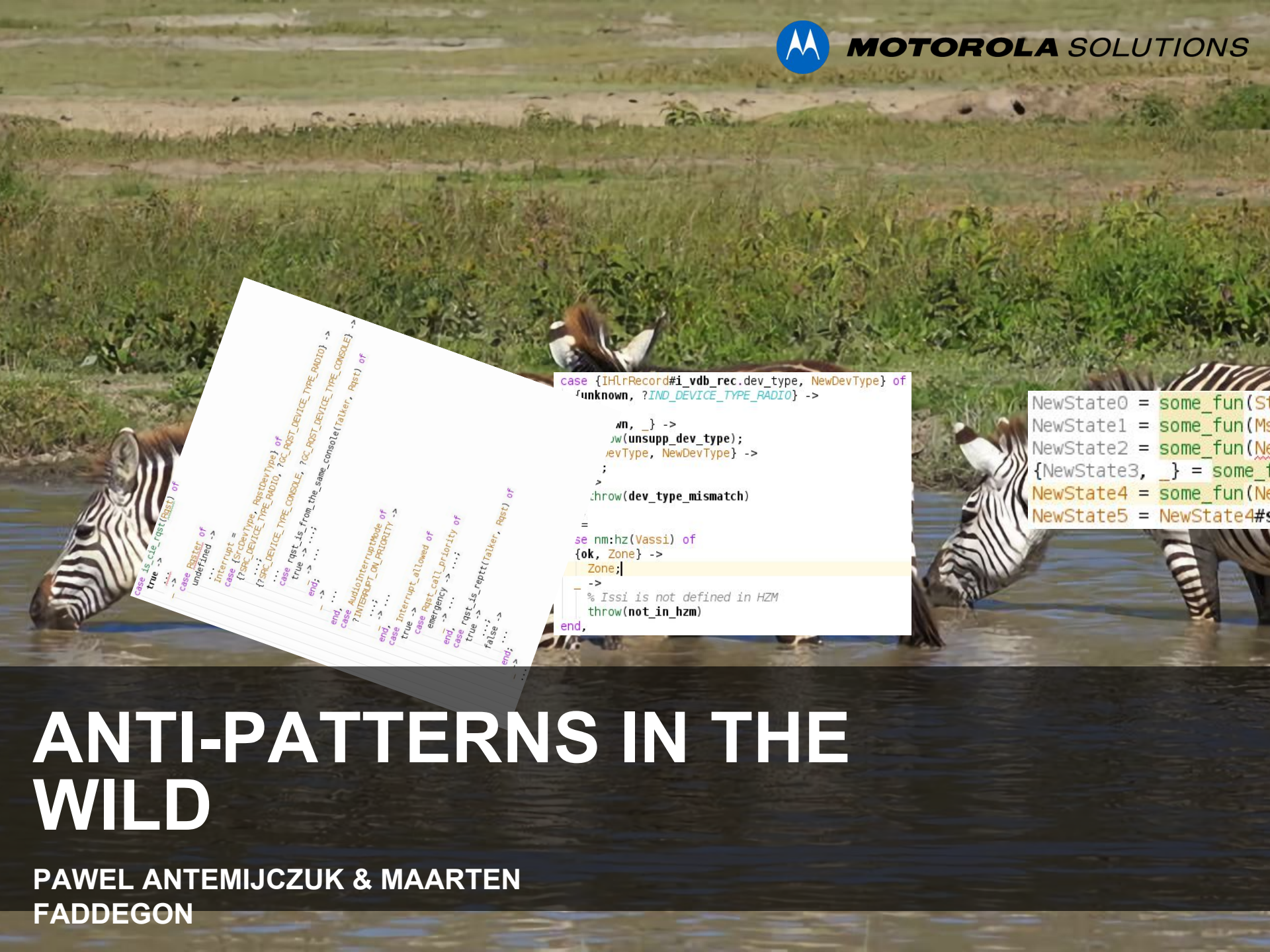


ANTI-PATTERNS IN THE WILD

PAWEL ANTEMIJCZUK & MAARTEN FADDEGON

OVERVIEW

Common anti-patterns

- Why do they happen?
- What to do about them
- Safe hunting

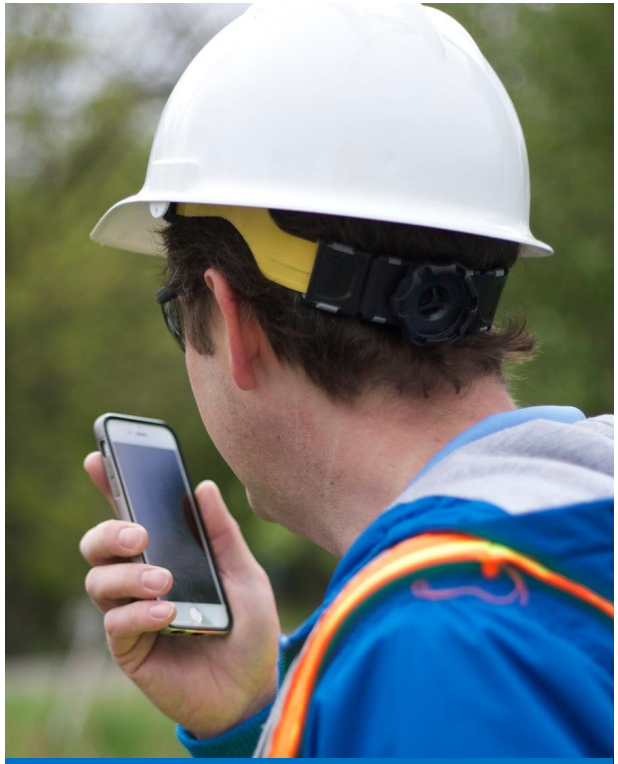
A LITTLE BACKGROUND

WHY THINGS ARE WHAT THEY ARE

MOTOROLA PRODUCTS

DIMETRA™
TETRA

ASTRO® 25
P25



WAVE™
LTE

ERLANG IN MSI

Sverige och Norge kopplar ihop sina radiosystem

2016-12-16 15:36 Av: Tommy Harnesk 4 KOMMENTARER

Som de första länderna i världen kopplar Sverige och Norge ihop sina nationella system för radiokommunikation – svenska Rakel med norska Nödnett.

