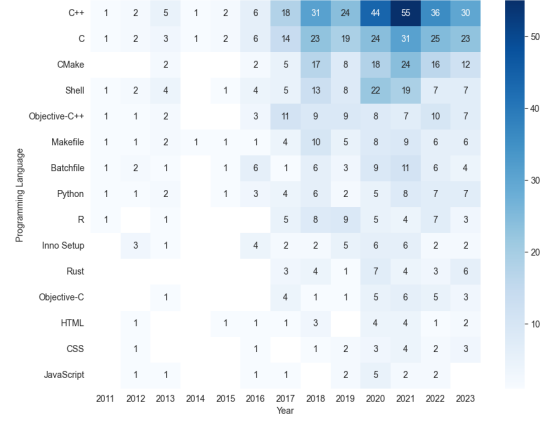


Fig. 8: Top 15 programming languages per year.

in 2011 (see Section IV). Its usage has steadily increased, peaking in 2021—the same year that saw the highest number of VST plugin creations on GitHub (Figure 5). However, its usage has declined in subsequent years, following the same downward trend as the plugin creation metric. This is correlated to the decrease in the creation of repositories (Figure 5). Additionally, other GPLs like *e.g.*, *Python*, *R*, and *Rust* are present in VST plugin development but are less common than C-like languages.

To gain a clearer understanding of VST plugin development, we analyzed the number of programming languages (PLs) used per repository to better grasp the technological stack employed in these projects. Most repositories use between 2 and 5 languages (mean = 3.8, std = 2.75). Notably, some projects use more than 10 languages, with one repository utilizing up to 15. This suggests that VST plugin development typically requires proficiency in at least two languages, often including both general-purpose and configuration languages.

B. Frameworks

Table IV shows the top-10 most used frameworks in our dataset. It can be seen that *JUCE* is used in 152 repositories (50.84% – *e.g.*, [QVbDev/quantumVerb](#)). The second most popular framework is *DPF* [42] which is used in 21 repositories (7% – *e.g.*, [Wasted-Audio/wstd-eq](#)). It is surprising to observe that 52 repositories (17.4% – *e.g.*, [Husenap/VSTakoyaki](#)) do not use any frameworks at all. When manually inspecting these 52 repositories, we discover that only 3 of them ([amsynth/amsynth](#), [sfztools/sfizz-ui](#), and [greatest-ape/OctaSine](#)) have stars, watchers, contributors, opened/closed issues. When compared to the other repositories, the stars, watchers, and contributors numbers are far greater than the median values for those metrics (see Table III), having a repository freshness of less than 50 days, thus suggesting that they are active projects.

C. Libraries

In Section V-A we observed that the most utilized PLs are C and C++. They can have two types of libraries, using header

TABLE IV: Top-10 VST plugin development frameworks.

Name	#Repos	#Stars	#Watchers	#Contributors
JUCE [43]	159	6.6k	257	50
DPF [44]	21	658	23	23
VST3-SDK [45]	9	1.6k	95	3
VST-SDK	8	-	-	-
nih-plug [46]	7	1.7k	41	30
Jamba [47]	4	122	8	1
VST-RS [48]	4	1k	26	40
WDL-OL [49]	3	935	101	14
iPlug2 [50]	3	1.9k	61	57
Cabbage [51]	3	517	25	11

[52] or compiled library [53] files. By manually analyzing our dataset, we observed that 243 repositories (81.3% – (e.g., [ffAudio/Frequalizer](#)) employ header files, while the other 56 repositories (18.7% – (e.g., [igorski/VSTSID](#)) use a combination of header and compiled library files. This is expected considering that the predominant framework, *JUCE* (Section V-B), is only available through the inclusion of header files. Additionally, the usage of header files enables flexibility in compiling source files with different configurations or implementations [54]. However, employing header files can introduce drawbacks such as increased compilation time [55], and code duplication (see Section VI-A).

TABLE V: Top-10 libraries used for developing VST plugins.

Name	Description	#Repos	PL
rand	Pseudo-random number generators for various distributions.	8	Python
os	Portable way of using operating system dependent functionality.	6	Python
vst	Support for creating VST2 plugins.	6	Rust
log	Lightweight logging facade.	5	Rust
simplelog	Logging facilities that can be easily combined.	5	Rust
Serde	Serializing and deserializing Rust data structures.	5	Rust
glob	Querying the file system with a particular pattern.	4	Rust
sys	Help Rust programs use C ("system") libraries.	4	Rust
shutil	Execute shell command pipelines and return <i>stdout</i> .	4	Rust
rust-vst	Support for creating VST2 plugins.	4	Rust

Table V shows the 10 most used libraries across repositories that do not utilize C/C++. We observe that the used libraries are not specific to audio signals processing; for example, *rand* [56] is featured in 9 repositories (e.g., [astral37/opus-parvulum](#)), *log* [57] featured in 6 repositories (e.g., [jcfischer/easylooper](#)), and *simplelog* [58], *serde* [59] alongside *os* [60] are featured in 5 repositories (e.g., [greatest-ape/OctaSine](#), [t-sin/soyboy-sp.vst3](#)). However, it is important to note that sound processing libraries are also used, but less than generic ones; examples of used sound processing libraries include *vst* [61], *rust-vst* [62], *fuzzball* [63], *easylooper* [64], *dd-plugs* [65], and *rvst* [66]).

