DIY refactoring in Wrangler

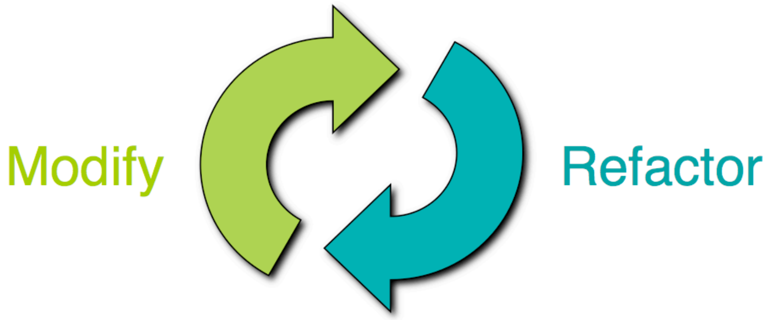
Huiqing Li and Simon Thompson

School of Computing
University of Kent

ProTest
property based testing

Refactoring

Change how a program works without changing what it does



Modify

Refactor

Why refactor?

Extension and reuse

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1},
        loop_a()
     end.
```

Let's turn this into a function

Why refactor?

Extension and reuse

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1},
        loop_a()
     end.
```

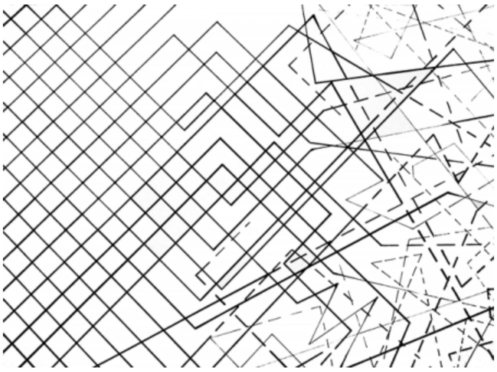
```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N),
        loop_a()
     end.

body(Msg,N) ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

