Appendix for "Understanding Rust in Engineering Practice: An Empirical Investigation of Usability, Toolchain Support, and Ecosystem Development"

Contents

A Interview Questions

We list the core questions used in our semi-structured interviews below. We organized them by topic.

1. How did you first come into contact with Rust, and what made you decide to use it for your current project? If possible, could you briefly introduce the research area or the project you're working on?
2. Does using Rust affect your team's efficiency in testing and fixing bugs? Are there significant differences between Rust and other languages in terms of collaborative development?
3. What are the advantages and disadvantages of using Rust in actual projects?
4. When debugging Rust programs, what difficulties do you encounter compared to other languages?
5. How do you choose which third-party libraries to use in your project? Do you have a specific evaluation strategy for the quality and reliability of these libraries? Are third-party libraries and frameworks rich enough to fully meet your needs?
6. How would you rate the quality of Rust's standard libraries and third-party libraries? Are the features provided sufficient? How well-documented are they, and how easy is it to learn to use them?
7. How does the rapid evolution of the Rust community (e.g., new features and library updates) impact the stability of industrial projects?
8. What automated testing frameworks and tools have you used in your projects, and have they been helpful in debugging and optimizing your project?
9. How does Rust's compilation time affect your development experience? Do you find that the strict compiler checks become a productivity bottleneck in fast-paced development environments?
10. When integrating Rust with other programming languages like C/C++ or Python, have you encountered any unexpected bugs due to Rust's FFI (Foreign Function Interface)? Has it increased maintenance challenges?
11. Is it easy to introduce errors or issues when modifying Rust code?
12. What challenges have you faced with Rust's toolchain (rustc, cargo, rustup, etc.) in cross-platform development?
13. What challenges have you encountered when deploying Rust applications?
14. What difficulties did you encounter when learning and using Rust?
    - Basic knowledge (ownership system, lifetimes, error handling, generics, traits, macros, concurrency models, etc.)
    - Poor code readability
    - Compiler error messages
    - Insufficient learning resources

- Poor quality of library documentation
- Community support (response rate, effectiveness of responses)

15. Regarding Rust's unique designs (such as those mentioned in the basic knowledge section), in terms of security and usability, which aspects do you think need further improvement?
16. Do you have any good suggestions for beginners learning Rust?
17. Have you encountered any issues when communicating on community platforms? What aspects do you think need further improvement?
18. In order to get more people to know and use Rust as their project development language, what areas do you think Rust needs to change or improve?
19. From a technical or theoretical perspective, what do you think are the current research hotspots or controversies related to Rust?

B Questionnaire Content

We present the full content of the online survey used in this study. We designed these questions based on the findings from our interviews. We list them here to ensure the transparency of our data collection process.

Q1. *(Single choice)* What is the size of your current organization?

- ☐ 0–10 people
- ☐ 10–50 people
- ☐ More than 50 people

Q2. *(Single choice)* How many years of programming experience do you have?

- ☐ 0–2 years
- ☐ 2–5 years
- ☐ More than 5 years

Q3. *(Multiple choice)* In what types of projects are you currently using the Rust programming language?

- ☐ Systems programming
- ☐ Web development
- ☐ Game development
- ☐ Data processing and analysis
- ☐ Blockchain and cryptography
- ☐ Network programming
- ☐ Cloud computing and microservices
- ☐ Tool development
- ☐ Others (please specify)

Q4. *(Multiple choice)* What do you consider to be the major difficulties in learning and using Rust?

- ☐ Fundamental concepts are difficult to master, such as ownership, lifetimes, macros, and traits
- ☐ Error messages are difficult to interpret, and locating the underlying issue can be time-consuming despite their detail
- ☐ The toolchain is not sufficiently mature; some IDE plugins or tools are not well developed or well integrated

- ☐ Third-party libraries or frameworks are less extensive compared with other programming languages
- ☐ Learning resources are insufficient, particularly introductory video tutorials, books, or Chinese-language materials
- ☐ Documentation is not sufficiently clear; official documentation lacks detailed explanations for certain advanced features or complex concepts
- ☐ Community support is not always timely; responses to questions may be slow and effective solutions may be difficult to find
- ☐ Others (please specify)

Q5. *(Multiple choice)* What do you consider to be the most helpful resources during your process of learning and using Rust?

