Do programming languages used in certain domains have an impact on bugs?

Hussain Abuwala
Cheriton School of Computer Science
University of Waterloo
hsabuwal@uwaterloo.ca

Jun Qing Lim
Cheriton School of Computer Science
University of Waterloo
jq3lim@uwaterloo.ca

Jumana
Cheriton School of Computer Science
University of Waterloo
jjumana@uwaterloo.ca

***Abstract*—In this paper, we explain the study we performed to investigate the sole influence of a programming language in a certain domain with the triggering of bug related commits. From the 5 domains, we find that JavaScript is the most popular choice in both front-end and back-end domains, with C++, Java and Swift being the most popular in the game, mobile and desktop domains, respectively. While calculating the median bug ratio of languages in various domains, we find that TypeScript is most bug prone in frontend and mobile domains, whereas, JavaScript, C++ and C are most bug prone for backend, game and desktop domains. Furthermore, TypeScript, JavaScript, C, C++ and Python are in the top 3 of highest median bug ratios for at least 2 of the 5 domains studied. When comparing the domains, desktop, game and mobile domains have the highest median bug ratios compared to front-end and back-end domains. Finally, in terms of security vulnerabilities, we find that C and C++ have the top 2 most security vulnerabilities. Hence, although there are other factors like complexity of a project that affects overall code quality, from the results of our study focusing solely on programming languages, we believe that the choice of programming languages in different domains can have an impact on the likelihood of introducing bugs in software development.**

***Index Terms*—programming languages, bug ratio, domains, security vulnerabilities, software development**

I. Introduction

Software developers and programmers in any software project are forced to invest a lot of time in detecting and fixing bugs. Much research has been conducted on finding the cause of bugs. Some research specifically focuses on certain parameters of a project like complexity, size, number of developers, project type, etc. to identify the influence of that parameter. Also, machine learning based and NLP based tools have been generated for the automatic detection of bugs in projects but yet this issue is still a challenge till this date as software development is evolving at a faster pace with time. One such paper which mainly focused on programming languages grabbed our attention [2].

For our study, we selected projects based on domains and then further filtered out the programming languages most popular in these domains. We understand that other factors might have greater influence on a bug's existence, and many may debate that bugs are not due to a language but the programmer writing the code. Though we absolutely agree with that, we still cannot neglect that programming languages are also prone to having errors. Also, even though programmers are responsible for opening a bug, we set out to study which language is prompting programmers in writing buggy codes in a certain domain and find through our analysis that the choice of a programming language in a domain does prompt bugs in a software development.

A. Motivation

Does a programming language affect the process of bug fixing? Will using one programming language over the other yield less bugs? Various speculations surrounding the suitability of a particular programming language for software development and maintenance is still a debatable concern. Although some of these arguments are intense, many people believe that the choice of programming language can affect the features of a software and how a software is developed and maintained.

At the time of this study, as we have access to WoC server, we create a data set utilizing the GitHub Repositories and the World of Code (WoC) infrastructure [1], containing 173M git repositories (with over 3.1B commits, 12.6B trees and 12.5B blobs) to explore and understand the relationships between programming languages and bugs.

We believe our findings might help developers in understanding the impacts or influences of a programming language in software maintenance and reliability. Also, it would allow them to choose the programming language for their project's domain by understanding the results we find in our study.

B. Research Questions

We begin our research in light to investigate ourselves the following research questions:

- **RQ1:** What are the popular programming languages used in each domain?
- **RQ2:** Which programming languages are more prone to bug commits in each domain?
- **RQ3:** In terms of domains, which domain leads to applications being more bug prone?
- **RQ4:** Which programming languages are more prone to security vulnerabilities?
- **RQ5:** Can the choice of programming language assist software developers in writing codes which are less prone to bugs?

The following parts of this paper is divided into sections. Section II describes our methodology of analysis, data collection and the tools used. Section III elaborates on the results we find through answering the research questions in Section I. Then, we discuss a little more in the discussion section IV, followed by the the related works in Section V. Section VI states the threats of validity of our study, and finally we conclude with a conclusion and a future work in sections VII and VIII.

II. Methodology

In this section, we describe the methodology used in our research in collecting the data. It consists of four steps 1) Programming Languages Collection, 2) Projects Collection, 3) Project Related Data Collection and 4) Security Data Collection.

