Refactoring Erlang with Wrangler

Huiqing Li
Simon Thompson

School of Computing
University of Kent

ProTest
property based testing

University of Kent | Computing

Overview

Refactoring.

Tools and tool building.

Clone detection.

Improve module structure.

Tool demo …

Introduction

All in the code

Functional programs embody their design in their code.

Successful programs evolve … as do their tests, makefiles etc.

```
loop(Frequencies) ->
  receive
    {request, Pid, allocate} ->
      {NewFrequencies, Reply} = allocate
(Frequencies, Pid),
      reply(Pid, Reply),
      loop(NewFrequencies);
    {request, Pid , {deallocate, Freq}} ->
      NewFrequencies=deallocate(Frequencies,
Freq),
      reply(Pid, ok),
      loop(NewFrequencies);
    {'EXIT', Pid, _Reason} ->
      NewFrequencies = exited(Frequencies, Pid),
      loop(NewFrequencies);
    {request, Pid, stop} ->
      reply(Pid, ok)
  end.

exited({Free, Allocated}, Pid) ->
  case lists:keysearch(Pid,2,Allocated) of
    {value,{Freq,Pid}} ->
       NewAllocated =    lists:keydelete(Freq,
1,Allocated),
      {[Freq|Free],NewAllocated};
    false ->
      {Free,Allocated}
  end.
```

ProTest property based testing

University of Kent Computing

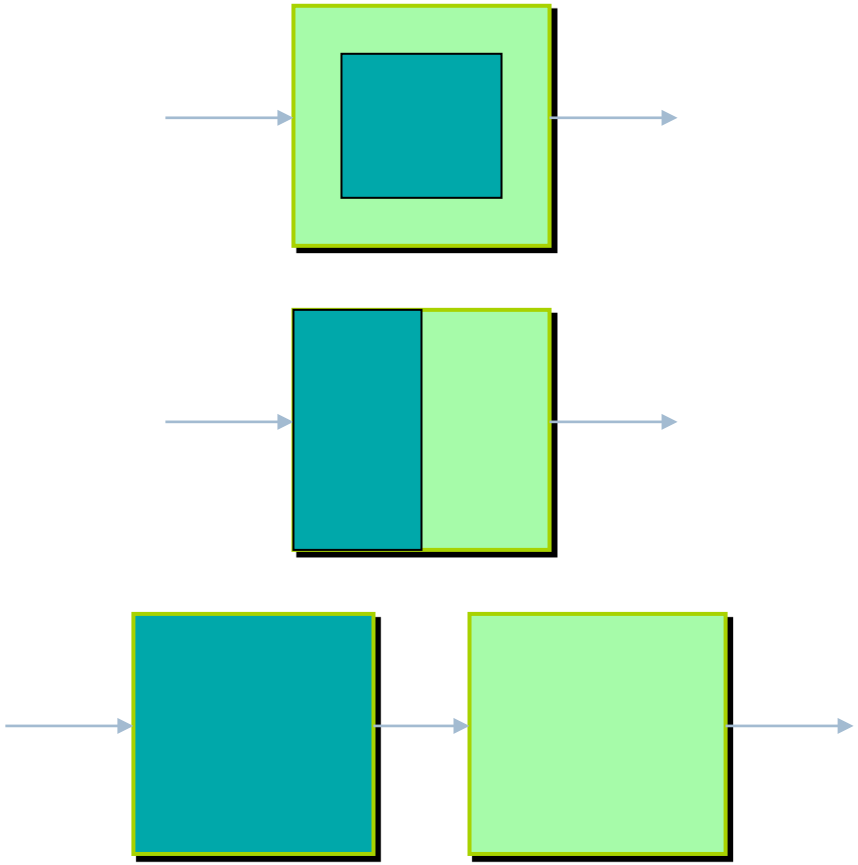
Soft-Ware

There's no single correct design …

… different options for different situations.

Maintain flexibility as the system evolves.

Refactor as you program.

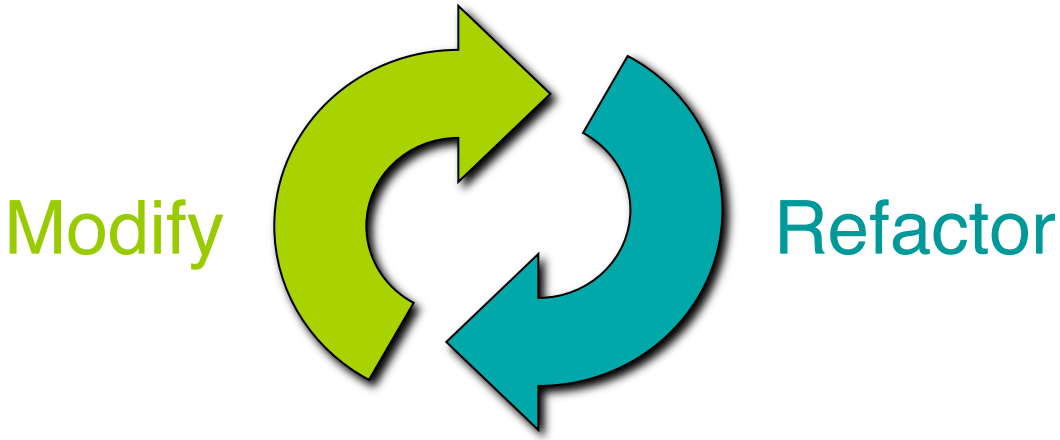


ProTest
property based testing

University of Kent | Computing

Refactoring

Refactoring means changing the design or structure of a program … without changing its behaviour.