Why refactor?

Counteract decay ... comprehension

"Clones considered harmful": detect and eliminate duplicate code.

Improve the module structure: remove loops, for example.



How to refactor?

By hand … using an editor.

Flexible … but error-prone.

Infeasible in the large.

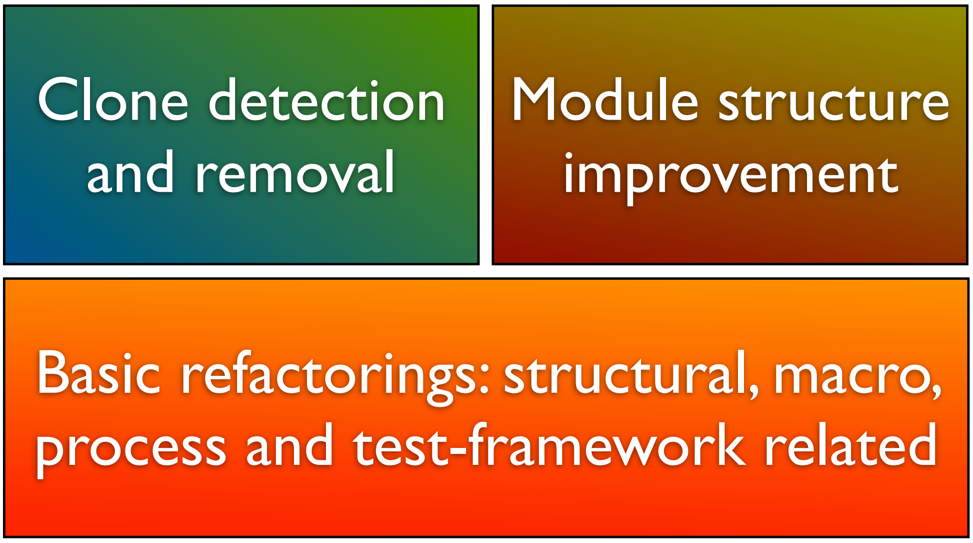
Tool supported.

Handle atoms, names, side-effects, …

Scalable to large-code bases.

Integrated with tests, macros, ...

Wrangler



Clone detection and removal

Module structure improvement

Basic refactorings: structural, macro, process and test-framework related

Wrangler in a nutshell

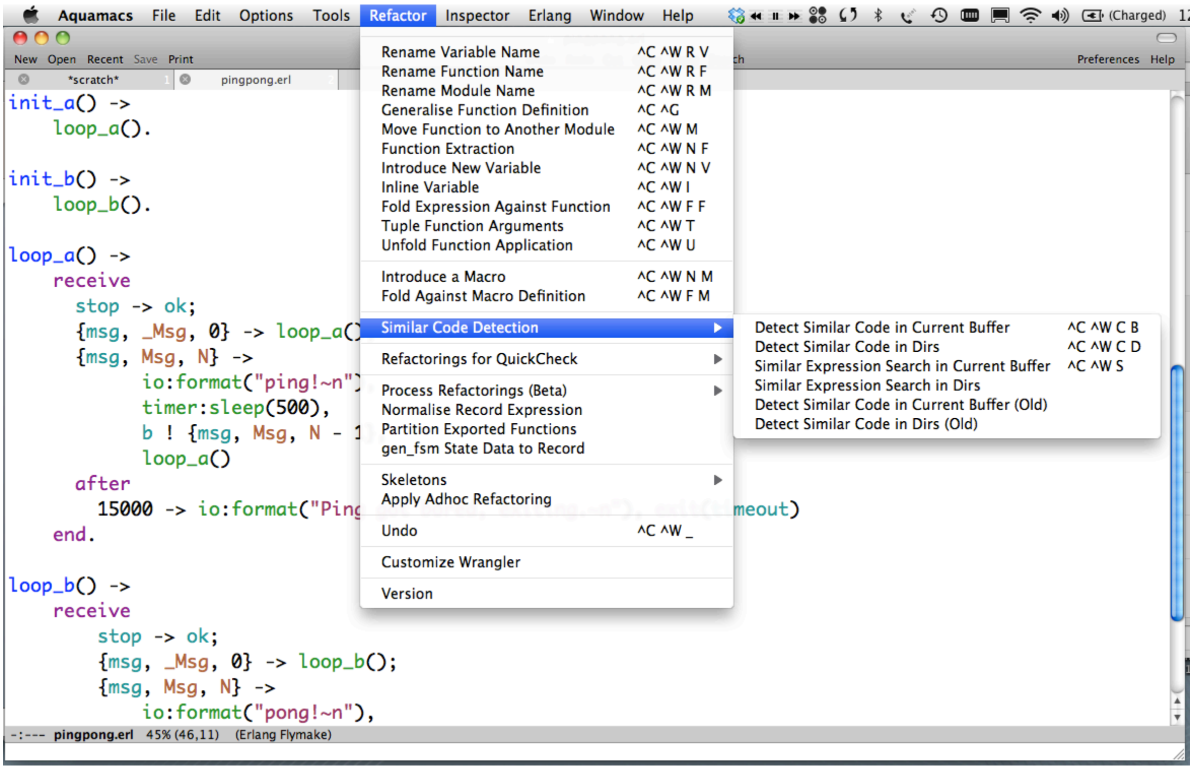
Automate the simple things, and …