A. Programming Languages Collection

Our methodology starts by collecting the top 10 programming languages in each project domain. We set our project domain as follow:

1) **frontend**: frontend-based projects
2) **backend**: backend-based projects
3) **game**: game-based projects
4) **mobile**: mobile-based projects
5) **desktop**: desktop-based projects

TABLE I: Topic Keywords

Domain	Topic Keywords
frontend	frontend
backend	backend
game	game, gamedev, game-development, game-engine
mobile	mobile, mobile-app, android, ios
desktop	macos, windows, linux

To gather the latest data of programming languages by popularity, we use GitHub API to query repositories with the domain topic keywords [5], as illustrated in Table I, giving us the list of repositories of the corresponding domain. Note that the term "project" and "repository" is used interchangeably here. Each project has a primary language tag associated with it that represents the dominant programming language for that project. We compute the frequency of each language assoicated with a project for each domain and choose the top-10 languages. The Table II illustrates the languages gathered.

Except for desktop domain (9 languages) and frontend domain (6 languages), for all the other domains, we collected top 10 languages. We excluded some languages for desktop and frontend because the frequency / popularity of those languages were relatively lower.

B. Projects Collection

Once the programming languages for each domain is collected, the list of top 100 projects is extracted through the following steps, depicted in Figure 1:

TABLE II: Programming Languages

Domain	Top 10 Programming Languages
backend	JavaScript, TypeScript, Python, PHP, Go, Java, Kotlin, C#, Rust, Ruby
frontend	JavaScript, TypeScript, Vue, CSS, HTML, Python
desktop	Swift, C++, Python, C, C#, Go, Rust, Objective-C, Dart
mobile	Java, Swift, JavaScript, Dart, TypeScript, Kotlin, Objective-C, C++, C#, Python
game	C++, C#, JavaScript, C, Java, Python, TypeScript, Go, Lua, Swift

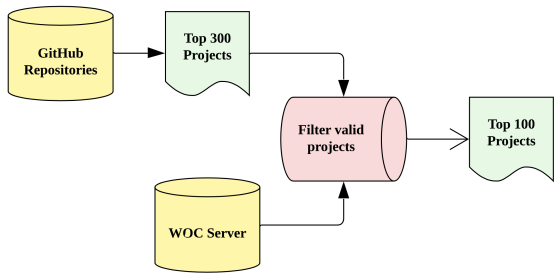

Fig. 1: Projects Collection

*1) **Collecting the top 300 projects of each language**:* To collect the top projects for each programming language, keyword-based topics are used to extract the list of projects through GitHub API (of repository search). Each domain of the programming languages uses the following keywords as query of topics in obtaining the list of repositories (projects), illustrated in Table I.

The list of repositories returned by the GitHub API are sorted in descending order of stars, metric for popularity of a project. The top 300 projects that are the most popular are then collected for the next step.

*2) **Filtering to obtain valid top 100 projects for each language**:* This step functions to filter valid top 100 projects, as some projects queried from GitHub API do not exist in the WoC server; as such this step is taken to ensure the validity of the existence of the projects and data consistency. Each project name is looked up over the WoC server and if the project does not have any commits such that the project (name) is deemed to be invalid, it is discarded and filtered out.

C. Project Related Data Collection

Once the top 100 projects for each programming language in each domain are extracted in the previous phase, the projects at this instance are considered to be valid. These projects are then further looked up over the WoC server to extract the following seven data fields, as shown in Figure 2:

1) **Total number of commits**
2) **Total number of authors**
3) **Total number of files**