Generalisation and renaming

```
-module (test).
-export([f/1]).

add_one ([H|T]) ->
  [H+1 | add_one(T)];

add_one ([]) -> [].

f(X) -> add_one(X).
```

```
-module (test).
-export([f/1]).

add_int (N, [H|T]) ->
 [H+N | add_int(N,T)];

add_int (N,[]) -> [].

f(X) -> add_int(1, X).
```

Generalisation

```
-export([printList/1]).

printList([H|T]) ->
  io:format("~p\n",[H]),
  printList(T);
printList([]) -> true.
```

```
-export([printList/2]).

printList(F,[H|T]) ->
  F(H),
  printList(F, T);
printList(F,[]) -> true.
```

```
printList([1,2,3])
```

```
printList(
  fun(H) ->
    io:format("~p\n", [H])
  end,
  [1,2,3]).
```

ProTest property based testing

University of Kent Computing

The tool

Refactoring tool support

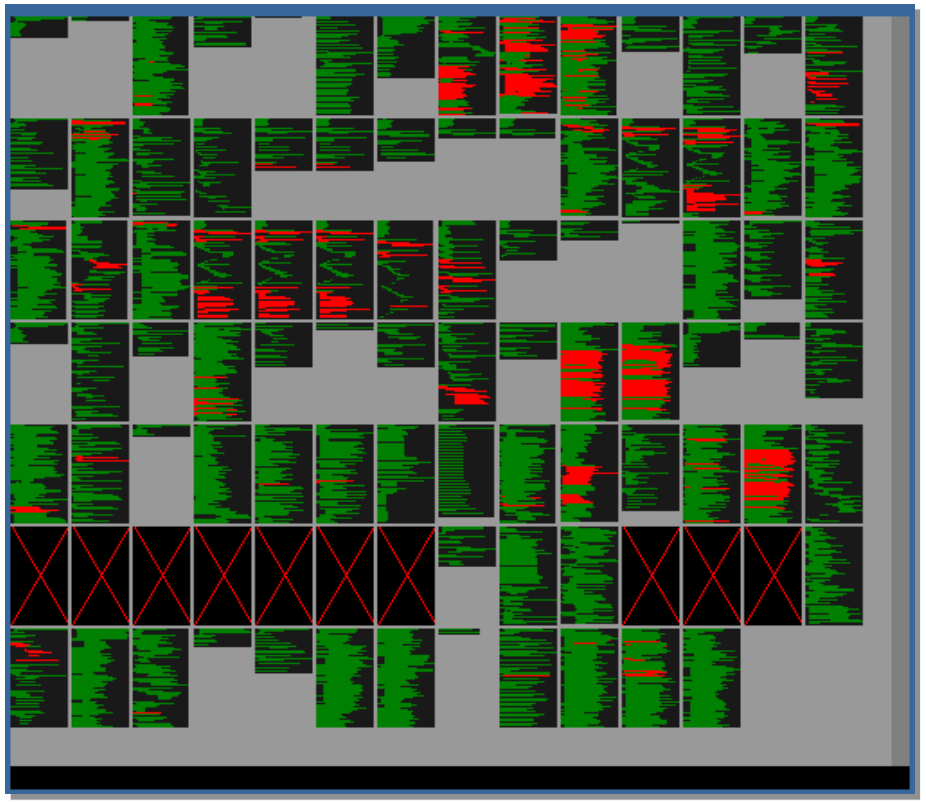
Bureaucratic and diffuse.

Tedious and error prone.

Semantics: scopes, types, modules, …

Undo/redo

Enhanced creativity



ProTest
property based testing

University of Kent | Computing

Wrangler

Refactoring tool for Erlang

Integrated into Emacs and Eclipse

Multiple modules

Structural, process, macro refactorings

Duplicate code detection …

… and elimination

Testing / refactoring

"Similar" code identification

Code Inspection

Property discovery

Wrangler

Clone detection
+ removal

Improve module structure

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

Design philosophy

Automate the simple actions …

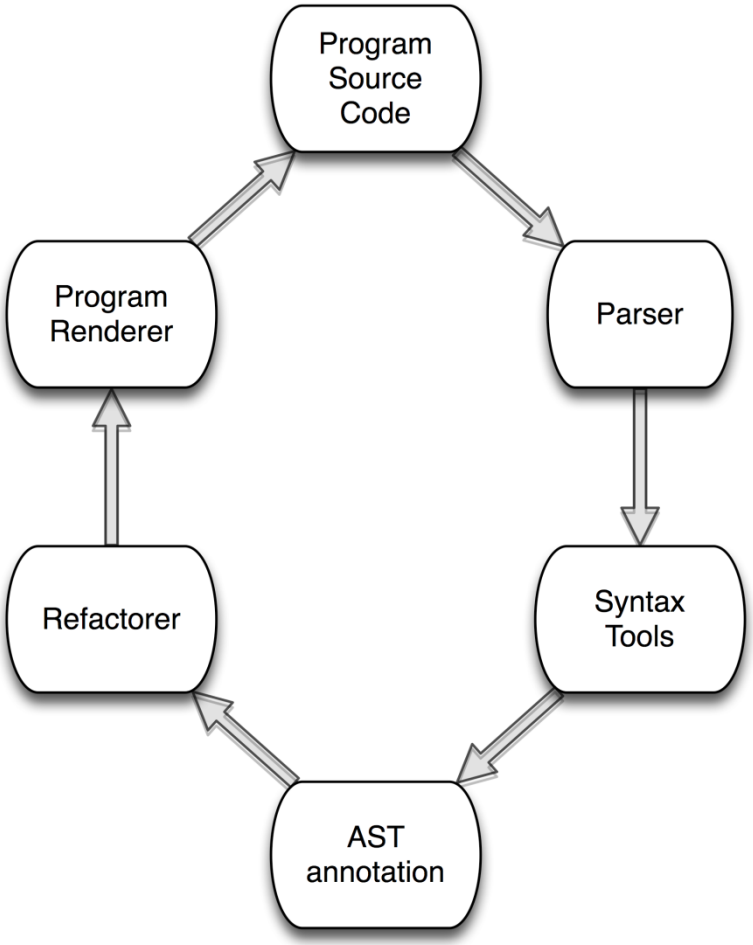
…as by hand they are tedious and error-prone.

Decision support for more complex tasks …

… don’t try to make them “push button”.

Clone detection experience validates this.

Architecture of Wrangler



ProTest property based testing

University of Kent Computing

Semantic analysis

Binding structure

- Dynamic atom creation, multiple binding occurrences, pattern semantics etc.

Module structure and projects

- No explicit projects for Erlang; cf Erlide / Emacs.

Type and effect information

- Need effect information for e.g. generalisation.

Erlang refactoring: challenges

Multiple binding occurrences of variables.