VI. CODE QUALITY PRACTICES (RQ3)

A. Number of code smells, bugs, and vulnerabilities

We screened all 299 repositories via SonarQube’s code quality analysis. SonarQube executes more than 6k rules defined across more than 30 programming languages [67] on the source code of a software project to generate issues belonging to three main families: (i) *code smells* (maintainability domain), (ii) *bugs* (reliability domain), and (iii) *vulnerabilities* (security domain) [68]. Table VI gives an overview of the results of the code quality analysis on all 299 repositories. SonarQube detected a relatively low number of issues (2,054 in total across 299 projects), with an average of 6.87 issues per project, and minimum and maximum of 0 and 20, respectively.

TABLE VI: Code quality analysis with SonarQube.

Issue type	Count	Repositories rating				
		A	B	C	D	E
Code smells (maintainability)	1,981	279	16	3	0	1
Bugs (reliability)	72	256	12	12	6	13
Vulnerabilities (security)	1	298	0	0	1	0

According to SonarQube documentation, **code smells** are “maintainability issue that makes your code confusing and difficult to maintain” [68]. Code smells are the most recurrent type of issue (1,981 in total); nevertheless, 279 projects out of 299 are rated as A by SonarQube (e.g., [surge-synthesizer/surge](#)), indicating that their remediation time is less than 5% of the time that has already invested into the project [68]. The remaining 20 projects are rated as B (remediation time between 6% and 10%, N=16), C (remediation time between 11% and 20%, N=3), and E (remediation time above 50%, N=1).

SonarQube supports more than 150 different types of **bugs**, ranging from unclosed input–output resources, classes compared by name, etc. The number of bugs is considerably lower than code smells, i.e., 72, still with a similar trend in terms of ratings: most projects are rated as A (zero bugs, N=256), e.g., [p-hlp/CTAGDRC](#), while the remaining ones are rated as B (at least one minor bug, N=12), C (at least one major bug, N=2), D (at least one critical bug, N=6) and E (at least one blocking bug, N=13).

SonarQube detected only one critical **vulnerability** in [joshua-maros/audiobench](#), leading to a D rating for the security of the project. All other projects are rated as A.

B. Cyclomatic and cognitive complexity

Figure 9 shows the distribution of the cyclomatic [69] and cognitive complexity [70] complexity metrics across the projects, as reported by SonarQube.

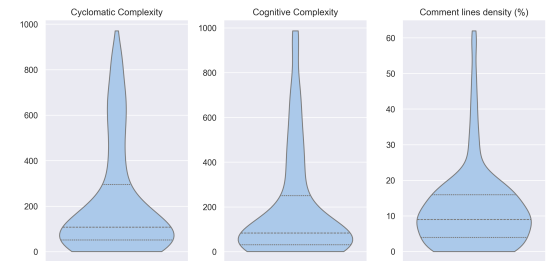


Fig. 9: Cyclomatic and cognitive complexity, comment density.

The median **cyclomatic complexity** is 108. According to Tom McCabe Jr [71], a cyclomatic complexity greater than 50 results in untestable code, and very high risk. However, there are exceptions, for example, the [azurls/penare](#) and [strikles/UQLRF-500](#) have a cyclomatic complexity of 5, meaning that their code implements simple procedures with an overall low risk. We observed that the size of the repository does not correlate with its cyclomatic complexity; for example, the *Penare* plugin is the largest project from our dataset with 64,785Kb, while *UQLRF-500* has a size of 2,6839Kb (See Section IV-B) and they have the same cyclomatic complexity.

The maximum value of cyclomatic complexity in our dataset is 971 for [MeijisIrlnd/Transfer](#).

The median **cognitive complexity** is 84. According to the evaluation by Campbell [72], we can state that most of VST plugins repositories on GitHub are relatively not difficult to understand projects from a mathematical perspective. However, it is important to take into consideration that SonarQube computes cognitive complexity based on a mathematical model [68], and not based on human understanding. The overall score is computed according to the following basic rules: The maximum value of cognitive complexity in our dataset is 987 for [michael-truscott/DoomVst](#).