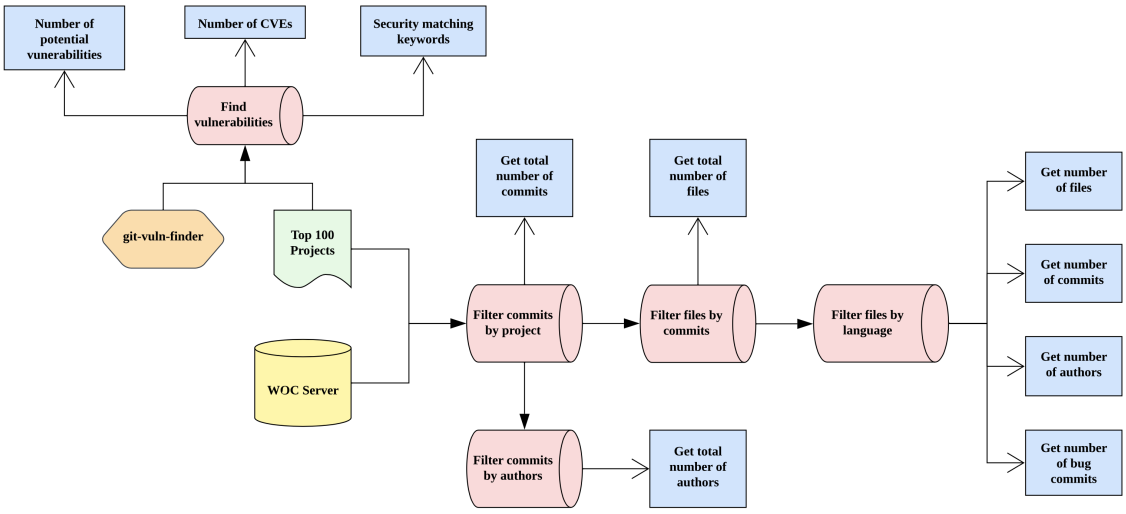

Fig. 2: Data + Security Data Collection

4) **Number of files (with language filtered)**
5) **Number of commits (with language filtered)**
6) **Number of authors (with language filtered)**
7) **Number of bug commits (with language filtered)**

The phrase 'with language filtered' refers to filtering the commits/files/authors that only include files of the corresponding programming language extension. For instance, the number of files in a project could be 10 and 6 of them are files with JavaScript extension; as such, the total number of files here is 10 and number of files (with language filtered) is 6. The list of programming language extensions are filtered based on Linguist's list of languages extensions [6], which is a popular and credible source of programming languages.

For identifying the bug commits, we use a keyword based approach on the commit messages, similar to the authors in [2]. In particular, we used the keywords "error", "bug", "fix", "issue", "mistake", "incorrect", "fault", "defect" and "flaw".

D. Security Data Collection

Our research also collects the software vulnerabilities information from the projects. For this, we use git-vuln-finder [7], which is an open-source security tool used to identify known and potential vulnerabilities in a Git repository. It analyzes a Git repository's history (of git commit messages) and identifies security vulnerabilities by scanning known patterns and code snippets that are associated with security issues. It detects a range of vulnerabilities, including those related to buffer overflows, command injection, SQL injection and more. The tools uses techniques from prior research by Barish, Michelson & Minton's work [8] and Zhou & Sharma's work [9] in mining security issues from git commit messages.

As such, we use this tool to further collect three security data fields from each project as follows, as illustrated in Figure 2:

1) **Number of potential vulnerabilities**: number of issues that could potentially be vulnerable (or prone to have security issue).
2) **Number of CVEs**: number of publicly disclosed security vulnerabilities.
3) **Keywords**: list of keywords that matched the different vulnerability patterns used by the tool.

III. Results

We would like to explain our findings through answering the research questions in Section I.

RQ1: What are the popular programming languages used in each domain?

Table II shows the top programming languages being used in each domain in terms of the frequency of their usage in the popular projects in each domain. For the front-end, the top 3 programming languages used are JavaScript, TypeScript and Vue. For the back-end, top 2 stays the same as front-end but the third most popular language is Python. For game domain, the top 3 programming languages are C++, C and JavaScript. For mobile, the top 3 programming languages are Java, Swift and JavaScript. Finally, for desktop, top 3 programming languages are Swift, C++ and C.

RQ2: Which programming languages are more prone to bug commits in each domain?

