

# Getting the right module structure: finding and fixing problems in your projects

Simon Thompson, Huiqing Li

School of Computing, University of Kent, UK

# Overview

Refactoring Erlang in Wrangler

Clone detection and elimination

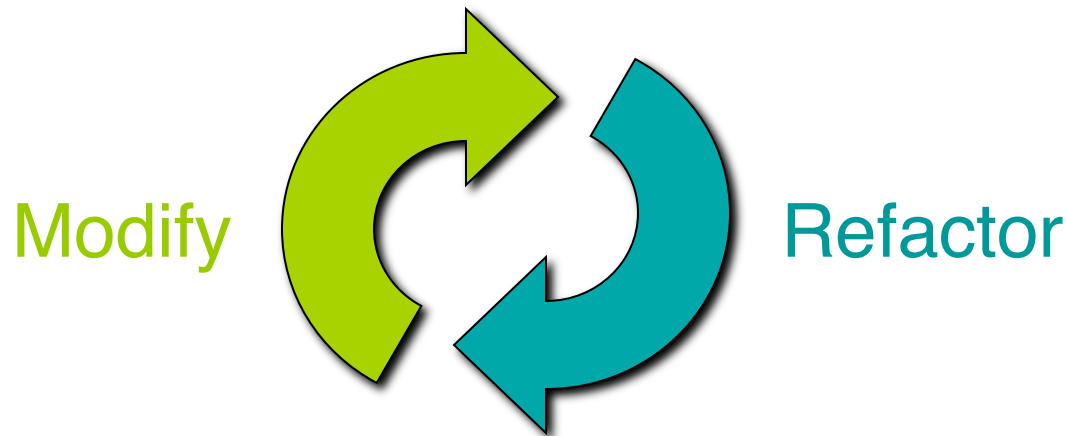
Case study: SIP message manipulation

Improving module structure

# Introduction

# Refactoring

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

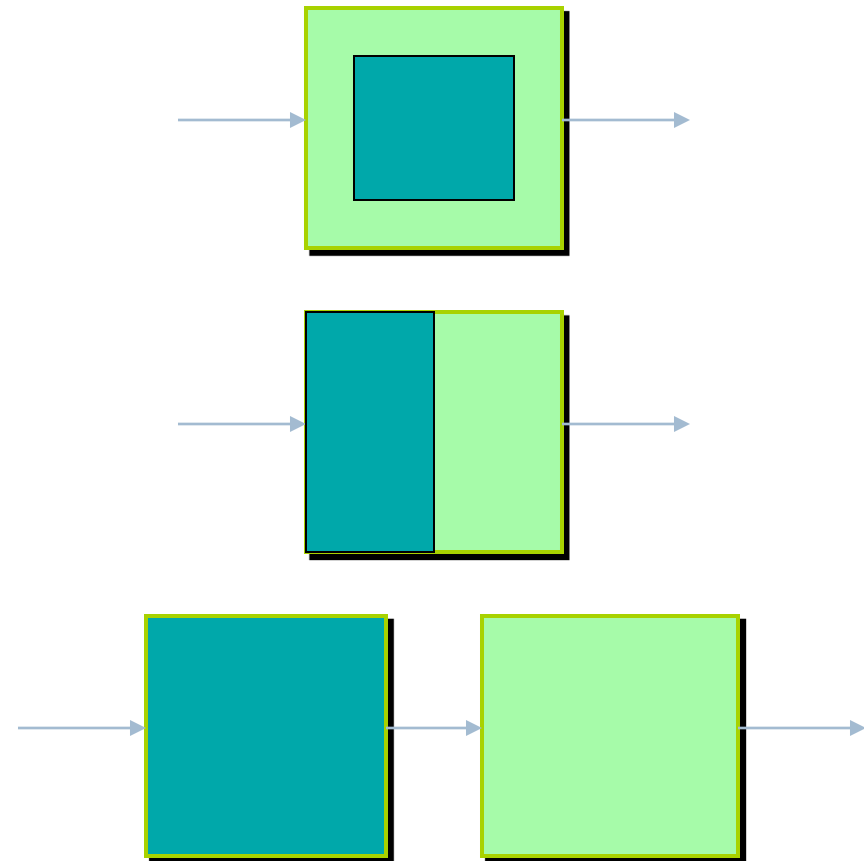


# Soft-ware

There's no single correct design ...

... different options for different situations.

