Demystifying Issues, Challenges, and Solutions for Multilingual Software Development

Haoran Yang
Washington State University
Pullman, WA, USA
haoran.yang2@wsu.edu

Weile Lian
Washington State University
Pullman, WA, USA
weile.lian@wsu.edu

Shaowei Wang
University of Manitoba
Winnipeg, Canada
shaowei.wang@umanitoba.ca

Haipeng Cai*
Washington State University
Pullman, WA, USA
haipeng.cai@wsu.edu

***Abstract*—Developing a software project using multiple languages together has been a dominant practice for years. Yet it remains unclear what *issues* developers encounter during the development, which *challenges* cause the issues, and what *solutions* developers receive. In this paper, we aim to answer these questions via a study on developer discussions on Stack Overflow. By manually analyzing 586 highly relevant posts spanning 14 years, we observed a large variety (11 categories) of *issues*, dominated by those with interfacing and data handling among different languages. Behind these issues, we found that a major challenge developers faced is the diversity and complexity in multilingual code building and interoperability. Another key challenge lies in developers' lack of particular technical background on the diverse features of various languages (e.g., threading and memory management mechanisms). Meanwhile, Stack Overflow itself served as a key source of solutions to these challenges—the majority (73%) of the posts received accepted answers eventually, and most in a week (36.5% within 24 hours and 25% in the next 6 days). Based on our findings on these issues, challenges, and solutions, we provide actionable insights and suggestions for both multi-language software researchers and developers.**

***Index Terms*—Multilingual software, development issues, language interfacing, software build, data format, interoperability**

I. Introduction

Using multiple computer languages in a single software project (i.e., *multilingual development*) is a prominent software practice [1]–[3]. In fact, multilingual development has been a norm for decades—over 80% projects sampled both in the industry and open-source world use two or more languages [1], [4]–[6], and become increasingly popular among developers [7]–[9]. This practice is easily justifiable by the increased productivity and flexibility of combining the complementary strengths of different languages [1], [2], [7], [10]–[13].

Meanwhile, as for single-language software development, developers adopting this practice (i.e., *multilingual developers*) need support of techniques and tools for assuring the quality of multi-language software. Intuitively, this need is more critical given the increased complexity of multilingual code [1]. Indeed, it is already found that using multiple languages tends to make the resulting software more prone to both functionality defects [14], [15] and security vulnerabilities [16]–[18]. To address such needs, it is imperative to understand the issues multilingual developers face and the underlying challenges that lead to those issues—so that researchers and tool providers can be well-informed regarding what to target and how best to meet the needs. Also, knowing what current solutions to those challenges are offers direct insights for designing automated tools while helping identify missing areas.

* Haipeng Cai is the corresponding author.

Existing research has looked at the multilingual development phenomenon, yet focusing mainly on the characteristics (e.g., prevalence [1], [3], [4], quality [14], [16], cross-language links [6], [7], [12], [19]) of the *end product* (i.e., the resulting multi-language software). To investigate practical issues/challenges in multilingual development, a common approach is to examine developers' discussions (questions, answers, comments, etc.) on relevant Q&A platforms such as Stack Overflow (SO) [20]. Indeed, SO has enabled numerous informative studies on developers' issues with software development concerning a broad range of topics [21]–[23]. However, no prior study has particularly addressed *multilingual* development issues encountered by developers.

In this paper, we set out to systematically examine the *issues*, *challenges*, and *solutions* regarding multilingual development by analyzing relevant SO posts. This choice of information source is justified by SO's role as a primary platform where developers exchange about software development and an educational resource that influences their development practices [24], [25] We started with the entire set of potentially relevant posts since the creation of SO until late 2021 and manually inspected 586 randomly sampled and highly relevant ones. For each post, we checked the entire thread of discussion, including the original question, answers, and comments. We frame our study around three key questions, as outlined below along with our major findings as respective answers.

RQ1: What are the issues encountered and discussed by multilingual developers? We found that developers encountered a variety (11 categories) of issues when developing multi-language software, among which the primary ones include those concerning how to *interface different languages* (accounting for 38% of the 586 inspected posts), *handle data across languages* (30%), and *build the holistic multi-language system* (15%). Issues were associated with specific language combinations. The top three language combinations involved were `PHP-JavaScript` (26% of the posts), `Python-C++` (10%), and `C++-C#` (9%), while almost half of all the 586 issues were with Web applications. For instance, 72% of Embedding issues were mainly encountered in `PHP-JavaScript` projects, while 30% of Error/Exception Handling issues were associated with the language combination of `Python-C++`.

RQ2: Which challenges do multilingual developers face as the root causes behind the issues? Build issues were mainly about compilation failures, version conflicts caused by language evolution, and project maintenance problems, due to challenges like insufficient documentation, uninformative compiler error messages, and inadequate tool support. Data handling issues were mostly about data conversion and third-part library usage difference, which are caused by challenges including variations in the configurations of libraries, complexity of cross-language data structures, and diversity in the type systems of different languages. Interfacing issues were often exhibited as failures in memory management and interoperations across languages, and the main causes were inconsistencies in memory management mechanisms between languages and incompatibilities (and even conflicts) in the data types of different languages.

RQ3: How are the challenges being solved by multilingual developers? For the documentation insufficiency challenge underlying the build issues, the solutions were to use external links to relevant information and multilingual code examples directly on SO as alternative documentation. The main solutions to the data conversion related challenges underlying data-handling issues were to use language-independent data format and check data-conversion (typically, relevant foreign) function calls against encoding/serialization errors. To overcome challenges with memory-management inconsistencies that led to interfacing issues, current solutions suggest avoiding pointer/memory operations across languages.

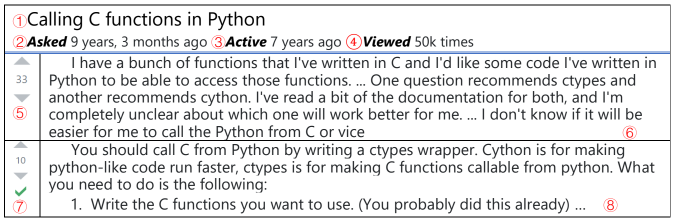
Based on our findings, we offer actionable suggestions to researchers and developers of multi-language software. For example, we suggest that multilingual developers refrain from directly managing threads and memory *across* languages—instead, they should manage them within individual languages and let the respective language's runtime to manage those low-level interoperations. Developers should be aware of the common issues/challenges of multilingual development when making decisions, e.g., they should carefully consider the typical issues/challenges that are often associated with particular language combinations when deciding which languages to use. For researchers, our results clearly call for efforts on developing tools to support for detecting data type/format conflicts across languages, recommending better API/usage in multilingual development, and building multi-language projects.

II. Background

This section defines key concepts and terminologies used in the rest of this paper that are necessary for understanding it.

A. Multi-Language Software and Multilingual Development

Multi-language software is one that uses more than one computer language, regardless of language classes (e.g., programming/modeling/descriptive), and the code written in each language (i.e., **language unit**) is *integral* to the system functionalities. The process/practice of developing multi-language software is referred to as **multilingual development**.


①Calling C functions in Python

②***Asked*** 9 years, 3 months ago ③***Active*** 7 years ago ④***Viewed*** 50k times

33 ⑤ I have a bunch of functions that I've written in C and I'd like some code I've written in Python to be able to access those functions. ... One question recommends ctypes and another recommends cython. I've read a bit of the documentation for both, and I'm completely unclear about which one will work better for me. ... I don't know if it will be easier for me to call the Python from C or vice ⑥

10 ✓ ⑦ You should call C from Python by writing a ctypes wrapper. Cython is for making python-like code run faster, ctypes is for making C functions callable from python. What you need to do is the following:

1. Write the C functions you want to use. (You probably did this already) ... ⑧

Fig. 1: An example Stack-Overflow post with an accepted answer illustrating that the `ctypes` is recommended to the questioner with advice on the usage of it [26].

B. Language Interfacing Mechanism

In a multi-language software project, developers have to consider and decide on the mechanisms by which the used languages interface with each other (i.e., language **interfacing mechanism** [16]). For example, in `Java-C` programs, a typical interfacing mechanism is that the Java code unit calls a C function via Java Native Interface (JNI).

Generally, there are two types of interfacing mechanism: explicit and implicit. An **explicit interfacing** is a mechanism by which the two languages interact via function invocations (e.g., via JNI for `Java-C` software)—one language unit invokes a function in another unit. There are two kinds of such functions: *foreign function* and *native function*. A **native function** is one written by the *application* developer in a language (i.e., *native language*) that is different from the caller's language. A **foreign function** is one written by the *language* developer in the caller's language, used for retrieving information from a different language (i.e., *foreign language*). For example, as shown in Figure 1, the answer suggested the developer uses the `ctypes` to call C functions from Python. In this case, the functions written in C are native functions.

