Contents

1 Introduction

Rust is a programming language that focusses on performance, security an reliability. The *Rust Project* is open source and dual-licensed under the Apache 2.0 and MIT license (Hoare and al. 2019). Rust 1.0 the first stable version was announced in May 2015 (The Rust Core Team 2015). This release also marked the start of the *commitment to stability* which promises stability on future Rust stable releases (Turon and Matsakis 2014). This new commitment also introduced a 6-week release cycle as well as development channels for language users and early adapters (Turon and Matsakis 2014). The latest stable compiler version 1.51.0 has been released on 25 of March 2021 (The Rust Core Team 2021). Developers and teams within the project put high effort into open communication. This focussed is formalized in the official *Code of conduct* (The Rust Team 2021b). The language with its connected tools has attracted over 5900 individual contributes as of writing this (The Rust Team 2021a).

The Rust project develops several tools besides the compiler itself. These tools are seen as a vital part in automating parts of the development process and collaboration among teams. *Clippy* is the official linter for Rust and is being developed in the *rust-clippy* repository. The linter contains over 450 lints which span from complexity and style lints over to restriction lints which might be required by certificates (The Rust Clippy Developers 2021). Clippy is written in Rust itself and interfaces with the compiler directly. This direct connection enables the use of the existing lexer, parser and connected diagnostic tools and ensures that the project stays up to date with the latest compiler changes. Since 2018 Clippy is distributed as a component of the Rust installation itself (Lusby 2018).

1.1 Problem

Clippy, like the compiler, puts great effort into giving helpful diagnostic messages. These messages consist out of a message, a direct quotation of the suboptimal code fragment and a direct suggestion how the code quality can be improved. The first message of a lint additionally includes a note why which lint is enabled in this location and a reference to the lint documentation in Clippy's lint list. This makes Clippy's lint list the second point of contact for new users with the project itself. The initial load time of the lint list is noticeably slower than most other websites in the Rust eco system.

Another pain point of the website are some technical deficiencies. The development documentation of the Rust project includes some technical guidelines that the current lint list doesn't fully fulfill.

1.2 Research question

The described problem in 1.1 leads to the following research question: *How can the internet presentation of the lint list for the rust-clippy project be improved?*

1.3 Overall goal

The primary goal of this paper is to review the previously mentioned problems and to find solutions for them.

1.4 Methodical

The start of this paper will collect the formal and informal specifications that Clippy's lint list should optimally fulfill. The next part will analyze the current state of the website. This chapter will conclude with a list of aspects that should be improved as well as a selection which this research will focus on.

The previous preparation leads to the solution finding process. This process will start with a technical explanation of the aspect to determine why this specification is important. The next step tries to find a solution for the found issues under the given circumstances. Additionally, this section will include a practical test if the suggested change would improved the focussed problem.

This paper will end with a summary of the reviewed aspects and a conclusion as well as a suggestion what further work can be done on this topic.

2 Requirements

The goal of this work is to improve the impression of Clippy's lint list. This section of the document will set a list of requirements to focus on in further research for this paper. These aspects are split up into:

- *Functional requirements*:
- *Non-functional requirements*:

This section is used to define a scope for further work. Listing all requirements which have already been implemented would draw focus from the key points. It is on the other hand important to keep them in mind. All requirements that have been implemented previously will therefor be summarized in the following functional requirement: *The functionality of the website should not be impacted by the implementation of new measures to improve the impression or usage.* This requirement covers functionality like the search feature, filter options and theming. The implementation of all of these has been completed at the point of writing this.

2.0.1 Functional requirements

- List formal requirements
- Say: These formal requirements have also been defined as formal specifications:
- *Formal specifications*: Specifications that are unambiguous and formally defined. Formal specifications have the advantage that the precise description and limited scope reduce misunderstandings. These specifications are usually a translation of user requirements which are often expressed in natural language. This translation forces a detailed analysis of the requirements which often catches errors in them, that were previously hidden by ambiguous language. The fulfillment of these specifications can also be verified using manual or tool-supported methods (Sommerville 2010, p. 334).
  - Technical security measured by Mozilla Observatory

2.0.2 Informal specifications

- We need SPEED
  - Why are these informal? This paper want's to focus and work on the technical background not on fining the most optimal solution

2.0.3 Focus

Or summary what ever works better

3 Measuring the specifications

3.1 Running benchmarks

1. Simple *Mozilla Observatory*
2. Load times -> difficult
    - Comparing only on one device as this load time is significant and we want significant improvements
    - rustfmt's website shows that fast loading times are possible

3.2 Summary of benchmarks

Hello expedia

4 Analysis of benchmark results

4.1 Technical problems

- Explaining the grade C from *Mozilla Observatory*
- This should definitely include scientific sources to make this a valid paper
    - The examiner noted that the paper outline seems interesting but that I need to take care to include scientific sources
- Explanation why the listed security risks are security risks

4.2 Slow loading times (Browser debug tools)

Hello

5 Solutions for unfulfilled specifications

- Reading GH page documentation
- Maybe contacting support

Hello

6 Conclusion

Hello

References

Hoare, Graydon and et al. (2019-01). *COPYRIGHT*. URL: https://github.com/rust-lang/rust/blob/master/COPYRIGHT (visited on 2021-04-20).

Lusby, Jane (2018-07). *Add clippy to the tools list #1461*. URL: https://github.com/rust-lang/rustup/pull/1461 (visited on 2021-04-17).

Sommerville, Ian (2010). *Software Engineering*. 9th edition. 501 Boylston Street, Suite 900, Boston, Massachusetts 02116: Pearson Education, Inc. ISBN: 978-0-13-703515-1.

The Rust Clippy Developers (2021-03). *Clippy*. URL: https://github.com/rust-lang/rust-clippy/blob/7fcd1/README.md (visited on 2021-04-17).

The Rust Core Team (2015-05). *Announcing Rust 1.0*. URL: https://blog.rust-lang.org/2015/05/15/Rust-1.0.html (visited on 2021-04-17).

– (2021-03). *Announcing Rust 1.51.0*. URL: https://blog.rust-lang.org/2021/03/25/Rust-1.51.0.html (visited on 2021-04-17).

The Rust Team (2021a-04). *All-time Contributors*. URL: https://thanks.rust-lang.org/rust/all-time/ (visited on 2021-04-20).

– (2021b). *Code of conduct*. URL: https://www.rust-lang.org/policies/code-of-conduct (visited on 2021-04-20).

Turon, Aaron and Niko Matsakis (2014-10). *Stability as a Deliverable*. URL: https://blog.rust-lang.org/2014/10/30/Stability.html (visited on 2021-04-20).

7 Attachments