Indirect function call or function spawn:

```
apply (lists, rev,  [[a,b,c]])
```

Multiple arities … multiple functions: `rev/1`

Concurrency

Refactoring within a design library: OTP.

Side-effects.

Static *vs* dynamic

Aim to check conditions statically.

Static analysis tools possible … but some aspects intractable: e.g. dynamically manufactured atoms.

Conservative *vs* liberal.

Compensation?

emacs@HL-LT

File Edit Options Buffers Tools Refactor Inspector Erlang Help

```
        {msg,_Msg,0} -> lo
        {msg,Msg,N} ->
            io:format("pin
            timer:sleep(50
            b!{msg,Msg,N+1
            loop_a()
        after 15000 -> io:

                        exi
    end.

loop_b() ->
    receive
        stop -> ok;
        {msg,_Msg,0} -> lo
        {msg,Msg,N} ->
            io:format("pon
            timer:sleep(50
            a!{msg,Msg,N+1
            loop_b()
        after 15000 -> io:

                        exi
    end.
```

--\--- pingpong.erl<2> Bot (51,0) (Erlang)

Rename Variable Name	C-c C-w r v
Rename Function Name	C-c C-w r f
Rename Module Name	C-c C-w r m
Generalise Function Definition	C-c C-g
Move Function to Another Module	C-c C-w m
Function Extraction	C-c C-w n f
Introduce New Variable	C-c C-w n v
Inline Variable	C-c C-w i
Fold Expression Against Function	C-c C-w f f
Tuple Function Arguments	C-c C-w t
Unfold Function Application	C-c C-w u
Introduce a Macro	C-c C-w n m
Fold Against Macro Definition	C-c C-w f m
Identical Code Detection	▸
Similar Code Detection	▸
Refactorings for QuickCheck	▸
Process Refactorings (Beta)	▸
Normalise Record Expression	
Partition Exported Functions	
gen_fsm State Data to Record	
Undo	C-c C-w _
Customize Wrangler	
Version	

emacs@HL-LT

File Edit Options Buffers Tools Refactor Inspector Erlang Help

- Instances of a Variable C-c C-w b
- Calls to a Function
- Dependencies of a Module
- Nested If Expressions
- Nested Case Expressions
- Nested Receive Expression
- Long Functions
- Large Modules
- Generate Function Callgraph
- Generate Module Graph
- Cyclic Module Dependency
- Improper Inter Module Dependency
- Show Non Tail Recursive Servers
- Incomplete Receive Patterns

```
    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(

                        exit(timeo
    end.

loop_b() ->
    receive
        stop -> ok;
        {msg,_Msg,0} -> loop_b();
        {msg,Msg,N} ->
            io:format("pong!~n"),
            timer:sleep(500),
            a!{msg,Msg,N+1},
            loop_b()
        after 15000 -> io:format("Pong got bored, "
                                 "exiting.~n"),
                        exit(timeout)
    end.
```

--\--- pingpong.erl<2> Bot (56,14) (Erlang)----------

Refactorings in Wrangler

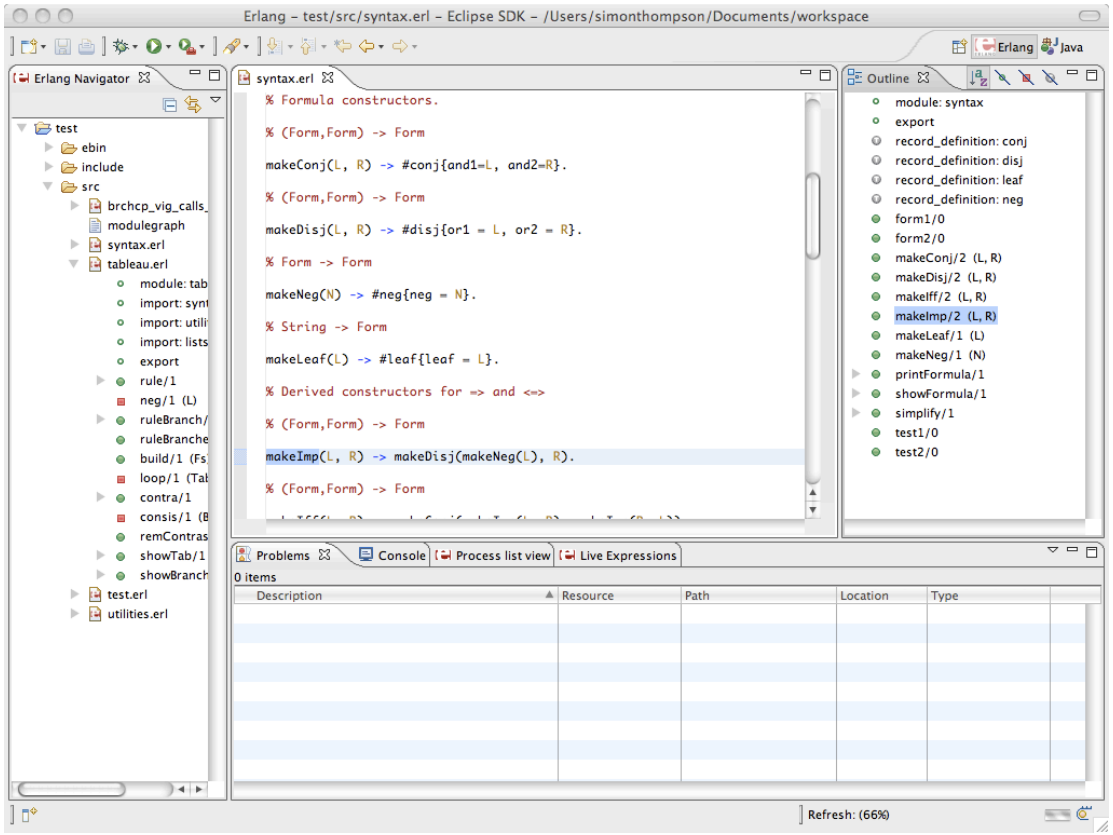
- Renaming variable, function, module, process
- Introduce/inline variables
- Function generalisation
- Move function(s) between modules.
- Function extraction
- Fold against definition
- Introduce and fold against macros.
- Tuple function arguments
- Register a process
- From function to process
- Add a tag to messages
- Quickcheck-related refactorings.

