Detecting Oxbow Code in Erlang Codebases with the Highest Degree of Certainty

Brujo Benavides

NextRoll Inc.

Laura M. Castro

Universidade da Coruña

Oxbow Code: An Origin Story

There was once a simple module...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A Simple Ticket...

Add sampling to lapp

Attach | Create subtask | Link issue | •••

Description

We want `lapp:run/1` to ignore a configurable percentage of the requests so we can A/B test the feature.

Activity

g | Zendesk Support | Oldest first ↑

Selected

YOUR PINNED FIELDS

Assignee	Unassign
Epic Link	None
Components	None
Labels	urgent

Somewhere in a Very Large Application™...

A parameter in the configuration

Pay attention to the configuration of the app.

```
{application,
 lapp,
 [{description, "A large application"},
  {vsn, "10.3.142"},
  {env, [{sample_rate, 0.5}]}]}.
```

Now a slightly more complex module...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

Now a slightly more complex module...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.
```

```
ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

Now a slightly more complex module...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.
```

```
ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

Now a slightly more complex module...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

But then, months after the previous ticket...

RTI-9409

Give feedback 1

Remove the sampling for lapp

Attach | Create subtask | Link issue

Description

The A/B tests proved that the feature works as needed.
We need to run the evaluation in all requests, effectively ignoring `sample_rate`.

Oldest first ↑

Selected

YOUR PINNED FIELDS

Assignee
Brujo Benavides

Epic Link
None

Components
None

Labels
urgent

NextRoll

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(Rand, Rate, _Param)
  when Rand >= Rate ->
  ignored;
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

ignore_test() ->
  application:set_env(
    lapp, sample_rate, 0.0),
  ?assertEqual(ignored, run(anything)).

evaluate_test() ->
  application:set_env(
    lapp, sample_rate, 1.0),
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
 maybe_evaluate(
  rand:uniform(), SampleRate, Param).
```

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

```
{application,
 lapp,
 [{description, "A large application"},
  {vsn, "10.3.142"},
  {env, [{sample_rate, 0.5}]}]}.
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

```
{application,
 lapp,
 [{description, "A large application"},
  {vsn, "10.3.142"},
  {env, [{sample_rate, 0.5}]}]}.
```

A seemingly simple change...

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

```
{application,
 lapp,
 [{description, "A large application"},
  {vsn, "10.3.142"},
  {env, [{sample_rate, 0.5}]}]}.
```

Success!

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-export([run/1]).

run(Param) ->
  Result = evaluate(Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

evaluate(Param) ->
  {evaluated, Param}.

evaluate_test() ->
  ?assertEqual({evaluated, anything},
               run(anything)).
```

```
{application,
 lapp,
 [{description, "A large application"},
  {vsn, "10.3.142"}]}.
```

Less code

*is **easier** to **maintain!***

Why don't we delete everything?

```
-module(lapp).
-include_lib("eunit/include/eunit.hrl").
-define(DEFAULT_SAMPLE_RATE, 0.25).
-export([run/1]).

run(Param) ->
  SampleRate =
    application:get_env(
      lapp, sample_rate,
      ?DEFAULT_SAMPLE_RATE),
  Result =
    maybe_evaluate(SampleRate, Param),
  logger:info(#{param => Param,
                result => Result}),
  Result.

maybe_evaluate(SampleRate, Param) ->
  maybe_evaluate(
    rand:uniform(), SampleRate, Param).
```

Is this used?

Is this checked?

```
maybe_evaluate(_Rand, _Rate, Param) ->
  evaluate(Param).

evaluate(Param) ->
  {evaluated, Param}.
```

Is this called?

I don't know.

Hank!

Hank!

A rebar3 plugin to detect oxbow code instances

Hank is...

Extensible

We defined ~10 rules to find different kinds of oxbow code, but you can define your own ones, too.

Accurate

Hank has dialyzer-levels of certainty: It doesn't produce false positives.

Configurable

You can analyze as much or as few code as you need.

Hank Rules

As of Hank 1.2.2, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

```
-module(my_module).
-export([new/0, used/1]).

-record(my_record, {used, unused}).

new() -> #my_record{}.

used(#my_record{used = Value}) -> Value.
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

```
-module(my_module).
-export([my_function/0]).

-define(UNUSED, unused).
-define(USED, used).

my_function() -> ?USED.
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

used.hrl

```
-define(A_MACRO, used_macro).
...
```

unused.hrl

```
-define(A_MACRO, unused_macro).
...
```

```
-module(my_module).
-include("used.hrl").

usage() -> ?A_MACRO.
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

sys.config

```
{my_app, [
    {used, “used”},
    {unused, “unused”}
]}.
```

```
-module(my_module).

usage() ->
    application:get_env(
        my_app,
        used,
        default_value
    ).
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

```
-module(my_module).