Alternatively, two language units may interact via **implicit interfacing**, where one unit accesses another unit's functionalities through IPC (interprocess communications)—via shared memory, network-based message passing, message queue, etc. For example, in `PHP-Java` applications, the PHP client interfaces with the Java server by message passing via sockets.

C. Embedding

In addition to having one language interface with another, another way of integrating units of different languages in a multi-language project is to embed one language unit within the code of another language. Typically, **embedding** is often used between two declarative languages (e.g., HTML embedding CSS), or a declarative language and an imperative language (e.g., HTML embedding JavaScript).

III. Methodology

A typical SO post has the following attributes that are used in our analyses, as illustrated in Figure 1.

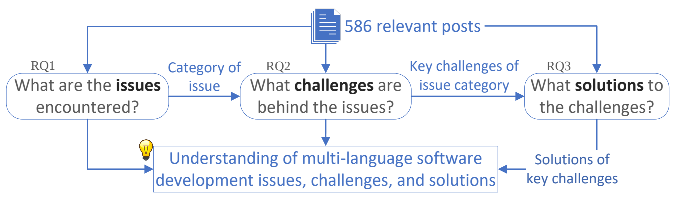

586 relevant posts
RQ1
What are the issues encountered?
Category of issue
RQ2
What challenges are behind the issues?
Key challenges of issue category
RQ3
What solutions to the challenges?
Understanding of multi-language software development issues, challenges, and solutions
Solutions of key challenges

Fig. 2: Our research questions and their relationships.

① **Title** summarizes the question of the asker (developer).
② **Created time** is the post creation time, in seconds.
③ **Active time** is the most recent time someone participated in the discussion of (i.e., posting answers to or comments on) this post, also accurate to seconds.
④ **View count** is the #times the post has been viewed.
⑤ **Vote** is the sheer #times the post has been liked (i.e., #upvotes - #downvotes).
⑥ **Question** describes the question (i.e., issue encountered) in detail. It may contain code snippets, screenshots, or links.
⑦ **Accepted answer** indicates that this answer has been selected as the solution by the asker, marked by a green tick.
⑧ **Answer** describes a potential solution to the issue of the asker. It may also contain code snippets, screenshots, or links.

A. *Research Questions*

Using the relevant posts, we seek to answer three research questions in relation, as shown in Figure 2. Below, we clarify the aim and justification for each question with respect to our study goal and describe our approach to answering it.

RQ1: Issues. First, we aimed to discover the problems that these developers have that may both impede their multilingual development productivity and compromise the quality of the resulting software product. Knowing the problems is the first step toward helping developers overcome those barriers. We plan to identify the first taxonomy of multilingual development issues that covers the entire software development life cycle (ranging from analysis and design to coding/testing and maintenance/evolution). This taxonomy provides an overarching guide for our further investigation into the underlying challenges and solutions. We also identified the language combinations associated with each category of issues.

To answer RQ1, we adopted an open coding approach to categorize each post, as we had no prior knowledge about it. Three of the authors created and validated the codebook with a common inter-agreement and consensus procedure, and then used it to categorize sample posts confirmed as highly relevant to our study, as elaborated in §III-C. As a result of categorizing these posts, we obtained the issue taxonomy mentioned above.

RQ2: Challenges. Next, we aimed to understand the root causes (i.e., *challenges*) underlying the issues identified in RQ1. This is essential for developing effective and sustainable solutions. We focused on the 4 most prevalent categories of issues and identified the major challenges for each. The result provides a guiding reference for our investigation into the current solutions to each of those categories of issues.

To answer RQ2, we clustered the 586 resulting posts from RQ1 by their issue categories. Then, authors 1 and 2 independently analyzed the posts in each category to summarize the common root causes across those posts based on the discussion in the posts while leveraging our prior knowledge about multilingual development and online resources. This was followed by meetings to resolve disagreements until a final consensus was reached for each post.

More specifically, for each issue category, we first read each post in it and assigned one/more tags that indicate the content and problems discussed. We then clustered these tags based on semantic similarities, and examined the posts in each cluster and identified causal factors of the problems. Finally, we consolidated the primary, common factors for each cluster as one of the root-causes (i.e., ‘challenges’) for the issue. As a result, for each issue category, we identified multiple root-causes/challenges, although due to space limit we will only present the primary one or two challenges.

RQ3: Solutions. Lastly, we aimed to reveal the current solutions that developers have been offered to the issues they encountered. The answer to RQ3 can provide a catalog of existing solutions which could be used as an immediately useful reference for developers. It also provides a picture of the current situation and practice of multilingual development, and helps identify a roadmap for future research. We focus on retrieving the common solutions for the 4 prevalent issue categories (discovered for RQ1) and their top underlying challenge (identified for RQ2).

To answer RQ3, we first selected the posts belonging to the studied 4 categories which had an accepted answer. Then, we examined the answers and the comments of each selected post. Finally, we consolidated the solutions specific to individual posts into common, more general ones for the challenge (root causes) behind each category of issues.

B. *Data Collection*

Given the complexity and diversity of each individual language and their combinations in multilingual software, we rely on manual inspection to analyze the content of posts. Thus, we need to identify a sufficient yet manageable number of relevant posts to study. As shown in Figure 3, we started with all the posts available on SO. We then used *two filters* below to identify relevant posts with high quality. Note that we consider a post **relevant** if it (1) discusses software development issues, and (2) the development issues are with *multi-language software* (§II-A) projects.

Filtering via language tags and #votes. To make our study manageable, we started with the most popular languages as *tags*, for which we referred to several language popularity rankings [27] and found common languages in those top-10 lists. We targeted the top-7 popular languages in 2020 (when we started this study), `JavaScript`, `Python`, `Java`, `C`, `C++`, `Shell`, and `PHP`, as an initial filter, as these have been mainstream programming languages for a long time [1]–[3]. We chose 7 because we referred to several top-language lists and there are 7 in common. We also used the #votes as a proxy for selecting potentially high-quality posts following prior studies [28], [29]. We kept posts where the original

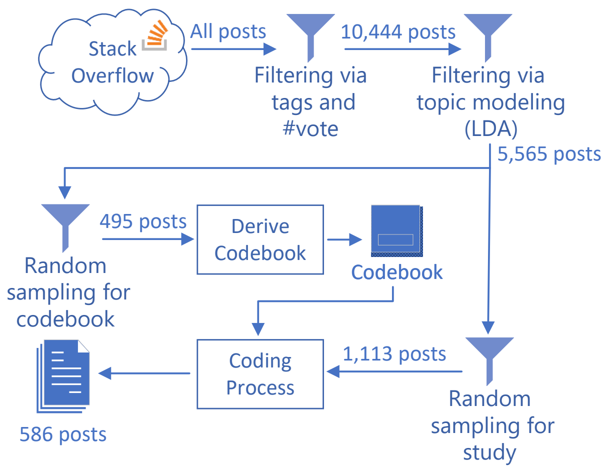

Stack Overflow
All posts
Filtering via tags and #vote
10,444 posts
Filtering via topic modeling (LDA)
5,565 posts
495 posts
Random sampling for codebook
Derive Codebook
Codebook
Coding Process
1,113 posts
Random sampling for study
586 posts

Fig. 3: The flow of our data collection process.

question received 6+ votes and prioritized our effort on those likely higher-quality posts. We tried a few other thresholds and chose 6 as it gave a good trade-off between the topic coverage of posts and an affordable amount of manual effort. To obtain the posts, we used the `Scrapy` tool [30] to crawl posts from SO that have at least two tags among the 7 selected languages. We crawled the SO website, rather than using a dump (which is released every three months), because we wanted to ensure that our study is up to date. After filtering via language tags and #votes, we ended up with 10,444 posts.

Filtering via topic modeling (LDA). Per our study goals, we would like to focus just on posts that are highly relevant to multilingual-development issues. Yet among the 10,444 posts, most were clearly irrelevant to any development issue per our quick sampling and inspection. However, filtering out irrelevant ones fully manually would be too tedious/costly. To help rule out such posts efficiently, we thus applied LDA [31] topic modeling as a filtering step. Specifically, we used LDA to extract a generic list of development-issue-relevant topic words, from which we then manually removed noisy ones (e.g., 'hence', 'solution') that are not indicative of development issues. Then, we kept a post if it contained some of the remaining topic words—we did not exclude a post just because it contains a non-indicative/noisy topic word. With the remaining ones (e.g., 'JNI', 'SWIG', 'socket') as keywords, we experimented with many potential sets of final keywords (including various word combinations as phrases) while randomly sampling 50 posts for which we manually labeled as ground truth to validate. For each potential set, we used it to detect relevant/irrelevant posts from the 50 and computed precision and recall as per the ground truth. We ended with a final keyword set that gave us 95% recall and 45.23% precision—other sets gave better precision but much lower recall. We favored recall because we intend to keep a high coverage of actually relevant posts even at the cost of greater manual effort for removing false positives in the next step. The recall of 95% is high enough for us to use this LDA-based filtering step as a filter, as it will not lead us to miss many true positives during our final post-relevancy confirmation. Filtering with the LDA left us with 5,565 posts.

