Labor Market Supply Through Open-Source Projects

Nikolaos Nikolaidis[1][0000-0002-7958-9393], Panagiotis Machairas[2], Apostolos Ampatzoglou[2][0000-0002-5764-7302], Alexander Chatzigeorgiou[2][0000-0002-5381-8418] and Nikolaos Mittas[1][0000-0003-3061-7864]

[1] Democritus University of Thrace, Kavala, Greece
[2] University of Macedonia, Thessaloniki, Greece

Abstract. The rapid evolution of information technology continually reshapes the labor market, creating a persistent demand for up-to-date technological skills. While Open-Source Software (OSS) repositories like GitHub offer a rich data source for tracking these skills, prior studies are often constrained by small sample sizes or reliance on subjective metadata. This paper introduces a novel, scalable methodology to identify technological skills by analyzing the file content of many repositories, providing an objective measure of technology usage at an unprecedented scale. Applying our methodology, we conduct an empirical study that reveals several key findings: (1) Python has emerged as a dominant language for new projects, alongside the foundational web stack (JavaScript, HTML, CSS); (2) developers overwhelmingly specialize, with the median contributor focusing on a single programming language; and (3) Aa clear technology lifecycle is visible, with a definitive shift away from older languages like C/C++ toward modern alternatives for new development. The analysis provides a data-driven portrait of the OSS ecosystem, offering valuable insights for educators, developers, and policymakers.

Keywords: Skills, Repositories, Developers, Open source, Software mining

1 Introduction

The contemporary world is characterized by an unprecedented pace of technological evolution, fundamentally reshaping industries, economies, and societal structures. Information technology acts as a primary catalyst for this transformation, profoundly influencing the labor market by creating new roles while rendering others obsolete [1]. This rapid change has created a persistent “skills gap”, where the competencies demanded by employers are often misaligned with the training and expertise available in the workforce [2]. For educators, policymakers, and professionals alike, the ability to accurately identify and track the most relevant and emerging technological skills is no longer just an advantage, but rather a critical necessity for economic competitiveness and individual career development.

In this dynamic landscape, the Open-Source Software (OSS) ecosystem, with platforms like GitHub hosting hundreds of millions of repositories, has emerged as a crucial and vast data source. OSS projects are not merely repositories of code; they

are living ecosystems that mirror the trends, collaborations, and technological adoptions of the broader software development industry [3]. The technologies used in these projects, from programming languages to frameworks and tools, serve as a real-time, large-scale proxy for the skills that are currently in demand. Analyzing this ecosystem offers a unique opportunity to understand the technological landscape in a way that traditional surveys or job market analyses cannot.

However, despite the richness of this data, existing research aimed at identifying technological skills from OSS platforms faces significant limitations. Many studies rely on small, curated samples of popular projects, which may not represent the full diversity of the ecosystem [4]. Others are limited by their methodology, often analyzing project metadata, README files, or user-provided tags, which can be incomplete, subjective, or outdated [5]. Furthermore, few approaches have been able to perform comprehensive, longitudinal analysis at a massive scale, leaving critical questions about the lifecycle and co-evolution of technologies unanswered. The sheer volume and heterogeneity of data on platforms like GitHub present a formidable technical challenge that has constrained the scope and depth of prior investigations.

To address these shortcomings, this paper introduces a novel and scalable methodology for identifying and analyzing technological skills across possibly the entirety of a git-based ecosystem. Our approach is designed for comprehensive data ingestion, moving beyond metadata to analyze the file content of projects, thereby providing a more objective and fine-grained measure of technology usage. The core contribution of this work is a methodology, and an accompanying tool, that can operate on a very big scale, making it possible to analyze trends across many projects.

The rest of the paper is organized in the following way: in Section 2 we present related work; in Section 3 we present the proposed methodology and tool. Section 4 contains the design of our study, along with the results. Finally, we conclude the paper in Section 5.

2 Related Work

A lot of information about developers' skills in the labor market can be found in the literature, thus we try to present some of those studies in this section. Firstly, a foundational theme in this area is the creation of developer profiles that function as an enhanced, evidence-based curriculum vitae (CV). For example, Greene and Fischer [6] developed CVExplorer, a tool that correlates developers with their skills by analyzing contributions, programming languages, and ReadMe files from GitHub repositories to create a web representation of a CV. Similarly, Lopes et al. [7] built EXTRACTPRO, a tool that consolidates self-reported skills with objective metrics like lines of code and number of commits to create searchable developer profiles. These approaches establish the value of using repository data to verify and discover developer expertise.