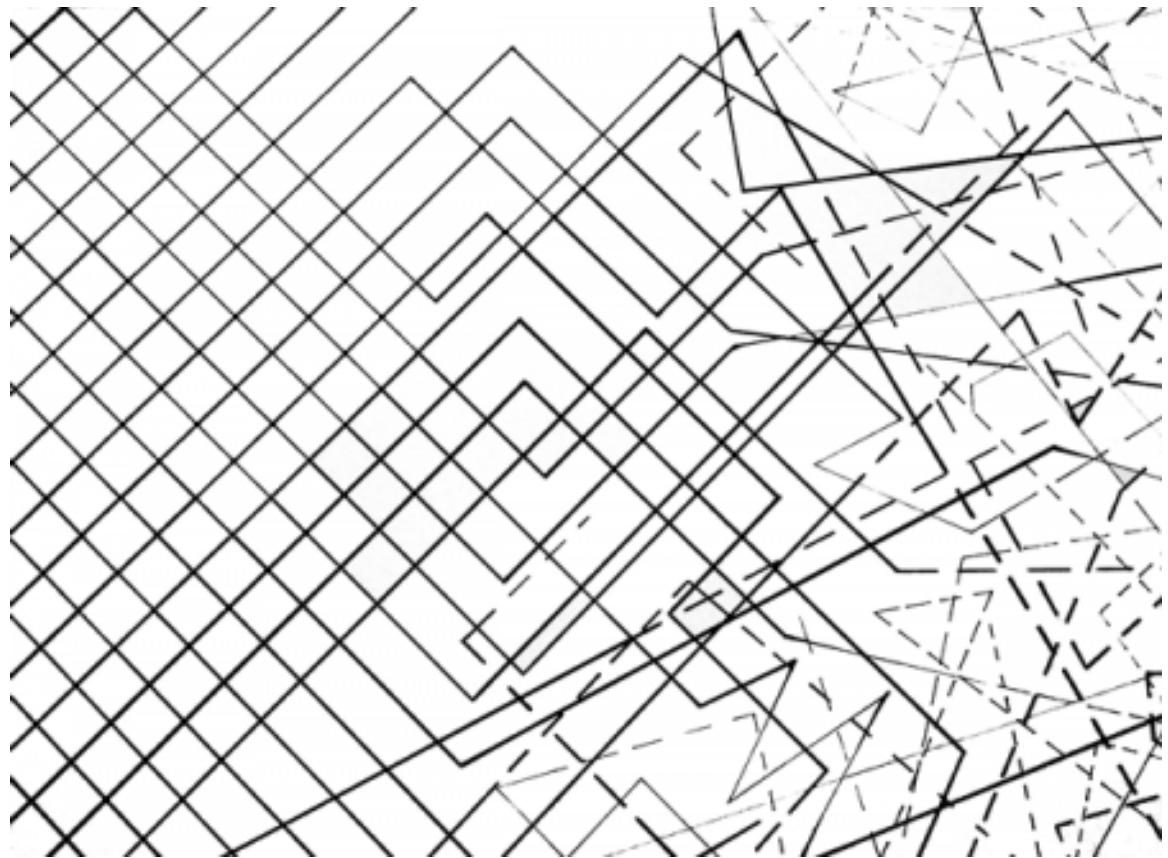
Maintain flexibility as the system evolves.



# From order to chaos ...

The best designs decay ...

- Clones
- Module structure "bad smells".
- ...



# 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).
```

# Refactoring tool support

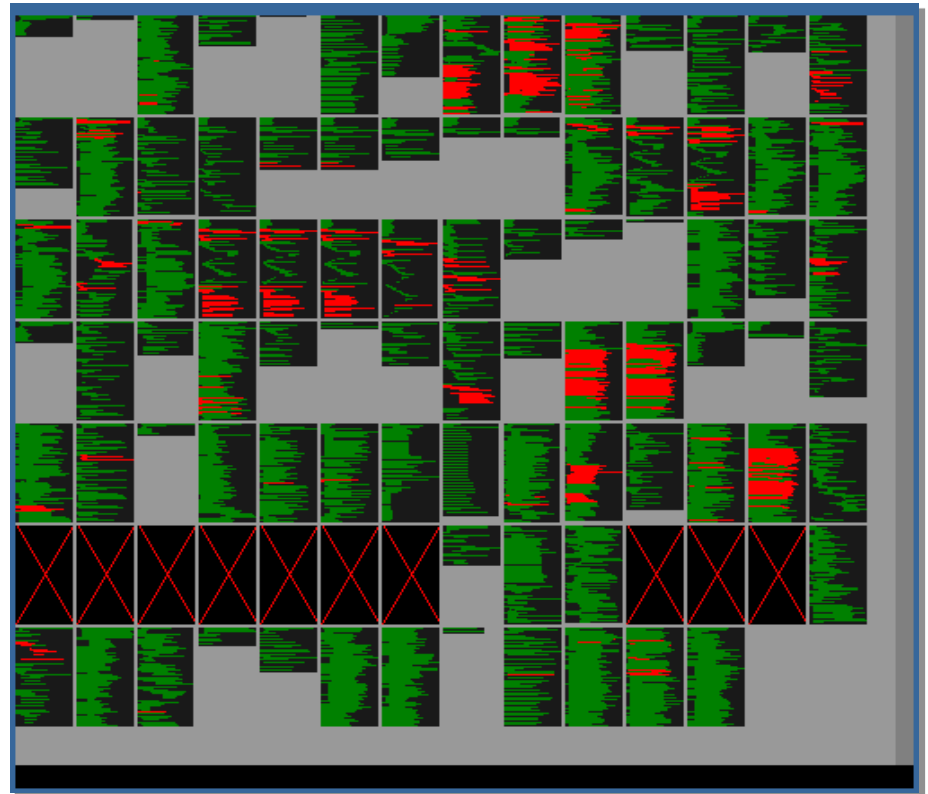
Bureaucratic and diffuse.

Tedious and error prone.

Semantics: scopes, types, modules, ...

Undo/redo

Enhanced creativity





# 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.