C. Post Categorization

For the next step, we need to derive a codebook and then apply it for the categorization of posts.

Random sampling for codebook. To manually generate the codebook, we first randomly sampled 495 out of the 5,565 posts for analysis. This sample size is statistically significant at 98% confidence level (CL) and 5% margin of error (ME).

Derive codebook. Three of the authors each created an initial list of issue categories independently based on the 495 posts. Then, they addressed any divergence until reaching a consensus on the final codebook of Table I. More specifically, each of them (1) carefully read the posts, (2) checked whether each post belonged to the current issue categories, and (3) created a new category if needed. When creating a new category, they first defined a label to describe the issue in the post, then created descriptions for this category and provided some common issues that should belong to this category, and added the post to the codebook as an example for this category.

Random sampling for study. Filtering with the LDA left us with 5,565 posts. Considering the typically substantial time cost of manually inspecting even one post, we randomly selected 20% of these posts, resulting in 1,113 posts that need to be manually confirmed in the coding process. This sample size is statistically significant at 99.9% CL and 5% ME.

Coding process. Finally, we manually determined the eventual relevancy of each of the 1,113 posts, confirmed that 586 were highly relevant, and categorized them using the codebook derived. We utilized *negotiated agreement* to ensure the reliability of the coding procedure, because it is helpful in research when the primary objective, just like ours here, is to provide new insights [32]. The final coding was performed by the three authors who derived the codebook reaching a consensus on each of these 586 posts, which (hereafter referred to as "relevant posts") are used for our further manual analysis to answer our three research questions.

IV. Results

A. RQ1: Issues

We present our issue taxonomy and then look into the association between issue categories and language combinations.

1) Issue Taxonomy (Categories): As shown in Figure 4, we categorized the 586 posts into 11 disjoint categories (the leaf nodes of the tree). The hierarchy also informs the distribution of the posts over these categories. At the top level, we have the 8 categories from the codebook we derived Table I . Then, we broke down the two categories with the most (222 and 174) posts to form the second level: **Interfacing choice** and **Invocation mechanism** under Interfacing, and **Data format/compatibility** and **Interoperability** under Data handling. Finally, the *Invocation mechanism* category is further broken to the third level as two subcategories corresponding to the two kinds of interfacing mechanisms (§II-B). We formed the taxonomy such that the number of posts in all leaf categories does not exceed 100, which facilitates deeper analyses (for RQ2 and RQ3).

TABLE I: Key codes used to categorize SO posts on multilingual development issues.

Code	Summary Description
Language choice	Developers ask/discuss about choosing the languages to use for their multi-language project.
Interfacing	Developers ask/discuss about calling interface (implicit or explicit) between two languages.
Embedding	Developers ask/discuss about embedding one language unit within another language unit.
Build	Developers discuss how to build (e.g., compile, install, configure) the multilingual code.
Efficiency/Performance	Developers are concerned about computing/storage efficiency of their multi-language systems.
Security	Developers have security/privacy/cryptography-related concerns with multilingual development.
Data handling	Developers discuss cross-language data processing and/or transferring data between languages.
Error/exception handling	Developers ask/discuss about handling errors and/or exceptions in multilingual code.

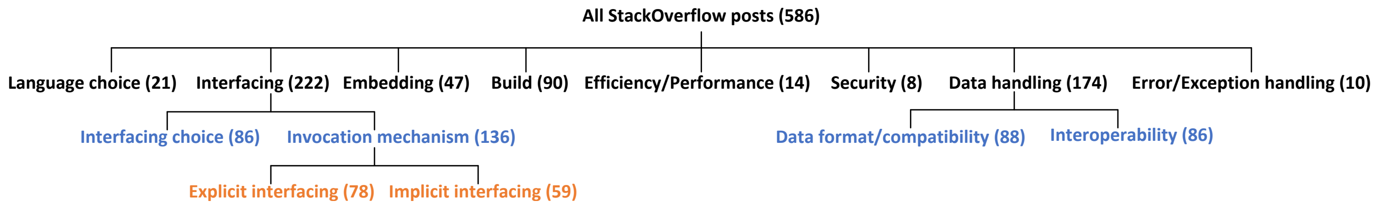

All StackOverflow posts (586)
Language choice (21)
Interfacing (222)
Embedding (47)
Build (90)
Efficiency/Performance (14)
Security (8)
Data handling (174)
Error/Exception handling (10)
Interfacing choice (86)
Invocation mechanism (136)
Data format/compatibility (88)
Interoperability (86)
Explicit interfacing (78)
Implicit interfacing (59)

Fig. 4: Our proposed *taxonomy of multilingual development issues* derived on the basis of the studied SO posts.

38% and 30% of all of the relevant posts are about Interfacing and Data handling, indicating that developers need more help on these issues.

We also found that 73% of the 586 posts have an accepted answer, versus 57% for all posts on SO per an earlier study [33]. Only 1% of these multilingual posts never received any answer. Moreover, we found no significant differences among the 11 issue categories regarding the response status. In terms of the response time, among those 73% of the 586 relevant posts, most of them got the accepted answer within seven days and nearly half got it in just 24 hours. Finally, we observed that the average view count per post has started rising since 2016 (after an 8-year plateau), and has then been sharply and steadily increasing since 2019.

The multilingual development community is active, with relevant questions getting quick responses within one day through a week, and different categories of issues received a similar degree of attention. Meanwhile, the community appeared to grow very fast in recent years.

2) Issue Categories versus Language Combinations: Figure 5 (top chart) shows the primary language combinations associated with each of the 8 top-level issue categories, with the distribution of all the 586 posts over the involved language combinations provided as a reference (bottom chart). The results in the bottom chart reveal that across all the issue categories, the top 3 language combinations involved were `PHP-JavaScript` (26% of the posts), `Python-C++` (10%), and `C++-C#` (9%). Note that this is not necessarily an indication that `PHP-JavaScript` projects are most likely subject to different kinds of multilingual development issues—it is possibly just a result of the highest popularity of this combination among the shown ones.

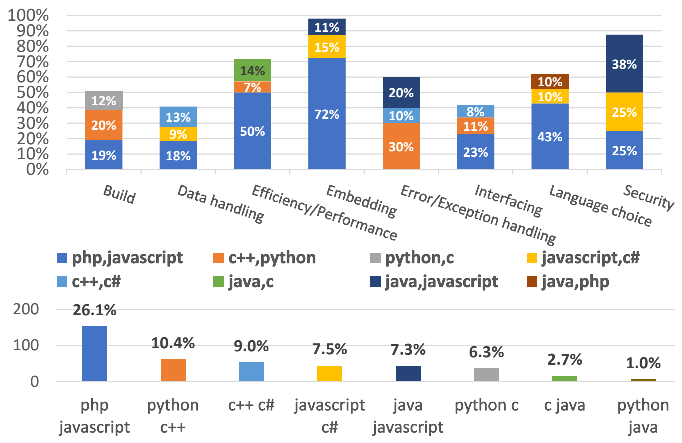
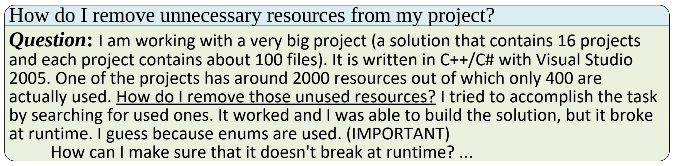


Fig. 5: The percentage distribution of top three language combinations involved in the posts of each issue category (top), and the percentage distribution of all the 586 posts over all such language combinations (bottom). The bottom chart serves as a reference for better understanding the top chart.

While the prevalence of language combinations affects the absolute proportions of posts associated with each combination, it is still noticeable that certain types of issues were typically associated with specific language combinations. For instance, 72% of Embedding issues were mainly encountered in `PHP-JavaScript` projects, 50% of the Efficiency/Performance issues were associated with `PHP-JavaScript`, and 38% of the Security issues were associated with `Java-JavaScript`. Meanwhile, Data handling issues were discussed almost evenly across different kinds of projects in terms of language combinations used. This contrast may also be elucidated by looking at the heights of the bars (in the top chart): the higher the bar for an issue category, the more dominating the top three language combinations that are associated with that category of issues.


How do I remove unnecessary resources from my project?

Question: I am working with a very big project (a solution that contains 16 projects and each project contains about 100 files). It is written in C++/C# with Visual Studio 2005. One of the projects has around 2000 resources out of which only 400 are actually used. How do I remove those unused resources? I tried to accomplish the task by searching for used ones. It worked and I was able to build the solution, but it broke at runtime. I guess because enums are used. (IMPORTANT)
How can I make sure that it doesn't break at runtime? ...

Fig. 6: An example illustrating that tool support for building multilingual code is lacking, as a challenge causing the issue here: unnecessary resources were not successfully removed during the build process [36].

Embedding and Efficiency/Performance issues were related to a few highly-dominant language combinations, while other issue categories involved more diverse combinations.