For measuring which languages are more prone to bug commits, we use a metric called "Bug Ratio", which is simply the number of bug commits (language filtered) divided by the number of commits (language filtered). The reason for using a ratio is to reduce the influence of the size of a project to have on the number of bug commits for a particular language. Using the raw value of number of bug commits (language filtered)

could possibly lead to projects with more overall commits to potentially have more bug commits. To visualize the data, we use box plots as its useful for visualizing and summarizing the distribution of a dataset, especially when comparing multiple groups, which in this case are the programming languages. For ease of comparing, all the box plots shown are sorted in descending order from left to right in terms of the median bug ratio. We sort and use the median of the bug ratio for each language rather than using the mean bug ratio for each language. This is mainly because our data is skewed. Although we collected top 100 projects for each language, the projects themselves vary in size and complexity. Hence, using mean can be affected by outliers and would not represent the center of the distribution well. Finally, we also use the median of the metric "the number of commits (language filtered)" to see if languages which have a higher median number of commits also have a higher median bug ratio.

Figure 3 shows the box plot for the bug ratio of programming languages in the front-end domain. TypeScript has the highest median bug ratio followed by JavaScript and Vue which have similar median bug ratios. On the other hand, Python, CSS and HTML have relatively lower bug ratio compared to the top 3 where HTML has the lowest median bug ratio among them. When we looked at the median number of commits for each language in the front-end domain, we found out that JavaScript has a much higher median value of 170 compared to TypeScript, which has 112 median number of commits. Despite this, TypeScript has a higher median bug ratio compared to JavaScript.

Figure 4 shows the box plot for the bug ratio of programming languages in the back-end domain. JavaScript has the highest median bug ratio followed by Python, Typescript, Go and PHP which have similar median bug ratios. On the other hand, Rust, Ruby, Java, C and Kotlin have relatively lower median bug ratios where Kotlin has the lowest median bug ratio. JavaScript and Python have the highest median number of commits (87 and 78 respectively), which is a possible reason for them being top 2 in median bug ratio. However, TypeScript is 6th in median number of commits with 44.5, still it has the 3rd highest median bug ratio.

Figure 5 shows the box plot for the bug ratio of programming languages in the game domain. C++ has the highest median bug ratio followed by C and C#. The ranking holds true also in terms of the median number of commits where C++, C and C# have the top 3 highest median number of commits with 3610, 495.5 and 443.5 commits respectively. On the other hand, Swift has the lowest median bug ratio among all.

Figure 6 shows the box plot for the bug ratio of programming languages in the mobile domain. TypeScript has the highest median bug ratio and Dart, Swift and Kotlin has the lowest median bug ratios. Java has by far the highest median number of commits (444.5), but has the 4th highest median bug ratio, whereas TypeScript is 8th in median number of commits (121), but is still has the highest median bug ratio. Although, Swift and Kotlin have the 2nd and 3rd highest median number of commits with 299 and 282 commits respectively, they have the 2 lowest median bug ratios among all.

Figure 7 shows the box plot for the bug ratio of programming languages in the desktop domain. C has the highest median bug ratio followed by Go, Python, C++ and C# which also have similar median bug ratios. Swift and Dart have the lowest median bug ratios. Although C++ has the highest median number of commits (1966), it has the 4th highest median bug ratio. Similarly, Swift has the 5th highest median number of commits (233), but has the 2nd lowest median bug ratio.

Finally, Figure 8 shows the box plot for the bug ratio of programming languages when all the domains are combined. C and C++ have the top 2 highest median bug ratio and they also have the top 2 highest median number of commits of 729 and 1319 respectively. Even though TypeScript has the 12th highest median number of commits (93), it has the 3rd highest median bug ratio. Swift and Kotlin have much lower median bug ratios even though they are among the higher ranked languages in terms of the median number of commits.

RQ3: In terms of domains, which domain leads to applications being more bug prone?

Figure 9 shows the box plot for bug ratio of across different domains. Desktop, mobile and game domains have the 3 highest median bug ratios compared to front-end and backend which have the 2 lowest median bug ratios. This ranking also holds true in terms of the median number of commits where desktop has the highest number of commits (289.0) and back-end has the lowest number of median commits (39).

RQ4: Which programming languages are more prone to security vulnerabilities?

Figure 10 shows the bar chart for median vulnerability ratio for various programming languages. For measuring security vulnerabilities, we use the metric median vulnerability ratio, which is the number of potential vulnerable commits divided by the total number of commits. After finding the vulnerability ratio for each language associated project, we take the median. C and C++ have the top 2 highest median vulnerability ratio and they also have the top 2 highest median number of total commits. Languages like Rust, Lua and Go have the 10th, 13th and 14th highest median number of total commits but they have the 3rd, 4th and 5th highest median vulnerability ratios.