A substantial portion of prior research has concentrated on identifying expertise within specific programming languages or popular library ecosystems. Santos et al. [4] proposed a method for identifying experts in specific Java libraries (e.g., Hadoop,

Elasticsearch) by measuring metrics like knowledge intensity and code proficiency. Montandon and Valente [8] focused on the JavaScript ecosystem, matching developers to their experience in libraries like React and Node.js. In a similar vein, Kourtzanidis et al. [9] built RepoSkillMiner, a tool that uses an NLP model to identify low-level skills specifically within .NET and front-end frameworks like React and Angular. While effective, these studies are often constrained to a single technological stack.

Other methodologies have sought to enrich skill profiles by integrating data from multiple platforms or by applying more advanced analytical techniques. Saxena and Pedanekar [10] correlated GitHub commit data with developer activity on Stack Overflow, matching code changes (specifically imports and method calls) with question-and-answer tags to build a "SkillMap". Hauff and Gousios [5] presented a pipeline that uses NLP on README files to identify entities and match developer profiles to job advertisements. More recently, Atzberger et al. [11] developed the CodeCV tool, which uses a Labeled Latent Dirichlet Allocation (LLDA) model on source code to calculate an expertise score for high-level concepts like machine learning. These studies demonstrate sophisticated ways to infer skills but can rely on metadata that may be incomplete or on complex models that are computationally intensive.

In contrast to the aforementioned approaches, our work introduces a methodology distinguished by its scale, objectivity, and language-agnostic nature. While previous studies often focus on a single ecosystem (e.g., Java), rely on interpreting metadata (e.g., README files), or are limited to smaller sample sizes, we present an approach that can analyze the entire file structure of many repositories. Instead of inferring skills from specific library calls or NLP, we use file extensions as a direct, objective proxy for technology usage.

3 Proposed Approach

In this section, we provide information about the methodology and the tool that we developed to get the repositories and analyze them.

3.1 Methodology

To investigate the supply of technological skills in the OSS ecosystem, this study employs a novel, multi-stage methodology for large-scale data collection and analysis. The approach is designed to overcome the limitations of traditional methods, such as API calls directly to GitHub, which have a rate limit (5,000 requests per hour), and reliance on subjective self-reported data, by directly processing repository contents at an unprecedented scale. The process is divided into three main phases: (1) Data Sourcing and Curation, (2) Granular Contribution and Skill Extraction, and (3) Continue real-time trends.

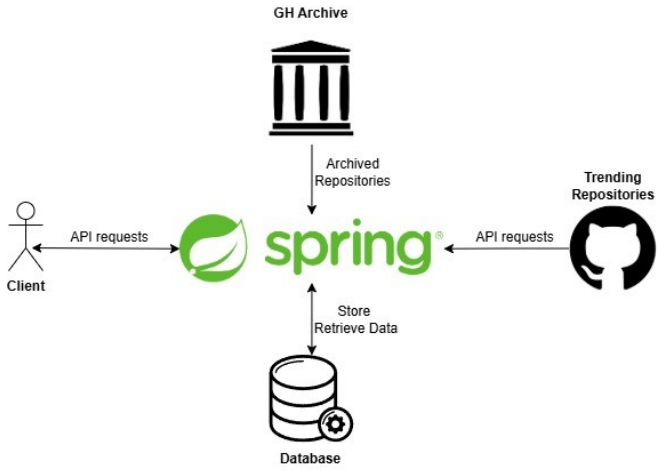

Fig. 1. Tool Architecture

Phase 1

The foundation of our analysis is a comprehensive dataset of active open-source projects. To circumvent the request limitations of the live GitHub API, we leverage GH Archive, a project that records all public GitHub event data and makes it available for bulk analysis.

To query the entire GH Archive dataset spanning from 2016 to the present, we utilized Google BigQuery. An initial, broad query identifies the names of all unique repositories that have registered any form of public event (e.g., push, fork, issue, watch event) during this period. This initial step yields a set of approximately 400 million repositories, demonstrating the scale of the operation.

To focus our analysis on repositories with a substantial development history and community engagement, we apply the following filtering process. Using BigQuery, we identify repositories that have received more than 50 stars, within distinct temporal windows (e.g., 2016-2019, 2020-2022, 2023-onwards).