Invigningen gjordes i samband med en fingerad bussolycka på E14 mellan Meråker i Norge och Storlien i Sverige. Vid övningen demonstrerades hur svenska och norska blåljusorganisationer (polis, räddningstjänst, ambulans, SOS Alarm med flera) kommunicerar vid en gemensam räddningsinsats.

ERLANG IN MSI

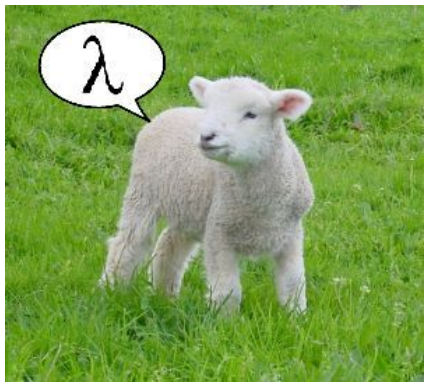
- **183000 lines of Erlang code in 531 modules**
  - 89 gen_fsm
  - 104 gen_server
- **71000 lines of Erlang headers**
- **7000 lines of C code**
- **2 C ports**
- **5 Erlang applications (3 releasables for 1 product)**

ERLANG IN MSI

Many iterations, with different teams of people

1. **Prototype, with The Prophet of Erlang**
2. **Product, with surprised C developers**
3. **Maintenance, with people enthusiastic about Erlang**

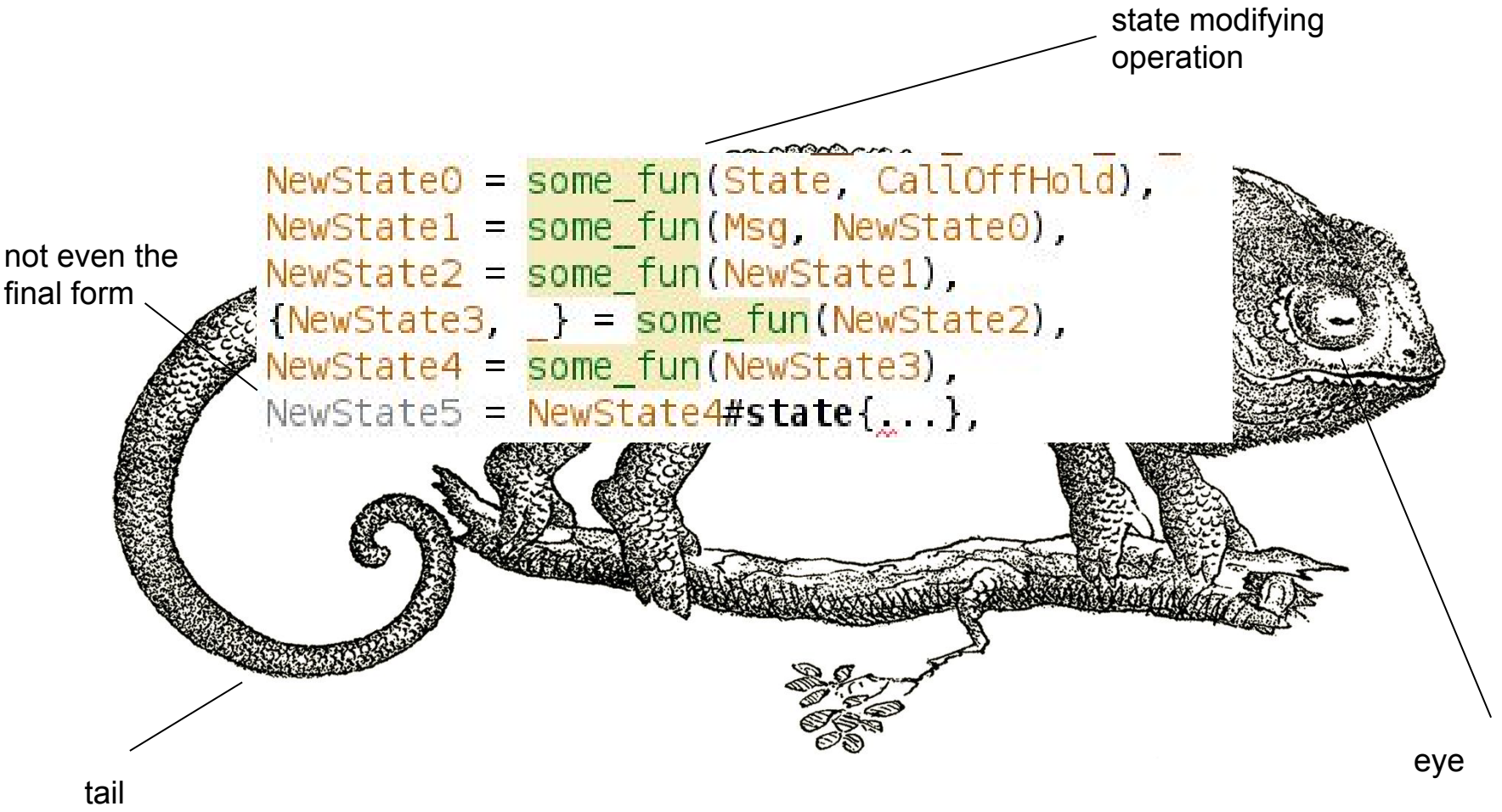
```
Ptr = case Policy of
  up ->
    case Last == bit_size(Bits) of
      true  -> 0;
      false -> Last+1
    end;
  down -> Last
end,
Rst = bit_size(Bits) - Ptr,
case {Bits, Policy} of
  {<<L:Ptr, 0:Rst>>, up}   ->
    {ok, Off, NLS} = get_bit(<<L:Ptr>>, low, Ptr),
    {ok, Off,      <<NLS/bitstring, 0:Rst>>};
```



THE COMMON SPECIMENS

THE ANTI-PATTERNS

THE SHIFTING STATE




```
NewState0 = some_fun(State, CallOffHold),
NewState1 = some_fun(Msg, NewState0),
NewState2 = some_fun(NewState1),
{NewState3, _} = some_fun(NewState2),
NewState4 = some_fun(NewState3),
NewState5 = NewState4#state{...},
```

THE SHIFTING STATE

- Immutability can be annoying to the unprepared
- A lot of things can happen in a single state machine transition
- Complicated protocol in an overcomplicated machine
- Too much data in the state