… provide decision support tools otherwise.

Embed in common IDEs: emacs, eclipse, …

Handle full language, multiple modules, tests, ...

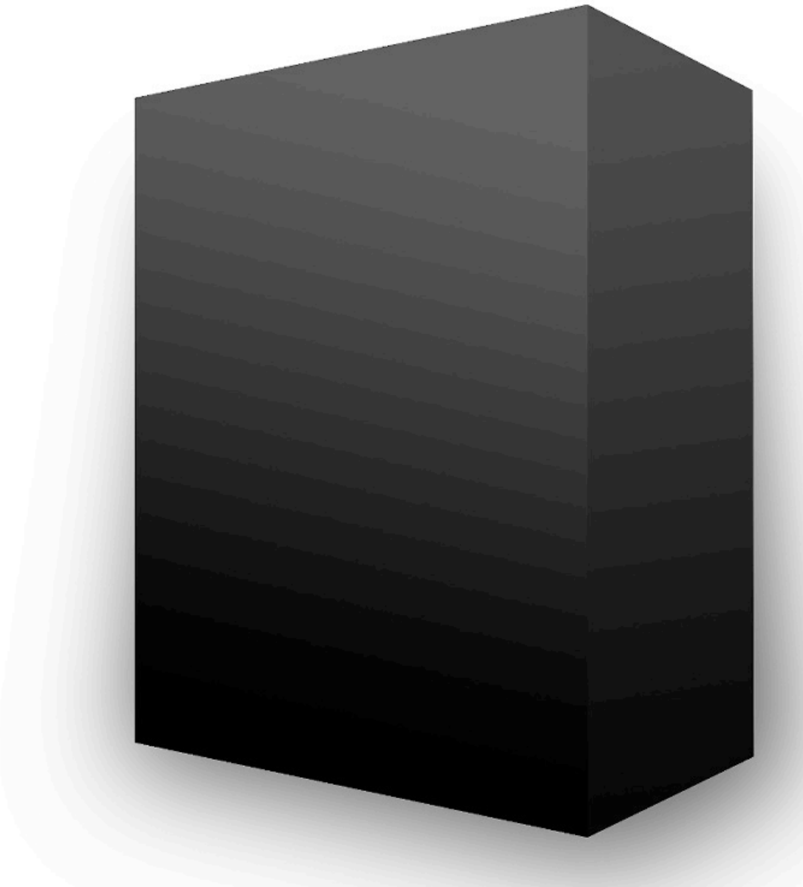
Faithful to layout and comments.

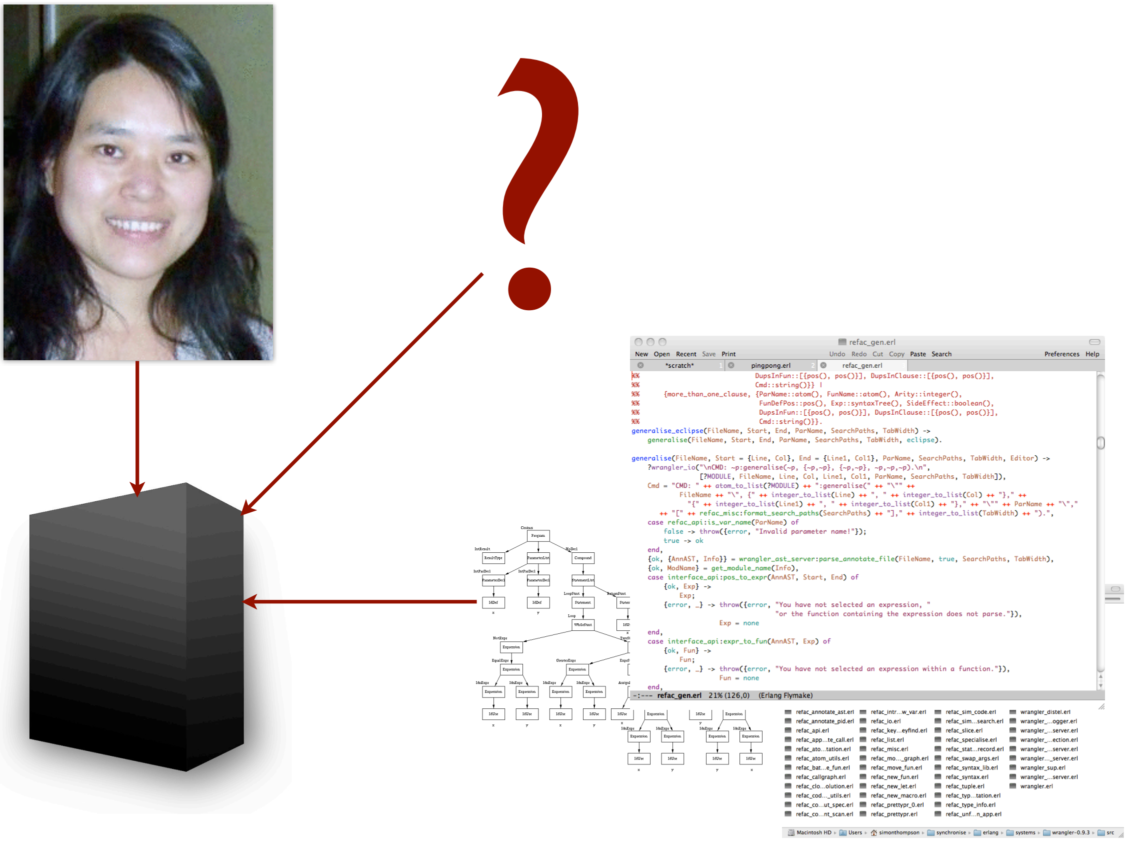
Build in Erlang and apply the tool to itself.


Aquamacs
File
Edit
Options
Tools
Refactor
Inspector
Erlang
Window
Help
(Charged)
New
Open
Recent
Save
Print
Preferences
Help
scratch
pingpong.erl
Rename Variable Name ^C ^W R V
Rename Function Name ^C ^W R F
Rename Module Name ^C ^W R M
Generalise Function Definition ^C ^G
Move Function to Another Module ^C ^W M
Function Extraction ^C ^W N F
Introduce New Variable ^C ^W N V
Inline Variable ^C ^W I
Fold Expression Against Function ^C ^W F F
Tuple Function Arguments ^C ^W T
Unfold Function Application ^C ^W U
Introduce a Macro ^C ^W N M
Fold Against Macro Definition ^C ^W F M
Similar Code Detection
Refactorings for QuickCheck
Process Refactorings (Beta)
Normalise Record Expression
Partition Exported Functions
gen_fsm State Data to Record
Skeletons
Apply Adhoc Refactoring
Undo ^C ^W _
Customize Wrangler
Version
Detect Similar Code in Current Buffer ^C ^W C B
Detect Similar Code in Dirs ^C ^W C D
Similar Expression Search in Current Buffer ^C ^W S
Similar Expression Search in Dirs
Detect Similar Code in Current Buffer (Old)
Detect Similar Code in Dirs (Old)
-:--- pingpong.erl 45% (46,11) (Erlang Flymake)