To avoid analyzing the same project multiple times, we process the filtered repository lists, with a custom script that ensures that a repository appearing in multiple periods is assigned only to the most recent period. The output of this phase is a curated set of unique, significant, and time-stamped repository names, ready for analysis.

Phase 2

With a curated list of target repositories, the second phase focuses on extracting empirical data about the technologies used and the developers who use them. This is performed by an automated pipeline that systematically clones and analyzes each repository.

For each project, we extract key metadata, including its creation date (from the first commit) and the date of its last commit, using standard Git commands. The core of our skill identification method involves a comprehensive scan of each repository's file

structure, by identifying and counting every unique file extension. This provides a direct, objective proxy for the programming languages, frameworks, and technologies employed within the project, moving beyond ambiguous repository tags or descriptions.

To link technologies to developers, we parse the complete Git commit log of each repository. We extract detailed information for every commit like the contributor's identity, the files changed in the commit, and the number of lines inserted and deleted for each file.

Phase 3

To complement the historical analysis of established repositories, our methodology incorporates a distinct process for capturing the leading edge of open-source development. We also collect the list of projects featured on GitHub's public "Trending" page.

These repositories can be part of the historical analysis but we categorize them as "trending" since they experiencing a current surge in community interest and activity. Once identified, these repositories are subjected to the same granular skill and contribution extraction pipeline described in Phase 2, allowing for a direct comparison between established technologies and those at the forefront of innovation.

3.2 DevSkill Tool

For the calculation of the proposed approach, we created the DevSkill tool. This tool was created as a web application, with a front-end, written in React and a back-end written in Java and the Spring framework. The web service exposes all the necessary functionalities through a RESTful API, whereas the web app makes the appropriate requests and demonstrates the appropriate results and views to the user. We should note that the code of both the frontend and backend, with extra information, can be found online[1]. A snapshot of the tool is visible in Figure 2, and the main functionalities of the application are the following.

Repositories. The user can browse the repositories that have been analyzed through the repositories page, where each one can see important information, such as their name, GitHub URL, creation date, last commit date, the extensions that this repository has, and contributors. Additionally, the user can search for a specific repository by name or URL, or filter repositories by programming language. Moreover, the user can see the top 5 repositories based on the number of contributors.

Contributors. The user can browse the analyzed contributors through the contributors' page, where they can see information such as GitHub username, email, the repositories the programmer has contributed to, the changes he has made to specific extensions and programming languages (deletions, insertions, and in total).

File Extensions. The extensions page shows the name of each extension, its programming language equivalent, the number of files the extension has been found in, the number of repositories it has been found in, and when it was last used. The user

[1] https://github.com/panagiotismach/devskill

can search by extension and programming language. Furthermore, two graphs show the top 5 extensions and languages.

Trending Repositories. Finally, the trending repositories page has the same functionality as the repositories page, with the only difference being the GitHub repositories it shows. This page shows only repositories belonging to the trending ones, according to GitHub[2].

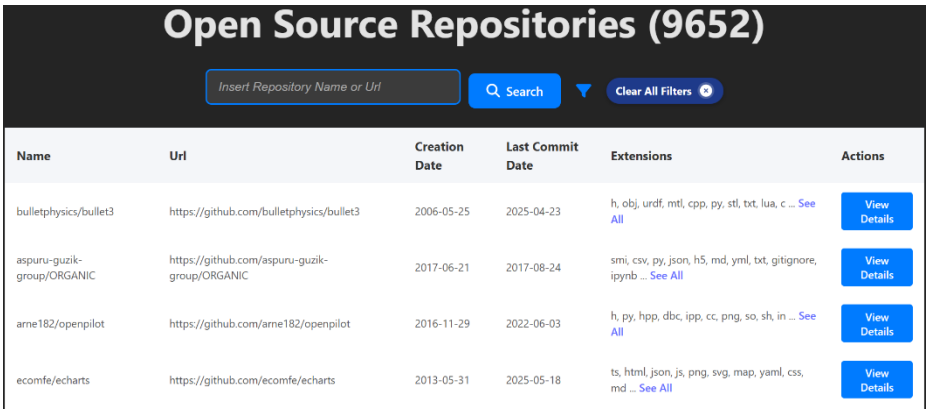

Fig. 2. Tool screenshot.

4 Case Study

To evaluate the proposed approach and tool, we have performed an initial exploratory empirical study, that was designed and reported based on the linear analytics guidelines of Runeson et al. [12].