- ☐ Community
- ☐ Documentation
- ☐ Large language models
- ☐ Discussions with colleagues or classmates
- ☐ Project mentors
- ☐ Online courses or video tutorials
- ☐ Others (please specify)

Q6. *(Multiple choice)* Which aspects do you find challenging during the initial stage of learning Rust?

- ☐ Understanding the ownership system, including ownership transfer and borrowing rules
- ☐ Understanding the use of lifetimes
- ☐ Understanding generics and trait constraints
- ☐ Understanding the application scenarios of `Result` and `Option`
- ☐ Understanding the organization of modules and crates
- ☐ Learning how to use Cargo to manage dependencies and build projects
- ☐ Learning the concurrency model, such as Tokio
- ☐ I did not encounter significant difficulties and was able to get started quickly
- ☐ Others (please specify)

Q7. *(Multiple choice)* Which aspects of Rust's unique design do you think require adaptation?

- ☐ Adapting to the strictness of the borrow checker
- ☐ Managing lifetimes correctly in complex scenarios
- ☐ Understanding how Rust ensures memory safety through ownership and borrowing
- ☐ Adapting to Rust's strong type system and type inference
- ☐ Handling pattern matching for complex data structures, such as nested enums and structs
- ☐ Understanding and using Rust's iterators and closures
- ☐ Understanding the use of declarative and procedural macros
- ☐ Adapting to module paths and understanding visibility rules
- ☐ Adapting to Rust's error-handling model, such as `Result` and `Option`
- ☐ Mastering Rust's concurrency model, such as Tokio
- ☐ Adapting to the foreign function interface (FFI) for cross-language interoperability
- ☐ Mastering disciplined and safe practices for writing `unsafe` code
- ☐ Others (please specify)

Q8. *(Multiple choice)* Do you find the Rust compiler's error messages helpful for learning?

- ☐ Very helpful; the compiler provides clear error messages and suggestions for resolution
- ☐ Helpful; the error messages allow me to locate issues quickly
- ☐ Moderately helpful; the error messages are acceptable but sometimes lack clarity
- ☐ Not helpful; the error messages are not intuitive and are difficult to understand
- ☐ Not helpful at all; the error messages make the learning process more confusing

Q9. *(Multiple choice)* When debugging Rust programs, what difficulties do you encounter compared with other programming languages?

- ☐ Debugging tools are less mature than those for C++ or Java, and configuration issues may occur especially on Windows

- ☐ Rust's borrow checker enforces memory-safety rules at compile time, requiring an understanding of variable lifetimes and scopes during debugging, which is usually unnecessary in many other languages
- ☐ When working with `unsafe` code (such as calling C libraries or operating with raw pointers), Rust's memory-safety guarantees no longer apply, leading to issues similar to those in C/C++
- ☐ Rust's concurrency model (such as `async/await` or threading) is safe but difficult to debug; asynchronous stack traces are often unclear and diagnosing deadlocks can be complex
- ☐ Rust's macros (such as `macro_rules!` or procedural macros) are difficult to trace during debugging because the expanded code may differ significantly from the original source
- ☐ Rust's error-handling mechanisms (such as `Result` and `Option`) are powerful, but in complex programs the propagation chain of errors may complicate debugging, particularly when multiple `?` operators are involved
- ☐ Others (please specify)

Q10. *(Multiple choice)* Which program analysis tools do you frequently use in your Rust projects?

- ☐ Clippy
- ☐ Cargo Bench
- ☐ Cargo Flamegraph
- ☐ Valgrind
- ☐ Miri
- ☐ Loom
- ☐ Tokio-console
- ☐ Tarpaulin
- ☐ I rarely use analysis tools

Q11. *(Multiple choice)* What aspects of the Rust analysis tool ecosystem do you think need the most improvement or supplementation?

- ☐ Providing more performance analysis tools
- ☐ Enhancing memory analysis and leak detection capabilities
- ☐ Improving compatibility and integration support between tools
- ☐ Providing more user-friendly UI or visualization tools

- ☐ Enriching documentation and tutorials for analysis tools to reduce the learning curve
- ☐ Enhancing integration with IDEs to improve development efficiency
- ☐ Others (please specify)

Q12. *(Multiple choice)* What new program analysis techniques or tools would you like to see in Rust?

- ☐ Performance optimization
- ☐ Memory leak detection
- ☐ Concurrency issue troubleshooting
- ☐ Code standardization
- ☐ Lexical and syntactical analysis tools
- ☐ Others (please specify)

Q13. *(Multiple choice)* Which automated testing frameworks and tools do you primarily use in your Rust projects?