```
init_a() ->
    loop_a().

init_b() ->
    loop_b().

loop_a() ->
    receive
        stop -> ok;
        {msg, _Msg, 0} -> loop_a()
        {msg, Msg, N} ->
            io:format("ping!~n")
            timer:sleep(500),
            b ! {msg, Msg, N - 1
            loop_a()
        after
            15000 -> io:format("Ping                      meout)
    end.

loop_b() ->
    receive
        stop -> ok;
        {msg, _Msg, 0} -> loop_b();
        {msg, Msg, N} ->
            io:format("pong!~n"),
```

Demo





```
%%                            DupsInFun::[{pos(), pos()}], DupsInClause::[{pos(), pos()}],
%%                            Cmd::string()}} |
%%      {more_than_one_clause, {ParName::atom(), FunName::atom(), Arity::integer(),
%%                             FunDefPos::pos(), Exp::syntaxTree(), SideEffect::boolean(),
%%                             DupsInFun::[{pos(), pos()}], DupsInClause::[{pos(), pos()}],
%%                             Cmd::string()}}.
generalise_eclipse(FileName, Start, End, ParName, SearchPaths, TabWidth) ->
    generalise(FileName, Start, End, ParName, SearchPaths, TabWidth, eclipse).

generalise(FileName, Start = {Line, Col}, End = {Line1, Col1}, ParName, SearchPaths, TabWidth, Editor) ->
    ?wrangler_io("\nCMD: ~p:generalise(~p, {~p,~p}, {~p,~p}, ~p,~p,~p).\n",
                 [?MODULE, FileName, Line, Col, Line1, Col1, ParName, SearchPaths, TabWidth]),
    Cmd = "CMD: " ++ atom_to_list(?MODULE) ++ ":generalise(" ++ "\"" ++
            FileName ++ "\", {" ++ integer_to_list(Line) ++ ", " ++ integer_to_list(Col) ++ "}," ++
              "{" ++ integer_to_list(Line1) ++ ", " ++ integer_to_list(Col1) ++ "}," ++ "\"" ++ ParName ++ "\","
       ++ "[" ++ refac_misc:format_search_paths(SearchPaths) ++ "]," ++ integer_to_list(TabWidth) ++ ").",
    case refac_api:is_var_name(ParName) of
        false -> throw({error, "Invalid parameter name!"});
        true -> ok
    end,
    {ok, {AnnAST, Info}} = wrangler_ast_server:parse_annotate_file(FileName, true, SearchPaths, TabWidth),
    {ok, ModName} = get_module_name(Info),
    case interface_api:pos_to_expr(AnnAST, Start, End) of
        {ok, Exp} ->
            Exp;
        {error, _} -> throw({error, "You have not selected an expression, "