C. Comment lines density

Code comments are meant to facilitate developers to better understand how the code works [73] and are considered key artifacts in many software engineering tasks related to maintenance and program comprehension [74]. Figure 9 shows the comment lines density across the mined repositories, computed by SonarQube’s formula: $CLD = CL / (NLOC + CL) * 100$. Where: CLD is the Comment Lines Density, CL is the number of comment lines, and $NLOC$ is the non-comment lines of code in the repository. Overall, the median comment density across the mined repositories is $\approx 9\%$, meaning that most repositories have significantly fewer comment lines than net LOCs. Previous work [75] highlighted the fact that active open-source projects have on average a density of 19%. [toasty-ghost/ASE_KM_Caverb](#) has the highest comment line density, at 62.00%. Note that there is debate on whether CLD is a good indicator of the quality of a software project and there is no fixed threshold for it; however, it can be seen as an objective indication about how much “meta” information the contributors are embedding in the code, typically positively impacting code comprehension [76].

D. Testing

To understand how VST plugin testing is conducted, we investigate whether developers test their VST plugins by (i) reviewing README files for mentions of testing practices, and (ii) inspecting repository code for unit tests. Out of the 299 repositories analyzed, 85 (28.42%) report some form of *manual testing* (e.g., [JosephTLyons/Track-Notes-1](#)), while 214 (71.58%) did not report any testing methods in their README files. Developers perform manual testing across various operating systems, including primarily Windows, Linux, and macOS. However, this testing activity primarily focuses on ensuring basic functionality, with no details provided about the hardware configurations used during testing or relevant quality attributes, such as performance, reliability, etc. Moreover, only 9 GitHub repositories (3%, e.g., [DamRsn/NeuralNote](#) and [surge-synthesizer/surge](#)) contain the *source code of unit tests*. Unit testing is the only software testing method employed across the analyzed repositories, with no evidence of specialized testing software, such as Selenium.

VII. DISCUSSION AND IMPLICATIONS

The GitHub ecosystem of VST plugins is young, and has potential. This ecosystem is still emerging, with a median repository age of *approx* 2.5 years and the majority of them developed by a single contributor. This observation is further supported by the fact that many projects have few commits and open/closed pull requests; additionally, the commit count (see Section IV) shows that most commits are the initial commit in the repository.

TABLE VII: Top-10 VST plugins (success stories).

VST project	#Stars	Year	#Contributors
werman/noise-suppression-for-voice	3,847	2018	12
surge-synthesizer/surge	2,627	2018	68
asb2m10/dexed	2,512	2013	17
DISTRHO/Cardinal	1,577	2021	14
jatinchowdhury18/AnalogTapeModel	940	2019	8
michaelwillis/dragonfly-reverb	769	2017	1
monadgroup/axiom	667	2017	3
DamRsn/NeuralNote	551	2023	3
greatest-ape/OctaSine	527	2019	1
leomccormack/SPARTA	463	2018	6

Nevertheless, we also observe the presence of *successful projects* within the VST plugins ecosystem, which are clear outliers in terms of repository activities (e.g., number of commits, and contributors), community engagement metrics (e.g., stars, forks, watchers, and pull requests), and maintainers’ efficiency. Table VII presents the top 10 VST projects by GitHub stars; developers can use this as inspiration for advancing their own projects, while musicians can rely on these as stable tools in their creative workflows.

What does a successful VST project look like? The case of Surge XT. Looking at the combination of metrics that we collected to answer our RQs, the *Surge XT* project [77] emerges as one of the most mature and successful VST projects in our analysis. *Surge XT* is a free and open-source hybrid synthesizer, originally developed as a commercial product at Vember Audio [78]. In September 2018, a partially completed version of Surge XT 1.6 was released on GitHub under GPL3.0 license [77]. Currently, the Surge XT plugin is maintained by the *surge-synthesizer* [79] organization on GitHub, which is a self-organized group of musicians, developers, testers, documenters, volunteers, and open-source enthusiasts collaborating on the Surge Synthesizer [80]. A key feature of the Surge XT repository is its comprehensive and detailed README, which includes: (i) an *overview of the project*; (ii) a *direct link to the binaries of the plugin*; (iii) a *Discord server* that serves as a direct line to developers and as a welcoming hub for Surge plugin users to connect, share insights, report bugs, and request features; (iv) a link to *official documentation* [81] that serves as a user manual where musical and technical aspects are covered, while also offering tutorials; (v) a link to a *developer guide* [82] describing a set of clear guidelines to contribute to the project; and (vi) a *build guide* on how to build the project for different platforms locally, and remotely via Azure pipelines [83].

Surge XT maintainers respond quickly to GitHub issues, with a median response time of 2.12 hours (compared to 14.12 hours for other VST projects in our dataset) and a median

closing time of 0 days. It is also one of the few repositories that perform unit testing and include the source code of their test cases in their GitHub repository (see [Section VI-D](#)).

By inspecting Surge XT’s documentation [\[81\]](#), we observe that developers prioritize transparency and thorough documentation. They meticulously describe Surge XT’s overall architecture (both textually and visually), from the synthesizer engine to each functionality it offers (e.g., LFOs [\[84\]](#), and oscillator algorithms [\[85\]](#)). Finally, developers are involved in addressing plugin-related issues, and they promote community interaction through a Discord server, enabling users to seek support for the Surge XT plugin and engaging in discussions with fellow users.