4.1 Research Questions

To study the technological skills from open-source projects and gain insights into the labor market, we have formulated and set the following research questions:
RQ$_1$: What are the dominant and emerging technological skills in open-source development over time?
RQ$_2$: What are the technological skills that tend to co-exist?
RQ$_3$: How do developers specialize?
RQ$_4$: Can we identify the lifecycle of each technology?

In **RQ**$_1$, we aim to identify which technologies have the largest footprint (dominant) and which have shown the most significant growth in recent years (emerging). Answering this provides a longitudinal perspective on the software development landscape, highlighting trends that are crucial for curriculum development, strategic tech-

[2] https://github.com/trending

nology adoption, and career planning. Software projects are rarely built with a single technology, the **RQ₂** question explores the concept of "*technological ecosystems*" or "*stacks*." By identifying which languages and tools frequently co-occur, we can understand the practical combinations of skills required for modern development, such as the relationship between frontend (e.g., JavaScript, TypeScript) and backend (e.g., Python, Go) technologies. **RQ₃** investigates the breadth of skills at the individual developer level. Understanding whether the OSS community is primarily composed of single-language specialists or multi-language generalists (polyglots) has direct implications for hiring strategies, team composition, and how developers should approach their skill development. Technologies follow a lifecycle of adoption, maturity, and eventual decline. **RQ₄** seeks to empirically map technologies to different stages of this lifecycle. By comparing the prevalence of languages in repositories from different time periods, we can identify potentially legacy technologies and distinguish them from those at the forefront of new project development.

4.2 Data Analysis

The analysis was performed in a subset of 100.000 GitHub projects, which were retrieved from the projects that had any type of action in GitHub (creation, commit, pull request, etc.) in these time frames (2016-2019, 2020-2022, 2023-onwards). Due to time constraints, we stopped the analysis in the 100.000 analyzed project, and we didn't explicitly used the trending repositories from GitHub in order to have a simple point of data. A custom Python script was developed, which is divided into two main stages: data preparation and a targeted analysis for each research question.

To answer RQ_1 (Language Trends), we performed a time-series analysis. Repositories were grouped by their creation year. For each year, we calculated the absolute number of new projects initiated for each of the top 15 most frequent programming languages (after filtering).

To answer RQ_2 (Common Skill Sets), we first identified the top 15 programming languages in our dataset. Using a transformer from the sklearn library, we constructed a binary matrix indicating the presence of these languages in each project. A co-occurrence matrix was then generated by calculating the dot product of this matrix with its transpose.

To answer RQ_3 (Developer Specialization), we analyzed the committer-contributions.csv dataset. We grouped the data by individual committer and for each committer, we counted the number of unique programming languages they contributed to.

To answer RQ_4 (Technology Lifecycles), we employed an era-based comparative analysis. We segmented the repositories into three distinct periods based on their creation year: 'Early' (pre-2008), 'Established' (2008-2019), and 'Modern' (2019+). For each era, we calculated the relative popularity of the top 15 languages, measured as their percentage share of all repositories created within that period. This normalization controls the overall growth in the number of projects over time.

4.3 Results

In this section, we present the findings from our analysis, addressing each research question in turn. The dataset and analyses related to this study were processed and stored on the RAISE platform; an open science platform developed as part of a European initiative. After the analyses is completed, a Persistent Identifier (PID)[3] is assigned to the corresponding experiment. This PID supports traceability, reproducibility, and long-term accessibility, in accordance with open science principles.

4.3.1 RQ1: Dominant and Emerging Skills Over Time

The analysis of technological skills over time, addressing RQ1, is presented in Figure 3. The figure plots the absolute number of new repositories created each year for the top 15 most frequent programming languages in our dataset.