B. RQ2: Challenges

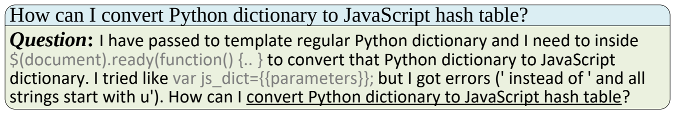
Out of the 11 distinct issue categories, we focused on the 4 most prevalent ones (accounting for 58% of all posts) to identify common challenges behind each.

1) Build: Build issues were found mainly concerning problems with installation, compilation, configuration, and packaging, for which we identified 2 common challenges. These two main challenges summarize the root causes of multilingual build issues encountered by developers, as seen in 10 out of the total of 90 posts on such (i.e., build) issues.

Challenge 1: *Documentations on the discussed build-related topic are insufficient or entirely lacking.* Many developers mentioned documentation related problems in their questions or comments. In particular, among the relevant posts, the developers faced two kinds of situations:

- *Missing documentation:* Documentation on the discussed topic is entirely lacking/missing. For example, the post [34] illustrates this situation. In this post, the `QWebChannel` JS API failed to setup in a `QWebEngineView` since the QT documentation is incomplete. The reason is that the documentation about `QWebChannel` is entirely missing.
- *Insufficient documentation*: Relevant documentations exist but they only provide some reference information or general ideas for solving respective problems. As shown in [35], the `Pybind11` documentation only addresses the special case and does not provide a complete method to developers; as a result, the developer could not figure out how to split code into multiple modules in a general situation.

Challenge 2: *Support for multi-language code build is severely lacking.* For example, compiler log messages are not enough, and developers seek to build related tools to improve work efficiency. However, the existing tools can not support multilingual projects well. Figure 6 indicates that the existing tool support failed to help the developer remove unnecessary resources from multi-language projects during the build process. The reason is that the build support is lacking in multi-language projects. Actually, the project was written in `C++-C#`, yet there is no supporting tool that can remove resources at runtime.


How can I convert Python dictionary to JavaScript hash table?

Question: I have passed to template regular Python dictionary and I need to inside $(document).ready(function() {.. } to convert that Python dictionary to JavaScript dictionary. I tried like var js_dict={{parameters}}; but I got errors (' instead of ' and all strings start with u'). How can I convert Python dictionary to JavaScript hash table?

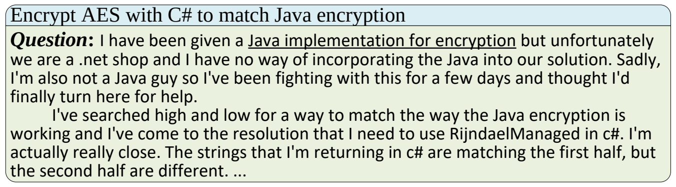
Fig. 7: An example showing that data conversion is difficult in multilingual coding, as a challenge causing the issue: converting the Python dictionary to JavaScript hash table failed in this particular case [37].

The build issue related posts are about failures in the build process, i.e., compilation failures, language version conflicts, and project maintenance problems. These problems are due to the challenges of missing/insufficient documentation and lacking necessary support.

2) Data Format/Compatibility: These issues usually occur before and after data exchange, causing program defects due to data format or compatibility problems. Such issues may also occur when the data formats between the languages are not compatible. Below, we elaborate on two primary challenges underlying data Format/Compatibility issues. These two major challenges summarize the root causes of such issues encountered by developers in 27 out of a total of 88 posts.

Challenge 1: *The languages involved in multi-language software development projects can have different typing strengths and type systems, making the conversions across such languages error-prone.* Meanwhile, for the same reason, it is difficult for developers to fully understand the data type validity requirements and data conversion rules across all the different languages. Type conversion is the most common problem with these symptoms. Each programming language has its own rules for how to convert types. Languages are also divided into strong typing and weak typing. Languages with weak typing perform many implicit conversions between data types and usually allows the compiler to interpret data items as having different representations arbitrarily. Although this is very convenient for developers, it introduces errors in data format for multilingual development projects. For example, Figure 7 illustrates that the data of the `dictionary` type in Python is hard to convert directly to a JS hash table. The reason is that Python and JavaScript have different systems to represent a dictionary internally.

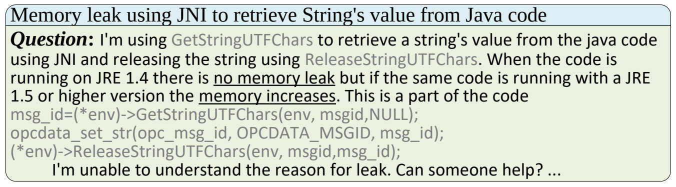
Challenge 2: *Different languages have varying third-party libraries even for the same algorithm, and accordingly, the libraries for these algorithms provide different functions for dissimilar languages.* This is a common error when working with the same algorithm but in other languages. Technically, the same algorithm should produce the same results in different languages. Yet in practice developers can get inconsistent results. This is because different languages set different algorithm parameters, leading to data generated in one language not being compatible with another, even though both use the same algorithm. The third-party libraries provide different functions with different parameters to achieve the purpose. In this case, it is difficult for developers to use the correct function when unfamiliar with it. As Figure 8 shows, the encryption result in the C# unit is different from that in the


Encrypt AES with C# to match Java encryption

Question: I have been given a Java implementation for encryption but unfortunately we are a .net shop and I have no way of incorporating the Java into our solution. Sadly, I'm also not a Java guy so I've been fighting with this for a few days and thought I'd finally turn here for help.

I've searched high and low for a way to match the way the Java encryption is working and I've come to the resolution that I need to use RijndaelManaged in c#. I'm actually really close. The strings that I'm returning in c# are matching the first half, but the second half are different. ...

Fig. 8: An example illustrating that there is encryption and decryption difficulty between different languages, as a challenge causing the data format/compatibility issue that the AES encryption result is different between C# and Java [38].


Memory leak using JNI to retrieve String's value from Java code

Question: I'm using GetStringUTFChars to retrieve a string's value from the java code using JNI and releasing the string using ReleaseStringUTFChars. When the code is running on JRE 1.4 there is no memory leak but if the same code is running with a JRE 1.5 or higher version the memory increases. This is a part of the code

```
msg_id=(*env)->GetStringUTFChars(env, msgid,NULL);
opcdata_set_str(opc_msg_id, OPCDATA_MSGID, msg_id);
(*env)->ReleaseStringUTFChars(env, msgid,msg_id);
```

I'm unable to understand the reason for leak. Can someone help? ...

Fig. 9: An example illustrating that garbage collection mechanism divergences across different languages lead to interoperation faults, as a challenge causing the interoperability issue that JNI retrieving strings resulted in a memory leak [39].

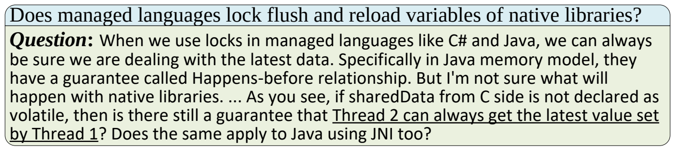
Java unit. The challenge is that even if the different languages encrypt using the same algorithm, the results can be different because of varying parameters.

The studies posts on Data Format/Compatibility issue were mainly about data conversion and third-party library usage differences. The challenges lie in the diversity in data type systems and variations in the configuration of the algorithms of the libraries.

3) Interoperability: This type of problem is relatively complex. In the interfacing phase, data related issues usually arise, except for formatting problems. For example, parameter configuration or type problems happening when data is transmitted are of this kind. We highlight below the two most significant challenges underlying these interoperability issues as seen in 29 posts, out of the total of 86 posts on such issues.

Challenge 1: *There are discrepancies in memory management mechanisms (e.g., allocation and recycling) across different languages, which cause interoperation faults (e.g., buffer overflow and memory leaks).* Some languages (such as C, C++) allow developers to manage their own memory. This seemingly convenient way may bring risks to the project in a multilingual environment. When multiple languages exist in a project simultaneously, if the garbage collector mechanism of each language is different, it will first cause development difficulties for developers. Second, if you want to pass data types such as pointers in development, it may cause problems in memory management. For example, Figure 9 illustrates that JNI generates memory leaks when retrieving strings from Java. The reason is that the different languages have distinct memory management mechanisms, which makes interoperation between two languages difficult.

Challenge 2: *Data types between languages can be incompatible or even have conflicts, which leads to failing interoperation between the languages.* Two languages that are


Does managed languages lock flush and reload variables of native libraries?

Question: When we use locks in managed languages like C# and Java, we can always be sure we are dealing with the latest data. Specifically in Java memory model, they have a guarantee called Happens-before relationship. But I'm not sure what will happen with native libraries. ... As you see, if sharedData from C side is not declared as volatile, then is there still a guarantee that Thread 2 can always get the latest value set by Thread 1? Does the same apply to Java using JNI too?