LESS RESPONSIBILITY

Too much information stored in the state of a single process

Example

- Problem: state machine of call controller has counters in state needed for messages sent to remote monitor
- Solution: create a dedicated process that keeps track of these counters

However

- **Complexity might be inherent**
- **Untangling can be non-trivial (might introduce defects)**

ASK WHAT YOU NEED

- **Many state changing functions**
  - do not demand all fields in the state
  - do not change all fields in the state
- **Example:**

```
fn(my_state, S1) ->
  S2      = f(S1),
  S3      = g(S2),
  {next_state, my_state, S3}.
```

```
fn(my_state, S = #state{x=X1, y=Y1}) ->
  Y2      = f(Y1),
  {X2,Y3} = g(X1, Y2),
  {next_state, my_state, S#state{x=X2, y=Y3}}
```

- **Chain of state passing can be hard to untangle**
- **Might reveal the "real" state that is needed (here `y`)**

HOMEBREW “STATE MONAD”

```
do(Z, [])     -> Z;
do(Z, [F|Fs]) -> do(F(Z), Fs).
```

no dependencies

every computation is wrapped in a (anonymous) function

```
do(State,
[ fun(S) -> S#state{src_site_id=SrcSiteId,
                    rqst_site_id=SrcSiteId,
                    csn_logging_number = CsnLoggingNumber,
                    rqst_dev_type = DevType}
  end
, fun(S) -> case UpdateType of %% AR-ISI-GC.40240, PZ Respond to PTT Update
              ?GRANT_RTP_UPDATE -> send_rapi(?RAPI_OPCODE_CAU_PTT_ID_UPDATE_ACTIVE,
                                             ptt, RqstSrc, S);
              ?ABORT_RTP_UPDATE -> send_rapi(?RAPI_OPCODE_CAU_PTT_ID_UPDATE_ACTIVE,
                                             ptt, RejSrc, S);
                                -> S
            end
  end
, fun(S) -> process_call_updt(state_wait_cz_grp_new_call_updt, S,
                              McId, Urid, PackedUpdateType, Call_type_mask, Updt_msg)
  end
]);
```

one kind of operation: takes state and evaluates to new state

ERLANDO STATE MONAD

`do` is expanded before compilation

an underscore '_' in a right-hand-side expression is wrapped in an anonymous function

```
StateT = state_t:new(identity_m),
StateT:exec(do([StateT ||
  StateT:put(State#state{src_site_id=SrcSiteId, rqst_site_id=SrcSiteId,
                         csn_logging_number = CsnLoggingNumber, rqst_dev_type = DevType}),
  StateT:modify(send_rapi_opcode_cau_ptt_id_update_active(UpdateType, _)),
  StateT:modify(process_call_updt(state_wait_cz_grp_new_call_updt, _, McId, Urid,
                                  PackedUpdateType, Call_type_mask, Updt_msg))
  ]), undefined);
```

different kinds of operations

CASE STUDY

```
S1 = State#state{rtp_resp_tmo_ref = undefined},
%% Late entry
StateName = state_wait_cz_grp_new_call_updt,
S2 = S1#state{rz_id              = RzId,
              src_zone_id        = RzId,
              cz_grp_add_active_call = undefined
},
send_stateless_rapi(?RAPI_OPCODE_RST_RADIO_SUBSCRIBER_REJECT, Msg, State),
Party = real_src_info(AddActiveCallMsg, S2),
S3 = S2#state{rqst = Party},
IsiMsg = construct_isi_connect_gc(S3),
send_to_ip2e1(IsiMsg, S3, StateName),
TmoRef = run_timer(?PING_INTERVAL, send_ping),
S4 = rqst_dev_type_to_src_dev_type(S3#state{ping_pong_tmo_ref = TmoRef,
                                            rqst_dev_type = RqstDevType,
                                            src_issi = Party#party.issi,
                                            src_mni = Party#party.mni}),
```

uses state, only has side-effect; no return value

computation returning value of other type than state …

… demanded by later computation

easy to mix up one of the many versions of the state (esp. when adding or removing operations)

ERLANDO STATE MONAD

```
StateT = state_t:new(identity_m),
NewState = StateT:exec(do([StateT ||
  StateT:put(State#state{rtp_resp_tmo_ref = undefined}),
  StateT:modify(_#state{rz_id              = RzId,
                        src_zone_id        = RzId,
                        cz_grp_add_active_call = undefined}),
  StateT:lift(send_stateless_rapi(?RAPI_OPCODE_RST_RADIO_SUBSCRIBER_REJECT, Msg, _)),
  Party <- StateT:return(real_src_info(AddActiveCallMsg, _)),
  StateT:modify(_#state{rqst = Party}),
  IsiMsg <- StateT:return(construct_isi_connect_gc(_)),
  StateT:lift(send_to_ip2e1(IsiMsg, _, StateName)),
  TmoRef <- StateT:return(run_timer(?PING_INTERVAL, send_ping)),
  State:modify(rqst_dev_type_to_src_dev_type(_#state{ping_pong_tmo_ref = TmoRef,
                                                     rqst_dev_type     = RqstDevType,
                                                     src_issi          = Party#party.issi,
                                                     src_mni           = Party#party.mni}))
  ]), undefined),
```

uses state, only has side-effect; no return value

computation returning value of other type than state …

… demanded by later computation

state is managed by monad, less mistakes

THE LONG CASE

long neck

case in case

case in case in case

case in case in case in case

secondary case in case in case in case

```
case is_cie_rqst(Rqst) of
  true ->
    ...
  _ ->
    case Rqster of
      undefined ->
        ...
        Interrupt =
        case {SrcDevType, RqstDevType} of
          {?SRC_DEVICE_TYPE_RADIO, ?GC_RQST_DEVICE_TYPE_RADIO} ->
            ...;
          {?SRC_DEVICE_TYPE_CONSOLE, ?GC_RQST_DEVICE_TYPE_CONSOLE} ->
            ...
            case rqst_is_from_the_same_console(Talker, Rqst) of
              true -> ...;
              _ -> ...
            end;
          _ ->
            ...
        end,
        case AudioInterruptMode of
          ?INTERRUPT_ON_PRIORITY ->
            ...;
          _ -> ...
        end,
        case Interrupt_allowed of
          true ->
            case Rqst_call_priority of
              emergency -> ...;
              _ -> ...
            end,
            case rqst_is_reptt(Talker, Rqst) of
              true ->
                ...;
              false ->
                ...
            end;
          _ ->
          ...
```

clause incognita

THE LONG CASE

- C-leftover for long sequential processing e.g. validation of a message received
- In C you do not create many small functions if there's no reuse
- No way to exit early, so cases keep nesting
- Sometimes cases are duplicated