- ☐ Rust's built-in testing framework
- ☐ Mockito
- ☐ QuickCheck
- ☐ Proptest
- ☐ Tarpaulin
- ☐ cargo-lcov
- ☐ criterion.rs
- ☐ assert-cmd
- ☐ I rarely use these automated testing frameworks and tools
- ☐ Others (please specify)

Q14. *(Multiple choice)* What is the primary purpose of using automated testing frameworks and tools in your projects?

- ☐ Unit testing
- ☐ Integration testing
- ☐ Performance testing

- ☐ Regression testing
- ☐ Code coverage analysis
- ☐ Others (please specify)

Q15. *(Multiple choice)* What are the main challenges you encounter during project testing?

- ☐ Instability of test cases; test cases often fail due to code changes, leading to high maintenance costs
- ☐ Insufficient test coverage; certain modules or features have inadequate test coverage, making comprehensive testing difficult
- ☐ Environment configuration issues; complex test environment setup makes cross-environment or cross-platform testing challenging
- ☐ Performance issues; automation test execution times are too long, affecting development efficiency and continuous integration speed
- ☐ Limitations of testing frameworks and tools; the current testing frameworks or tools do not meet certain specific requirements
- ☐ Others (please specify)

Q16. *(Multiple choice)* How do you think the challenges encountered during project testing can be reduced or resolved?

- ☐ Improve testing frameworks and tools to enhance test coverage and support for testing asynchronous code performance, etc.
- ☐ Increase the scalability and stability of automated tests to reduce frequent failures caused by code changes
- ☐ Optimize test execution time to improve testing efficiency
- ☐ Enhance automation of test environment configuration and cross-platform support
- ☐ Others (please specify)

Q17. *(Multiple choice)* What aspects of Rust's design do you like the most?

- ☐ Consistent syntax design
- ☐ Powerful type system and static type checking
- ☐ Ownership and borrowing mechanisms
- ☐ Rich functionality, enabling both low-level and high-level programming
- ☐ Strong concurrency model

- ☐ I think Rust's design is average, with nothing I particularly like
- ☐ Others (please specify)

Q18. *(Multiple choice)* What aspects of Rust's design do you dislike?

- ☐ The compiler checks are too strict
- ☐ Lifetime management
- ☐ Error handling
- ☐ Using Rust's macro system
- ☐ Handling and debugging panics
- ☐ I really like Rust's design, and currently, I don't think there are any drawbacks to it
- ☐ Others (please specify)

Q19. *(Single choice)* How has the ownership and borrowing mechanism in Rust affected your development experience?

- ☐ Very helpful; it helps me manage memory better and reduces errors
- ☐ Somewhat helpful; although it is complex, I can understand it and benefit from it
- ☐ Average; it was somewhat difficult to understand, but I have gradually adapted to it
- ☐ Somewhat obstructive; the ownership system confuses or inconveniences me
- ☐ Very obstructive; I find this mechanism overly complicated and it impacts development efficiency

Q20. *(Multiple choice)* Which design aspects of Rust do you think affect its ease of use?

- ☐ Ownership and borrowing rules; managing memory through ownership and borrowing is difficult to learn
- ☐ Lifetime management; Rust enforces explicit management of lifetimes
- ☐ Error handling model; Rust enforces explicit error handling
- ☐ Design of generics and traits; they make the code more complex, and beginners need time to master them
- ☐ Default immutability; all variables are immutable by default in Rust
- ☐ Concurrency and asynchronous programming; the complexity and concurrency safety requirements pose challenges for many developers, especially in multi-threading and asynchronous programming

- ☐ Rust compiler is very strict and has high requirements for the program; although the error messages are detailed, they can sometimes be cumbersome and hard to pinpoint errors
- ☐ Incomplete toolchain and IDE support
- ☐ Others (please specify)

Q21. *(Multiple choice)* What areas do you think Rust could improve in terms of code readability?

- ☐ Provide Rust coding standards
- ☐ Improve documentation comment tools
- ☐ Provide more code formatting tools)
- ☐ Others (please specify)

Q22. *(Multiple choice)* What areas do you think Rust could improve in terms of code maintainability?

- ☐ Provide better refactoring tool support
- ☐ Improve dependency management tools
- ☐ Provide more comprehensive testing tools
- ☐ Improve compile-time error messages
- ☐ Others (please specify)

Q23. *(Multiple choice)* Which areas are most prone to errors or issues when modifying code?