After inspecting Surge XT’s source code and documentation, and discussing our results with its development team on Discord [\[86\]](#), we observe the following:

- The project provides a stable automated CI infrastructure that includes GitHub workflows and Open Build Service for (nightly) builds that support multiple distributions and hardware architectures.
- Surge XT can also be built in headless mode, i.e., as a standalone executable of Surge XT without a UI, facilitating debugging and testing activities.
- The Surge XT’s software architecture is explicitly documented [\[87\]](#) and made openly available to contributors, where we identified six interesting traits: (i) There is a clear separation of concerns, where third-party libraries and individual functionalities are kept independent from each other; (ii) The Surge XT core engine defining the DSP and voice handling logic (see [Section II](#)) lives in its own `common` module; (iii) A dedicated GUI module contains all functionalities related to the GUI, with custom UI widget classes, and various UI-related helpers; (iv) the `SurgeStorage` module acts as a centralized global entity containing a shared set of parameters available to all modules at runtime. In a way, it acts as a *blackboard component* [\[88\]](#). (v) Surge XT makes an heavy use of the *adapter design pattern* for masking the complexity and platform-specific aspects of the used third-party libraries/modules; (vi) The DSP engine of Surge XT uses dependency injection via CRTP [\[89\]](#) in order to avoid the overhead of virtual methods calls at runtime.

Finally, during this research, it emerged that some of the strongest indicators of Surge XT’s success are about the *social sustainability* of the project, rather than its technical aspects. Specifically, by quoting parts of a discussion we had with members of the Surge XT community in January 2024: “[they] had good people working enthusiastically on every part of it [Surge XT] (coding, design, product ideas, testing, doc, build infrastructure, sound design)”, “the starting point was super strong but the owner of the starting point was open to change”, “[they] instituted a mostly-don’t-be-jerks rule and that stuck, [they] usually apologize when [they] break it, [they] say please and thank you most of the time, and [they] aim to be respectful of everyone’s contribution”, and “anyone is welcome to join

the community, as long as they observe [the previous rule]”.

VST plugin development has high entry barriers. Characterization of the *typical* VST developer is challenging. Most repositories are under 3 years old, often with contributors who aren’t primarily software developers. Many projects have a single commit and are infrequently updated, with an average freshness of 1.8 years. However, contributors have a median experience of 9 years on GitHub.

The most used programming languages for VST projects on GitHub are C++ and C (see [Section V-A](#)). Their popularity can be justified by three reasons. First, according to the TIOBE index [\[90\]](#), C++ and C are the second and fourth most used programming languages as of November 2024, respectively. Second, C++ and C allow close-to-hardware performance, which is crucial in real-time audio processing, where latency is critical. Lastly, many essential frameworks and libraries, such as the VST SDK [\[91\]](#), JUCE [\[35\]](#), RackAFX [\[92\]](#), ASIO [\[93\]](#) (a Steinberg library for low-latency, high-fidelity audio streaming), and Audio Toolkit [\[94\]](#) (an open-source collection for audio processing), are written in C++. The higher popularity of textual C-like languages for VST development results in a higher entry barrier for novice programmers, especially for creatives/musicians without a technical background [\[95\]](#). Moreover, most repositories contain source code in multiple programming languages ([Section V-A](#)), further increasing the technical skills needed to contribute to these projects.

A good starting point for novice VST developers include: (i) the inspection of repositories in [Table VII](#) with a focus on the software design patterns, coding style, and tools used in successful projects, (ii) the usage of template projects to reduce the development time of their VST plugins (e.g., Pample Juce [\[96\]](#)), and (iii) the study of educational resources on VST development in general (e.g., [\[97\]](#)).

Software Engineering practices for VST plugins are still immature. With only 9 out of 299 repositories containing test cases, we can safely conclude that testing practices in the VST ecosystem are scarce (see [Section VI](#)). Moreover, according to the cyclomatic complexity results, the source code of most VST projects can be classified as *untestable*. These are not the only metrics to be considered when assessing the technical level of a software project, but they are already a first indication of the immaturity of software engineering practices for VST plugins. Moreover, despite the variety of programming languages, libraries, and frameworks used in VST plugin development on GitHub (see [Sections IV](#) and [V](#)), *standardized development guidelines are lacking*, particularly regarding best practices in terms of architecture, development, and testing of VST plugins. Researchers can play a crucial role in filling this gap by using this study’s results as a starting point for conducting surveys targeting developers to gather a better understanding of the current development practices and needs of VST plugin developers, as well as developing specialized tooling that can support creatives and musicians develop their VST plugins (e.g., IDEs, interfaces, notations, and domain-specific languages). These insights can guide the design and validation of improved development tools for VST,

particularly in testing and assessing code quality in terms of performance and reliability.