THE THROWING VALIDATOR

Exceptions to interrupt the control flow

Horns

```
case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
  {unknown, ?IND_DEVICE_TYPE_RADIO} ->
    ok;
  {unknown, _} ->
    throw(unsupp_dev_type);
  {NewDevType, NewDevType} ->
    ok;
  _ ->
    throw(dev_type_mismatch)
end,
Hz =
case nm:hz(Vassi) of
  {ok, Zone} ->
    Zone;
  _ ->
    % Issi is not defined in HZM
    throw(not_in_hzm)
end,
```

Sequential C-like processing

THE THROWING VALIDATOR

- Attempt to fix sequential, nested cases
- Kingdom for a return
- Exceptions seemed like a good idea at the time...

CASE STUDY

```
IHlrRecord = case i_vdb:get_ihlr_record(Vassi) of
                {get_ihlr_rec_ok, MsRecord} ->
                  MsRecord;
                _ ->
                  throw(no_ihlr_record)
              end,

case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
  {unknown, ?IND_DEVICE_TYPE_RADIO} ->
    ok;
  {unknown, _} ->
    throw(unsupp_dev_type);
  {NewDevType, NewDevType} ->
    ok;
  _ ->
    throw(dev_type_mismatch)
end,
Hz =
case nm:hz(Vassi) of
  {ok, Zone} ->
    Zone;
  _ ->
    % Issi is not defined in HZM
    throw(not_in_hzm)
end,
ZeroBasedIndZone = Hz - 1,
case ZeroBasedIndZone of
  ZbIsiZcNum ->
```

throw used to emulate a C-like return statement

expression might just be a sanity check…

… or the expression might evaluate to a value needed later

HOMEBREW “ERROR MONAD”

```
-type action() :: fun((Argument :: term()) ->
  {continue, Next_argument :: term()} | {return, R :: term()}).
-type last_action() :: fun((Argument :: term()) -> term()).
-spec return_or_continue(maybe_improper_list(action(), last_action()),
                         First_argument :: term()) -> term().
return_or_continue([], Arg) ->
  Arg;
return_or_continue([Action | List_of_actions], Arg) ->
  case Action(Arg) of
    {continue, Next_arg} ->
      return_or_continue(List_of_actions, Next_arg);
    {return, R} ->
      R
  end.
```

one type of operation: takes a state and returns a new state

ERLANDO ERROR MONAD

when expression evaluates to {error, <reason>}
then {error, <reason>} is the result
and remaining expressions are not evaluated

```
do([error_m ||
  %% AR-ISI-MM.41200
  IHlrRecord <- case i_vdb:get_ihlr_record(Vassi) of
                  {get_ihlr_rec_ok, MsRecord} -> {ok, MsRecord};
                  _                           -> {error, no_ihlr_record}
                end,
  _DevType   <- case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
                  {unknown, ?IND_DEVICE_TYPE_RADIO} -> {ok, NewDevType};
                  {unknown, _}                      -> {error, unsupp_dev_type};
                  {NewDevType, NewDevType}          -> {ok, NewDevType};
                  _                                 -> {error, dev_type_mismatch}
                end,
  Hz         <- nm:hz(Vassi),
  _          <- case (Hz-1) of
                  ZbIsiZcNum -> {ok, ZbIsiZcNum};
                  % subscriber belongs to other zone inside Dimetra
                  _ -> {error, wrong_hlr}
                end,
```