RQ5: Can the choice of programming language assist software developers in writing code that is less prone to bugs?

The choice of programming language can have an impact on the likelihood of introducing bugs in software development. Different programming languages have different features and

characteristics that affect the ability of developers to write code that is less prone to errors. For example, from our results, it can be seen that modern languages like Swift Kotlin are lesser prone to bugs, possibly due to having features like static typing, NULL safety and modern syntax. On the other hand, languages such as C, C++, Python and JavaScript are more bug prone due to their own unique features.

However, it is important to note that while the choice of programming language can certainly impact the likelihood of introducing bugs, it is not the only factor. The skills and practices of individual developers, as well as the overall development process and tools used, also play a significant role in producing high-quality, bug-free code. Adherence to best practices and coding standards, such as proper naming conventions and consistent formatting, can help reduce the likelihood of introducing bugs. Finally, factors such as testing, well-design software architecture, code reviews, debugging tools and continuous integration and deployment are all important in assisting software developers in writing code that is less prone to bugs.

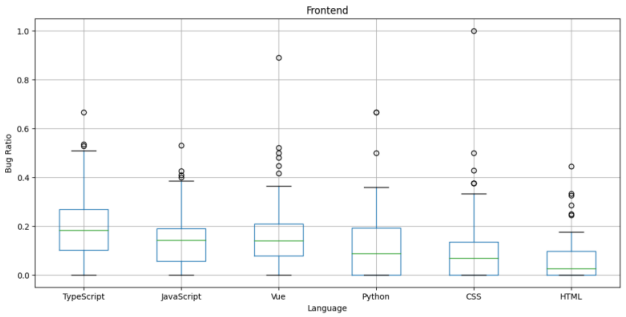

Fig. 3: Bug Ratio for Frontend

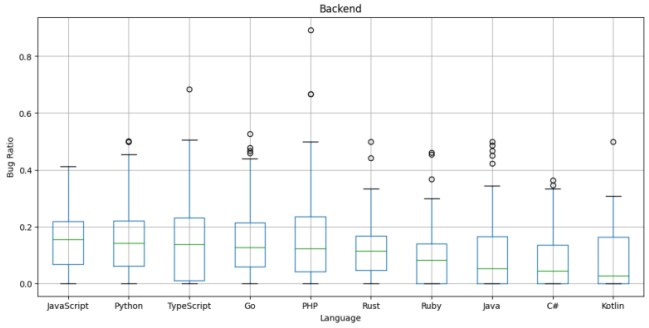

Fig. 4: Bug Ratio for Backend

IV. Discussion

Languages such as Swift and Kotlin tend to have a lower median bug ratio in respective domains as well as overall across all domains. This can be attributed to multiple factors. Both are relatively modern languages introduced in 2014 (Swift) and 2016 (Kotlin) respectively. Both Swift and Kotlin were released to address the shortcomings of their

