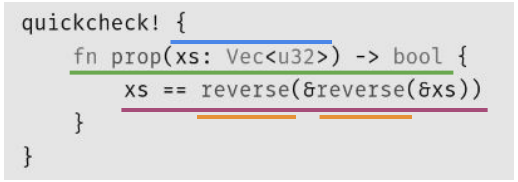
# Property Based Testing in Rust, How is it Used?

A case study of the `quickcheck` crate used in open source repositories

**Max Derbenwick**
M.A.Derbenwick@student.tudelft.nl
BSc Computer Science and Engineering
Supervisors: Sára Juhošová, Andreea Costea

TU Delft — Delft University of Technology

## Introduction



```
quickcheck! {
    fn prop(xs: Vec<u32>) -> bool {
        xs == reverse(&reverse(&xs))
    }
}
```

**Properties** are tested on many inputs and should hold true [2]
**Inputs** are arbitrarily generated using a **generator**
**Invariants** are what should hold true of the system for all inputs
**System Under Test (SUT)** is the code/system being tested

## Research Questions

Our research explores the following for Quickcheck in Rust:

- What themes emerge in the properties being tested?
- How are these properties implemented?
- What role does property-based testing (PBT) play in the overall testing and correctness guarantee within software repositories?
- How and when are generators implemented?
- In which cases is shrinking support explicitly added?

## References

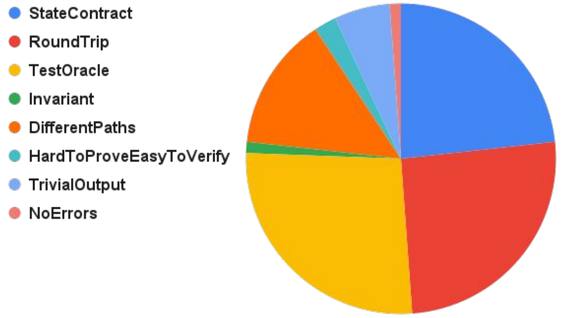
[1] Rashina Hoda. Qualitative Research with Socio-Technical Grounded Theory. Springer Cham, Cham, Switzerland, September 2024.

[2] David MacIver. What is property based testing? https://hypothesis.works/articles/what-is-property-based-testing/, May 2016.

[3] Scott Wlaschin. Choosing properties for property-based testing. https://fsharpforfunandprofit.com/posts/property-based-testing-2/, Dec 2014.

## Findings

| Repository | Regular Test Count | PBT Count | Expanded PBT Count |
|---|---|---|---|
| indexmap | 106 | 33 | 33 |
| time | 506 | 61 | 61 |
| regex | 398 | 5 | 5 |
| itertools | 140 | 203 | 203 |
| memchr | 920 | 24 | 115 |
| byteorder | 32 | 3 | 30 |
| http | 132 | 1 | 1 |
| h2 | 55 | 1 | 1 |
| crc32fast | 1 | 5 | 5 |
| flate2 | 62 | 5 | 5 |
| num-bigint | 205 | 47 | 47 |
| unicode-segmentation | 9 | 5 | 5 |
| bumpalo | 78 | 19 | 19 |

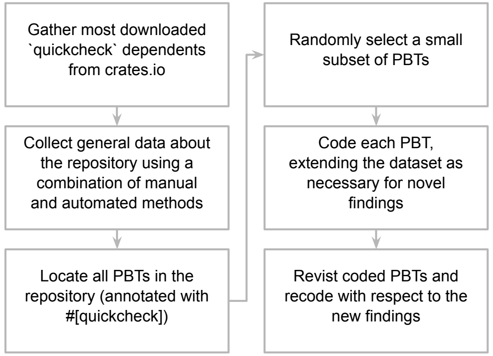


**State contract** PBTs verify SUT-specific contracts about a state after mutation

**Round trip** PBTs ensure that composing two inverse operations lead to an identity [3]

**Test oracle** PBTs compare the SUT against a reference implementation [3]

## Methodology



Gather most downloaded `quickcheck` dependents from crates.io

Collect general data about the repository using a combination of manual and automated methods

Locate all PBTs in the repository (annotated with #[quickcheck])

Randomly select a small subset of PBTs

Code each PBT, extending the dataset as necessary for novel findings

Revist coded PBTs and recode with respect to the new findings

## Analysis and Conclusions

1. **Simple and obvious** PBTs are prevalent.
2. Properties prefer **fewer assertions** and **fewer SUT invocations**.
3. PBTs make up a **small portion** of overall testing suites.
4. **Assumptions** (input filters) occur most prevalently within **test oracle** PBTs.
5. **Custom generators** appear almost exclusively **when the SUT is the input**.
6. Because the default behavior of *Quickcheck* arbitraries is not to shrink, **custom generators are usually paired with custom shrinkers**.