- ☐ Dependency management
- ☐ Changes to function signatures
- ☐ Lifetime issues
- ☐ Modifications to concurrent code
- ☐ Others (please specify)

Q24. *(Multiple choice)* What challenges have you encountered when developing Rust for cross-platform projects?

- ☐ Lack of official support for certain operating systems/platforms
- ☐ Dependency issues during the build process
- ☐ Longer compilation times or performance degradation on certain platforms

- ☐ Insufficient cross-platform debugging support
- ☐ Inconsistent behavior of compiled programs across platforms
- ☐ Interoperability with other languages, including safe usage of FFI, memory management, and lifetime handling
- ☐ Insufficient documentation or community support, making cross-platform issues difficult to resolve
- ☐ Other issues (please specify)

Q25. *(Multiple choice)* What issues have you encountered during the deployment of Rust applications?

- ☐ Dependency management issues: certain libraries or dependencies fail to build or are incompatible in production environments
- ☐ Long compilation times: Rust applications have long compilation times, affecting the deployment cycle
- ☐ Cross-platform deployment difficulties: compatibility issues between different operating systems or platforms, leading to deployment failures or instability
- ☐ Containerization/virtualization issues: problems encountered when deploying in Docker or virtual machines, such as image building, dependencies, etc.
- ☐ Complex configuration management: difficulty in managing configuration files and maintaining consistency across multiple environments
- ☐ Others (please specify)

Q26. *(Multiple choice)* In which areas would you like to see improvements in Rust's deployment process?

- ☐ Provide simpler cross-platform deployment tools or support
- ☐ Improve compilation speed to reduce waiting times during deployment
- ☐ Enhance containerization support and simplify integration with tools like Docker
- ☐ Provide better automated deployment tools or scripts
- ☐ Others (please specify)

Q27. *(Single choice)* Do you think the official documentation is sufficient to help you understand Rust?

- ☐ Yes, the official documentation is clear and comprehensive, and almost answers all questions
- ☐ In most cases, the official documentation resolves my issues, but sometimes I need to consult other resources

- ☐ Average; the official documentation addresses basic issues but lacks detail for deeper content or complex problems
- ☐ Not sufficient; the official documentation is not clear, and some concepts are difficult to understand, requiring reliance on external resources
- ☐ Completely insufficient; the official documentation hardly answers my questions, and learning Rust relies heavily on other sources

Q28. *(Single choice)* Do you think the third-party libraries and frameworks in the Rust ecosystem are rich enough?

- ☐ Very rich; I can almost find all the functionality I need
- ☐ Sufficiently rich; most commonly used functionalities are supported
- ☐ Fairly rich, but some functionalities are still missing
- ☐ Not rich enough; many functionalities lack suitable library support
- ☐ Very limited; the ecosystem is immature, and there are few library options

Q29. *(Multiple choice)* What do you think is the main reason when you cannot find a relevant support library or have trouble using a library correctly?

- ☐ Cannot find the corresponding library, possibly due to the library's name being unintuitive or lacking clear identification
- ☐ The relevant library has not been widely supported in the Rust ecosystem
- ☐ The functionality I need has no corresponding library, requiring me to implement it myself
- ☐ The library documentation and functions are unclear, lacking complete examples or explanations
- ☐ Could not find the official documentation, or the documentation does not cover the functionality I need
- ☐ The library's API design is complex, and there is a lack of easy-to-understand Chinese documentation or tutorials
- ☐ Unclear how to correctly import the library, and there is no guidance on how to do so
- ☐ Others (please specify)

Q30. *(Single choice)* Do you think the third-party libraries in Rust have sufficient documentation and example code?

- ☐ Very sufficient; almost all libraries have detailed documentation and example code

- ☐ Average; most libraries have adequate documentation and example code
- ☐ Insufficient; some libraries have brief documentation and limited example code
- ☐ Very insufficient; many libraries lack documentation or example code, making them difficult to use

Q31. *(Multiple choice)* What factors do you consider most when choosing a third-party library?

- ☐ The library's stability and activity
- ☐ Whether the documentation is complete and easy to learn
- ☐ Whether the library's performance meets project requirements
- ☐ Whether there is widespread community support and discussion
- ☐ Whether it aligns with my coding style and project requirements
- ☐ Whether it is easy to integrate with other libraries or toolsets
- ☐ Others (please specify)

Q32. *(Single choice)* How do you find the async programming libraries in Rust?