All these refactorings work across multiple-module projects and respect macro definitions.

Integration with ErlIDE

Tighter control of what's a project.

Potential for adoption by newcomers to the Erlang community.



ProTest property based testing

University of Kent Computing

Tool Demo

Clone detection

Duplicate code considered harmful

It's a *bad smell* …

- increases chance of bug propagation,
- increases size of the code,
- increases compile time, and,
- increases the cost of maintenance.

But … it's not always a problem.

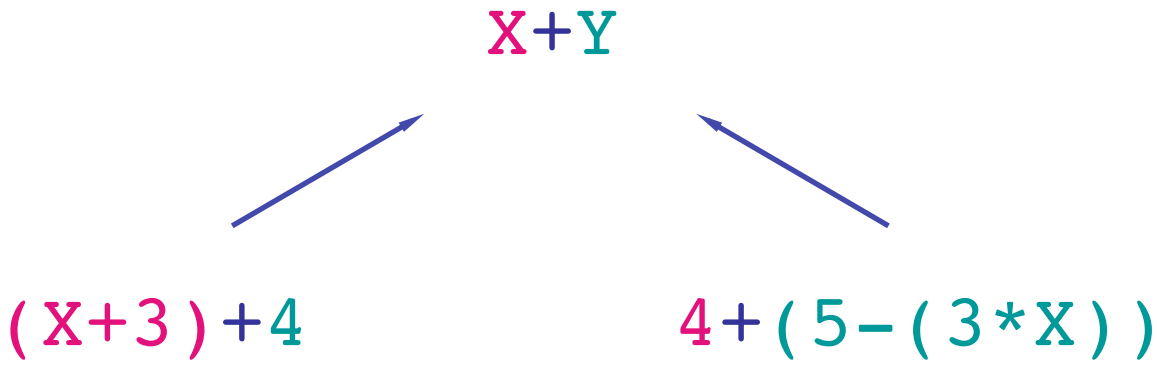
What is ‘identical’ code?

variable+number

`X+4` `Y+5`

Identical if values of literals and variables ignored, but respecting binding structure.

What is ‘similar’ code?



The anti-unification gives the (most specific) common generalisation.

Clone detection

- The Wrangler clone detector
  - relatively efficient
  - no false positives
- User-guided interactive removal of clones.
- Integrated into development environments, but can also be run from an Erlang shell.

Detection	Expression search
All clones in a project meeting the threshold parameters …	All instances of expressions similar to this expression …
… and their common generalisation.	… and their common generalisation.
Default threshold: ≥ 5 expressions and similarity of ≥ 0.8.	Default threshold: similarity ≥ 0.8.

Similarity

Threshold: anti-unifier should be big enough relative to the class members:

$$\text{similarity} = \min\left(\frac{\|\texttt{X+Y}\|}{\|\texttt{(X+3)+4}\|}, \frac{\|\texttt{X+Y}\|}{\|\texttt{4+(5-(3*X))}\|}\right)$$

Can also threshold length of expression sequence, or number of tokens, or … .

emacs@HL-LT

File Edit Options Buffers Tools Errors Help

```
Similar detection finished with *** 30 *** clone(s) found.

Clone 1. This code appears 21 times:
c:/cygwin/home/hl/test/smm_SUITE.erl:1906.4-1912.71:
   new_fun()
c:/cygwin/home/hl/test/smm_SUITE.erl:2181.4-2187.71:
   new_fun()
c:/cygwin/home/hl/test/smm_SUITE.erl:229.4-235.71:
   new_fun()
        .
        .
        .
c:/cygwin/home/hl/test/smm_SUITE.erl:836.4-842.71:
   new_fun()
c:/cygwin/home/hl/test/smm_SUITE.erl:1059.4-1065.71:
   new_fun()
c:/cygwin/home/hl/test/smm_SUITE.erl:2908.4-2914.71:
   new_fun()

The cloned expression/function after generalisation:

new_fun() ->
    RSSetResult = ?SMM_IMPORT_FILE_BASIC(?SMM_RULESET_FILE_1,no),
    ?TRIAL(ok,RSSetResult),

    AmountOfRuleSets = ?SMM_RULESET_FILE_1_COUNT,
    ?OM_CHECK(AmountOfRuleSets,?MP_BS,ets,info,[sbgRuleSetTable,size]),
    ?OM_CHECK(AmountOfRuleSets,?SGC_BS,ets,info,[smmRuleSet,size]),
    AmountOfRuleSets.
```

-1**- *erl-output* 1% (20,0) (Fundamental Compilation)----------

ProTest property based testing

University of Kent Computing

emacs@HL-LT

File Edit Options Buffers Tools Errors Help

```
Clone 29. This code appears twice:
c:/cygwin/home/hl/test/smm_SUITE.erl:2904.4-2989.69:
   new_fun("This test case will check the use of "
         "ISM support table for 'Rule Set Usage'.~nInst"
         "ances will be changed when filter names "
         "are changed.~n")
c:/cygwin/home/hl/test/smm_SUITE.erl:2760.4-2845.69:
   new_fun("This test case will check the use of "
         "ISM support table for 'Rule Set Usage'.~nInst"
         "ances will be removed when rule sets "
         "are removed.~n")

The cloned expression/function after generalisation:

new_fun(NewVar_1) ->
    ?COMMENT(
       NewVar_1,[]),

    RSSetResult = ?SMM_IMPORT_FILE_BASIC(?SMM_RULESET_FILE_1,no),
    ?TRIAL(ok,RSSetResult),

    AmountOfRuleSets = ?SMM_RULESET_FILE_1_COUNT,
    ?OM_CHECK(AmountOfRuleSets,?MP_BS,ets,info,[sbgRuleSetTable,size]),
    ?OM_CHECK(AmountOfRuleSets,?SGC_BS,ets,info,[smmRuleSet,size]),

    FilterStateAtom = notUsed,

    FilterName1 = "Filter_1",
    CreateFilter1 = ?SMM_CREATE_FILTER(FilterName1),
-1\**-  *erl-output*   93% (2316,0)   (Fundamental Compilation)----------
```

ProTest property based testing

University of Kent Computing

Tool Demo

Improve Module Structure

Maintaining modularity

Modularity tends to deteriorate over time.