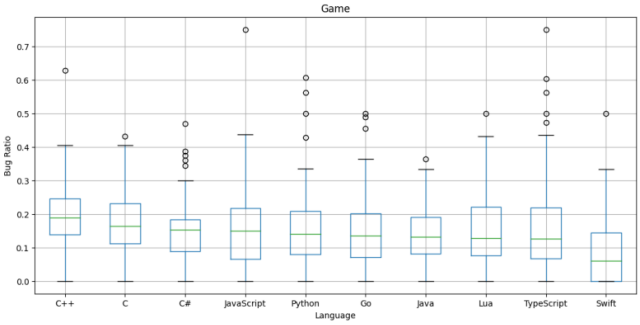

Fig. 5: Bug Ratio for Game

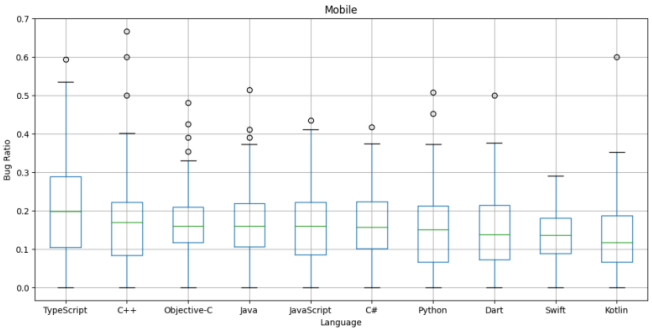

Fig. 6: Bug Ratio for Mobile

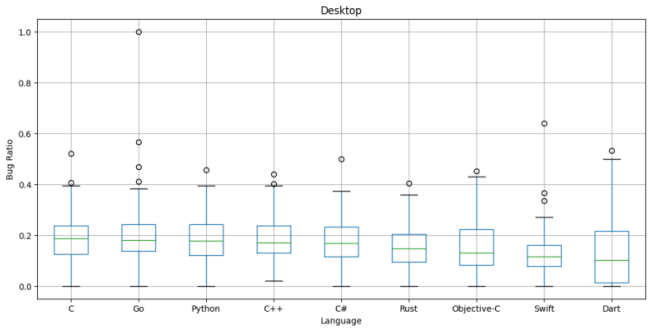

Fig. 7: Bug Ratio for Desktop

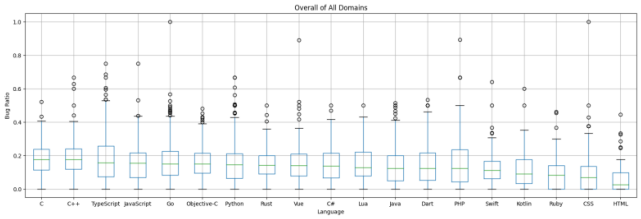

Fig. 8: Overview on Overall Domain

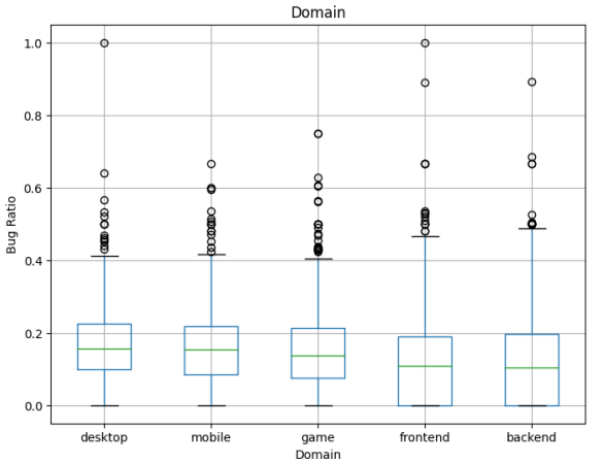

Fig. 9: Bug Ratio for ALL Domains

predecessors Objective-C and Java respectively. They are statically typed languages where variables are checked at compile time reducing the likelihood of errors caused by type mismatches or invalid assignments. Also, they have features such as Null Safety built in to prevent null pointer exceptions. Being relatively modern, they have modern syntax which make them more concise and readable which can also reduce the likelihood of errors.

TypeScript came as a big surprise as it had one of the higher median bug ratios. It was released in 2012 by Microsoft as a superset of JavaScript to address some of its shortcomings such as its lack of static typing, which can lead to errors in larger codebases. It also includes features such as classes, interfaces, and enums, which make it easier to write complex and maintainable code. One possible reason for its higher median bug ratio could be that the developers are using “any” keyword as a type. This effectively turns off the type checking that TypeScript provides and defeats the purpose of using TypeScript in the first place.

Other than TypeScript, languages such as C, C++, Python and JavaScript are in the top 3 of highest median bug ratios for at least 2 of the 5 domains studied. C and C++ both require manual memory management which can be error prone and lead to issues such as buffer overflows, null pointer dereferences etc. Both also allow for pointer arithmetic that can also lead to other issues. Furthermore both are low level languages that lack high-level abstractions, which can make it difficult to write safe and maintainable code. For JavaScript and Python, both are dynamically typed languages, which means that type errors may not be caught until runtime. JavaScript is also weakly typed which can lead to unexpected type conversions and errors.