Fig. 10: An example illustrating that multiple threading makes the interface have special requirements in the multi-language project, as a challenge causing the explicit interfacing issue that the developer was confused about declaring the 'volatile' data type in multithreading between `C` and `C#`. [41]

not able to communicate normally interact through language interaction APIs. This issue is usually caused by poor semantic interoperability, which may exist in the data transmission process and lead the APIs to be buggy. In the post [40], the string-type data in C# is hard to match the parameter type `wchar_t *` returned from the C++ function. The challenge is caused by the conflict between different data type systems.

Interoperability issue posts were mainly about memory management and failures in the interoperations between languages. These challenges were due to discrepancies in memory management mechanisms and incompatibilities in data types across different languages.

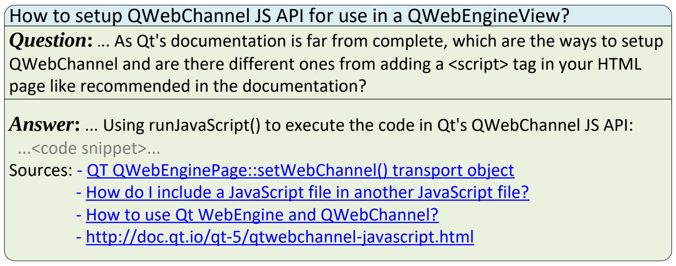
4) Explicit Interfacing: Explicit interfacing is often realized via a foreign function interface (FFI), e.g., JNI, CPython, etc. As the interfacing mechanism is already chosen, the questioner usually asks about errors related to the selected interface. We elaborate on the top challenge below, which explains Explicit Interfacing issues seen in 6 out of 78 posts on such issues.

Challenge 1: *Multiple threading further complicates the correct usage of explicit interfaces, as data transfer or value sharing via these interfaces across threads may have special/additional requirements.* Many developers have always had trouble with thread control, and many challenges exist in the single language development environment. In a multilingual environment, the multithreading mechanism may be completely different between languages. For example, the C++ language does not support multithreading directly, and multithreading programming in C++ is implemented by calling low-level functions of the operating system. Figure 10 shows that the developer wants to use the thread of the C# side to get the value from the C side. And the developer is confused about whether the variable needs to be declared as 'volatile' on the C side in the multi-language project. So the challenge is that there is an additional requirement for C side variables.

The studied posts on Explicit Interfacing issues were mainly about threading and foreign function calls. The challenges were due to complicated multiple threading and varying multithreading requirements.

C. RQ3: Solutions

We further looked into the current solutions to the primary (first) challenge for each of the 4 prevalent issue categories for which we identified the common challenges for RQ2.


How to setup QWebChannel JS API for use in a QWebEngineView?

Question: ... As Qt's documentation is far from complete, which are the ways to setup QWebChannel and are there different ones from adding a <script> tag in your HTML page like recommended in the documentation?

Answer: ... Using runJavaScript() to execute the code in Qt's QWebChannel JS API:
...<code snippet>...
Sources: - QT QWebEnginePage::setWebChannel() transport object
- How do I include a JavaScript file in another JavaScript file?
- How to use Qt WebEngine and QWebChannel?
- http://doc.qt.io/qt-5/qtwebchannel-javascript.html

Fig. 11: An example illustrating that external references and code example can provide useful information for developers to overcome documentation insufficiency [34].

1) Documentation Insufficiency Challenge: Overall, no generic solutions to this challenge were found. Nevertheless, we observed a few useful suggestions.

Solution 1: According to the accepted answers, external links can help the asker. Some posts even provide valuable parts of the corresponding documents directly.

Figure 11 shows that the developer encountered the challenge with insufficient documentation that led to not being able to set up the `QWebChannel` JavaScript API for use in `QWebEngineView`. In addition, the developer is confused about the <script> tag usage of `QWebChannel` on the HTML page. To address this problem, the developer was provided some alternative sources to solve Qt's `QwebChannel` JS API problem and <script> tag usage confusion. In sum, the developer can look for alternative sources on this Q&A platform (i.e., SO) itself or other developer discussions forums.

Solution 2: Accepted answers to the studied posts also suggest that code examples, provided sometimes, helped developers understand how to solve problems immediately.

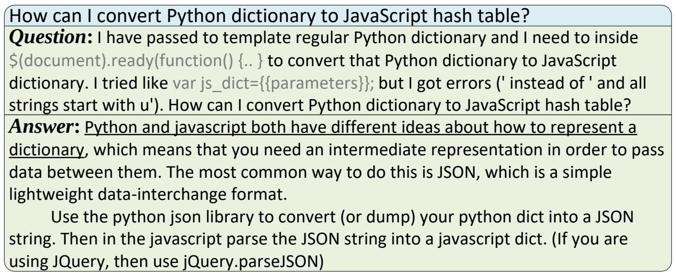
As shown in Figure 11, for the issue encountered by the developer as described above, in addition to the solution via alternative sources, the developer was also provided with a code example that demonstrates how to set up the `QWebChannel` JavaScript API and a code example on using a <script> tag in the web page. In sum, the developer can look for code examples when the challenge with insufficient documentation is encountered.

For the Documentation Insufficiency challenge under the Build issues, the solutions provided were to offer external links to and/or code examples about relevant information as alternatives (to the documentation).

2) Error-Prone Data Conversion Challenge: We found three generic solutions for the challenge with the error-proneness of data conversion underlying the *Data format* issue.

Solution 1: According to the accepted answers, we found that the questioner's function for data conversion is not applicable. Questioners should check whether the usage of data conversion functions is correct (e.g., against wrong choices of functions, function call parameter mismatch).

The post [42] shows that the developer encountered the challenge with converting a PHP array to a JavaScript object. However, the conversion was unsuccessful, causing a syntactic error, as the developer tried to use the function `implode()` for that purpose. The suggested solution was to use the `json_encode()` function, instead of `implode()`, for the data conversion. In sum, the developer should check whether the function they used is correct.


How can I convert Python dictionary to JavaScript hash table?

Question: I have passed to template regular Python dictionary and I need to inside $(document).ready(function() {.. } to convert that Python dictionary to JavaScript dictionary. I tried like var js_dict={{parameters}}; but I got errors (' instead of ' and all strings start with u'). How can I convert Python dictionary to JavaScript hash table?

Answer: Python and javascript both have different ideas about how to represent a dictionary, which means that you need an intermediate representation in order to pass data between them. The most common way to do this is JSON, which is a simple lightweight data-interchange format.

Use the python json library to convert (or dump) your python dict into a JSON string. Then in the javascript parse the JSON string into a javascript dict. (If you are using JQuery, then use jQuery.parseJSON)

Fig. 12: An example illustrating that there is a common way for the dictionary conversion between Python and JavaScript, which is using JSON [37].

Solution 2: Two languages can use foreign functions (e.g., in `SWIG`, `ctypes`, etc.) to realize data type conversion. The developer should look for dedicated foreign functions that can be used for correct data conversion across languages.

For example, the post [43] illustrates that the developer faced a challenge with converting the `jobject` data type to the `jstring` type. However, the conversion was unsuccessful since the JNI code got an error at compile time. To address this problem, the suggested solution was to use the foreign function properly so as to avoid any errors like the one in this case. It includes choosing the correct foreign function and modifying parameters as needed. In sum, the developer should check the dedicated foreign functions for data conversion.

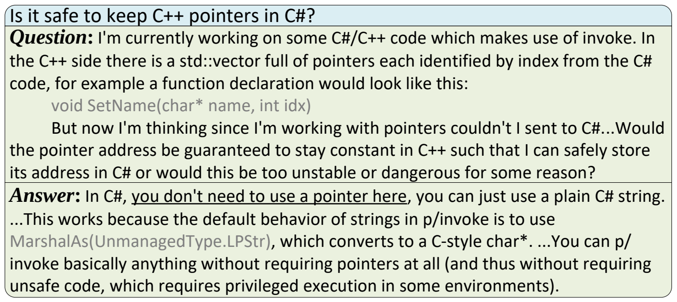
Solution 3: Some special multi-language projects based on network transmission use language-independent data-interchange formats (e.g., JSON, XML) for data conversion. Many languages have good compatibility with these data interchange formats, so using them to convert data is simple and fast while preventing data conversion related issues.

Figure 12 illustrates a developer encountering the challenge with converting a Python dictionary to its corresponding JavaScript dictionary. The developer tried to convert it directly. However, it was unsuccessful since the Python data type differs from the counterpart in JavaScript. The suggested solution was to choose JSON, a language-independent and data-interchange format, to pass data between Python and JavaScript to address this problem. In sum, the solution was to use the data-interchange format to accomplish data conversion.

For the Data Conversion challenge underlying the Data Format/Compatibility issues, the solutions were checking conversion function calls (e.g., foreign function) and using the language-independent and data-interchange format.

3) Memory-Access-Mechanism Discrepancy Challenge: We found two generic solutions for this challenge, which causes the Interoperability issue.