Repair with incremental modularity maintenance.

Four modularity “bad smells”.

Cyclic module dependencies.

Export of functions that are “really” internal.

Modules with multiple purposes.

Very large modules.

Refactoring: move functions

Move a group of functions from one module to another.

Which functions to move? Move to where? How?

Wrangler provides:

1. Modularity smell detection
2. Refactoring suggestions
3. Refactoring

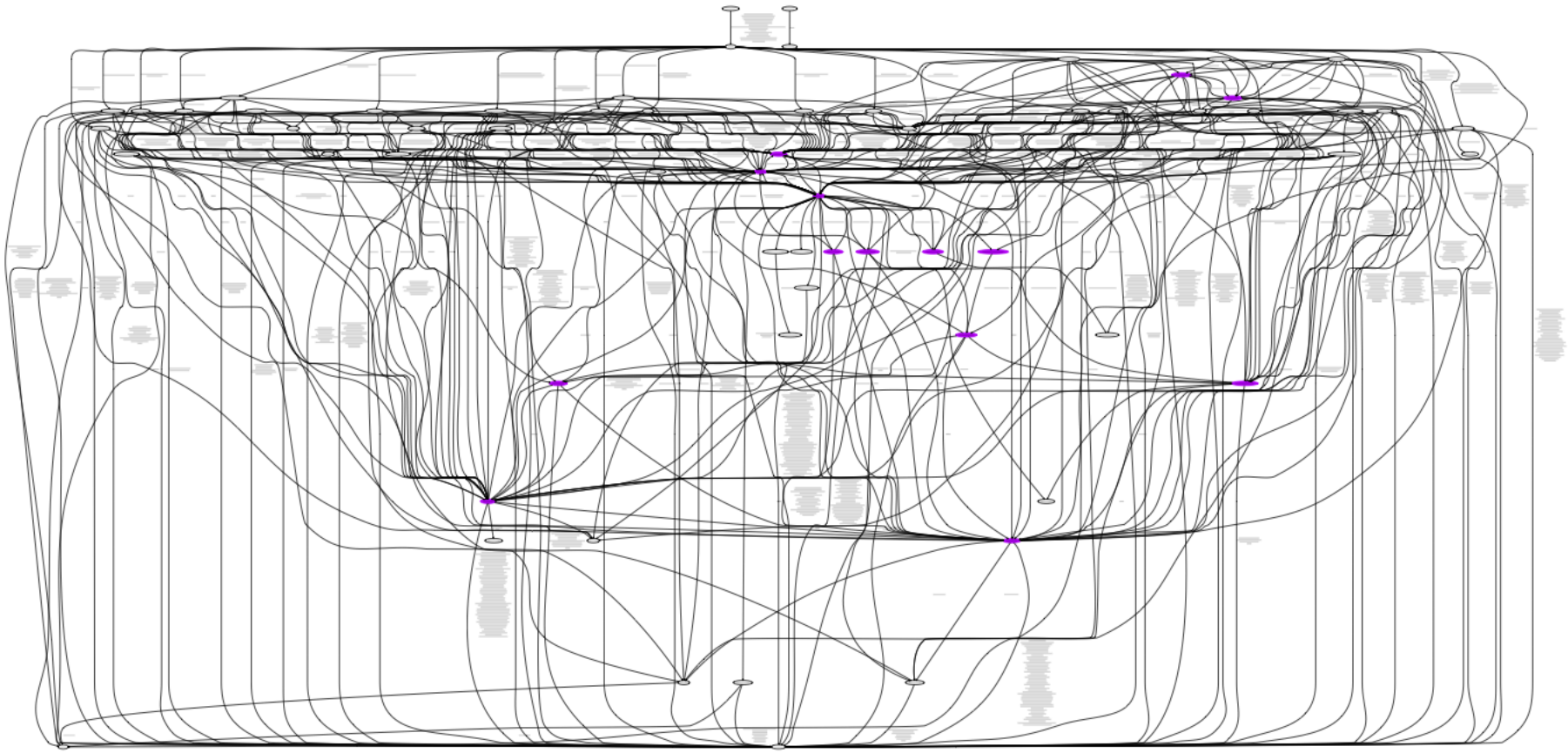
“Dogfooding” Wrangler

Case study of Wrangler-0.8.7

56 Erlang modules, 40 kloc (inc. comments).

- Improper dependencies: sharing implementation between refactorings.
- Cyclic dependencies: need to split modules.
- Multiple goals: `refac_syntax_lib` 7 clusters.