SonarQube, one of the most used code quality inspection tools, detected very few issues in general. This result might be an indication of two phenomena: either VST plugins are perfect (this is doubtful given the number of open issues and PRs) or SonarQube is not a good fit for this type of software. We do not have the data to give a definitive answer on this, but overall, this can be considered an indication of the *need for VST-specific tools for debugging and assessing the source code of VST plugins*. This is a promising line of research, both for software engineering researchers and tool vendors.

Lack of highly-visible initiatives from major players. Based on the collected data, we can speculate that many VST projects are created out of passion for music production or personal interest (e.g., [mariusz96/blue-synthesiser](#)). Another reason is that many VST plugins, though not demos or templates, are developed primarily for personal learning or to enhance programming skills in languages like C++, C, or Python (see [Figure 8](#)). Overall, we found few projects tied to larger initiatives from foundations or organizations, which is a missed opportunity. Based on our data, the foundations for a vibrant open-source ecosystem around VST technology are present (see [Table VII](#)), but they require more support to flourish. We urge major players in the music industry, such as Roland, Korg, Moog, Yamaha, Arturia, and Behringer, to invest in highly-visible initiatives like hackathons, financial support, or in-kind contributions to open-source projects. This would strengthen the community and drive progress toward better virtual instruments, improved development tools for VST plugins, and a more open, accessible future for music.

VIII. THREATS TO VALIDITY

Internal Validity. This study paid special attention to the reproducibility of our results by following a systematic mining process (see [Section III](#)). Overall the systematic process was conducted following this steps: (i) store the all the data in a CSV file; (ii) extract relevant fields from the CSV file, containing the data of the 299 repositories to answer the research questions; (iii) enhance the dataset with additional data obtained via SonarQube, and issue-metrics; (iv) utilize analysis methods for addressing each research question, combining quantitative analysis with qualitative analysis. Finally, we include a replication package [\[28\]](#) containing the scripts for mining the repositories and analyzing them, SonarQube analysis reports, and both raw and curated data, to facilitate the replication of our study.

External Validity. While our study provides valuable insights into the VST plugin development practices, it is important to acknowledge that GitHub does not fully represent all development practices that occur in this niche. There could be other open-source communities (e.g., GitLab [\[98\]](#), Bitbucket [\[99\]](#)) and closed-source communities.

Construct Validity. The selection of metrics and data for analysis is driven by our RQs, thus, we aim to ensure that the data collected accurately reflects the repository characteristics,

community involvement and engagement, and development practices. To this end, we used Cohen’s Kappa coefficient to guarantee a high level of agreement in the data collection and the application of the selection procedure. However, it is important to acknowledge potential limitations: (i) mining date *27th of September, 2023*: mined data (e.g., the number of commits), might be subject to change over time; (ii) bias: we aimed to reduce as much as possible the degree of bias, by checking the level of agreement between the authors, but this does not guarantee its complete removal; (iii) since we are not the contributors of the analysed repositories, we do not have full access to the decision-making processes behind them.

Conclusion Validity. We mitigate threats to the conclusion validity by (i) including a replication package to allow third-party researchers to verify and check the obtained results; (ii) documenting our mining process to ensure the integrity and replicability of our dataset; (iii) employing quantitative and qualitative analysis methods to strengthen the reliability of our findings. However, it is important to acknowledge potential risks: (i) given the exploratory nature of this study, we did not perform statistical tests within the data; (ii) bias in the definition of the inclusion and exclusion protocol.

IX. CONCLUSION AND FUTURE WORK

This paper explores the development practices of open-source VST plugins on GitHub. For this goal, we mined $\approx 1.5k$ GitHub repositories. After applying a rigorous selection procedure, we created a dataset of 299 GitHub repositories containing open-source VST projects. Then, we analyzed all repositories both quantitatively and qualitatively to understand the development practices of the VST plugins development community on GitHub. Based on the experimental data, we conclude that: (i) most VST projects are VST effects (VSTfx) and are maintained by a single contributor on GitHub, with the majority of them created in the last 5 years; (ii) most VST projects are developed using C-like languages, the JUCE framework, and several non-VST-specific libraries; (iii) SonarQube detected primarily code smells in VST projects, few bugs, and only one vulnerability, highlighting the need for domain-specific code analysis tools for VST development; (iv) the detected cyclomatic complexity of the analysed projects indicates a high risk in terms of their testability; (v) testing practices are still immature, with only 9 repositories containing unit tests.

As next steps, in addition to investigating a selection of the future research directions discussed in [Section VII](#), we are evaluating the application of qualitative research methods to obtain additional and deeper insights into the VST plugin development community and its practices and needs.

ACKNOWLEDGMENTS

We are grateful to the Surge XT developer community, especially [baconpaul](#), [EvilDragon](#), and [Andreya](#) for their valuable feedback on the initial draft of this article and their insights into the Surge XT community dynamics.