Solution 1: In the face of complex memory management mechanisms in a multilingual environment, developers can try to avoid using pointers across languages in the project


Is it safe to keep C++ pointers in C#?

***Question*:** I'm currently working on some C#/C++ code which makes use of invoke. In the C++ side there is a std::vector full of pointers each identified by index from the C# code, for example a function declaration would look like this:

void SetName(char* name, int idx)

But now I'm thinking since I'm working with pointers couldn't I sent to C#...Would the pointer address be guaranteed to stay constant in C++ such that I can safely store its address in C# or would this be too unstable or dangerous for some reason?

***Answer*:** In C#, you don't need to use a pointer here, you can just use a plain C# string. ...This works because the default behavior of strings in p/invoke is to use MarshalAs(UnmanagedType.LPStr), which converts to a C-style char*. ...You can p/invoke basically anything without requiring pointers at all (and thus without requiring unsafe code, which requires privileged execution in some environments).

Fig. 13: An example illustrating that it is not necessary to use the *pointer* date type in C#; instead, the *string* date type may be used to solve the problem [44].

or reduce the use of data types requiring memory allocation/releasing manually.

Figure 13 shows that keeping C++ pointers in C# poses a challenge for the developer. The developer was concerned that storing pointer addresses from the C++ code in the C# unit can be unstable or dangerous because programming in C++ requires the pointer address to remain unchanged. To address this problem, the suggested solution was to use *strings* in `p/invoke`, because such a string can be converted to a C-style `char*` using the `MarshalAs()` function in `p/invoke`. In sum, the solution is to avoid using the pointers in C# to keep the pointer address unchanged in C++.

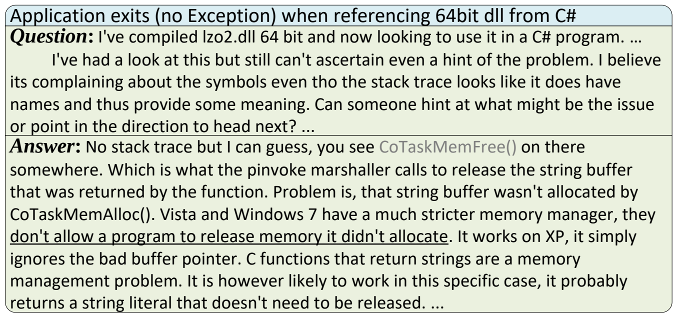
Solution 2: Another common cause of memory issues is improper memory allocation and release [45]. Because the memory management mechanisms between various languages are often different, it is easy to hit errors if developers allocate or release memory across languages. For example, when the memory allocated in one language is released from another, the limited access of memory across two languages due to permission issues may result in errors.

As shown in Figure 14, the developer encountered a problem that the application exits when calling DLL from C# and no exception was thrown. The developer was convinced that the DLL file does not have an issue. According to the answer, we know that it is a memory management problem, in which the developer wanted to use a C# function named `CoTaskMemFree()` to release memory. However, because of the stricter memory manager of Vista and Windows 7, C# cannot release the memory allocated in the C++ DLL. The suggested solution was to stop the marshaller from trying to release the string. In sum, the solution is to avoid releasing the memory from another language.

For the Memory-Access-Mechanism Discrepancies challenge underlying the Interoperability issues, the solutions were to be cautious against or even avoid pointer and memory operations across languages.

4) Threading-Induced Complication Challenge: We found two generic solutions for threading-induced complications underlying the Explicit Interfacing issue.

Solution 1: The multithreading mechanism may be different between languages. For some simple multi-threaded interactions (e.g., using threads to call functions without data exchange or mutual exclusion between threads, etc.), developers should manage threads within individual languages and avoid managing the same thread across languages (e.g., creating a thread of one language from another language) if possible.

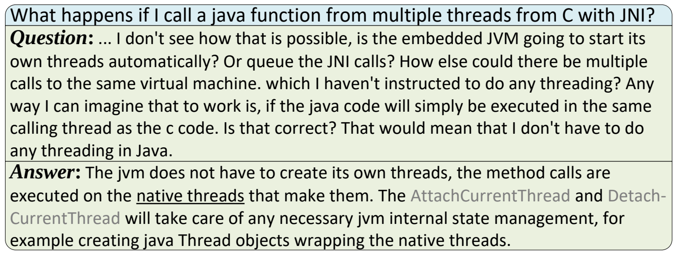

Application exits (no Exception) when referencing 64bit dll from C#

***Question*:** I've compiled lzo2.dll 64 bit and now looking to use it in a C# program. ...

I've had a look at this but still can't ascertain even a hint of the problem. I believe its complaining about the symbols even tho the stack trace looks like it does have names and thus provide some meaning. Can someone hint at what might be the issue or point in the direction to head next? ...

***Answer*:** No stack trace but I can guess, you see CoTaskMemFree() on there somewhere. Which is what the pinvoke marshaller calls to release the string buffer that was returned by the function. Problem is, that string buffer wasn't allocated by CoTaskMemAlloc(). Vista and Windows 7 have a much stricter memory manager, they don't allow a program to release memory it didn't allocate. It works on XP, it simply ignores the bad buffer pointer. C functions that return strings are a memory management problem. It is however likely to work in this specific case, it probably returns a string literal that doesn't need to be released. ...

Fig. 14: An example illustrating that C# code cannot release the memory allocated by C++ code using the function `CoTaskMemFree()` [46].


What happens if I call a java function from multiple threads from C with JNI?

***Question*:** ... I don't see how that is possible, is the embedded JVM going to start its own threads automatically? Or queue the JNI calls? How else could there be multiple calls to the same virtual machine. which I haven't instructed to do any threading? Any way I can imagine that to work is, if the java code will simply be executed in the same calling thread as the c code. Is that correct? That would mean that I don't have to do any threading in Java.

***Answer*:** The jvm does not have to create its own threads, the method calls are executed on the native threads that make them. The AttachCurrentThread and DetachCurrentThread will take care of any necessary jvm internal state management, for example creating java Thread objects wrapping the native threads.

Fig. 15: An example illustrates that the `JVM` can use its own thread mechanism to handle multi-threading in JNI. [48]

As shown in Figure 15, the developer did not understand how to call a foreign function (written in Java) from multiple native threads (created in C) and believed that this was not possible, or it would require multiple (Java) threads to be created accordingly by the JVM. Apparently, this challenge was encountered because the developer was not familiar with Java's *native threading* [47] (i.e., threading in native code) mechanisms via JNI.

The suggested solution is that the `JVM` does not have to create its own threads in order for the native threads to call the foreign function concurrently. Rather, the concurrent calls are executed in the native threads (not Java/JVM threads). Meanwhile, JNI allows for managing these native threads by wrapping them in Java thread objects or attaching/detaching them (using relevant APIs such as `AttachCurrentThread()` and `DetachCurrentThread()`).

Solution 2: For multilingual projects, the use of threads is likely to lead to more complex thread activities (e.g., mutual exclusion between threads) [49], [50], which often causes issues with explicit cross-language interfacing. Suppose threading management across languages is inevitable to deal with such complexities. In that case, the developers can use mutex (i.e., language-specific thread synchronization mechanism like `GIL` (*global interpreter lock*) for Python) properly to manage the threads across the languages involved.

For example, the post [51] shows that the developer encountered the challenge with C++ and Python multi-threading working with each other. However, the Python thread will stop when returning to the main thread of the C++ program, because the developer did not use proper thread synchronization (e.g., `GIL`) to manage these threads.

The suggested solution is that the main thread should hold the GIL, and the developer can use the GIL to manage each thread properly. In sum, the developer can use the GIL to manage the threads between C++ and Python.

For the Threading-Induced Complication challenge underlying the Explicit Interfacing issues, the solutions were avoiding managing threads across languages and, if it is inevitable, using a mutex (i.e., language-specific thread synchronization mechanism) to manage threads.

Note that none of the solutions presented for RQ3 were provided by the authors—they were given by developers, often in a *very-specific* manner. When we extracted *common* solutions, we also tried to keep them as specific as necessary for being *actionable*. Also, these specific solutions solve the challenges for *different* language combinations. Each solution was summarized from *a number of* posts—these posts often involve various language combinations and accordingly the solution applies to all those combinations. In fact, the solutions identified were rarely tied to a few specific language combinations, although each illustrating/example post we provide in the paper typically involved only one.

V. Discussion

A. Implications of our findings

1) Actionable Suggestions for Developers: **Our findings reveal the potential risks and challenges for multilingual development and developers should be aware of them when making decisions.** In RQ1, we reveal the frequent issues encountered by developers, such as issues about interfacing and data handling across languages, and the build of a holistic multi-language system. More importantly, certain types of issues were typically associated with some specific language combinations, e.g., 72% of Embedding issues were mainly encountered in `PHP-JavaScript` projects. In RQ2, we reveal the causes of the challenges for multilingual development, such as data handling and interfacing challenges across different languages due to the incompatibility of data type systems, and difficulty in memory and multi-threading operation and management across languages.

