# rustyRTS: Regression Test Selection for Rust

## Context

### Rust

Rust is a relatively new systems programming language, designed to enforce safety (e.g. in the context of memory allocation or parallelism) while being competitive in terms of performance [1].

It comes with an incremental compiler, written in Rust itself. On compilation, the source code is transformed and optimized through several intermediate representations. One such representation is called the *Mid-level Intermediate Representation* (MIR). The MIR consists of so-called *bodies*, which correspond to functions in source code. Using a compiler driver, the MIR may be analyzed or even modified during compilation. [2]

### Regression Test Selection

Regression Test Selection (RTS) tries to minimize testing time by only re-executing tests that are affected by changes in the code. Existing approaches either try to overestimate these dependencies based on static analysis or use information on dependencies, which is dynamically collected during a previous execution. RTS is considered *safe* if it succeeds to selects all tests that are affected by changes. [3]

To identify changes in source code, some existing solutions for RTS for other languages apply specialized checksums [4,5,6]. The Rust compiler essentially does the same to determine which parts of the code need to be recompiled [2]. Its checksum subsystem may also be used in an approach to RTS for Rust.

#### Static

Similar to existing static RTS tools for other languages [3,5,6], static RTS for Rust may be realized by generating a dependency graph based on the MIR. By tracking modified bodies through this graph, the affected tests can be identified.

#### Dynamic

By injecting additional function calls into the MIR, Rust code may be instrumented to trace bodies that are executed (i.e. functions that are called) during the execution of a test. When these traces and the set of modified bodies overlap, the corresponding test is affected and needs to be executed again.

### Mutation Testing

Mutation Testing is a technique for evaluating the quality of a test set. Certain faults are purposefully introduced into the source code of a program, resulting in so-called *mutants*. The tests are then executed on these mutants, to check whether the tests are able to detect the introduced faults or not. The output of mutation testing is a score called the *mutation score*, which denotes the proportion of detected faults. [7]

As described by Zhu et al. [8], Mutation Testing may be used to evaluate RTS tools. During their study on RTS for Java however, Mutation Testing turned out to be infeasible due to the fact that no suitable mutation tool for Java has been available [8]. For Rust, in contrast a tool called *cargo mutants* exists which appears to be well suited and, when adapted slightly, may assist in evaluating RTS for Rust.

## Problem & Research Gap

Despite Rust having a vivid and open-source ecosystem [1], there is currently no strategy or tool for Regression Test Selection in Rust publicly available. Consequently, there are no previous studies on the effectiveness of RTS in this

language.

## Contribution

This research project contributes to research by presenting and evaluating two different approaches to RTS in Rust (dynamic and static).

## Working Plan

1. Research related literature on mutation testing and evaluation strategies for regression test selection tools
2. Design and implement a system for evaluating the safety, precision and performance of RTS for Rust: a) Apply RTS to open-source projects, record which tests are executed along with the execution time of the tests and the end-to-end testing time b) Use Mutation Testing to verify that all tests failing in consequence of a mutation are actually selected by RTS

## Research Questions

1. Are there some tests that are not selected even though they fail because of a changes in source code (e.g. a mutation)? If this is the case, what is the reason for these tests not being selected?
2. How much does dynamic respectively static RTS for Rust decrease the number of executed tests? Is there a difference in selection ratio between unit and integration tests?
3. How much performance overhead does RTS for Rust generate? Does it provide an improvement in end-to-end testing time in comparison to retest-all?

## References


[1] Bugden, W., & Alahmar, A. (2022). Rust: The Programming Language for Safety and Performance. *Igscong'22*, *June*.
[2] The Rust Project Developers. (2018). Rust Compiler Development Guide (rustc-dev-guide) [Computer software]. https://github.com/rust-lang/rustc-dev-guide
[3] Legunsen, O., Hariri, F., Shi, A., Lu, Y., Zhang, L., & Marinov, D. (2016). An extensive study of static regression test selection in modern software evolution. *Proceedings of ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 583–594. https://doi.org/10.1145/2950290.2950361
[4] Gligoric, M., Eloussi, L., & Marinov, D. (2015). Ekstazi: Lightweight Test Selection. *Proceedings - International Conference on Software Engineering*, *2*, 713–716. https://doi.org/10.1109/ICSE.2015.230
[5] Legunsen, O., Shi, A., & Marinov, D. (2017). STARTS: STAtic regression test selection. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 949–954. https://doi.org/10.1109/ASE.2017.8115710
[6] Fu, B., Misailovic, S., & Gligoric, M. (2019). Resurgence of Regression Test Selection for C++. *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, 323–334. https://doi.org/10.1109/ICST.2019.00039
[7] Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. In *IEEE Transactions on Software Engineering* (Vol. 37, Issue 5, pp. 649–678). https://doi.org/10.1109/TSE.2010.62
[8] Zhu, C., Legunsen, O., Shi, A., & Gligoric, M. (2019). A Framework for Checking Regression Test Selection Tools. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 430–441. https://doi.org/10.1109/ICSE.2019.00056
[9] Pool, M., Moka D., Godwin S., Diekmann T. (2022). cargo-mutants [Computer software]. https://github.com/sourcefrog/cargo-mutants