```

-module(test_kill).

-compile(export_all).

foo() ->
    register(foo, spawn(test_kill,foo,[]))

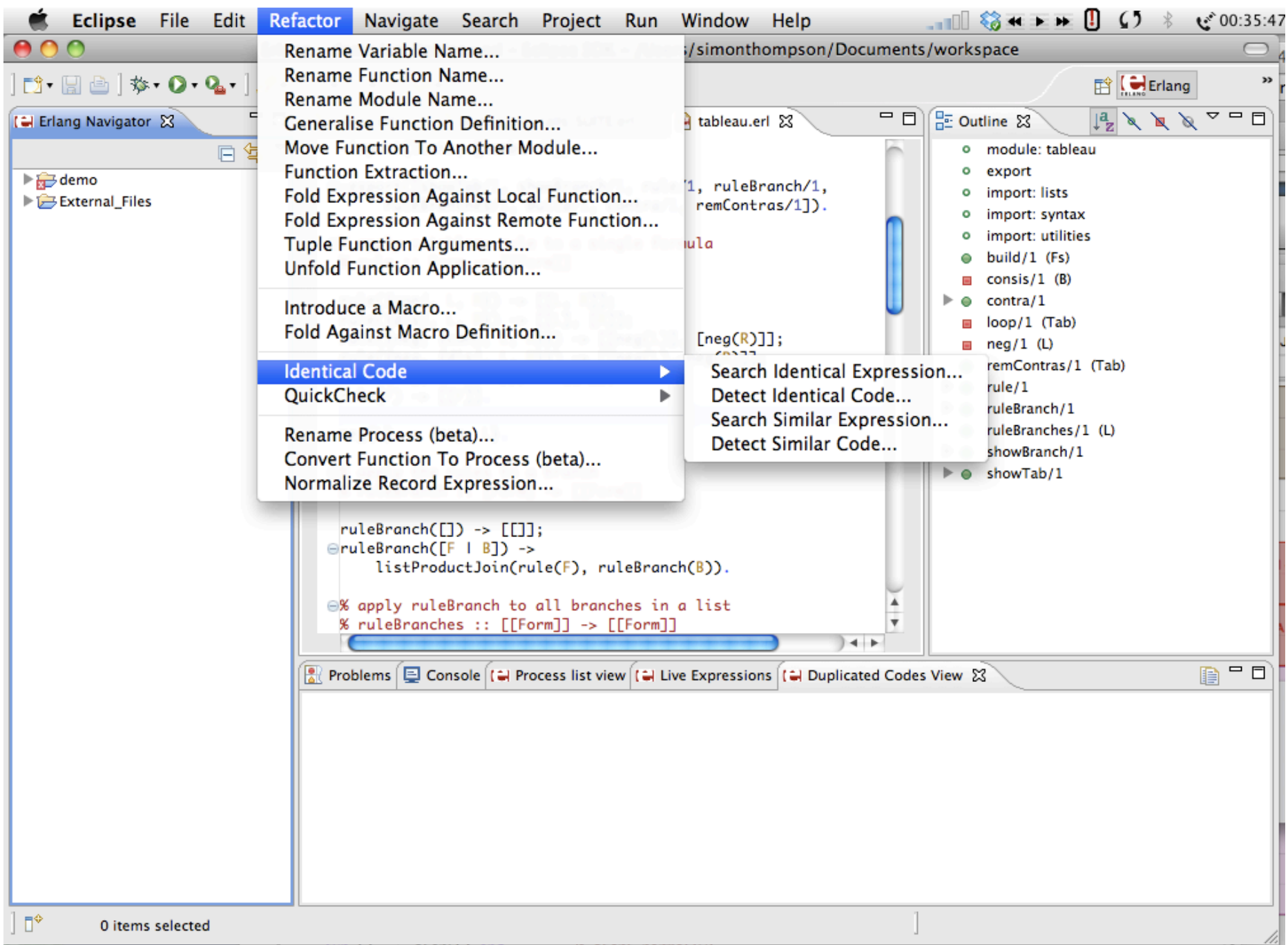
foo() ->
    register(bar,spawn_link(test_kill,bar,[]))
    receive
    after 1000 ->
        io:format("tick~n",[]),
        foo()
    end.