Such results reveal the potential risks and challenges in multilingual development, and provide insights to developers for decision-making given their context. For instance, developers should be aware of such risks and challenges when deciding on 1) if they should use single-language or multilingual development, and 2) if they use multilingual development, what languages should be selected and what system design across languages should be avoided or minimized to avoid potential risks. As nnother example, some language combinations are prone to being associated with a particular issue. Therefore, developers need to consider this when selecting languages based on their system design and requirements.

Our findings provide concrete guidelines for dealing with challenges in multilingual development. In RQ3, we summarized the solutions proposed in SO answers for solving common challenges underlying multilingual development issues, particularly in data handling and interfacing across languages. For instance, for the last two challenges examined in RQ3, the current solutions are to manage threads/pointers/memory within each language unit because doing so across languages would likely cause issues due to language semantics disparity; For the other two, the solutions are not to look for isolation (e.g., using a more universal data format like JSON). We suggest developers be aware of these common solutions when developing multilingual projects.

2) Actionable Suggestions for Researchers: **Future research is encouraged to develop techniques and tools to detect data type/format incompatibilities/conflicts across languages.** In §IV-B2, we observed that error-prone data conversions are frequently caused by data format/compatibility issues. Therefore, future research on detecting such issues is encouraged. For strongly-typed languages, one possible direction is to perform static analysis on two languages and infer the types/format of the converted data across languages. Once the types/formats are inferred, conflict detection could be performed. For languages with weak typing, the detection is more challenging since the type can only be determined during runtime. One possible direction is using machine learning techniques to infer type statically based on the surrounding code following prior studies [52], [53].

Developers often encounter issues with using or choosing the correct APIs in various tasks when handling data across languages. Future research should provide better support for recommending correct APIs and their usage. In §IV-B3, we observe that some issues are raised due to developers not being familiar with multilingual APIs and their usage or using the wrong APIs [54]. Therefore, one direction is to develop an API recommender for multilingual projects. Unfortunately, current research on API recommendations is not typically designed for multilingual development contexts. For instance, most of the-state-of-art API recommendation approaches are ML-based and heavily rely on training data [55], [56], while no multilingual code corpus is available [57]. Future researchers should create a multilingual project dataset, and then train the API recommendation model for multilingual development following the approach in prior studies [55], [58].

Future research is encouraged to develop tools to support the building of multilingual projects. As discussed in §IV-B1, tool support for building multilingual code is severely lacking. Each individual language's build environment and configuration are different, current building tools do not support or do not support very well the build of multi-language projects. Such lacking of tool support is particularly fatal in DevOps, which requires all build and deployment processes to be done automatically and continuously [59]. In other words, the lack of build tools may have impeded the DevOps workflow. We suggest future research should invest more efforts in developing multilingual-software-compatible build solutions with more multilingual build support tools. Moreover, the some of aforementioned data type/format incompatibility/conflict detection and cross-language API misuse detection tools could be integrated in such build support.

B. Threat to validity

Threats to internal validity. We depend heavily on manual analysis and our hand labeling is subject to personal prejudices. To reduce human bias throughout the labeling process, two authors examined each post, and labeled the challenge and solution independently, with discrepancies discussed until consensus was established. In RQ1, we incorporated the LDA model to filter out irrelevant posts and reduce the number of posts for manual labeling. Our approach may miss some relevant posts. However, our approach achieves a recall of 95% and we believe we cover the majority of actually relevant posts. Also, we filtered out posts with fewer votes. This may cause us to ignore some useful posts, resulting in biased results. However, we believe that the low votes of a post usually indicate relatively low value of that post.

Threats to external validity. We focused on analyzing posts from SO as the single data source for our study. Thus, our findings may not be generalizable to other programming-related Q&A websites. This raises a validity threat concerning the extent to which SO can be considered an accurate reflection of challenges multilingual developers actually have in the field.

Yet SO is the best data source we can access for our study. It is well-known as a big and popular knowledge repository where developers post questions and get answers, and has been widely used by prior software-engineering studies [24], [60], [61]. Thus, we assume it reasonably reflects developers' issues/challenges, including those on multilingual development. Nonetheless, this assumption implies a validity threat. Thus, we encourage future research to investigate the problem via other forms (e.g., surveys) and platforms (e.g., GitHub issues).

Threats to construct validity. As described in §III-B, our data collection started by focusing on a few popular languages. To obtain a better/more comprehensive taxonomy, ideally we would want to drop the language-choice-based filter. We applied this filter to make the study (which largely relies on manual effort) more manageable. Also, we took the current language choices as we believe addressing the issues related to the most popular languages would serve a potentially broader audience—they are common languages over several most-popular-language lists (including one from Stack Overflow itself [62]). Alternative/different choices may or may not lead to a better taxonomy. Nevertheless, our current choices constitute a construct validity threat.

VI. Related Work

A. Studies on Multilingual Software Development

Many studies have been done for multi-language software development [1]–[3], [9], [10], [12], [63]–[66]. The concept of multi-language software development (then called mixed language programming) was first proposed by Einarsson and Gentleman [66]. Later, Abidi et al. surveyed 93 developers to assess the impact of multi-language design practices on software quality [2]. Different from their study, we studied SO posts. Mayer et al. mined 1,150 open-source projects on GitHub [12] and found multilingual programming is prevalent in open-source projects. Mayer et al. also found that cross-language links (i.e., points in the system where code in two languages is connected) are common in multilingual development, and such links can cause problems for developers [1]. We observed similar issues from SO posts (e.g., issues related to data handling and interfacing across two languages). In sum, different from prior studies focusing on understanding the practice of multilingual developments by mining open-source projects and surveying developers, we reveal the issues, challenges, and solutions by analyzing SO posts.

B. Studying Development Issues using Stack Overflow

A number of studies used Stack Overflow as a source to study the issues encountered by developers for various domains [21]–[23], [60], [61], [67]. Abdalkareem et al. explored the impact of code reuse from SO on Android development [21] and found that after reusing the code from SO, the development issue increases significantly. Meng et al. manually analyzed SO posts to understand the existing secure coding practice and identify risks of adopting code snippets from SO as well as the gap between specification and implementation [22]. Yang et al. studied security-related topics and its trends over time by analyzing SO posts [60]. Wang et al. studied issues and challenges when developing big data applications by examining SO posts related to Apache Spark [61]. In contrast, we focus on studying a new topic—the issues, challenges, and solutions on multilingual development.

VII. Conclusion

While much prior work has studied multilingual development, there has been no in-depth study of the difficulties faced by developers during the process. This paper does a manual examination of developer discussions on Stack Overflow to dissect the issues, challenges, and solutions encountered during multi-language software development. We describe the types of challenges in multilingual development by manually examining the posts on Stack Overflow. Then, we summarize and demonstrate the primary current solutions to each dominant issue's primary challenge (i.e, root cause). Actionable insights and recommendations to both researchers and developers of multi-language software are provided through the consolidation of empirical findings.

VIII. Data Availability

Open science. Source code and datasets are all available in our artifact package and has been made publicly accessible.

Acknowledgment

We thank the reviewers for their constructive comments which helped us improve our original manuscript. This research was supported by National Science Foundation (NSF) under Grant CCF-2146233 and Office of Naval Research (ONR) under Grant N000142212111.

References

[1] P. Mayer, M. Kirsch, and M. A. Le, "On multi-language software development, cross-language links and accompanying tools: A survey of professional software developers," *Journal of Software Engineering Research and Development*, vol. 5, pp. 1–33, 2017.

[2] M. Abidi, M. Grichi, and F. Khomh, "Behind the scenes: Developers' perception of multi-language practices," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, 2019, pp. 72–81.

[3] W. Li, N. Meng, L. Li, and H. Cai, "Understanding language selection in multi-language software projects on GitHub," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings*, 2021, pp. 256–257.

[4] F. Tomassetti and M. Torchiano, "An empirical assessment of polyglot-ism in GitHub," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–4.

[5] C. Jones, *Software engineering best practices: Lessons from successful projects in the top companies*. McGraw-Hill Education, 2010.

[6] D. P. Delorey, C. D. Knutson, and C. Giraud-Carrier, "Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects," in *Second International Workshop on Public Data about Software Development*, 2007.

[7] P. S. Kochhar, D. Wijedasa, and D. Lo, "A large scale study of multiple programming languages and code quality," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, vol. 1, 2016, pp. 563–573.

[8] K. Kontogiannis, P. Linos, and K. Wong, "Comprehension and maintenance of large-scale multi-language software applications," in *2006 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 497–500.

[9] M. Abidi, M. S. Rahman, M. Openja, and F. Khomh, "Are multi-language design smells fault-prone? An empirical study," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 3, pp. 1–56, 2021.

[10] M. Grichi, E. E. Eghan, and B. Adams, "On the impact of multi-language development in machine learning frameworks," in *2020 IEEE International Conference on Software Maintenance and Evolution*, 2020, pp. 546–556.