In terms of bug ratios of languages across domains, it seems that regardless of the domains, TypeScript, JavaScript, Python, C and C++ are in the top 3 of highest median bug ratios for at least 2 of the 5 domains studied. In terms of median bug ratios for each domain (Figure 9), regardless of the language, the desktop, game and mobile domain have the highest median bug ratios compared to front-end and back-end domains. The reason for this can be applications in these domains need to be developed to run on multiple operating systems and each platform has its own unique features and API, which can make development more complex.

In terms of security vulnerabilities, C and C++ have the top 2 highest median vulnerability ratio. The main reason is the lack of built-in security features. Unlike newer programming languages like Java or Python, C and C++ do not have built-in security features such as memory safety mechanisms, bounds checking, or automatic garbage collection. This puts the burden of ensuring program security on the programmer, who must manually check and validate input, manage memory, and prevent vulnerabilities such as buffer overflow.

V. Related Work

Multiple research has been conducted to investigate the impacts of programming language choices and bugs in software development. During our study, we came across the following related works:

The paper ”How often do single-statement bugs occur? the Manysstubs4J dataset” [4] presents a new dataset called Manysstubs4J, which consists of single-statement bugs mined from open-source Java projects and determines how often single-statement bugs occur in software projects and identifies characteristics of these bugs. Another paper by Zhu et al. [3] investigates how developers fix their own simple bugs compared to how they fix bugs reported by others.

Ray et al [2] presents a study of the relationship between programming languages and code quality in open source projects hosted on GitHub. This paper was very insightful and mainly led to our interest of exploring the recent popular projects’ evolution in terms of programming languages used in specific domains and the bugs found. This paper’s related work also led us to more similar researches done like controlled experiments on undergraduate students in developing compilers, influence of programming languages in developers’ effort, comparison between C and C++ and more [10]–[16]. Our work is different the paper [2] in terms of scalability (i.e. analyzing above 4000 projects and above 10 million commits) and mainly in terms of the approach being taken from domain.

VI. Threats to Validity

Firstly, we did not filter out duplicate projects across domains during data collection. As our dataset contains only 143 duplicate projects and our analysis is done domain-wise, this threat is mitigated. Secondly, we did not consider other aspects like the age, size, complexity, the skill sets of the developers, etc. As we can certainly say that these factors have significant impacts on the triggering of bugs, we deliberately overlook them and just focus on one aspect that is the programming language to understand its sole influence on bugs. Finally, for identifying bug fix commits related to a specific language, we assumed that if a bug fix commit has files of a particular programming language, then that bug fix commit is counted towards that programming language. However, it can be the

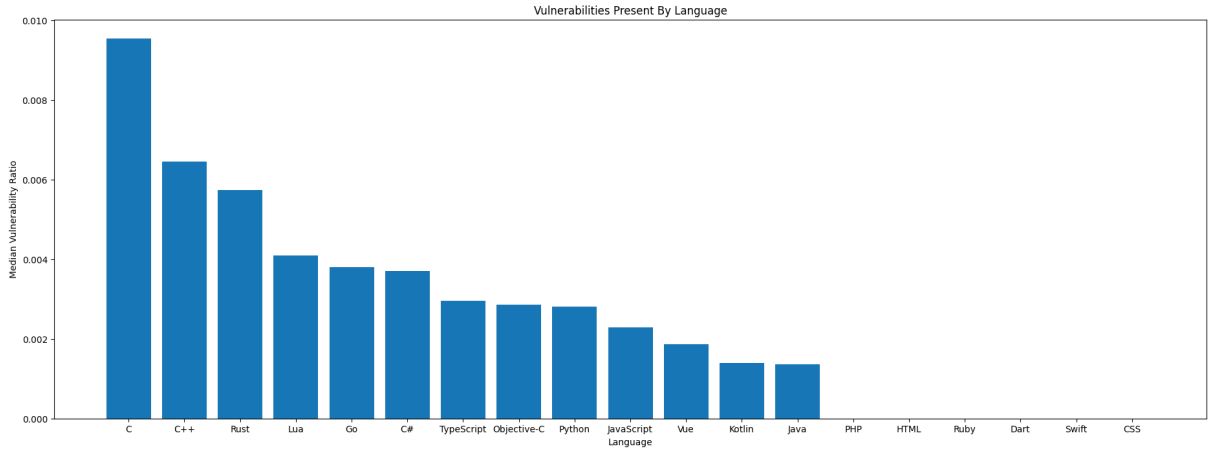

Fig. 10: General Overview

case that the bug fix commit can be associated with some other programming language. This can lead to an overestimation.