bar() ->
    receive
    after 1000 ->
        io:format("tock~n",[]),
        bar()
    end.

```

- 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
- 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)



- Rename Variable Name...
- Rename Function Name...
- Rename Module Name...
- Generalise Function Definition...
- Move Function To Another Module...
- Function Extraction...
- Fold Expression Against Local Function...
- Fold Expression Against Remote Function...
- Tuple Function Arguments...
- Unfold Function Application...

- Introduce a Macro...
- Fold Against Macro Definition...

- Identical Code** ▶
- QuickCheck ▶

- Rename Process (beta)...
- Convert Function To Process (beta)...
- Normalize Record Expression...

- Search Identical Expression...
- Detect Identical Code...
- Search Similar Expression...
- Detect Similar Code...

```

ruleBranch([]) -> [];
ruleBranch([F | B]) ->
    listProductJoin(rule(F), ruleBranch(B)).

% apply ruleBranch to all branches in a list
% ruleBranches :: [[Form]] -> [[Form]]

```

Outline

- module: tableau
- export
- import: lists
- import: syntax
- import: utilities
- build/1 (Fs)
- consis/1 (B)
- contra/1
- loop/1 (Tab)
- neg/1 (L)
- remContras/1 (Tab)
- rule/1
- ruleBranch/1
- ruleBranches/1 (L)
- showBranch/1
- showTab/1

- Problems
- Console
- Process list view
- Live Expressions
- Duplicated Codes View

# 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.

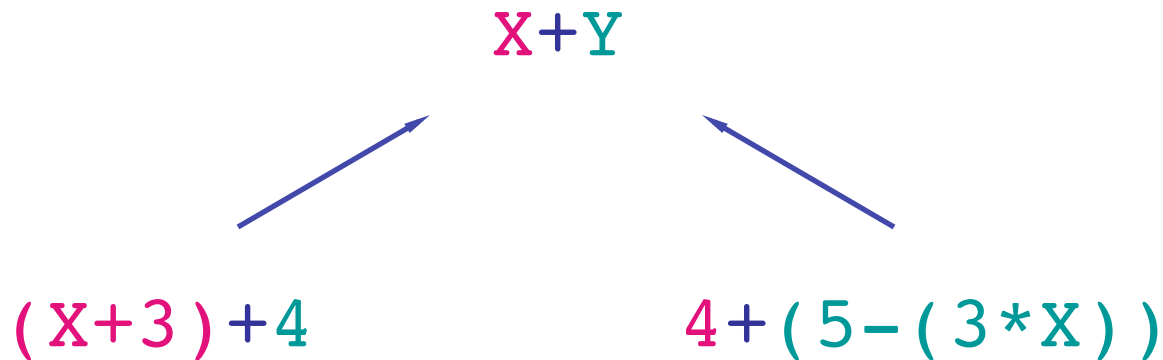
But ... it's not always a problem.

# Clone detection

- Hybrid clone detector