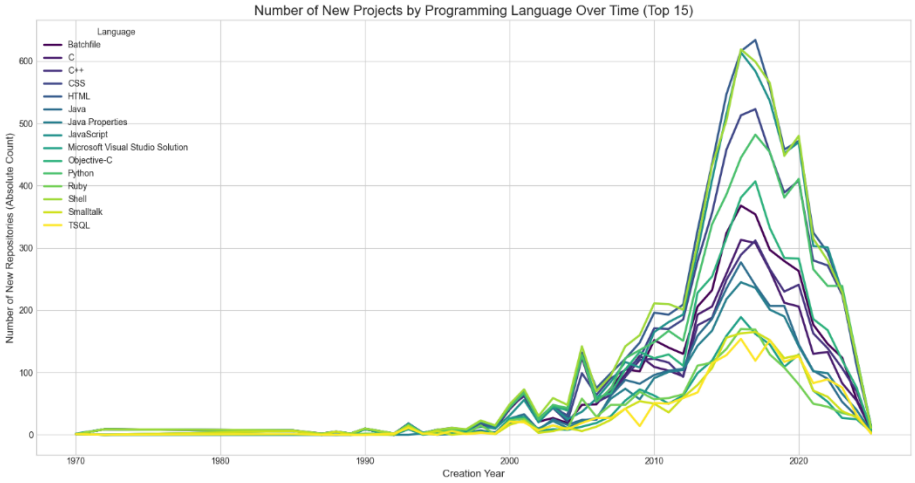

Fig. 3. Number of New Projects by Programming Language Over Time (Top 15)

Several key trends are immediately apparent from the figure. First, the period from roughly 2008 to 2020 marks a significant boom in open-source project creation across all major languages. The decline observed in the most recent years is likely an artifact of data collection lead times and does not necessarily indicate a downturn in activity.

A clear cluster of dominant technologies can be identified by the lines occupying the highest positions on the chart during the peak years. JavaScript, HTML, CSS, Shell, and Python consistently represent the largest volume of new projects. This finding underscores the centrality of web technologies (JavaScript, HTML, CSS) and versatile, high-level scripting languages (Python, Shell) in the modern open-source ecosystem.

While many languages show growth, Python exhibits a particularly steep trajectory, starting from a modest base in the early 2000s and rapidly ascending to become

[3] https://doi.org/21.T15999/raise/88

one of the most dominant languages by the mid-2010s. This pattern strongly characterizes it as a key emerging skill over the last decade. JavaScript also shows a meteoric rise, solidifying its position from an essential web tool to a cornerstone of software development.

Older, more established languages like C and C++ show steady, continuous growth but lack the explosive adoption rate of Python or JavaScript. Other languages like Objective-C display a distinct lifecycle, with a clear peak around the mid-2010s followed by a noticeable decline, likely correlated with the introduction and promotion of Swift for Apple ecosystem development. Similarly, languages like Ruby and Smalltalk appear to have peaked earlier and at a lower volume, suggesting they now occupy more niche roles compared to the dominant players.

4.3.2 RQ2: Common Technological Skill Sets

To answer RQ2, we analyzed the co-occurrence of technologies within the same projects. Figure 4 presents a heatmap of the co-occurrence frequencies for the top 15 languages. The diagonal shows the total number of projects for each language, while the off-diagonal cells show the number of projects where the corresponding pair of languages appears together.