SIDE BY SIDE COMPARISON

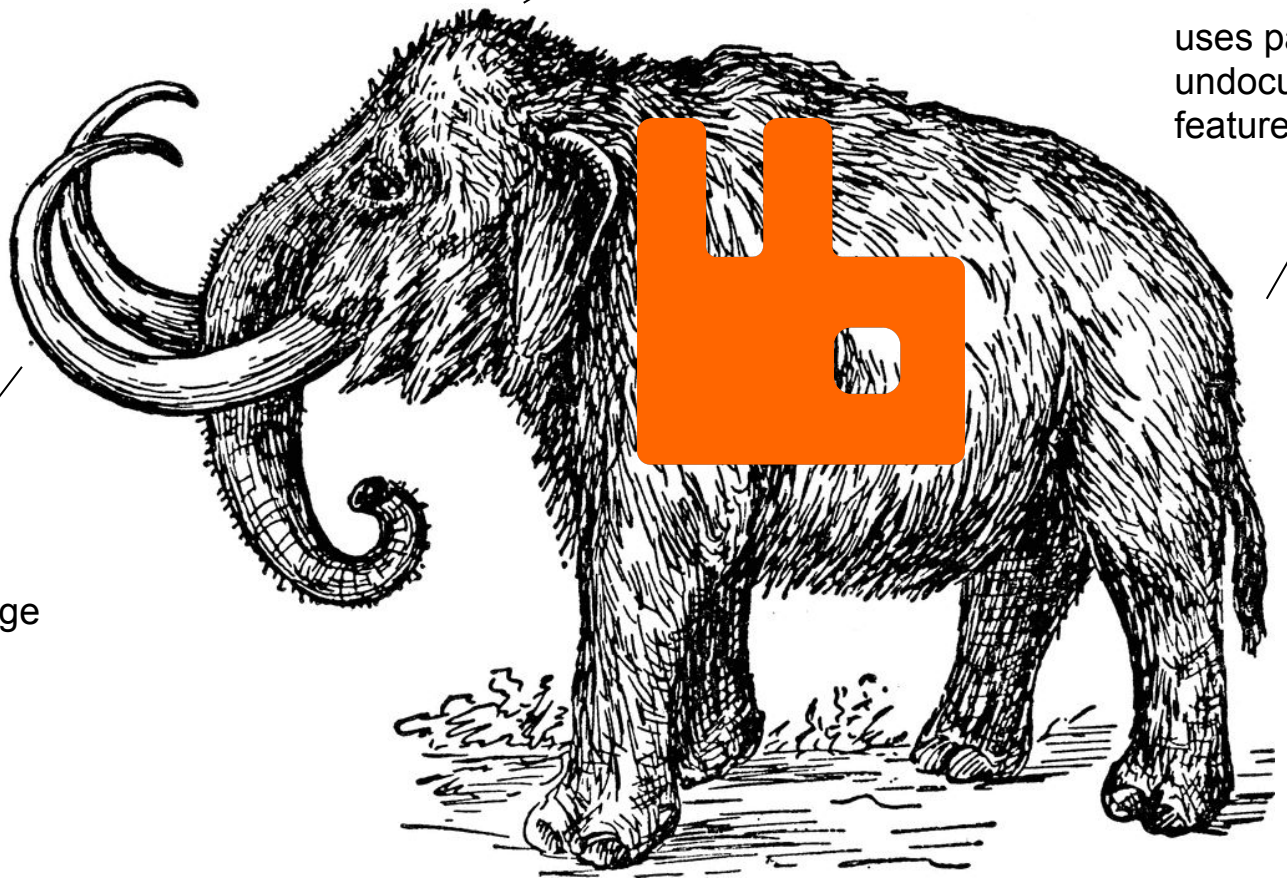
```
IHlrRecord = case i_vdb:get_ihlr_record(Vassi) of
               {get_ihlr_rec_ok, MsRecord} ->
                 MsRecord;
               _ ->
                 throw(no_ihlr_record)
             end,

case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
  {unknown, ?IND_DEVICE_TYPE_RADIO} ->
    ok;
  {unknown, _} ->
    throw(unsupp_dev_type);
  {NewDevType, NewDevType} ->
    ok;
  _ ->
    throw(dev_type_mismatch)
end,
Hz =
case nm:hz(Vassi) of
  {ok, Zone} ->
    Zone;
  _ ->
    % Issi is not defined in HZM
    throw(not_in_hzm)
end,
ZeroBasedIndZone = Hz - 1,
case ZeroBasedIndZone of
  ZbIsiZcNum ->
    ok;
  _ ->
    % subscriber belongs to other zone inside Dimetra
    throw(wrong_hlr)
end,
```

```
do([error_m ||
  %% AR-ISI-MM.41200
  IHlrRecord <- case i_vdb:get_ihlr_record(Vassi) of
                  {get_ihlr_rec_ok, MsRecord} -> {ok, MsRecord};
                  _                           -> {error, no_ihlr_record}
                end,
  _DevType   <- case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
                  {unknown, ?IND_DEVICE_TYPE_RADIO} -> {ok, NewDevType};
                  {unknown, _}                      -> {error, unsupp_dev_type};
                  {NewDevType, NewDevType}          -> {ok, NewDevType};
                  _                                 -> {error, dev_type_mismatch}
                end,
  Hz         <- nm:hz(Vassi),
  _          <- case (Hz-1) of
                  ZbIsiZcNum -> {ok, ZbIsiZcNum};
                  % subscriber belongs to other zone inside Dimetra
                  _ -> {error, wrong_hlr}
                end,
```

ERLANDO IN PRODUCTION?

not part of the Erlang release

uses parse transform and undocumented language features

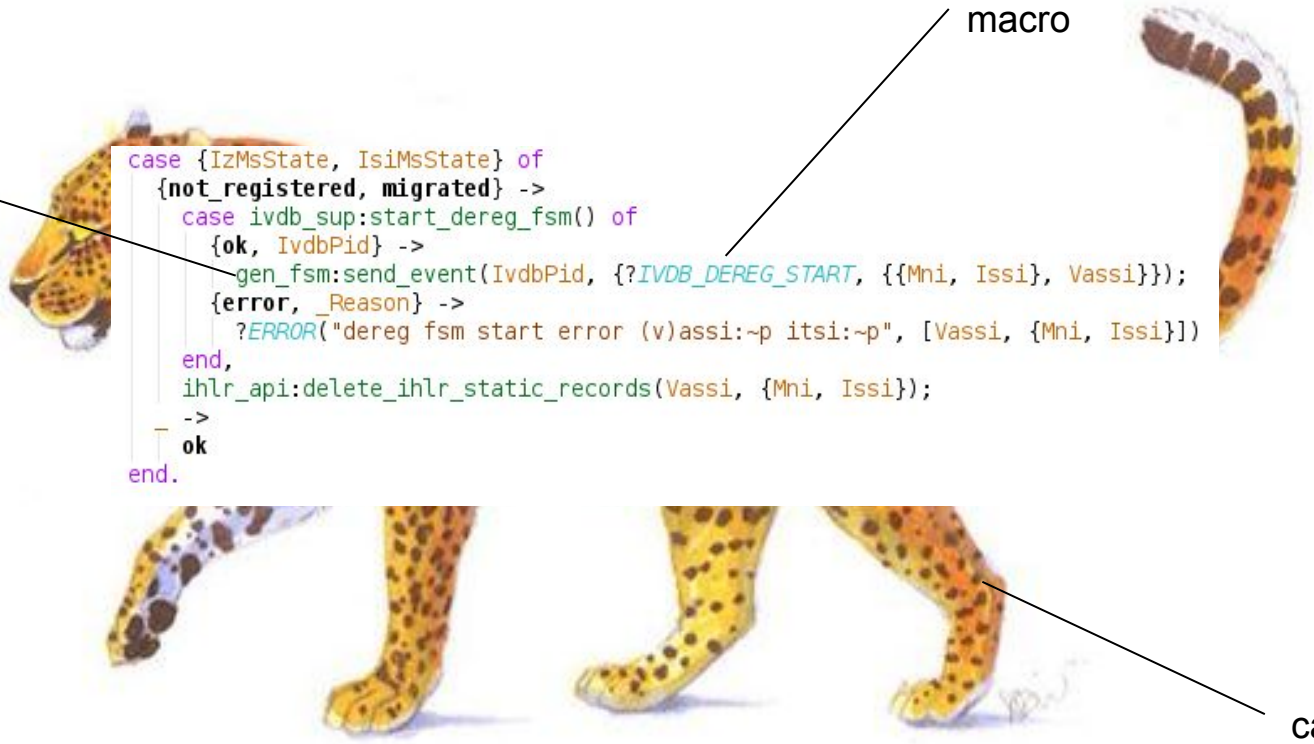


Clearer code reduces change of defects

THE MYSTERIOUS API

a mysterious macro

unidentified process' API



```
case {IzMsState, IsiMsState} of
  {not_registered, migrated} ->
    case ivdb_sup:start_dereg_fsm() of
      {ok, IvdbPid} ->
        gen_fsm:send_event(IvdbPid, {?IVDB_DEREG_START, {{Mni, Issi}, Vassi}});
      {error, _Reason} ->
        ?ERROR("dereg fsm start error (v)assi:~p itsi:~p", [Vassi, {Mni, Issi}])
    end,
    ihlr_api:delete_ihlr_static_records(Vassi, {Mni, Issi});
  _ ->
    ok
end.
```

camouflage spots

THE MYSTERIOUS API

- Macro helps with typos in atom names
- gen_fsm call kind of looks like an API already
- General confusion with processes (which process executes what code)