                                    "or the function containing the expression does not parse."}),
                      Exp = none
    end,
    case interface_api:expr_to_fun(AnnAST, Exp) of
        {ok, Fun} ->
            Fun;
        {error, _} -> throw({error, "You have not selected an expression within a function."}),
                      Fun = none
    end,
-:---  refac_gen.erl   21% (126,0)   (Erlang Flymake)
```

refac_annotate_ast.erl
refac_annotate_pid.erl
refac_api.erl
refac_app...te_call.erl
refac_ato...tation.erl
refac_atom_utils.erl
refac_bat...e_fun.erl
refac_callgraph.erl
refac_clo...olution.erl
refac_cod..._utils.erl
refac_co...ut_spec.erl
refac_co...nt_scan.erl
refac_intr...w_var.erl
refac_io.erl
refac_key...eyfind.erl
refac_list.erl
refac_misc.erl
refac_mo..._graph.erl
refac_move_fun.erl
refac_new_fun.erl
refac_new_let.erl
refac_new_macro.erl
refac_prettypr_0.erl
refac_prettypr.erl
refac_sim_code.erl
refac_sim...search.erl
refac_slice.erl
refac_specialise.erl
refac_stat...record.erl
refac_swap_args.erl
refac_syntax_lib.erl
refac_syntax.erl
refac_tuple.erl
refac_typ...tation.erl
refac_type_info.erl
refac_unf...n_app.erl
wrangler_distel.erl
wrangler_...ogger.erl
wrangler_...server.erl
wrangler_...ection.erl
wrangler_...server.erl
wrangler..._server.erl
wrangler_sup.erl
wrangler_...server.erl
wrangler.erl

Macintosh HD ▸ Users ▸ simonthompson ▸ synchronise ▸ erlang ▸ systems ▸ wrangler-0.9.3 ▸ src

Design criteria

We assume you can program Erlang …

… but don’t want to learn the internal syntax or details of our representation and libraries.

We aim for simplicity and clarity …

… rather than complete coverage.

Integration

Describe refactorings by a `behaviour`.

Integration with emacs for execution …

… which gives preview, undo, interactive behaviour etc. “for free”.

Generalisation

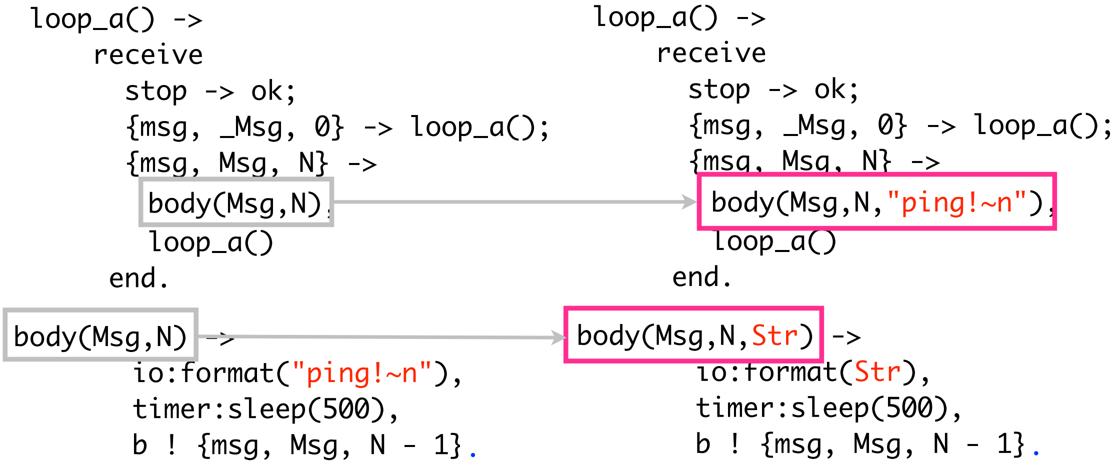
Describe expressions in Erlang ...

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N),
        loop_a()
    end.

body(Msg,N) ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N,"ping!~n"),
        loop_a()
    end.

body(Msg,N,Str) ->
        io:format(Str),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

Generalisation

... how expressions are transformed ...

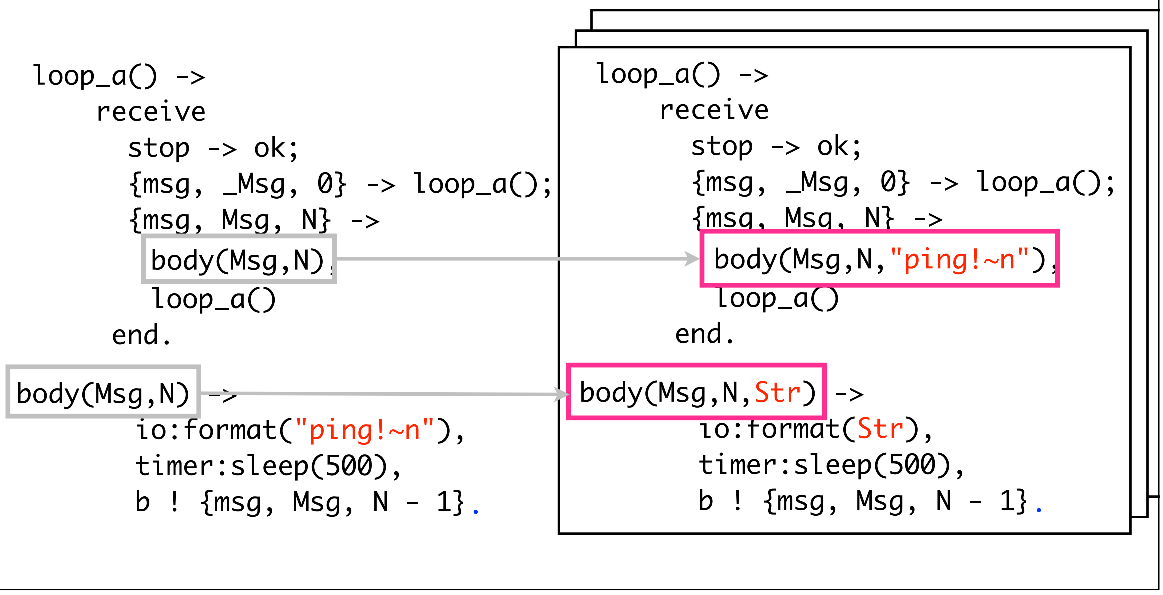