  - relatively efficient (suffix tree)
  - no false positives (AST analysis)
- User-guided interactive removal of clones.
- Integrated into development environments.



# What is 'similar' code?



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

# Detection

All clones in a project meeting the threshold parameters ...

... and their common generalisations.

Default threshold:  
 $\geq 5$  expressions and  
similarity of  $\geq 0.8$ .

# Expression search

All instances of expressions similar to this expression ...

... and their common generalisation.

Default threshold:  
 $\geq 20$  tokens.

# Similarity

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

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

Can also threshold number of expressions, number of tokens, number of new variables or . . . .

# Demo

# SIP Case Study

# Why test code particularly?

Many people touch the code.

Write some tests ... write more by copy, paste and modify.

Similarly with long-standing projects, with a large element of legacy code.

# “Who you gonna call?”

Can reduce by 20% just by aggressively removing all the clones identified ...

... what results is of **no value at all**.

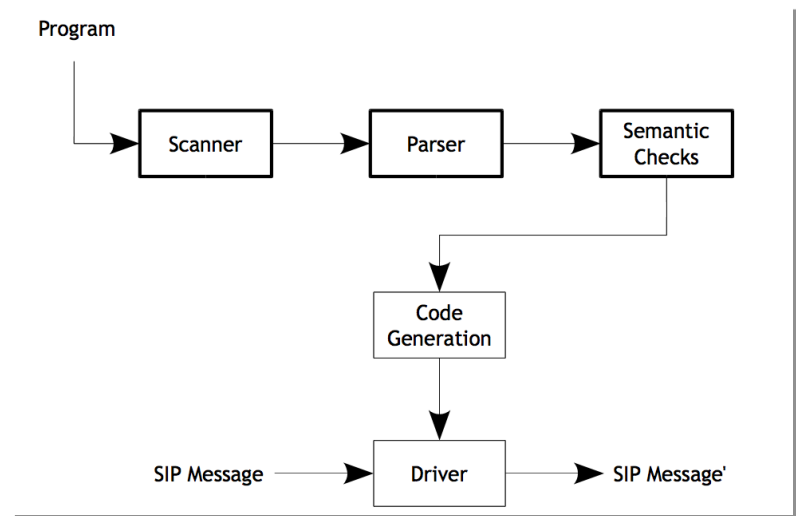
Need to call in the domain experts.

# SIP case study



SIP message manipulation allows rewriting rules to transform messages.

Test `smm_SUITE.erl`,  
2658 LOC.

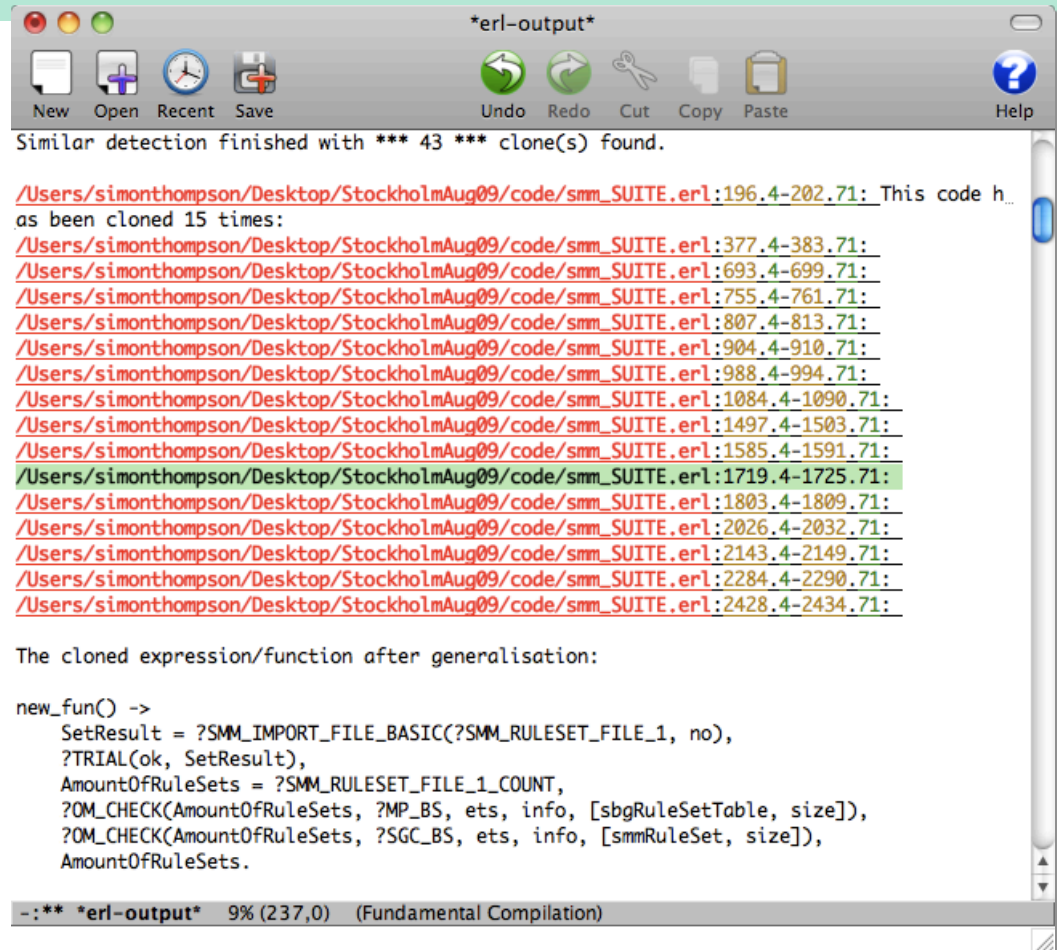




# Step 1

The largest clone class has 15 members.

The suggested function has no parameters, so the code is literally repeated.