THE MYSTERIOUS API

- Define
- Your
- APIs

THE ABSOLUTE COORDINATOR

mean stare

gproc lookup

```
handle_call({send_iz_to_stack,
             {iz_call_trans_id, Cti},
             Msg}, _From, State) ->
  Reply =
    case prot:lookup({cti_id, Cti}) of
      {ok, {Module, Pid}} ->
        Module:send_iz_to_stack(Pid, Msg),
        ok;
      {error, not_found} ->
        process_iz_msg(Msg)
    end,
  {reply, Reply, State};
```

send based on lookup

THE ABSOLUTE COORDINATOR

- **The simplest architecture**
- **Frees you from thinking about more complex and more Erlang solutions**

THE ABSOLUTE COORDINATOR

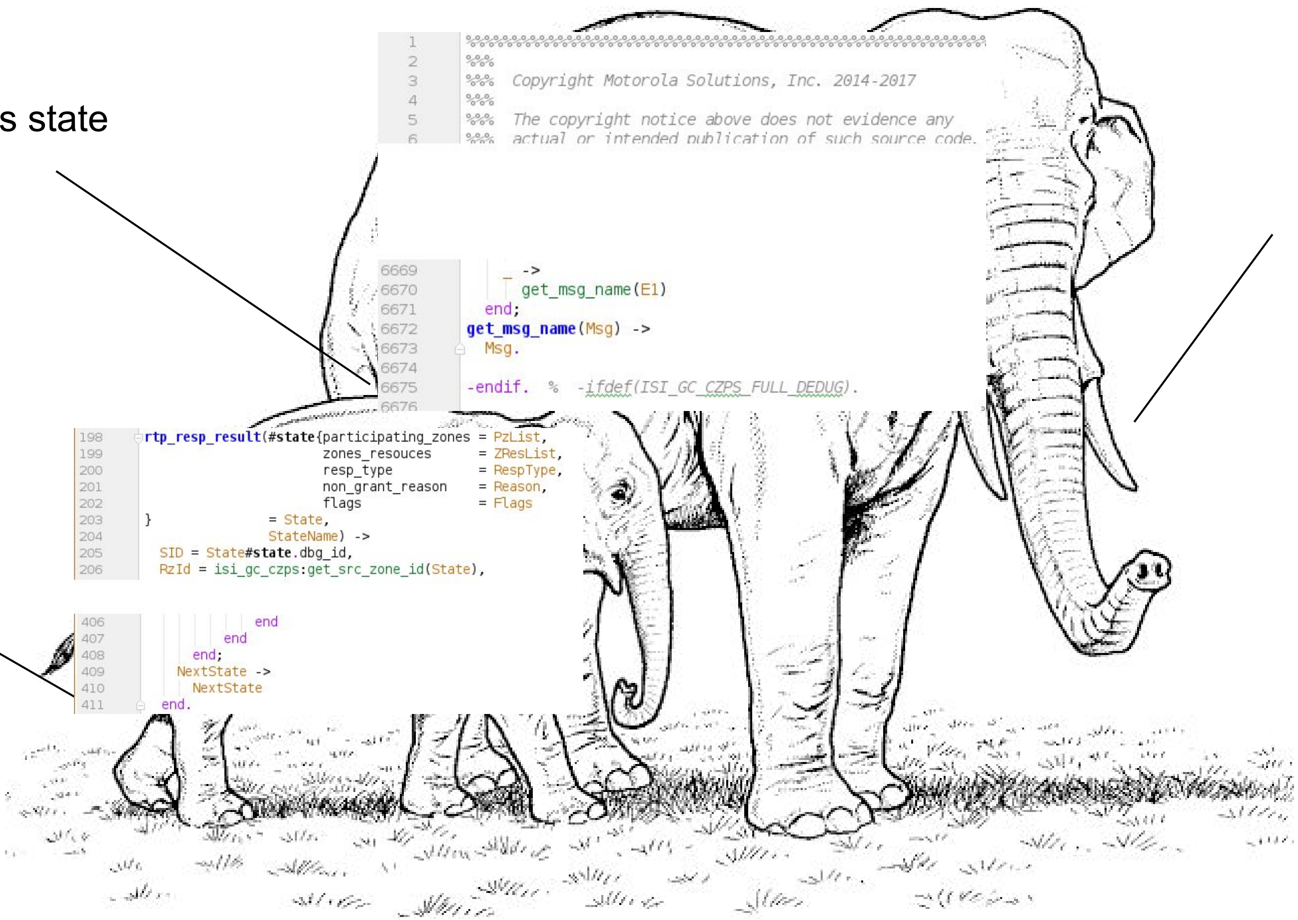
- Difficult problem to solve - at the end of the day, there is only one socket
- Coordinator's responsibility can be limited
- Multiple processes can talk to each other directly
- gproc helps solve the problem (but it's not free)
- Processes can let each other know their pids