```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N),
        loop_a()
    end.

body(Msg,N) ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N,"ping!~n"),
        loop_a()
    end.

body(Msg,N,Str) ->
        io:format(Str),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

Generalisation

... and its context and scope.



```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N),
        loop_a()
     end.

body(Msg,N) ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N,"ping!~n"),
        loop_a()
     end.

body(Msg,N,Str) ->
        io:format(Str),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

Generalisation

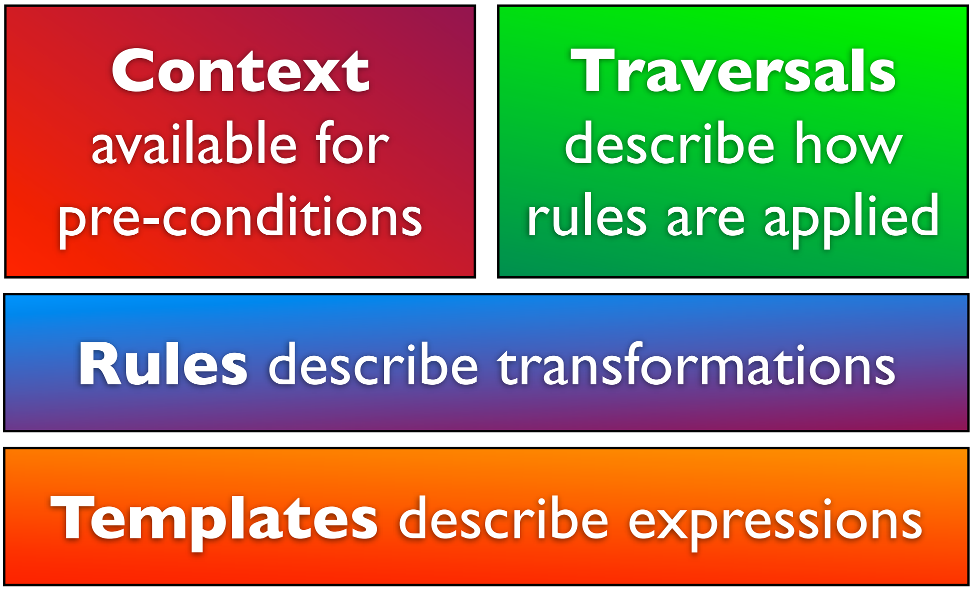
Pre-conditions for refactorings

```
loop_a() ->
    receive
      stop -> ok;
      {msg, _Msg, 0} -> loop_a();
      {msg, Msg, N} ->
        body(Msg,N),
        loop_a()
     end.

body(Msg,N) ->
        io:format("ping!~n"),
        timer:sleep(500),
        b ! {msg, Msg, N - 1}.
```

Can't generalise over an expression that contains free variables ...

… or use the same name as an existing variable for the new variable.

Wrangler API



Context available for pre-conditions

Traversals describe how rules are applied

Rules describe transformations

Templates describe expressions

Templates

Templates are enclosed in the `?T` macro call.

Meta-variables in templates are Erlang variables ending in `@`, e.g. `F@`, `Arg@@`, `Guards@@@`.

```
?T("M:F@(1,2)")

?T("spawn(Args@@)")

?T("spawn(Arg1@,
    Arg2@,Args@@)")
```

`F@` matches a single element.

`Args@@` matches a sequence of elements of some kind.

Rules

```
?RULE(Template, NewCode, Cond)
```

The old code, the new code and the pre-condition.

```
rule({M,F,A}, N) ->
 ?RULE(?T("F@(Args@@)"),
       begin
         NewArgs@@=delete(N, Args@@),
         ?QUOTE("F@(NewArgs@@)")
       end,
       refac_api:fun_define_info(F@) == {M,F,A}).

delete(N, List) ->  … delete Nth elem of List …
```

Information in the AAST

Wrangler uses the `syntax_tools` AST, augmented with information about the program semantics.

API functions provide access to this.

Variables bound, free and visible at a node.

Location information.

All bindings (if a vbl).

Where defined (if a fn).

Atom usage info: name, function, module etc.

Process info …

Collecting information

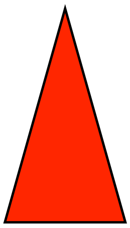
```
?COLLECT(Template, Collector, Cond)
```

- The template to match.
- The information to extract ("collect").
- Condition on when to collect the information.

```
?COLLECT(?T("Body@@, V@=Expr@, V@"),
       {_File@, refac_api:start_end_loc(_This@)},
       refac_api:type(V@) == variable).
```

`_File@` current file `_This@` subtree matching `?T(…)`

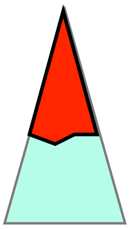
Traversals



```
?FULL_TD_TP(Rules, Scope)
```

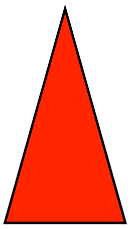
- Traverse top-down
- At each node, apply first of `Rules` to succeed …
- `TP` = "Type preserving".

Traversals



?STOP_TD_TU(Collectors, Scope)

- Traverse top-down
- … apply all of the Collectors to succeed …
- … only visit sub-nodes if no collector has fired.
- TU = "Type unifying".



?FULL_TD_TP(Rules, Scope)

- Traverse top-down
- At each node, apply first of Rules to succeed …
- TP = "Type preserving".