-export([external/1]).

external(Param) ->
    internal(Param, some:computation()).

internal(Used, _Unused) when Used > 0 ->
    {valid, Used};
internal(_Used, _Unused) ->
    invalid.
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

```
-module(my_module).

-export([used/1]).

-callback used() -> used.
-callback unused() -> unused.

used(Module) ->
    Module:used().
```

OPINIONATED

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files

```
%% @todo Come up with a small-enough
examples that fit in a slide for these
two rules.
```

Hank Rules

As of Hank 1.0.0, we defined 8 rules:

- Unused **record fields**
- Unused **macros**
- Unused **header files**
- Unused **configuration options**
- Unnecessary **function arguments**
- Unused **callbacks**
- Single use **header files**
- Single use **attributes** in header files
- Your **own rules**

```
%%% @doc A rule to detect nothing.
-module(empty_rule).

-behaviour(hank_rule).

-export([analyze/2, ignored/2]).

-spec analyze(hank_rule:asts(),
              hank_context:t()) ->
                 [hank_rule:result()].
analyze(_ASTs, _Context) -> [].

-spec ignored(
    hank_rule:ignore_pattern(),
    term()) -> boolean().
ignored(_Pattern, _IgnoreSpec) -> false.
```

Accuracy

“

Dialyzer is **never wrong**.
It may not report **all** discrepancies.
But if it emits a warning,
there is a **problem** there.

“

Hank is **never wrong**.
It may not report **all** oxbow code.
But if it emits a warning,
there is **unused code** there.

Results

Erlang/OTP

> 2M lines of code
> 20 years in development

The **oldest** Erlang codebase, the own implementation of the language and its standard libraries.
We found:

- > **1000** unused **macros**
- > **200** unused **record fields**
- > **1000** unused **function arguments**
- > **15** unused **header files**
- **1** unused **configuration**

2600hz/Kazoo

> 400K lines of code
> 10 years in development

A software platform designed to provide carrier-grade VoIP features. One of the **largest open-source** Erlang repositories.
We found:

- > **150** unused **macros**
- > **20** unused **record fields**
- > **500** unused **function arguments**
- **3** unused **header files**
- **3** unused **configuration**

esl/MongooseIM

> 180K lines of code
> 8 years in development

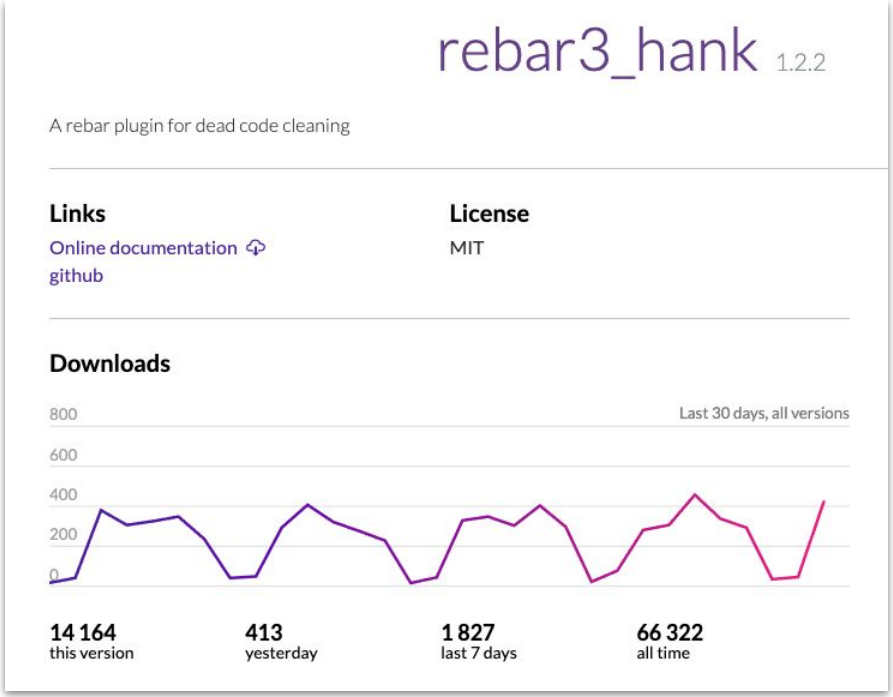
A very large Erlang system. An instant messaging platform built for heavy use with **very careful maintenance**.
We found:

- **90** unused **macros**
- **7** unused **record fields**
- > **200** unused **function arguments**

Some Stats

Who uses Hank?

- **NextRoll**
- **Klarna**
- **Miniclip**
- **Nova**
- **30+** other open-source repos
- ...and more!



NextRoll

Future Work


LET 'EM GO. YOU'LL STAY HERE
WITH ME. WE'LL GO BOWLING.
WHAT'S BOTHERING 'EM?

Future Work

Other Langs

While it can't be used for other languages, **Hank** can serve as an inspiration to implement similar tools for them.

The rules that we defined and the techniques we used can be used as a starting point.

More Rules

We haven't find all the possible instances of **oxbow code** out there.

New rules can be defined and implemented, reducing the existence of oxbow code in Erlang projects even further.

Improvements

- Better **parsing**
  - OTP tools can be drastically improved.
  - Alternative parsers can be used.
- Performance **optimizations**
  - Hank algorithms have lots of room for improvement and analysis.
- More **edge cases**
  - To find false positives and remove them.

To take home


WHEN YOU GO HOME TONIGHT,
THERE'S GONNA BE ANOTHER STORY
ON YOUR HOUSE.

“

Hank is a **rebar3 plugin** that provides a **static analyzer** of **Erlang code** to **detect** and **report** instances of **oxbow code** with a high degree of **accuracy.**

Thank you!

Brujo Benavides
@elbrujohalcon

Laura M. Castro
@lauramcastro