THE MINOR OFFENDERS

enormous state machine

long single clause function

tusks



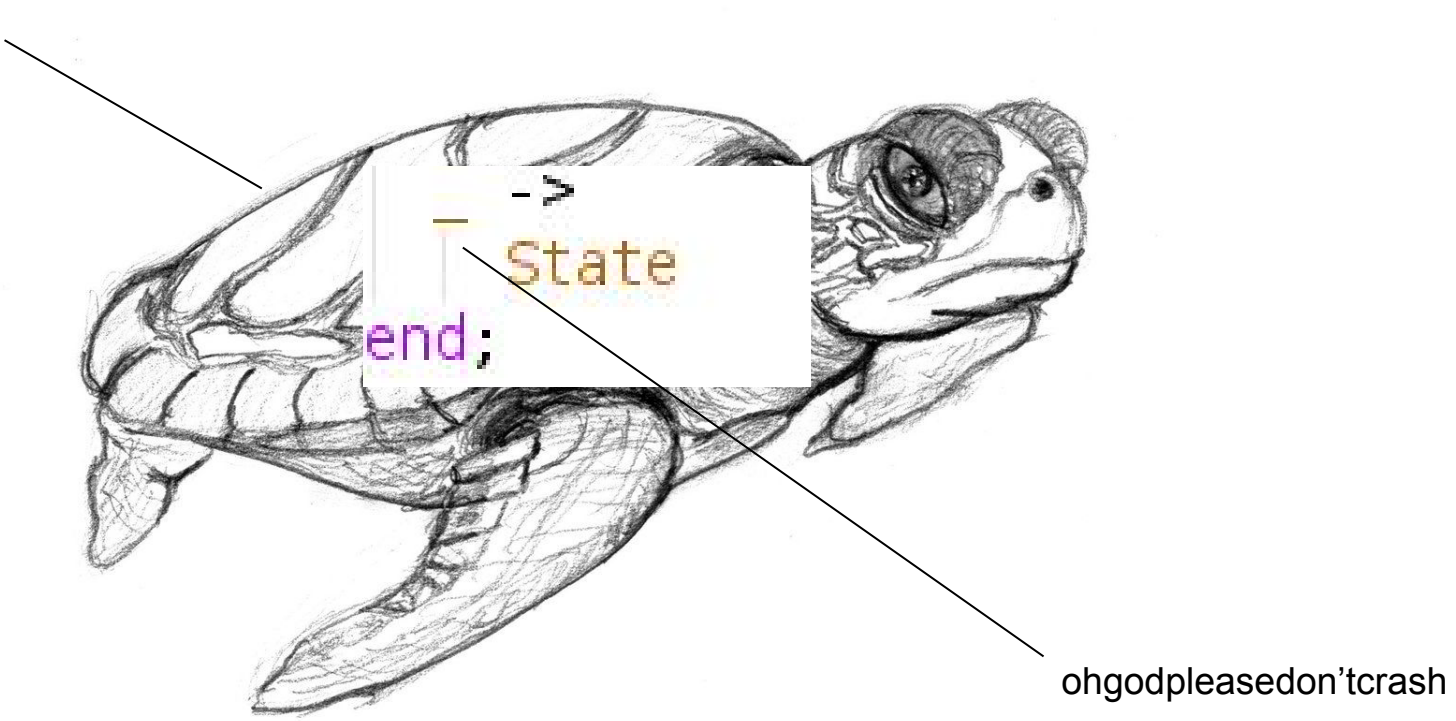
THE MINOR OFFENDERS

same module name everywhere

stripes

THE MINOR OFFENDERS

hard
shell



ohgodpleasedon’tcrash

THE MINOR OFFENDERS

- More C habits
- Big functions that enclose all functionality, not making small ones if there is no reuse
- Complicated domain with complicated protocols
- Easier to make one process than split properly into several
- Desire for interfaces or virtual classes
- Defend against everything

HUNTING THE PROBLEMS

GENERAL TOOLS

THE ARSENAL

THE REFACTOR QUICKCHECK

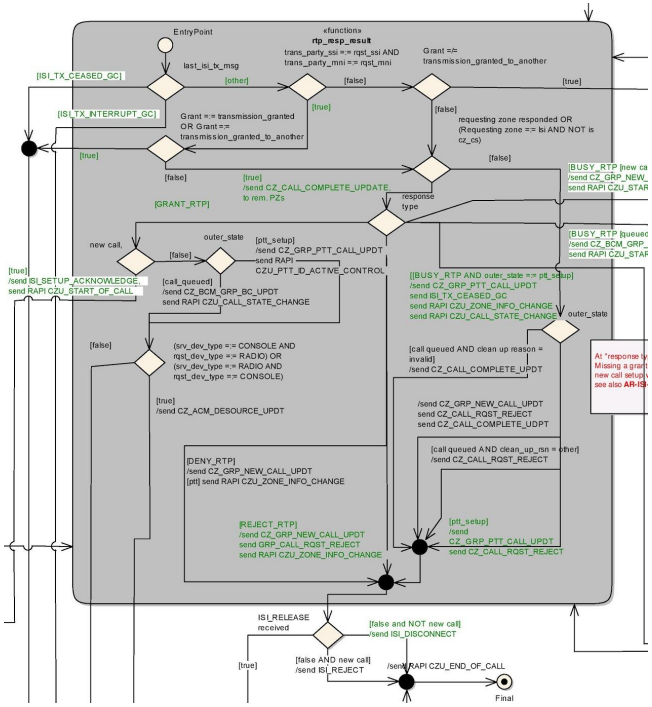
- Get the pre-refactor code
- Refactor it
- Mock out all the external calls you must keep (including the sequence)
- Generate the outputs of all decision points
- Run

THE REFACTOR QUICKCHECK

```
is_rtp_response_equiv(NumTests) ->
  Collect = fun({Transition, ToState, EndState}, B) ->
    collect({{Transition, ToState}, EndState#state.eqc_exit_path}, B)
          end,
  equiv_ref:check(fun pre_ref_function/1,
                  fun post_ref_function/1,
                  equiv_generator(),
                  fun mocking_setup/0,
                  fun mocking_dynamic_setup/1, Collect, NumTests).
```

