Auto-Analyzing Ruby

For Fun and Quality

About

Dave Raffensperger

@draffensperger

davidraff.com

cru.org

Why auto-analyze Ruby code?

- To enforce a style guide (rubocop)
- To catch security issues (brakeman)
- For smell checks/metrics (reek, rails_best_practices)
- What about catching potential bugs from bad use of dynamic typing? (Undefined method, etc.)
  - ruby-lint gem statically analyzes Ruby code to do type checks but won't see dynamically defined methods

Ruby code analysis is hard

- Ruby is awesomely dynamic and it's great for rapid development
- So a perfect `calls_undefined_method?` static analyzer is impossible:

```
require ‘ideal_ruby_static_analysis’

if calls_undefined_method?(__FILE__)
  # Wrong! Actually let’s define say_hello now..
  define_method :say_hello do
    puts 'Hello world!'
  end
else
  # Wrong! About to call undefined method say_hello
end

say_hello
```

Fewer of these would be nice ...

#101 NameError: NameError: uninitialized constant Paperclip::Storage::S3::AWS production

#662 NameError: NameError: undefined local variable or method `users' for #<AccountList::Merge:0x007f4c97a52cc0> Did you mean? user production

#251 NoMethodError: NoMethodError: undefined method `sync_contacts' for #<MailChimpAccount:0x007f95bafbbfb0> production

#884 NoMethodError: undefined method `id' for nil:NilClass production

- **Integration tests are surest way to quality check Ruby**
- But code analysis can 1) make unit tests smarter and 2) provide some safety in the absence of tests

Ruby code analysis toolbox

- **Reflection** lets you inspect the structure of methods at runtime. E.g. `Person.instance_method(:intro)`
- **The parser gem** that converts Ruby source code into an “abstract syntax tree” (simplified and easy to analyze representation of the code)
- **Ruby’s Tracepoint API** that lets you trace Ruby virtual machine events like method calls, method returns, exceptions, moving to a new line, etc. (powers the byebug gem)

Three ways to catch undefined methods

These combine reflection, the parser gem, and Tracepoint to:

1. Help unit tests catch inter-class bad method calls
2. Catch undefined instance methods without tests
3. Catch bad calls on method arguments via sampled types

Code Analysis #1: smarter RSpec verifying doubles

- You can have 100% unit test coverage and still get bad method calls in the “seams” between classes
- Verifying doubles (e.g. `instance_double`) help mitigate that
- A little static analysis we can make it even smarter

Code Analysis #2: Rake out bad method calls

- Ruby and especially Rails apps define a lot of methods at runtime
- So let's combine code parsing and runtime code loading (running the Rake task in the app environment)
- That allows us to catch (some) undefined instance methods even in the absence of specs
- Could easily add to your CI build (Travis, Jenkins, etc.)

Code Analysis #3: Check calls via sampled types

- Sample implicit method signatures in deployed app
- Git diff to invalidate possibly changed signatures based on callers
- Check local changes for bad calls on arguments based on sampled signatures
- Other possible uses for sampled types:
  - Use as the starting point for more complete type analysis (similar to ruby-lint gem)
  - Automatically convert RSpec doubles to instance_doubles
  - Generate YARD docs
- Sampling only a fraction of requests may make this practical on a real app

Questions?

Dave Raffensperger

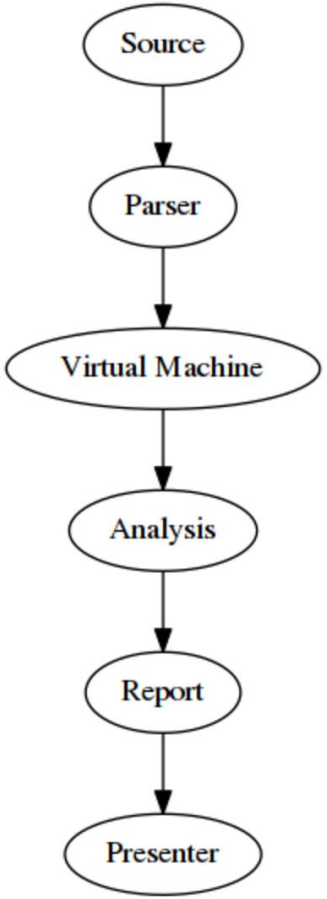

@draffensperger

davidraff.com

Gem and links to code snippets: github.com/draffensperger/type_tracer

Pure static analysis: ruby-lint gem

- Simulates code with a virtual machine to do type checking
- It doesn't load your code into its runtime (by design) and so can't see dynamically defined methods
- Has generated definitions files for Rails and a rake task to generate them for other gems



Combining dynamic and static code analysis

We'll cover the **static/dynamic Ruby analysis toolbox**, then go through ways to apply it to catch (some) bad method calls:

1. **Help unit tests catch bad inter-class "seam" method calls** by making RSpec's `instance_double` check for arguments to stubbed methods that would cause an undefined method error
2. **Check for undefined instance methods** with a Rake task run in the app's environment so it can handle dynamic method definitions
3. **Check a modification to a method for undefined calls on arguments** using type signatures sampled from a deployed app