REFERENCES

- [1] C. Music, “Early DAWs: the software that changed music production forever,” *MusicRadar*, Feb. 2020. [Online]. Available: <https://www.musicradar.com/news/early-daws-the-software-that-changed-music-production-forever>
- [2] A. Reuter, “Who let the daws out? the digital in a new generation of the digital audio workstation,” *Popular Music and Society*, vol. 45, no. 2, pp. 113–128, 2022.
- [3] “DAW Software | Equipboard,” Oct. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: <https://equipboard.com/c/daw?sort=most-used>
- [4] “Virtual studio technology definition.” [Online]. Available: https://en.wikipedia.org/wiki/Virtual_Studio_Technology
- [5] “VST - VST 3 Developer Portal,” Jul. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: https://steinbergmedia.github.io/vst3_dev_portal
- [6] W. Pirkle, *Designing audio effect plugins in C++: for AAX, AU, and VST3 with DSP theory*. Routledge, 2019.
- [7] “Audio Plug-ins Software Application Market - Global Industry Analysis and Forecast (2024 -2030),” Aug. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: <https://www.maximizemarketresearch.com/market-report/global-audio-plug-ins-software-application-market/100413>
- [8] Roland Corporation, “Roland - Roland Cloud,” Oct. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: https://www.roland.com/global/categories/roland_cloud
- [9] “JUCE,” Oct. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: <https://github.com/juce-framework/JUCE>
- [10] “Made With JUCE - JUCE,” Apr. 2024, [Online; accessed 17. Oct. 2024]. [Online]. Available: <https://juce.com/made-with-juce>
- [11] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “An in-depth study of the promises and perils of mining github,” *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.
- [12] “Digital audio workstation definition.” [Online]. Available: https://en.wikipedia.org/wiki/Digital_audio_workstation
- [13] “Clap official page,” 2025. [Online]. Available: <https://cleveraudio.org>
- [14] “Aax sdk official page,” 2025. [Online]. Available: <https://developer.avid.com/aax>
- [15] “Lv2 official page,” 2025. [Online]. Available: <https://lv2plug.in>
- [16] “Ableton Live,” Oct. 2024, [Online; accessed 24. Oct. 2024]. [Online]. Available: <https://www.ableton.com/en/live>
- [17] D. M. Huber, *The MIDI manual: a practical guide to MIDI in the project studio*. Routledge, 2012.
- [18] C. Cannam, L. A. Figueira, and M. D. Plumbley, “Sound software: Towards software reuse in audio and music research,” in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 2745–2748.
- [19] I. Damjanovic, L. A. Figueira, C. Cannam, and M. D. Plumbley, “soundsoftware. ac. uk survey report,” 2011, [Online; accessed 27. Jan. 2025]. [Online]. Available: <http://code.soundsoftware.ac.uk/documents/17>
- [20] G. Bulet and A. Hindle, “An empirical study of end-user programmers in the computer music community,” in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR ’15. IEEE Press, 2015, p. 292–302.
- [21] A. Islam, K. Eng, and A. Hindle, “Opening the valve on pure-data: Usage patterns and programming practices of a data-flow based visual programming language,” in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 492–497.
- [22] S. Stickland, R. Athauda, and N. Scott, “Design and evaluation of a scalable real-time online digital audio workstation collaboration framework,” *Journal of the Audio Engineering Society*, vol. 69, no. 6, pp. 410–431, 2021.
- [23] I. J. Clester and J. Freeman, “Composing with generative systems in the digital audio workstation,” in *Joint Proceedings of the ACM IUI Workshops*, 2023.
- [24] M. Choetkiertikul, A. Hoonlor, C. Ragkhitwetsagul, S. Pongpaichet, T. Sunetmanta, T. Sette Wong, V. Jiravatvanich, and U. Kaewpichai, “Mining the characteristics of jupyter notebooks in data science projects,” 2023.
