Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Testing Servo, the Parallel Browser Engine Project

David Schumann, Johannes Linke

Agenda

- Project Overview
- 5 V&V Questions
- Current Testing Status
- Test Automation
- Automatic Static Analysis

Project Overview

- New browser engine by Mozilla
- Long Term: Replace Firefox's current engine Gecko
- Goals in 2016:
  - Ship one Rust component in Firefox Nightly
  - Windows port
  - Explore new areas for performance improvements
  - Continue adding web platform features

5 V&V Questions

- When does V & V start? When is it complete?
  - Began with 6th commit, probably never ends.
- What particular techniques should be applied during development?
  - Generating and checking coverage support on every PR
- How can we assess the readiness of a product?
  - Hard to say for a research project. Main goal is still far away
- How can we control the quality of successive releases?
  - See 2. V&V Question. Mozillas internal processes are not known
- How can the development process itself be improved?
  - The process is already solid. Hard to improve without investing significant amount of time and money.

Current Testing Status

- ~110,000 LOC, ~11,000 tests
- Called by "./mach test"
  - ./mach test-wpt
  - ./mach test-unit
  - ./mach test-jquery
  - ./mach test-tidy
- Currently about 70-80 intermittently failing tests

Validation and Defect Testing

- Test suite is focused on validation testing
- Some scattered defect testing (e.g. acid2 test)

Development, Release and User Testing

- Hard to separate
- Core development team combines developers, release testers and users
- Outside perspective on readiness and validity might be missing

Unit-, Integration- and System-testing

- Unit tests only make up a fraction of the test suite
- Integration tests are used in some dependencies
- >95% of tests execute the servo binary and tests it's output

Test Automation - cargo test

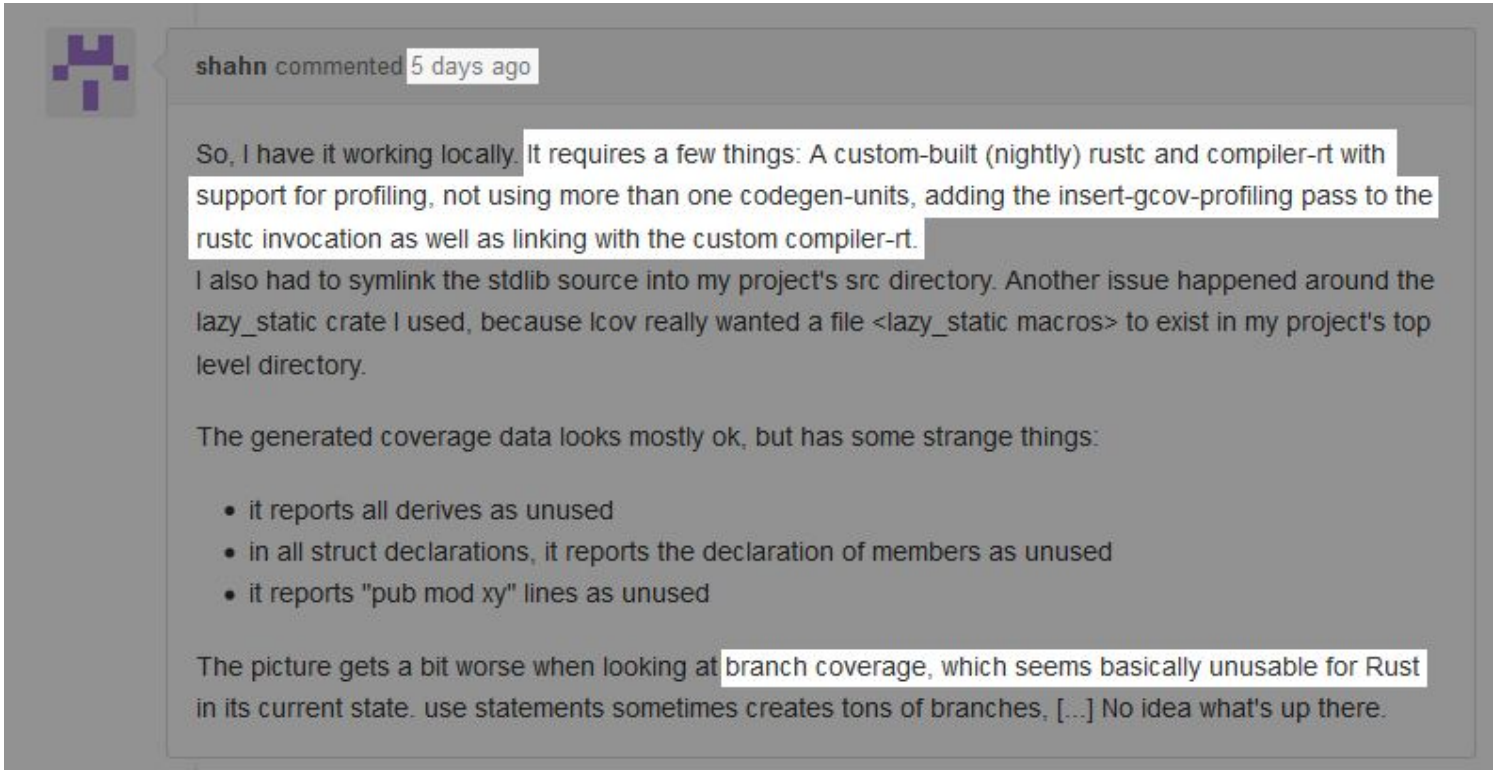
- To test your Rust program:
  - 1. Annotate method with #[test]
  - 2. run ‘cargo test’
  - 3. No third step required

```
johannes@johannes-VirtualBox:~/servo/servo-sync/components/gfx$ cargo test
     Running /home/johannes/servo/servo-sync/target/debug/deps/gfx_tests-9666be7e60be2090

running 6 tests
test text_util::test_transform_compress_whitespace ... ok
test text_util::test_transform_compress_none ... ok
test text_util::test_transform_compress_whitespace_newline ... ok
test text_util::test_transform_discard_newline ... ok
test text_util::test_transform_compress_whitespace_newline_no_incoming ... ok
test font_cache_thread::test_local_web_font ... ok

test result: ok. 6 passed; 0 failed; 0 ignored; 0 measured
```