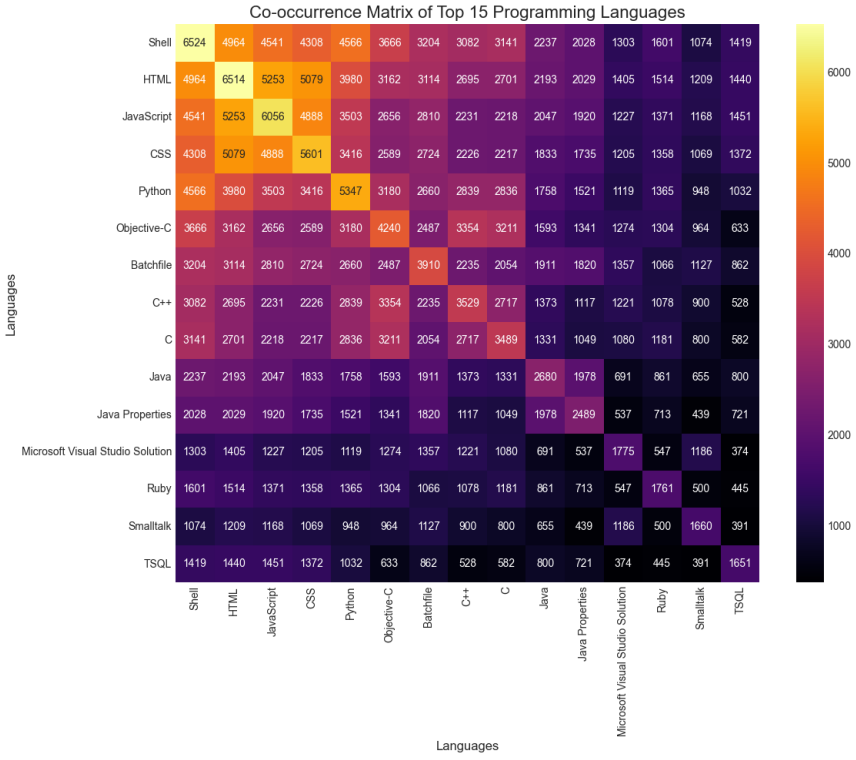

Fig. 4. Co-occurrence Matrix of Top 15 Programming Languages

The most prominent finding is an extremely strong and tightly coupled cluster of web technologies. The highest co-occurrence values on the matrix are between HTML and JavaScript (5,253 projects), HTML and CSS (5,079 projects), and JavaScript and CSS (4,888 projects). This confirms the existence of a foundational web development skill set, where proficiency in one of these technologies is highly likely to be paired with proficiency in the others.

Shell demonstrates a very high degree of co-occurrence with nearly all other dominant languages, particularly JavaScript (4,541), HTML (4,964), and Python (4,566). This suggests that Shell scripting is a ubiquitous "glue" technology, used for automation, build processes, and deployment across a wide variety of project types, from web applications to data science.

A strong pairing is evident between C and C++ (2,717 projects), reflecting their common use in systems-level programming where C libraries are often integrated into C++ projects. Python also shows a strong connection to HTML (3,980 projects), indicating its frequent use as a backend language for web applications (e.g., via frameworks like Django or Flask). Similarly, Java's co-occurrence with Java Properties (1,978) and HTML (2,193) points to its established role in enterprise and web application development.

4.3.3 RQ3: Developer Specialization

To address RQ3 and understand the extent of developer specialization, we analyzed the number of unique programming languages each committer contributed to across all their projects in the dataset. Our findings strongly indicate that specialization, rather than polyglotism, is the predominant pattern in the open-source community. The core of this finding is presented in the following figure, which presents the distribution of language counts per committer.

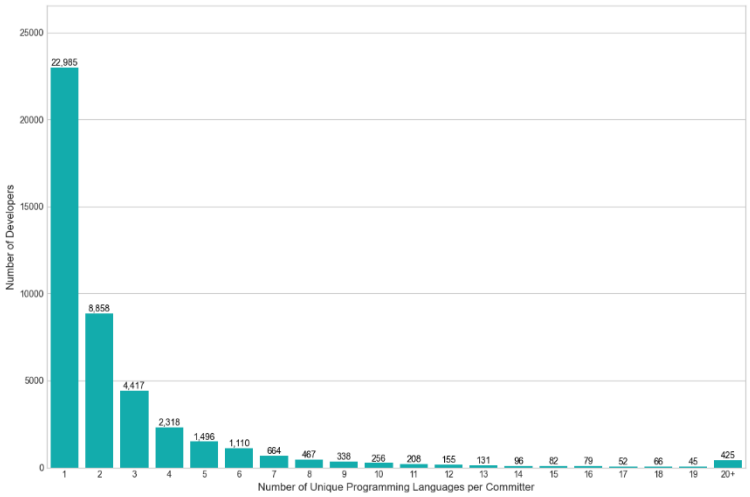

Fig. 5. Distribution of Developer Specialization

The most striking result from this table is that the median (50th percentile) number of languages per developer is just one. This indicates that half of all developers in our dataset have only contributed to projects using a single programming language, classifying them as specialists. Furthermore, the 75th percentile is three, meaning that three-quarters of the entire developer population contributes to three or fewer languages. This reinforces the conclusion that most developers focus their efforts on a narrow set of programming languages.

A detailed analysis of the full distribution further supports this observation. We found a classic long-tail distribution: the number of developers who contribute to only one programming language is the largest group by a significant margin, numbering in the tens of thousands. This is followed by a steep drop-off. The number of committers contributing to two languages is the next largest group, and the count continues to decrease sharply as the number of languages increases.

Conversely, the number of true "polyglot" developers—those contributing to a wide array of technologies (e.g., 10 or more)—is exceptionally small, representing a tiny fraction of the overall committer population. While these highly versatile individuals exist, they are clear outliers.

4.3.4 RQ4: Technology Lifecycles

Our final research question sought to identify the lifecycle of technologies by analyzing their relative popularity across different eras. For this analysis, we categorized projects into three distinct periods: 'Early' (created before 2008), 'Established' (2008-2019), and 'Modern' (2019 and later). Figure 6 shows the share of repositories in each of the top 15 languages claimed within each era. This normalization allows us to observe shifts in technological focus, independent of the overall growth of open source.