- ☐ Very good; the async programming libraries are feature-complete and meet most of my needs
- ☐ Average; the async programming libraries meet basic needs, but I occasionally encounter issues
- ☐ Not very good; the async programming libraries are cumbersome to use and unstable
- ☐ Difficult to learn and master the async programming libraries
- ☐ I do not use async programming and will not provide an opinion

Q33. *(Multiple choice)* What community support issues have you encountered during your Rust development?

- ☐ The documentation is not detailed enough, making it difficult for the community to provide sufficient help
- ☐ Community discussions are scattered, making it hard to find answers to related issues
- ☐ The feedback speed from the community is slow, and it usually takes a long time to get effective responses
- ☐ The community atmosphere is not friendly, and negative feedback is often encountered when asking questions or discussing topics

- ☐ Lack of Chinese support, and many technical articles and discussions require reliance on translation tools
- ☐ I have not encountered such issues; community support has been smooth
- ☐ Others (please specify)

Q34. *(Multiple choice)* What deficiencies or issues do you think exist in Rust learning resources?

- ☐ The coverage is not broad enough, lacking a complete system from beginner to advanced topics
- ☐ Insufficient in-depth materials on advanced topics
- ☐ Scarcity of resources in specific fields
- ☐ Poor availability of resources, especially a lack of high-quality Chinese materials
- ☐ There is a sufficient quantity of resources, but their quality varies, making it difficult to find reliable learning materials
- ☐ Others (please specify)

C Full Prompt Template and Annotation Configuration

C.1 Prompt Template

The following is the complete prompt template used for assigning multiple tags using a large language model. It corresponds to the code used in our data processing script for this study.

- **Task.** Analyze a post about Rust. It contains a title and a description. Generate descriptive tags that reflect the problems, needs, or topics. Focus on understanding the meaning of the post rather than simply matching keywords to a fixed list of labels.
- **Consider broad thematic areas.**

  1.Core language features: Fundamental constructs of the Rust language determine how programs manage memory and ensure safety. They help structure code, handle data, and control flow, they support the development of modular, maintainable, and concurrent software systems.

  2.Learning resources and community support: Availability and quality of materials, guidance, and social support that help developers learn Rust, solve problems, and collaborate with the wider community;

  3.Development experience and tools: Overall developer workflow, including how programming tools, compilers, IDEs, testing, debugging, and dependency management affect productivity, code quality, and problem-solving;

  4.Use cases and cross-platform support: This includes practical uses of Rust in many different scenarios, environments, and platforms. It also covers the challenges developers face when they adapt Rust to specific hardware, operating systems, or limits of the system.

  5.Performance and language comparison: Efficiency, reliability, and practical performance of Rust programs in real-world contexts, as well as conceptual or practical comparisons with other programming languages;

  6.Ecosystem and future development: This covers how libraries, frameworks, tools, and documentation in the Rust ecosystem grow and improve. It also looks at trends, future changes, and gaps that affect whether teams adopt Rust and how productive they remain over time.
- **Instructions.** Carefully read the post content and identify the key issues or topics; Generate 2-4 descriptive tags that summarize the essence of the post, based on the semantic understanding of the content; If the content is vague or cannot be clearly classified, use fallback tags: [“core language features”, “learning resources”, “development experience”, “application scenarios”, “performance comparison”, “ecosystem future”]
- **Output format.** Tags: [“tag1”, “tag2”, ...]

C.2 LLM Configuration

We used the DeepSeek model to perform multi label annotation tasks that relied on semantic understanding. The specific configuration details are as follows.

- Model: DeepSeek-V3.1
- Max tokens: 200
- Temperature: 0.6
- Retry strategy: up to 3 attempts with exponential backoff
- Inference mode: Chat completions API, non-streaming

D Model Parameters for Label Aggregation

This section provides the detailed configuration for the algorithms used in this study. We include these parameters to ensure the reproducibility of our results.

D.1 Model and Embedding Configuration

- Model: all-mpnet-base-v2
- Batch size: 64
- Normalize embeddings: True
- Convert to numpy: True

D.2 HDBSCAN Configuration

- Min cluster size: 12
- Min samples: 5
- Cluster selection method: “eom”
- Prediction data: True
- Low-frequency Filter : 5