THE REFACTOR QUICKCHECK

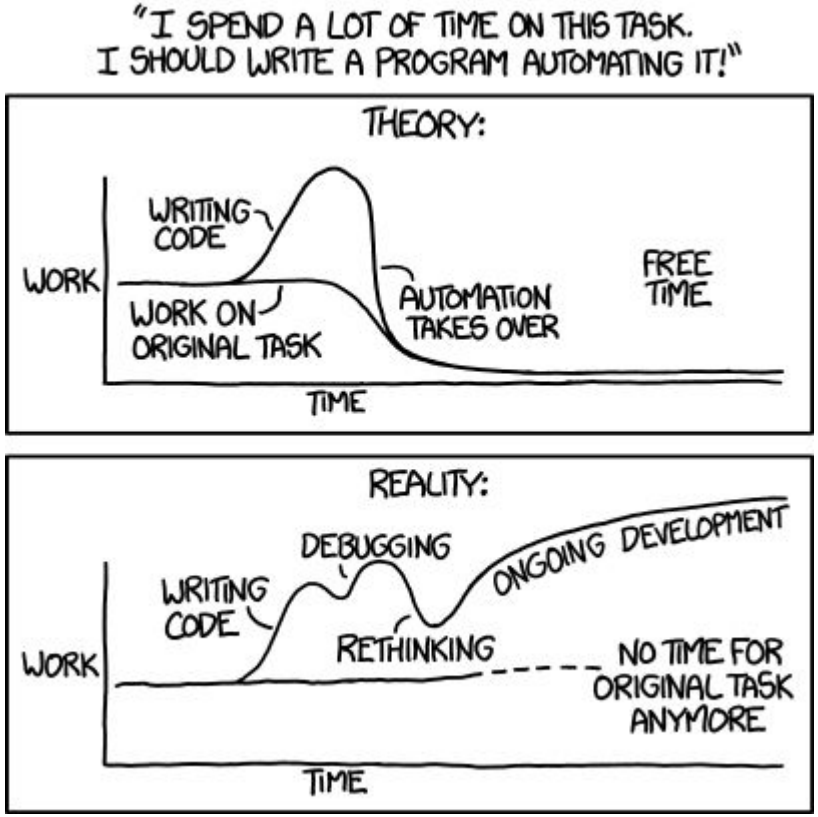
- **214 lines in 1 clause in 1 function**



- **212 lines**
- **3 functions**
- **10 clauses for main processing - one for each scenario**
- **0 new defects**

WRANGLER

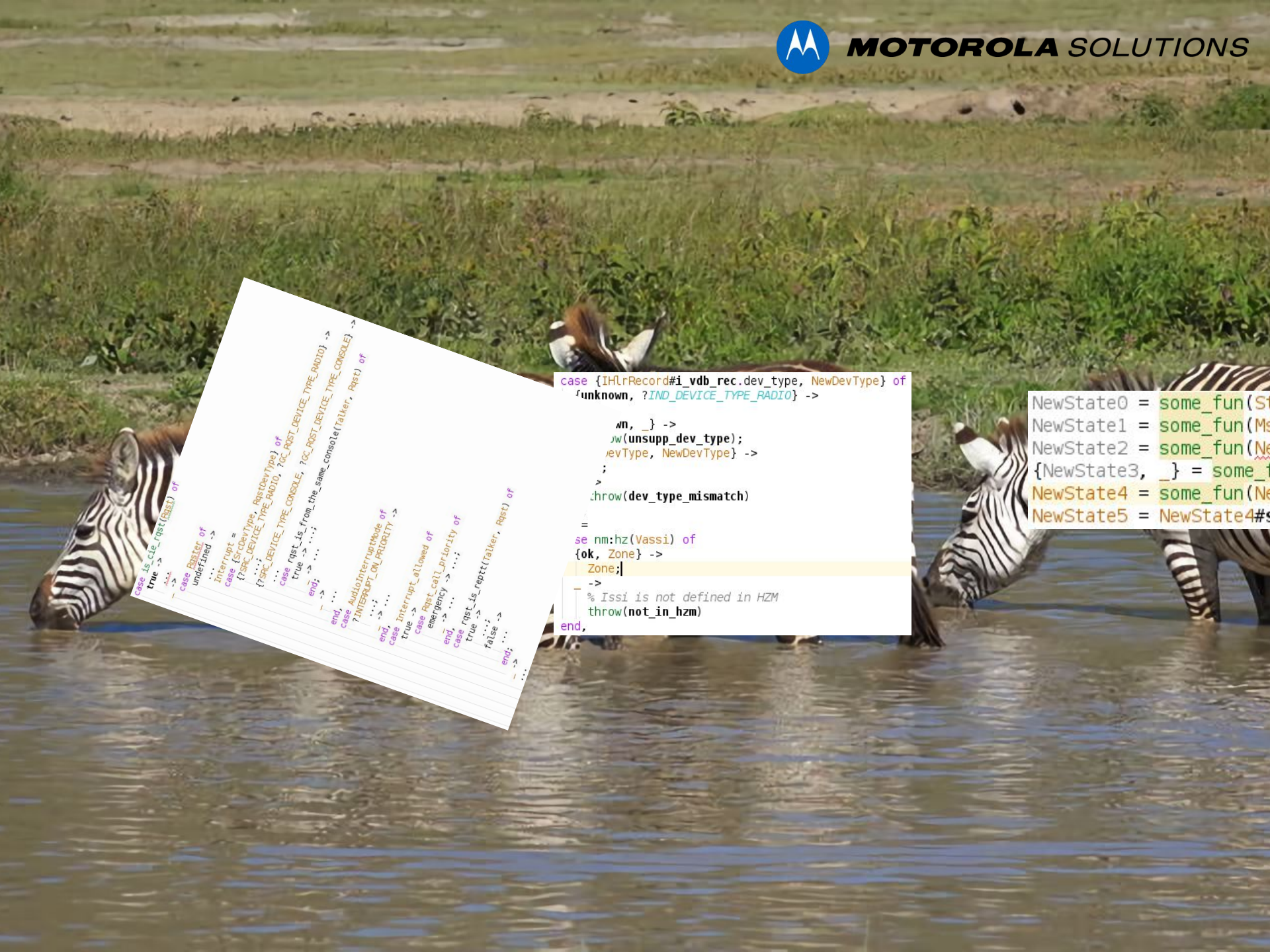
- **Semi-automated refactoring**
- **Reduces risk of introducing new mistakes**
- **Desired refactoring might not be supported**
- **Tool can be extended**
  - learning curve
  - is the transformation correct?



source: https://xkcd.com/1319/

ELVIS

- Automated code quality check
- Can check the most obvious things, however can’t really prevent bad code
- Worth considering e.g. in pull request approval
- New code goes through review
- A lot of old code


MOTOROLA SOLUTIONS
case {IHlrRecord#i_vdb_rec.dev_type, NewDevType} of
{unknown, ?IND_DEVICE_TYPE_RADIO} ->
throw(dev_type_mismatch)
{ok, Zone} ->
Zone;
% Issi is not defined in HZM
throw(not_in_hzm)
end,
NewState0 = some_fun(St
NewState1 = some_fun(Ms
NewState2 = some_fun(Ne
{NewState3, _} = some_f
NewState4 = some_fun(Ne
NewState5 = NewState4#s