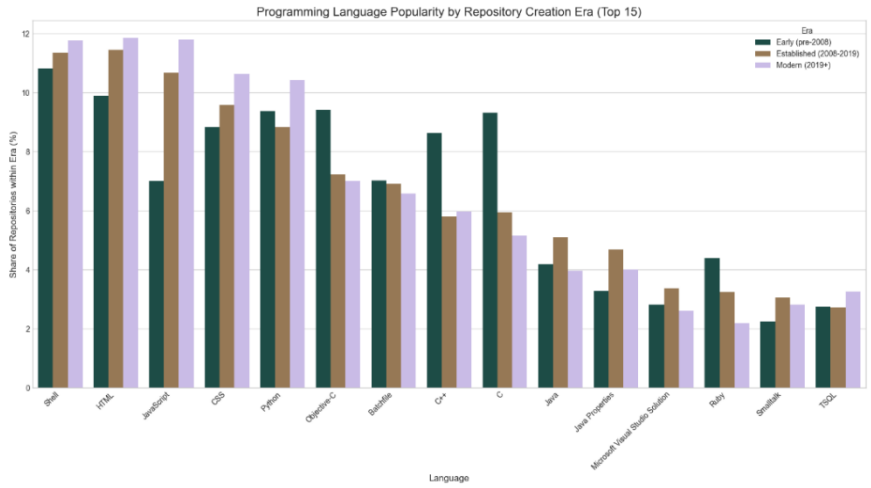

Fig. 6. Programming Language Popularity by Repository Creation Era (Top 15).

The analysis identifies three primary trajectories. First, Ascendant Technologies like Python, JavaScript, HTML, and CSS show increasing dominance, with their highest relative share of new projects occurring in the 'Modern' era. This highlights the consolidation of the web stack and the rise of high-level languages for general-purpose development.

Second, Mature and Legacy Technologies exhibit declining shares of new projects. C and C++, foundational in the 'Early' era, have seen a relatively steady decline. The contraction is more pronounced for Objective-C and Ruby, which peaked in the 'Established' era; Objective-C's decline directly reflects its replacement by Swift. Java also presents as a mature technology, with a modest decrease from its peak in the 'Established' period, though it remains an enterprise standard.

Third, a few Stable Technologies, most notably Shell, maintain a consistent share across all eras. This underscores Shell's enduring role as a foundational tool for automation and system administration, independent of primary development trends.

In summary, our findings affirm that technology lifecycles are observable through shifts in relative adoption. The data clearly indicates a long-term pivot in open-source development from traditional systems languages toward high-level dynamic languages and web technologies.

5 Threats to Validity

In line with established guidelines for empirical software engineering research, we acknowledge several potential threats to the validity of our study. We categorize them into construct, internal, and external validity.

Construct Validity. Construct validity concerns the extent to which our study measures what it claims to be measuring. Our primary threat is the use of file presence (via extensions) as a proxy for technological skills. Our methodology can confirm that a technology like Python is present in a project, but it cannot measure the depth of expertise of a contributor. A project with a single, trivial script is treated the same as one with a complex, multi-module system. Similarly, a contributor making a one-line fix is counted the same as a core architect. We mitigate this threat by operating at a massive scale, assuming that these nuances average out and that the overall trends reflect the collective focus of the community. However, our findings should be interpreted as measures of technology adoption and contribution frequency, not proficiency. Moreover, our analysis relies entirely on a languages.yml mapping file from GitHub's Linguist project. While this is the industry standard and the most comprehensive resource of its kind, it is not infallible. Some file extensions can be ambiguous, and the mapping may not be perfectly up to date with the very latest emerging technologies. By using the same tool that GitHub uses for its own language statistics, we ensure our results are consistent with the platform's native analysis, which we consider a strong mitigation.

Internal Validity. Internal validity relates to confounding factors that could influence our results. For RQ4, we defined three eras ('Early', 'Established', 'Modern') with specific year boundaries. These boundaries are inherently arbitrary. Shifting the

boundaries by a year or two could slightly alter the percentage shares for each language. We mitigated this by choosing boundaries that correspond to known shifts in the industry (e.g., the pre-2008 era representing the early days of Git and GitHub, the post-2018 era representing the modern cloud-native/AI boom). The broad trends we observed—such as the decline of C and the rise of Python—were robust enough to be visible regardless of minor boundary shifts. Moreover, the time-series plot for RQ1 shows a sharp decline in project creation in the most recent years. This is highly likely to be an artifact of the data collection process (e.g., the data dump was created at a specific point in time and is incomplete for the latest period) rather than a true decline in open-source activity. We acknowledge this explicitly and caution against interpreting the rightmost edge of the graph as a real-world trend.