```
*erl-output*
New Open Recent Save Undo Redo Cut Copy Paste Help
Similar detection finished with *** 43 *** clone(s) found.

/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:196.4-202.71: This code h...
as been cloned 15 times:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:377.4-383.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:693.4-699.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:755.4-761.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:807.4-813.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:904.4-910.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:988.4-994.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:1084.4-1090.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:1497.4-1503.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:1585.4-1591.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:1719.4-1725.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:1803.4-1809.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2026.4-2032.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2143.4-2149.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2284.4-2290.71:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2428.4-2434.71:

The cloned expression/function after generalisation:

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

-: ** *erl-output* 9% (237,0) (Fundamental Compilation)
```

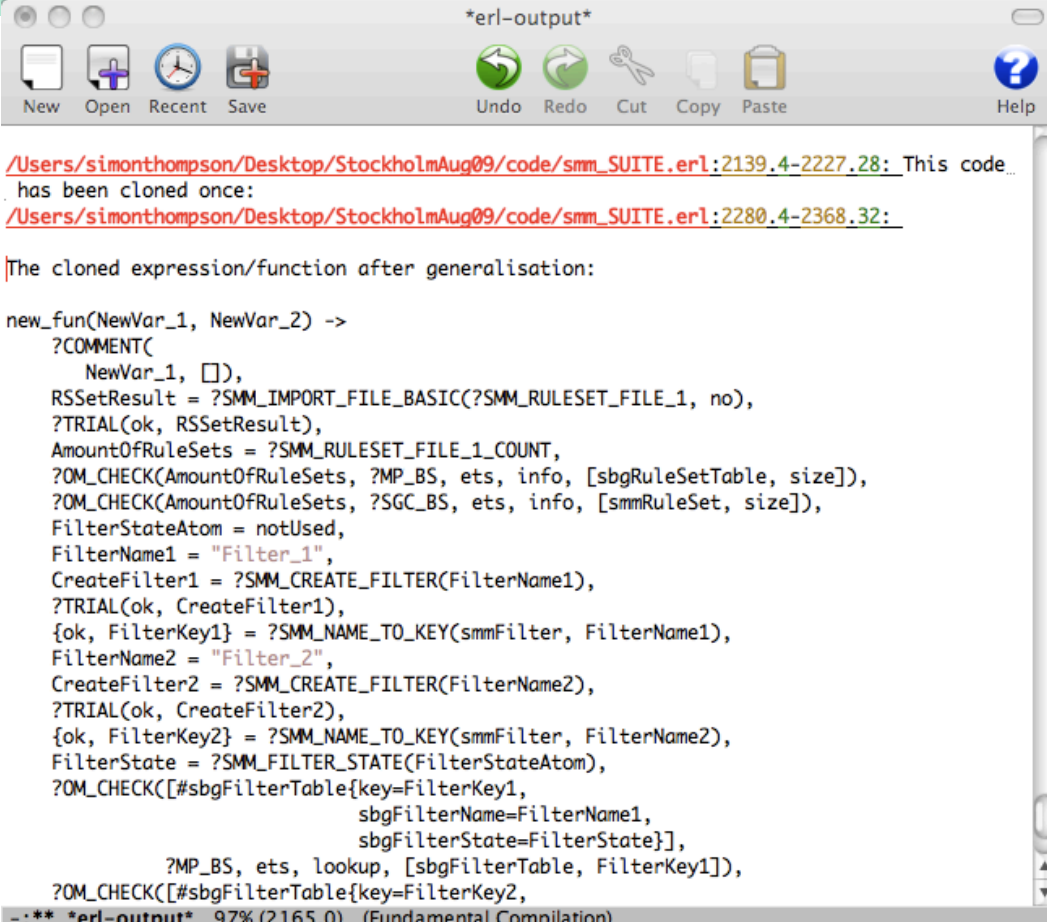
# Not step 1

The largest clone has 88 lines, and 2 parameters.

But what does it represent?

What to call it?

Best to work bottom up.