- [25] I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago, and D. Garlan, “Mining guidelines for architecting robotics software,” *Journal of Systems and Software*, vol. 178, p. 110969, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121221000662>
- [26] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, ser. Computer Science. Springer, 2012.
- [27] F. Shull, J. Singer, and D. I. Sjöberg, *Guide to advanced empirical software engineering*. Springer, 2007.
- [28] I. M. Bogdan Andrei, Mauricio Verano Merino, “Replication package of this study,” 2025. [Online]. Available: <https://doi.org/10.5281/zenodo.14810650>
- [29] V. R. Basili and H. D. Rombach, “The tame project: Towards improvement-oriented software environments,” *IEEE Transactions on software engineering*, vol. 14, no. 6, pp. 758–773, 1988.
- [30] “Preliminary search on GitHub,” Nov. 2024, [Online; accessed 8. Nov. 2024]. [Online]. Available: <https://github.com/search?q=vst+plugin&stars&o=desc&type=repositories>
- [31] “Sonarqube official website.” [Online]. Available: <https://www.sonarsource.com/products/sonarqube/>
- [32] I. Malavolta, T. A. Ghaleb, I. David, J. van Rooijen, and M. Stoelinga, “Engineering mobile apps for disaster management - the case of covid-19 apps in the google play store,” *IEEE Software*, vol. 39, no. 3, pp. 31–42, November 2021. [Online]. Available: http://www.ivanomalavolta.com/files/papers/IEEE_Software_COVID_2021.pdf
- [33] V. Lenarduzzi, N. Saarimäki, and D. Taibi, “Some sonarqube issues have a significant but small effect on faults and changes. a large-scale empirical study,” *Journal of Systems and Software*, vol. 170, p. 110750, 2020.
- [34] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.
- [35] JUCE, “The official documentation of juce framework.” [Online]. Available: <https://juce.com/>
- [36] pongasoft organization, “pongasoft organization github profile.” [Online]. Available: <https://github.com/pongasoft>
- [37] wolf-plugins organization, “wolf-plugins organization github profile.” [Online]. Available: <https://github.com/wolf-plugins>
- [38] G. O. System, “Gpl-3.0,” <https://www.gnu.org/licenses/gpl-3.0.html>, 2023, [Online, accessed 24 September 2024].
- [39] MIT, “Mit license,” https://en.wikipedia.org/wiki/MIT_License, 2009, [Online, accessed 24 September 2024].
- [40] GitHub, “Licensing a repository official documentation.” [Online]. Available: <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository>
- [41] J. Russell and M. Laan, “Inno setup,” <https://jrsoftware.org/isinfo.php>, [Online, accessed 28 October 2024].
- [42] “Dpf framework github repository.” [Online]. Available: <https://github.com/DISTRHO/DPF>
- [43] JUCE, “Juce framework.” <https://github.com/juce-framework/JUCE>, [Online, accessed 28 October 2024].
- [44] DISTRHO, “Dpf - distrho plugin framework,” <https://github.com/DISTRHO/DPF>, [Online, accessed 28 October 2024].
- [45] Steinberg, “Vst3 - sdk,” <https://github.com/steinbergmedia/vst3sdk>, [Online, accessed 28 October 2024].
- [46] R. van der Helm, “Nih-plugin,” <https://github.com/robbert-vdh/nih-plugin>, [Online, accessed 28 October 2024].
- [47] pongasoft, “Jamba,” <https://github.com/pongasoft/jamba>, [Online, accessed 28 October 2024].
- [48] RustAudio, “Vst-rs,” <https://github.com/RustAudio/vst-rs>, [Online, accessed 28 October 2024].
- [49] O. Larkin, “Wdl-ol,” <https://github.com/olilarkin/wdl-ol>, [Online, accessed 28 October 2024].
- [50] iPlug, “iPlug,” <https://github.com/iPlug2/iPlug2>, [Online, accessed 28 October 2024].
- [51] R. Walsh, “Cabbage,” <https://github.com/rorywalsh/cabbage>, [Online, accessed 28 October 2024].
- [52] IBM, “Header files definition,” <https://www.ibm.com/docs/en/aix/7.2?topic=reference-header-files>
- [53] —, “Compiled library files definition,” <https://www.ibm.com/docs/en/xl-c-and-cpp-linux/16.1.0?topic=library-compiling-linking>
- [54] Z. Djalalian, “Preprocessor for c++ class implementation,” Master’s thesis, Concordia University, 2000, unpublished. [Online]. Available: <https://spectrum.library.concordia.ca/id/eprint/1056/>