```
-module(refac_swap_args).

-behaviour(gen_refac).

-export([…]).

-include("../include/gen_refac.hrl").

-import(refac_api, [fun_define_info/1]).

input_pars() -> ["Parameter Index 1: ", "Parameter Index 2: "].

select_focus(_Args=#args{current_file_name=File,
                         cursor_pos=Pos}) ->
    interface_api:pos_to_fun_def(File, Pos).


pre_cond_check(_Args=#args{focus_sel=FunDef,
                           user_inputs=[I, J]}) ->
    …
         true ->
                ok;
         false ->
                {error, "Index 1 and Index 2 are the same."} …
    .

transform(Args=#args{current_file_name=File,focus_sel=FunDef,
                     user_inputs=[I, J]}) ->
    …
    {ok, Res}=transform_in_cur_file(Args, {M,F,A}, I1, J1),
    case refac_api:is_exported({F,A}, File) of
        true ->
            {ok, Res1}=transform_in_client_files(Args, {M,F,A}, I1, J
         …
    end.
```

Behaviour gen_refac encapsulates what a refactoring needs to provide.

input_pars: prompts for interactive input

select_focus: what to do with focus information.

pre_cond_check: check preconditions

transform: if the pre-condition is ok, do the transform.

```
transform_in_cur_file(_Args=#args{current_file_name=File},MFA, I, J)->
    …
      ?FULL_TD_TP([rule1(MFA, I, J), … ],[File])
    -.

transform_in_client_files(_Args=#args{current_file_name=File,
                                     search_paths=SearchPaths},
                          MFA, I, J) ->
    ?FULL_TD_TP([rule2(MFA, I, J),
                 rule3(MFA, I, J),
                 rule4(MFA, I, J),
                 rule5(MFA, I, J),
                 rule6(MFA, I, J)],
                refac_api:client_files(File, SearchPaths)).
```

Transformations defined by means of a template language …

… rules applied in full, top-down manner in this case.

```
%% transform the function definition itself.

rule1({M,F,A}, I, J) ->
    ?RULE("f@(Args@@) -> Bs@@;", begin NewArgs@@=swap(Args@@,I,J),
                                      ?QUOTE("f@(NewArgs@@)->Bs@@;")
                                end,
          fun_define_info(f@)== {M,F,A}.).

%% the following rules transform the different kinds of
%% application scenarios of the function.

rule2({M,F,A}, I, J) ->
    ?RULE("F@(Args@@)", begin NewArgs@@=swap(Args@@, I, J),
                              ?QUOTE("F@(NewArgs@@)")
                        end,
          fun_define_info(F@) == {M, F, A}).
```

Demo

Finding out more

Latest release of Wrangler: 0.9.3

`www.cs.kent.ac.uk/projects/wrangler`

Documentation for

- `refac_api`
- `interface_api`
- `gen_refac`

within Wrangler documentation.

Examples including

- `refac_swap_args.erl`
- `refac_specialise.erl`
- `refac_keysearch_to_key find.erl`

in the `doc` directory.

Other approaches

Use `syntax_tools` or parse transforms?

Gives a nice high-level interface to AST …

… but all the analysis is up to you, and

… no integration with IDE.

```erlang
-module(refac_replace_append).

-behaviour(gen_refac).

-export([input_par_prompts/0, select_focus/1,
         check_pre_cond/1, selective/0,transform/1]).

input_par_prompts() -> [].

select_focus(_Args) -> {ok, none}.

check_pre_cond(_Args) -> ok.

selective() -> true.

transform(_Args=#args{search_paths=SearchPaths}) ->
  ?FULL_TD_TP([rule_replace_append()], SearchPaths).

rule_replace_append() ->
  ?RULE(?T("F@(L1@, L2@)"),
        ?QUOTE("L1@++L2@"),
        {lists, append, 2}==refac_api:fun_define_info(F@)).
```

Questions?

www.cs.kent.ac.uk/projects/wrangler

ProTest
property based testing