[11] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 155–165.

[12] P. Mayer and A. Bauer, "An empirical analysis of the utilization of multiple programming languages in open source projects," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, 2015, pp. 1–10.

[13] P. Mayer, "A taxonomy of cross-language linking mechanisms in open source frameworks," *Computing*, vol. 99, no. 7, pp. 701–724, 2017.

[14] M. Grichi, M. Abidi, F. Jaafar, E. E. Eghan, and B. Adams, "On the impact of interlanguage dependencies in multilanguage systems empirical case study on Java native interface applications (JNI)," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 428–440, 2020.

[15] H. Yang, W. Li, and H. Cai, "Language-agnostic dynamic analysis of multilingual code: Promises, pitfalls, and prospects," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Ideas, Visions and Reflections*, 2022, pp. 1621–1626.

[16] W. Li, M. Jiang, X. Luo, and H. Cai, "PolyCruise: A cross-language dynamic information flow analysis," in *31st USENIX Security Symposium*, 2022, pp. 2513–2530.

[17] W. Li, L. Li, and H. Cai, "On the vulnerability proneness of multilingual code," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 847–859.

[18] W. Li, J. Ruan, G. Yi, L. Cheng, X. Luo, and H. Cai, "PolyFuzz: Holistic greybox fuzzing of multi-language systems," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[19] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère, "Popularity, interoperability, and impact of programming languages in 100,000 open source projects," in *2013 IEEE 37th annual computer software and applications conference*, 2013, pp. 303–312.

[20] "Stack Overflow," 2008 Accesssed: 2022-3-17. [Online]. Available: https://stackoverflow.com/

[21] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from StackOverflow: An exploratory study on android apps," *Information and Software Technology*, vol. 88, pp. 148–158, 2017.

[22] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. A. Argoty, "Secure coding practices in Java: Challenges and vulnerabilities," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 372–383.

[23] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[24] Y. Wu, S. Wang, C.-P. Bezemer, and K. Inoue, "How do developers utilize source code from Stack Overflow?" *Empirical Software Engineering*, vol. 24, no. 2, pp. 637–673, 2019.

[25] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Software Engineering*, vol. 22, no. 6, pp. 3149–3185.

[26] "Calling C functions in Python," 2013. [Online]. Available: https://stackoverflow.com/questions/16647186/calling-c-functions-in-python

[27] V. Puzhevich, "Top programming languages to use in 2020," 2020. [Online]. Available: https://scand.com/company/blog/top-programming-languages-to-use-in-2020/

[28] S. Wang, T.-H. Chen, and A. E. Hassan, "Understanding the factors for fast answers in technical Q&A websites," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1552–1593, 2018.

[29] A. Bhatia, S. Wang, M. Asaduzzaman, and A. E. Hassan, "A study of bug management using the Stack Exchange question and answering platform," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 502–518, 2020.

[30] "Scrapy," Accesssed: 2022-3-17. [Online]. Available: https://scrapy.org

[31] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[32] E. R. Morrissey, "Sources of error in the coding of questionnaire data," *Sociological methods & research*, vol. 3, no. 2, pp. 209–232, 1974.

[33] N. Gantayat, P. Dhoolia, R. Padhye, S. Mani, and V. S. Sinha, "The synergy between voting and acceptance of answers on StackOverflow-or the lack thereof," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 406–409.

[34] "How to setup QWebChannel JS API for use in a QWebEngineView?" 2016. [Online]. Available: https://stackoverflow.com/questions/39649807/how-to-setup-qwebchannel-js-api-for-use-in-a-qwebengineview

[35] "With pybind11, how to split my code into multiple modules/files?" 2018. [Online]. Available: https://stackoverflow.com/questions/53762552/with-pybind11-how-to-split-my-code-into-multiple-modules-files

[36] "How do I remove unnecessary resources from my project?" 2009. [Online]. Available: https://stackoverflow.com/questions/1496731/how-do-i-remove-unnecessary-resources-from-my-project

[37] "How can I convert Python dictionary to JavaScript hash table?" 2012. [Online]. Available: https://stackoverflow.com/questions/10073564/how-can-i-convert-python-dictionary-to-javascript-hash-table

[38] "Encrypt AES with C# to match Java encryption," 2014. [Online]. Available: https://stackoverflow.com/questions/21890805/encrypt-aes-with-c-sharp-to-match-java-encryption

[39] "Memory leak using JNI to retrieve string's value from Java code," 2009. [Online]. Available: https://stackoverflow.com/questions/915790/memory-leak-using-jni-to-retrieve-strings-value-from-java-code

[40] "C# calling native C++ all functions: What types to use?" 2011. [Online]. Available: https://stackoverflow.com/questions/5368720/c-sharp-calling-native-c-all-functions-what-types-to-use

[41] "Does managed languages lock flush and reload variables of native libraries?" 2019. [Online]. Available: https://stackoverflow.com/questions/56787106/does-managed-languages-lock-flush-and-reload-variables-of-native-libraries

[42] "Convert PHP associative array into JavaScript object," 2014. [Online]. Available: https://stackoverflow.com/questions/21153805/convert-php-associative-array-into-javascript-object

[43] "How to convert JObject to JString," 2012. [Online]. Available: https://stackoverflow.com/questions/14036004/how-to-convert-jobject-to-jstring

[44] "Is it safe to keep C++ pointers in C#?" 2011. [Online]. Available: https://stackoverflow.com/questions/7057022/is-it-safe-to-keep-c-pointers-in-c

[45] W. Li, H. Cai, Y. Sui, and D. Manz, "PCA: Memory leak detection using partial call-path analysis," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Tool Demos*, 2020, pp. 1621–1625.

[46] "Application exits (no exception) when referencing 64bit DLL from C#," 2011. [Online]. Available: https://stackoverflow.com/questions/8241732/application-exits-no-exception-when-referencing-64bit-dll-from-c-sharp

[47] IBM, "Understanding Java and native thread details," https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=threads-understanding-java-native-thread-details, 2019.

[48] "Multithreading with Python and C API," 2015. [Online]. Available: https://stackoverflow.com/questions/29595222/multithreading-with-python-and-c-api

[49] X. Fu, H. Cai, W. Li, and L. LI, "Seads: Scalable and cost-effective dynamic dependence analysis of distributed systems via reinforcement learning," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 1, pp. 1–45, 2020.

[50] H. Cai and X. Fu, "D^2Abs: A framework for dynamic dependence analysis of distributed programs," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 4733–4761, 2021.

[51] "What happens if I call a Java function from multiple threads from C with JNI?" 2011. [Online]. Available: https://stackoverflow.com/questions/8654519/what-happens-if-i-call-a-java-function-from-multiple-threads-from-c-with-jni

[52] A. M. Mir, E. Latoškinas, S. Proksch, and G. Gousios, "Type4Py: Practical deep similarity learning-based type inference for Python," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2241–2252.

[53] Z. Xu, X. Zhang, L. Chen, K. Pei, and B. Xu, "Python probabilistic type inference with natural language support," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 607–618.

[54] J. Wang, L. Li, K. Liu, and H. Cai, "Exploring how deprecated python library apis are (not) handled," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 233–244.

[55] C. Chen, X. Peng, Z. Xing, J. Sun, X. Wang, Y. Zhao, and W. Zhao, "Holistic combination of structural and textual code information for context based API recommendation," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2987–3009, 2021.

[56] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 631–642.

[57] W. Li, L. Li, and H. Cai, "PolyFax: A toolkit for characterizing multi-language software," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Tool Demos*, 2022, pp. 1662–1666.

[58] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 858–868.

[59] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[60] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? A large-scale study of Stack Overflow posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[61] Z. Wang, T.-H. P. Chen, H. Zhang, and S. Wang, "An empirical study on the challenges that developers encounter when developing Apache Spark applications," *Journal of Systems and Software*, vol. 194, p. 111488, 2022.

[62] "Popular programming languages on stack overflow," 2021. [Online]. Available: https://www.stackscale.com/blog/popular-programming-languages-2021/

[63] M. Grichi, "Towards understanding modern multi-language software systems," Ph.D. dissertation, Ecole Polytechnique, Montreal (Canada), 2020.

[64] M. Lopes and A. Hora, "How and why we end up with complex methods: A multi-language study," *Empirical Software Engineering*, vol. 27, no. 5, pp. 1–42, 2022.

[65] S. Buro, R. L. Crole, and I. Mastroeni, "On multi-language abstraction: Towards a static analysis of multi-language programs," in *Static Analysis: 27th International Symposium, SAS 2020, Virtual Event*, 2020, pp. 310–332.

[66] B. Einarsson and W. M. Gentleman, "Mixed language programming," *Software: Practice and Experience*, vol. 14, no. 4, pp. 383–395, 1984.

[67] M. Bagherzadeh and R. Khatchadourian, "Going big: A large-scale study on what big data developers ask," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 432–442.