External Validity. External validity concerns the generalizability of our findings. Our study is based exclusively on public repositories on GitHub. This ecosystem heavily over-represents open-source projects, web technologies, and scripting languages, while heavily under-representing closed-source, enterprise software (e.g., COBOL, SAP ABAP, proprietary .NET systems) and certain embedded systems development. Therefore, our conclusions are generalizable to the open-source community on GitHub, which is a massive and influential part of the industry, but they cannot be directly generalized to all software development worldwide. Furthermore, our dataset captures projects and contributions that exist now. Projects that were created and later deleted from GitHub are not included. Similarly, developers who have deleted their accounts would be absent. This introduces a survivorship bias, although its impact on broad technological trends is likely to be minimal.

6 Conclusions

This paper sets out to address the critical challenge of identifying and tracking technological skills in a rapidly evolving software development landscape. Traditional methods for skills analysis are often limited in scope or rely on subjective data, failing to capture the full dynamism of the industry. To overcome these limitations, we proposed a novel and scalable methodology for analyzing technological trends across the entirety of a git-based open-source ecosystem like GitHub.

Our primary contribution is a methodology, implemented as a robust analysis tool, that identifies technologies by directly parsing the file extensions within a big set of repositories. This data-driven approach moves beyond metadata to provide an objective, fine-grained, and comprehensive view of technology adoption. Applying this tool to a large-scale dataset, we conducted an empirical study that yielded significant insights into the open-source world.

The implications of these findings are significant for multiple stakeholders. For educators, they provide a data-driven mandate to prioritize curricula around Python, modern web development, and essential automation skills with Shell. For developers, our results affirm that specialization is a common and viable career strategy, and they highlight the specific technology ecosystems that are currently most relevant. For hiring managers, these insights can help shape realistic job descriptions and team-

building strategies, emphasizing the need for specialists who can form a cohesive, stack-oriented team.

Acknowledgments. The work presented here is part of the project "SKILLAB: Monitoring the Demand and Supply of Skills in the European Labour Market" and received funding from the European Union's Horizon 2023 research and innovation programme under grant agreement No 101132663.

References

1. Frey, C. B., & Osborne, M. A. (2017). The future of employment: How susceptible are jobs to computerisation?. Technological forecasting and social change, 114, 254-280.
2. Acemoglu, D., & Autor, D. (2011). Skills, tasks and technologies: Implications for employment and earnings. In Handbook of labor economics (Vol. 4, pp. 1043-1171). Elsevier.
3. Hassan, A. E. (2008, September). The road ahead for mining software repositories. In 2008 frontiers of software maintenance (pp. 48-57). IEEE.
4. Santos, A., Souza, M., Oliveira, J., & Figueiredo, E. (2018, September). Mining software repositories to identify library experts. In Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse (pp. 83-91).
5. Hauff, C., & Gousios, G. (2015, May). Matching GitHub developer profiles to job advertisements. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (pp. 362-366). IEEE.
6. Greene, G. J., & Fischer, B. (2016, August). Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In Proceedings of the 31st IEEE/ACM international conference on automated software engineering (pp. 804-809).
7. GM Lopes, J., Alves, J., & Figueiredo, E. (2022, October). EXTRACTPRO: A Data Mining Tool for Developer Profile Generation based on Source Code Analysis. In Proceedings of the XXXVI Brazilian Symposium on Software Engineering (pp. 112-117).
8. Montandon, J. E., & Valente, M. T. (2022, July). Mining the technical skills of open source developers. In Concurso de Teses e Dissertações (CTD) (pp. 1-10). SBC.
9. Kourtzanidis, S., Chatzigeorgiou, A., & Ampatzoglou, A. (2020, December). RepoSkillMiner: identifying software expertise from GitHub repositories using natural language processing. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (pp. 1353-1357).
10. Saxena, R., & Pedanekar, N. (2017, February). I know what you coded last summer: Mining candidate expertise from github repositories. In Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (pp. 299-302).
11. Atzberger, D., Scordialo, N., Cech, T., Scheibel, W., Trapp, M., & Döllner, J. (2022, October). CodeCV: Mining expertise of GitHub users from coding activities. In 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 143-147). IEEE.
12. Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). Case study research in software engineering: Guidelines and examples. John Wiley & Sons.