VII. Conclusion

To conclude, we conducted a study to understand whether programming languages of a specific domain is related to bugs triggering in the projects. We collected data from GitHub repositories and used WoC to find the required information for each of the projects. We started with domain specific languages selecting popular projects within them from GitHub. We find that JavaScript is the most popular choice in both front-end and back-end domains, while C++, Java and Swift are the most popular in game, mobile and desktop development. TypeScript leads with the bug ratio in frontend and mobile domains, while JavaScript, C++ and Swift lead in backend, game and desktop. Finally, C and C++ have the most security vulnerabilities. Hence, although there are other factors like complexity of a project that affects overall code quality, from the results of our study, we believe that the choice of programming languages in different domains can have an impact on the likelihood of introducing bugs in software development.

VIII. Future Work

Future work should consider the scale of the project when comparing languages. By scale, we mean the size of the project, number of users (more users will find more bugs), number of platforms the project needs to support, etc. If we can control for scale, then comparing projects of different programming languages is much fairer.

References

[1] Ma, Yuxing, et al. "World of code: an infrastructure for mining the universe of open source VCS data." 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019.

[2] Ray, B., Posnett, D., Filkov, V., Devanbu, P. (2014). A large scale study of programming languages and code quality in github. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. https://doi.org/10.1145/2635868.2635922

[3] Zhu, Wenhan, and Michael W. Godfrey. "Mea culpa: How developers fix their own simple bugs differently from other developers." 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, 2021.

[4] Karampatsis, Rafael-Michael, and Charles Sutton. "How often do single-statement bugs occur? the manysstubs4j dataset." Proceedings of the 17th International Conference on Mining Software Repositories. 2020.

[5] Searching for repositories. GitHub Docs. (n.d.). Retrieved March 2023, from https://docs.github.com/en/search-github/searching-on-github/searching-for-repositories#search-by-topic

[6] Github. (n.d.). Linguist/languages.yml at master · github/linguist. GitHub. Retrieved March 2023, from https://github.com/github/linguist/blob/master/lib/linguist/languages.yml

[7] Cve-Search. (n.d.). CVE-search/Git-VULN-Finder: Finding potential software vulnerabilities from Git commit messages. GitHub. Retrieved March 2023, from https://github.com/cve-search/git-vuln-finder

[8] Barish, Greg, Matthew Michelson, and Steven Minton. "Mining commit log messages to identify risky code." Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2017.

[9] Zhou, Yaqin, and Asankhaya Sharma. "Automated identification of security issues from commit messages and bug reports." Proceedings of the 2017 11th joint meeting on foundations of software engineering. 2017.

[10] P. Bhattacharya and I. Neamtiu. Assessing programming language impact on development and maintenance: A study on c and c++. In Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pages 171–180, New York, NY, USA, 2011. ACM.

[11] J. Armstrong, R. Virding, C. Wikström, and M. Williams. Concurrent programming in erlang. 1993.

[12] S. Hanenberg. An experiment about static and dynamic type systems: Doubts about the positive impact of static type systems on development time. In Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '10, pages 22–35, New York, NY, USA, 2010. ACM.

[13] R. Harrison, L. Smaraweera, M. Dobie, and P. Lewis. Comparing programming paradigms: an evaluation of functional and object-oriented programs. Software Engineering Journal, 11(4):247–254, 1996.

[14] S. Kleinschmager, S. Hanenberg, R. Robbes, É. Tanter, and A. Stefik. Do static type systems improve the maintainability of software systems? an empirical study. In Program Comprehension (ICPC), 2012 IEEE 20th International Conference on, pages 153–162. IEEE, 2012.

[15] L. A. Meyerovich and A. S. Rabkin. Empirical analysis of programming language adoption. In Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages applications, pages 1–18. ACM, 2013.

[16] V. Pankratius, F. Schmidt, and G. Garretón. Combining functional and imperative programming for multicore software: an empirical study evaluating scala and java. In Proceedings of the 2012 International Conference on Software Engineering, pages 123–133. IEEE Press, 2012