Test Automation - gcov



shahn commented 5 days ago

So, I have it working locally. It requires a few things: A custom-built (nightly) rustc and compiler-rt with support for profiling, not using more than one codegen-units, adding the insert-gcov-profiling pass to the rustc invocation as well as linking with the custom compiler-rt.
I also had to symlink the stdlib source into my project's src directory. Another issue happened around the lazy_static crate I used, because lcov really wanted a file <lazy_static macros> to exist in my project's top level directory.

The generated coverage data looks mostly ok, but has some strange things:

- it reports all derives as unused
- in all struct declarations, it reports the declaration of members as unused
- it reports "pub mod xy" lines as unused

The picture gets a bit worse when looking at branch coverage, which seems basically unusable for Rust in its current state. use statements sometimes creates tons of branches, [...] No idea what's up there.

Test Automation - kcov

- Works on all binaries with debug symbols
- Inserts breakpoints
- Dead slow (30s per test)
- Broken accumulation

Test Automation - afl.rs

- American Fuzzy Lop in Rust
- Feeds input data over stdin to program under test
- Mutates input data to get branch coverage
- Tries to crash the program

Test Automation - doctests

```
johannes@johannes-VirtualBox:~/servo/doctests-tests$ cat src/lib.rs
/// This function adds two to its argument.
/// # Examples
/// ```
/// use doctests_tests::add_two;
/// assert_eq!(6, add_two(3));
/// ```
pub fn add_two(a: i32) -> i32 {
    a + 2
}
```

```
johannes@johannes-VirtualBox:~/servo/doctests-tests$ cargo test
   Doc-tests doctests-tests

running 1 test
test add_two_0 ... FAILED

failures:

---- add_two_0 stdout ----
thread '<main>' panicked at 'assertion failed: `(left == right)` (left: `6`, right: `5`)', <anon>:4
', ../src/librustdoc/test.rs:282

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured
```

Automatic Static Analysis: Ownership

```
fn foo() {
    let v = vec![1, 2, 3];
}
```

variable binding

'owned' by a scope

takes ownership of v

```
fn take(v: Vec<i32>) {
    // do stuff with v
}
let v = vec![1, 2, 3];
take(v);
```

'move' v into take

Automatic Static Analysis: Borrowing

take v by reference

```
fn borrow(v: &Vec<i32>) {
    // do stuff with v
}

let v = vec![1, 2, 3];

borrow(&v);
```

'borrow' v to method

Automatic Static Analysis: Prevented Bugs

- No accesses to uninitialized memory
- No double frees
- No use after free
- No null pointers
- No data races
- No iterator invalidation
- No memory leaks (kinda)
- No garbage collector