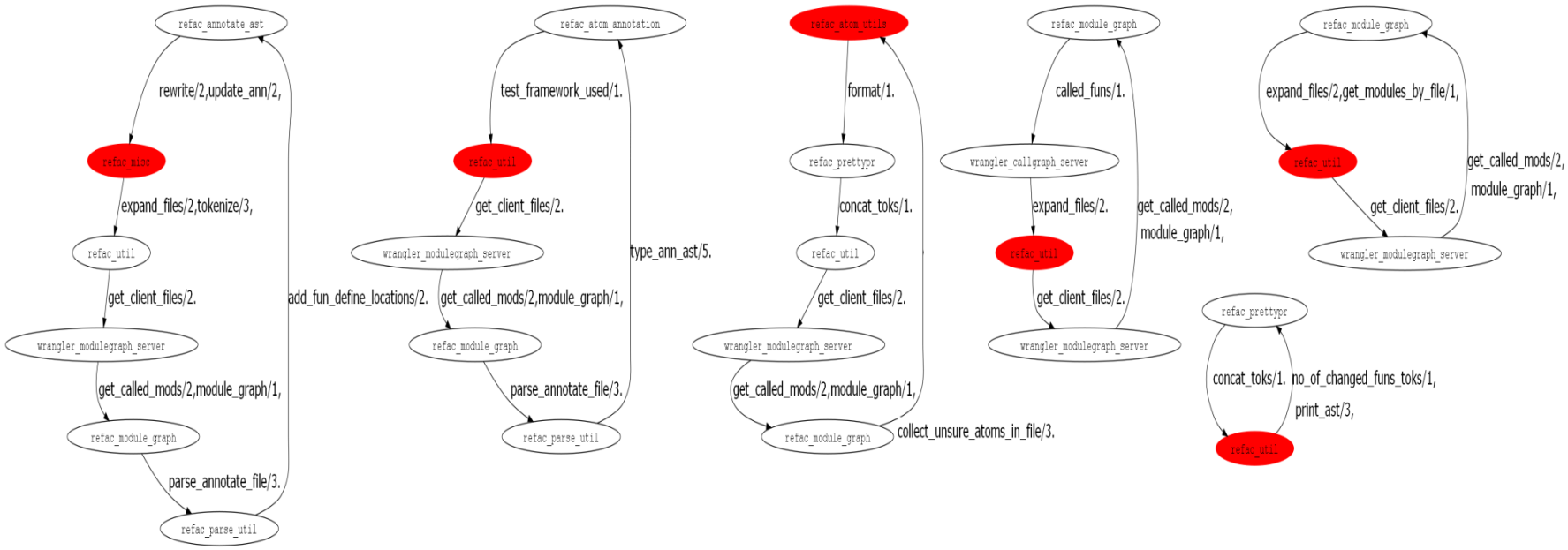
Wrangler module graph



ProTest
property based testing

University of Kent | Computing

Wrangler Cycles

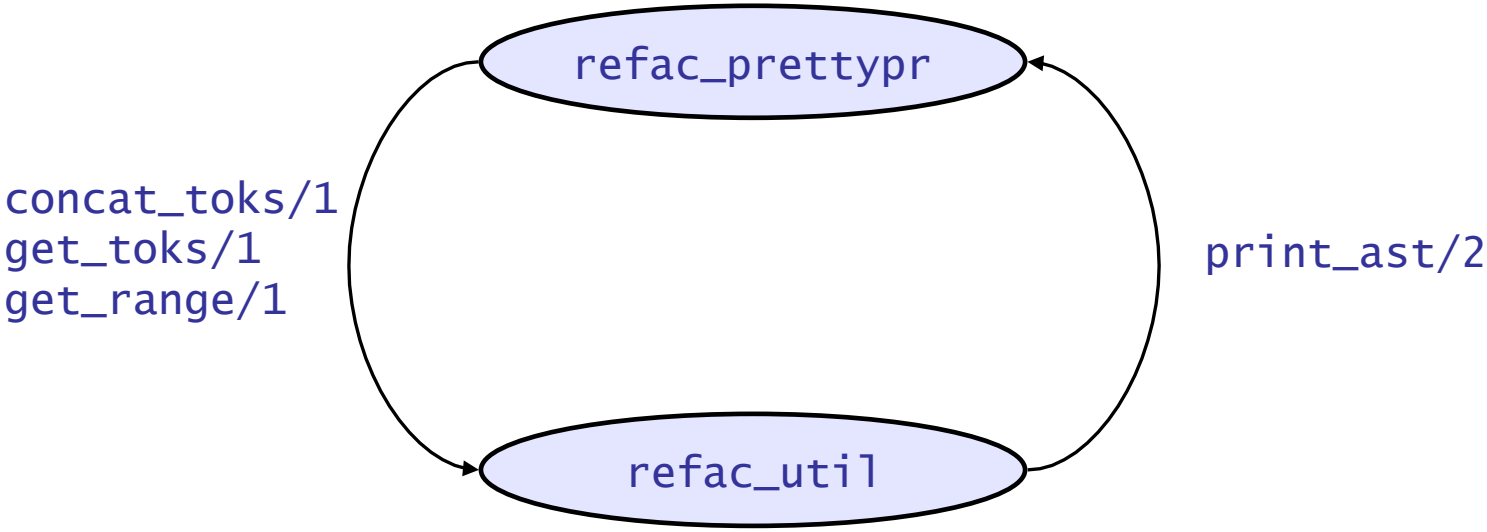


- Nodes in red are modules that need inspection.

ProTest property based testing

University of Kent Computing

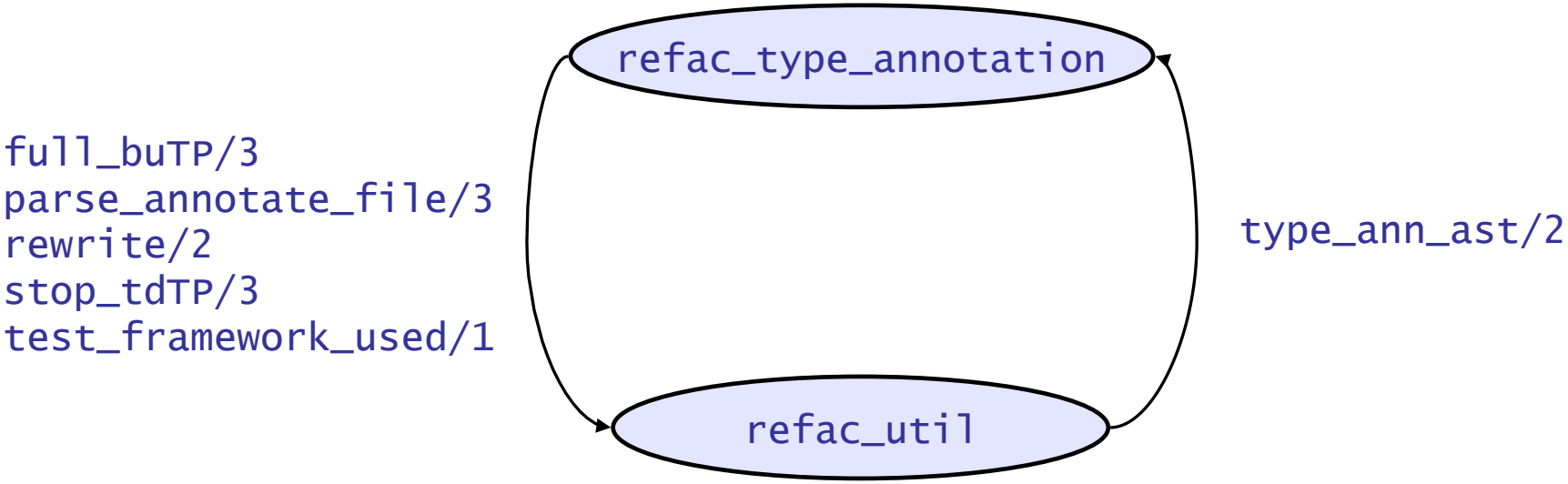
Inter-layer dependency




```
Inter-layer cyclic module dependency found:
   [refac_prettypr, refac_util, refac_prettypr]

Refactoring suggestion:
move_fun(refac_util, [{refac_util,write_refactored_files,1},
                      {refac_util,write_refactored_files,3},
                      {refac_util,write_refactored_files,4}],
                     user_supplied_target_mod).
```