- [55] Y. Yu, H. Dayani-Fard, and J. Mylopoulos, "Removing false code dependencies to speedup software build processes." 01 2003, pp. 343–352.
- [56] Rust, "The official documentation of "rand" library." [Online]. Available: <https://docs.rs/rand/latest/rand/>
- [57] —, "The official documentation of "log" library." [Online]. Available: <https://docs.rs/log/latest/log/>
- [58] —, "The official documentation of "simplelog" library." [Online]. Available: <https://docs.rs/simplelog/latest/simplelog/>
- [59] S. RS, "Serde rs," <https://github.com/serde-rs/serde> [Online, accessed 24 September 2024].
- [60] Python, "The official documentation of "os" library." [Online]. Available: <https://docs.python.org/3/library/os.html>
- [61] Rust, "The official documentation of "vst" library." [Online]. Available: <https://docs.rs/vst/latest/vst/>
- [62] R. organization, "'vst-rs' library official github repository," 2017. [Online]. Available: <https://github.com/RustAudio/vst-rs>
- [63] fake-industries organization, "'fuzzball' vst plugin official github repository," 2020. [Online]. Available: <https://github.com/fake-industries/fuzzball>
- [64] J.-C. Fischer, "'easylooper' vst plugin official github repository," 2018. [Online]. Available: <https://github.com/jcfischer/easylooper>
- [65] monomadic, "'dd-plugs' vst plugin official github repository," 2017. [Online]. Available: <https://github.com/monomadic/dd-plugs>
- [66] E. Stankov, "'rvst' vst plugin official github repository," 2018. [Online]. Available: <https://github.com/EmilianStankov/rvst>
- [67] Nov. 2024, [Online; accessed 4. Nov. 2024]. [Online]. Available: <https://rules.sonarsource.com>
- [68] "Sonarqube metrics definition." [Online]. Available: <https://docs.sonarsource.com/sonarqube/latest/user-guide/metric-definitions/>
- [69] "Cyclomatic complexity definition." [Online]. Available: https://en.wikipedia.org/wiki/Cyclomatic_complexity
- [70] "Cognitive complexity definition." [Online]. Available: https://en.wikipedia.org/wiki/Cognitive_complexity
- [71] T. M. Jr., "Software quality metrics to identify risk," 2008. [Online]. Available: <https://web.archive.org/web/20220329072759/http://www.mccabe.com/ppt/SoftwareQualityMetricsToIdentifyRisk.ppt>
- [72] G. A. Campbell, "Cognitive complexity." [Online]. Available: <https://www.sonarsource.com/docs/CognitiveComplexity.pdf>
- [73] J. K. Ousterhout, *A philosophy of software design*. Yaknyam Press Palo Alto, CA, USA, 2018, vol. 98.
- [74] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz, "A decade of code comment quality assessment: A systematic literature review," *Journal of Systems and Software*, vol. 195, p. 111515, 2023.
- [75] O. Arafat and D. Riehle, "The comment density of open source software code," in *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 2009, pp. 195–198.
- [76] L. Pascarella and A. Bacchelli, "Classifying code comments in java open-source software systems," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 227–237.
- [77] surge-synthesizer organization, "'surge' vst plugin official github repository," 2018. [Online]. Available: <https://github.com/surge-synthesizer/surge>
- [78] V. Audio, "Vember audio official website," <https://vemberaudio.se/>
- [79] "Surge synthesizer organization official website." [Online]. Available: <https://surge-synth-team.org/>
- [80] S. Williams, "Baconpaul from the Surge Synthesizer Team - JUCE," *JUCE*, Jun. 2024. [Online]. Available: <https://juce.com/made-with-juce/baconpaul-from-the-surge-synthesizer-team>
- [81] "Surge xt official documentation." [Online]. Available: <https://surge-synthesizer.github.io/manual-xt/>
- [82] "Surge official developer guide." [Online]. Available: <https://github.com/surge-synthesizer/surge/blob/main/doc/Developer%20Guide.md>
- [83] "Azure pipelines official documentation." [Online]. Available: <https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops>
- [84] Wikipedia, "Low frequency oscillation wikipedia page," https://en.wikipedia.org/wiki/Low-frequency_oscillation
- [85] —, "Electronic oscillator wikipedia page," https://en.wikipedia.org/wiki/Electronic_oscillator
- [86] "Discord - Surge Synth Team," Jan. 2025, [Online; accessed 3. Feb. 2025]. [Online]. Available: <https://discord.com/channels/744319641211633774/744324663383031821>
- [87] "Surge Architecture," Feb. 2025, [Online; accessed 3. Feb. 2025]. [Online]. Available: <https://github.com/surge-synthesizer/surge/blob/main/doc/Surge%20Architecture.md>
- [88] F. Buschmann, R. Meunier, H. Rohnert, P. Sornmerlad, and M. Stal, *Pattern-oriented software architecture: a system of patterns. Volume 1*. Wiley, 2001.
- [89] D. Abrahams and A. Gurtovoy, *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Pearson Education, 2004.
- [90] "Tiobe index - november 2023." [Online]. Available: <https://www.tiobe.com/tiobe-index/>
- [91] Steinberg, "The official documentation of vst sdk framework." [Online]. Available: <https://www.steinberg.net/developers/>
- [92] W. Pirkle, "The official documentation of rackafx framework." [Online]. Available: <https://www.willpirkle.com/rackafx/>
- [93] A. for all, "The official documentation of asio library." [Online]. Available: <https://asio4all.org/>
- [94] A. Toolkit, "The official documentation of audio toolkit library set." [Online]. Available: <https://www.audio-tk.com/>
- [95] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 2004, pp. 199–206.
- [96] "pamplejuce," Nov. 2024, [Online; accessed 8. Nov. 2024]. [Online]. Available: <https://github.com/sudara/pamplejuce>
- [97] "Audio-Plugin-Development-Resources," Oct. 2024, [Online; accessed 18. Oct. 2024]. [Online]. Available: <https://github.com/jaredrayton/Audio-Plugin-Development-Resources>
- [98] "Gitlab definition." [Online]. Available: <https://about.gitlab.com/>
- [99] "Bitbucket definition." [Online]. Available: <https://bitbucket.org/product>