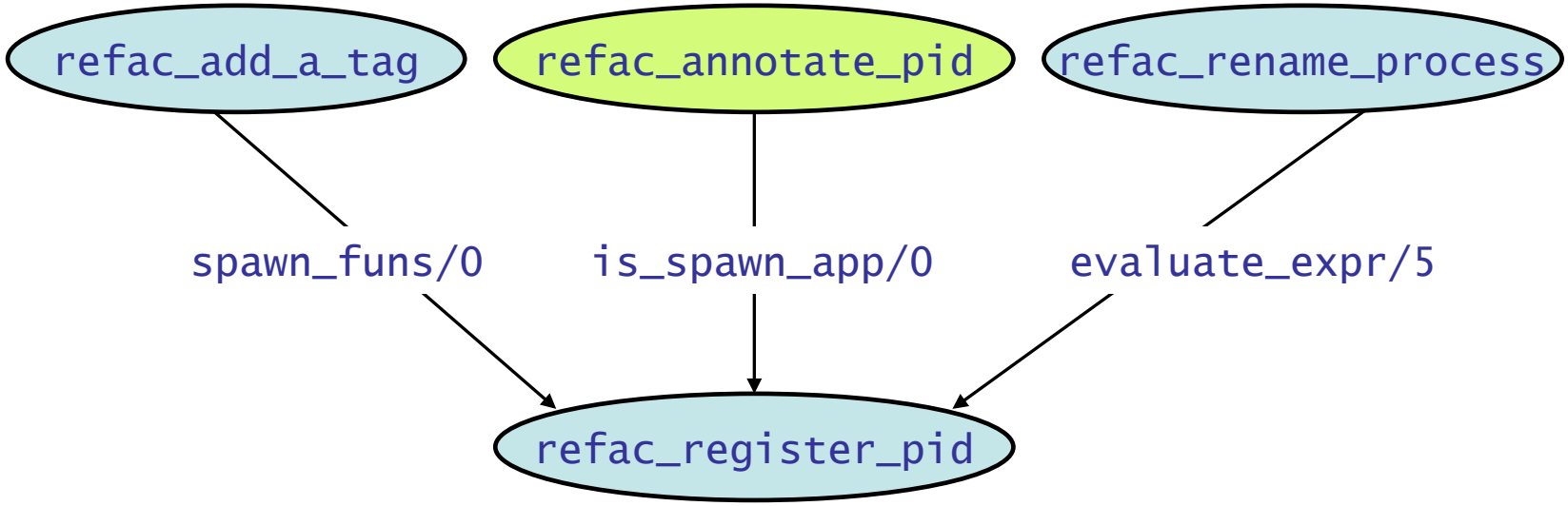
Intra-layer dependency



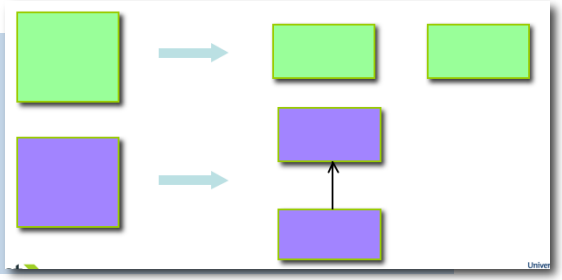
Identifying "API" functions

- Identify by examining call graph.
- API functions are those …
  - … not used internally,
  - … "close to" other API functions.
- Others are seen as *internal*, external calls to these are deemed *improper*.

Improper dependency



refac_syntax_lib.erl



Report on multi-goal modules: 12/56.

Agglomerative hierarchical algorithm.

Functions represented by feature lists … fed into Jaccard metric.

```
Module: refac_syntax_lib
Cluster 1, Indegree:25, OutDegree:1,
[{map,2}, {map_subtrees,2},
 {mapfold,3},{mapfold_subtrees,3},
 {fold,3}, {fold_subtrees,3}]

Cluster 2, Indegree:0, OutDegree:0,
[{foldl_listlist,3},{mapfoldl_listlist,3}]

Cluster 3, Indegree:0, OutDegree:0,
[{new_variable_name,1},{new_variable_names,2},
 {new_variable_name,2},{new_variable_names,3}]

Cluster 4, Indegree:4, OutDegree:1,
[{annotate_bindings,2},{annotate_bindings,3},
 {var_annotate_clause,4},{vann_clause,4},
 {annotate_bindings,1}]

 …
```

Future work

Incremental detection of module bad smells, e.g. in overnight builds.

Partition module exports according to client modules.

Case studies.

Improve module structure

- Refactoring
  - Move function(s) from one module to another.
    - select a function definition to move a single function, an export list to move a collect of functions.
  - Partition module exports.

emacs@HL-LT

File Edit Options Buffers Tools Refactor Inspector Erlang Help

```
%% @type syntaxTree() = refac_syntax:syntaxTree(). An abstract syntax
%% tree. See the <code>erl_syntax</code> module for details.

-module(refac_syntax_lib).

-export([ analyze_application/1,
          analyze_attribute/1, analyze_export_attribute/1,
          analyze_file_attribute/1, analyze_form/1,
          analyze_forms/1, analyze_function/1,
          analyze_function_name/1, analyze_implicit_fun/1,
          analyze_import_attribute/1, analyze_module_attribute/1,
          analyze_record_attribute/1, analyze_record_expr/1,
          analyze_record_field/1, analyze_rule/1,
          analyze_wild_attribute/1, annotate_bindings/1,
          annotate_bindings/2, annotate_bindings/3,
          fold/3, fold_subtrees/3,
          foldl_listlist/3, function_name_expansions/1,
          is_fail_expr/1, limit/2, limit/3, map/2, map_subtrees/2,
          mapfold/3, mapfold_subtrees/3, mapfoldl_listlist/3,
          new_variable_name/1, new_variable_name/2,
          new_variable_names/2, new_variable_names/3,
          strip_comments/1, to_comment/1, to_comment/2,
          to_comment/3, variables/1, var_annotate_clause/4]).

%% ====================================================================
%% @spec map(Function, Tree::syntaxTree()) -> syntaxTree()
%%
%%          Function = (syntaxTree()) -> syntaxTree()
%%
%% @doc Applies a function to each node of a syntax tree. The result of
%% each application replaces the corresponding original node. The order
%% of traversal is bottom-up.
%%
--(Unix)---  refac_syntax_lib.erl    2% (51,54)    SVN-1561   (Erlang)-------------
```

ProTest property based testing

University of Kent Computing

emacs@HL-LT

File Edit Options Buffers Tools Refactor Inspector Erlang Help

```
%% @type syntaxTree() = re
%% tree. See the <code>erl

-module(refac_syntax_lib).

-export([ analyze_applicat
          analyze_attribute
          analyze_file_attr
          analyze_forms/1,
          analyze_function_
          analyze_import_at
          analyze_record_at
          analyze_record_fi
          analyze_wild_attr
          annotate_bindings
          fold/3, fold_subt
          foldl_listlist/3,
          is_fail_expr/1, l
          mapfold/3, mapfol
          new_variable_name
          new_variable_name
          strip_comments/1,
          to_comment/3, var

%% ======================
%% @spec map(Function, Tre
%%
%%          Function = (sy
%%
%% @doc Applies a function
%% each application replac
%% of traversal is bottom-
%%
--(Unix)---  refac_syntax_lib.erl
```

Rename Variable Name	C-c C-w r v
Rename Function Name	C-c C-w r f
Rename Module Name	C-c C-w r m
Generalise Function Definition	C-c C-g
Move Function to Another Module	C-c C-w m
Function Extraction	C-c C-w n f
Introduce New Variable	C-c C-w n v
Inline Variable	C-c C-w i
Fold Expression Against Function	C-c C-w f f
Tuple Function Arguments	C-c C-w t
Unfold Function Application	C-c C-w u
Introduce a Macro	C-c C-w n m
Fold Against Macro Definition	C-c C-w f m
Identical Code Detection	▸
Similar Code Detection	▸
Refactorings for QuickCheck	▸
Process Refactorings (Beta)	▸
Normalise Record Expression	
Partition Exported Functions	
gen_fsm State Data to Record	
Undo	C-c C-w _
Customize Wrangler	
Version	

ProTest property based testing

University of Kent Computing

emacs@HL-LT

File Edit Options Buffers Tools Refactor Inspector Erlang Help

```
%% @type syntaxTree() = refac_syntax:syntaxTree(). An abstract syntax
%% tree. See the <code>erl_syntax</code> module for details.

-module(refac_syntax_lib).

-export([is_fail_expr/1, analyze_function_name/1,
         analyze_implicit_fun/1, analyze_application/1,
         analyze_module_attribute/1, analyze_export_attribute/1,
         analyze_import_attribute/1, analyze_wild_attribute/1,
         analyze_function/1, analyze_rule/1,
         analyze_file_attribute/1, analyze_record_attribute/1,
         analyze_record_field/1, analyze_record_expr/1,
         analyze_forms/1, analyze_form/1, analyze_attribute/1]).

-export([map/2, map_subtrees/2, mapfold/3,
         mapfold_subtrees/3, strip_comments/1, fold/3,
         fold_subtrees/3]).

-export([new_variable_name/1, new_variable_names/2,
         new_variable_name/2, new_variable_names/3]).

-export([annotate_bindings/2, annotate_bindings/3,
         var_annotate_clause/4, annotate_bindings/1]).

-export([to_comment/1, to_comment/2, to_comment/3]).

-export([foldl_listlist/3, mapfoldl_listlist/3]).

-export([limit/2, limit/3]).

-export([function_name_expansions/1]).

%% ====================================================================
--(Unix)---  refac_syntax_lib.erl    2% (38,0)    SVN:1561  (Erlang)----------
```

ProTest property based testing

University of Kent Computing

Tool Demo

Hands-on

Installation: Mac OS X and Linux

Requires: Erlang release R11B-5 or later

Installation: Mac OS X and Linux

Download Wrangler from

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

or get it from the memory stick …

In the wrangler directory

```
./configure
make
sudo make install
```

Installation: Mac OS X and Linux

Add to `~/.emacs` file:

```
(add-to-list 'load-path
        "/usr/local/share/wrangler/elisp")
(require 'wrangler)
```

If you're installing emacs now, then you add the following lines to your `~/.emacs` file

```
(setq load-path (cons "/usr/local/otp/lib/tools-<ToolsVer>/emacs"
                      load-path))
(setq erlang-root-dir "/usr/local/otp")
(setq exec-path (cons "/usr/local/otp/bin" exec-path))
(require 'erlang-start)
```

Installation: Windows

Requires R11B-5 or later + Emacs

Download installer from

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

Requires no other actions.

Installation: Eclipse + ErlIDE

Requires Erlang R11B-5 or later, if it isn't already present on your system.

On Windows systems, use a path with no spaces in it.

Install Eclipse 3.5, if you didn't already.

All the details at

http://erlide.sourceforge.net/

Starting Wrangler in Emacs

Open emacs, and open a `.erl` file.

`M-x erlang-refactor-on` or ...

... `C-c, C-r`

New menus: Refactor and Inspector

Customise for dir

Undo `C-c, C-w, _`

Preview Feature

Preview changes before confirming the change

Emacs `ediff` is used.

Stopping Wrangler in Emacs

`M-x erlang-refactor-off` to stop Wrangler

Shortcut `C-c, C-r`

Tutorial materials

Exercises:

http://www.cs.kent.ac.uk/projects/wrangler/Misc/WranglerExercise.{doc.pdf}

Code:

http://www.cs.kent.ac.uk/projects/wrangler/Misc/wrangler_ex.tar.gz

Carrying on …

Try on your own project code …

Feedback:

`erlang-refactor@kent.ac.uk` or
`H.Li@kent.ac.uk`