```
*erl-output*
New Open Recent Save Undo Redo Cut Copy Paste Help

/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2139.4-2227.28: This code
has been cloned once:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:2280.4-2368.32:

The cloned expression/function after generalisation:

new_fun(NewVar_1, NewVar_2) ->
  ?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),
  ?TRIAL(ok, CreateFilter1),
  {ok, FilterKey1} = ?SMM_NAME_TO_KEY(smmFilter, FilterName1),
  FilterName2 = "Filter_2",
  CreateFilter2 = ?SMM_CREATE_FILTER(FilterName2),
  ?TRIAL(ok, CreateFilter2),
  {ok, FilterKey2} = ?SMM_NAME_TO_KEY(smmFilter, FilterName2),
  FilterState = ?SMM_FILTER_STATE(FilterStateAtom),
  ?OM_CHECK([#sbgFilterTable{key=FilterKey1,
    sbgFilterName=FilterName1,
    sbgFilterState=FilterState}],
    ?MP_BS, ets, lookup, [sbgFilterTable, FilterKey1]),
  ?OM_CHECK([#sbgFilterTable{key=FilterKey2,
```

-.:\*\* \*erl-output\* 97% (2165,0) (Fundamental Compilation)

# The general pattern

Identify a clone.

Introduce the corresponding generalisation.

Eliminate all the clone instances.

So what's the complication?

# What is the complication?

Which clone to choose?

Include all the code?

How to name functions and variables?

When and how to generalise?

'Widows' and 'orphans'

# Module structure inspection

# 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



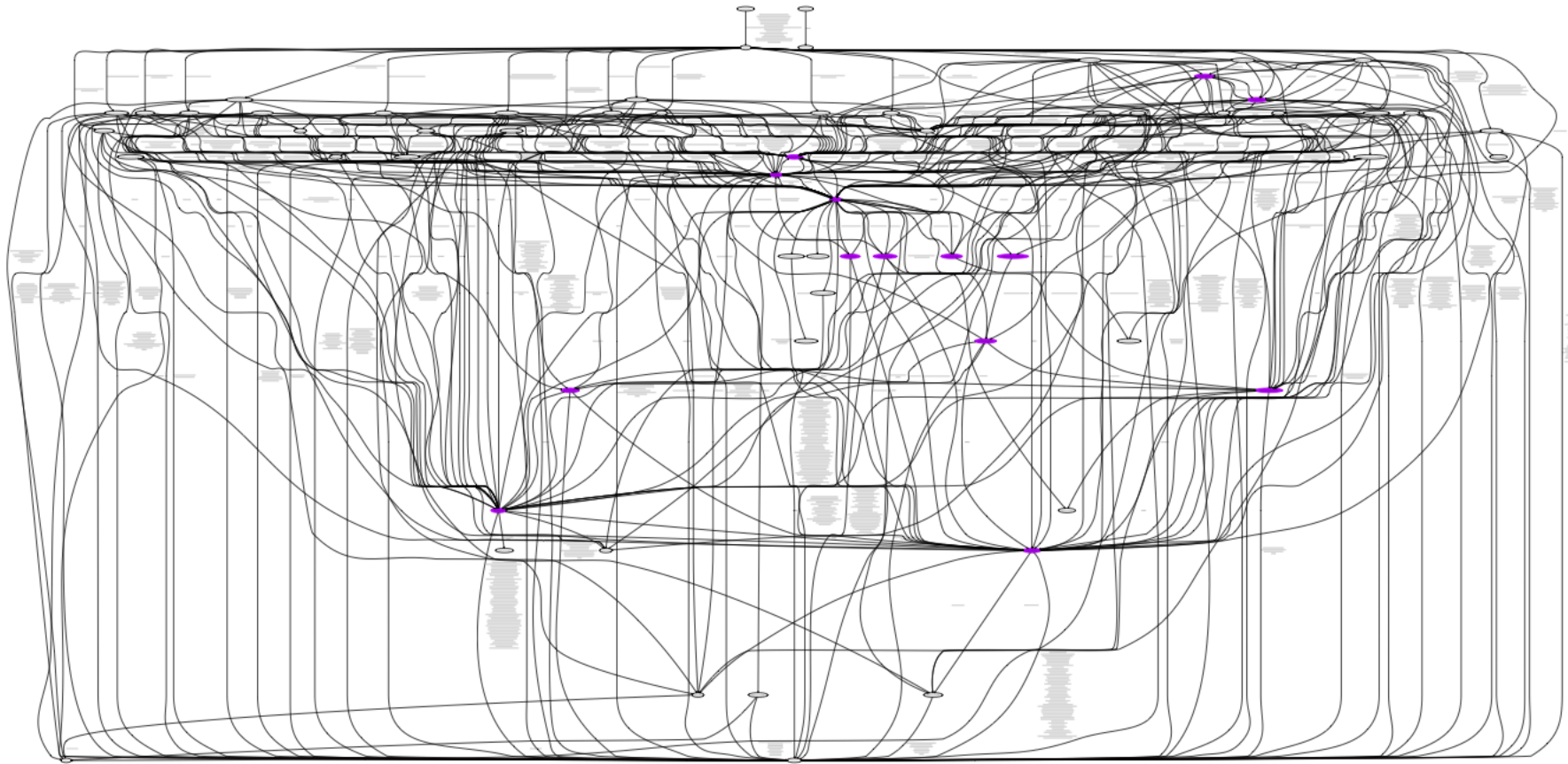
Case study of Wrangler-0.8.7

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.

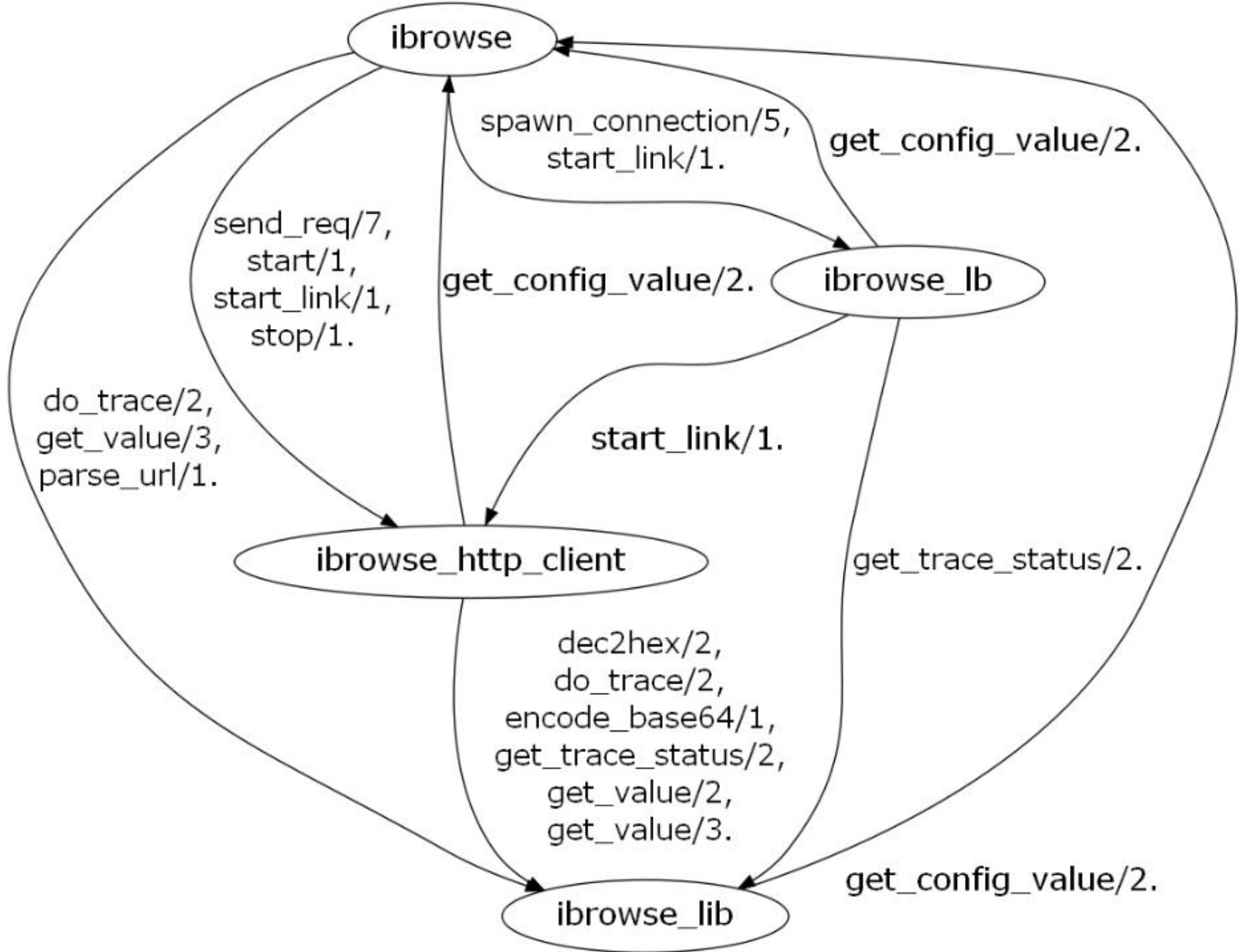


# Wrangler module graph

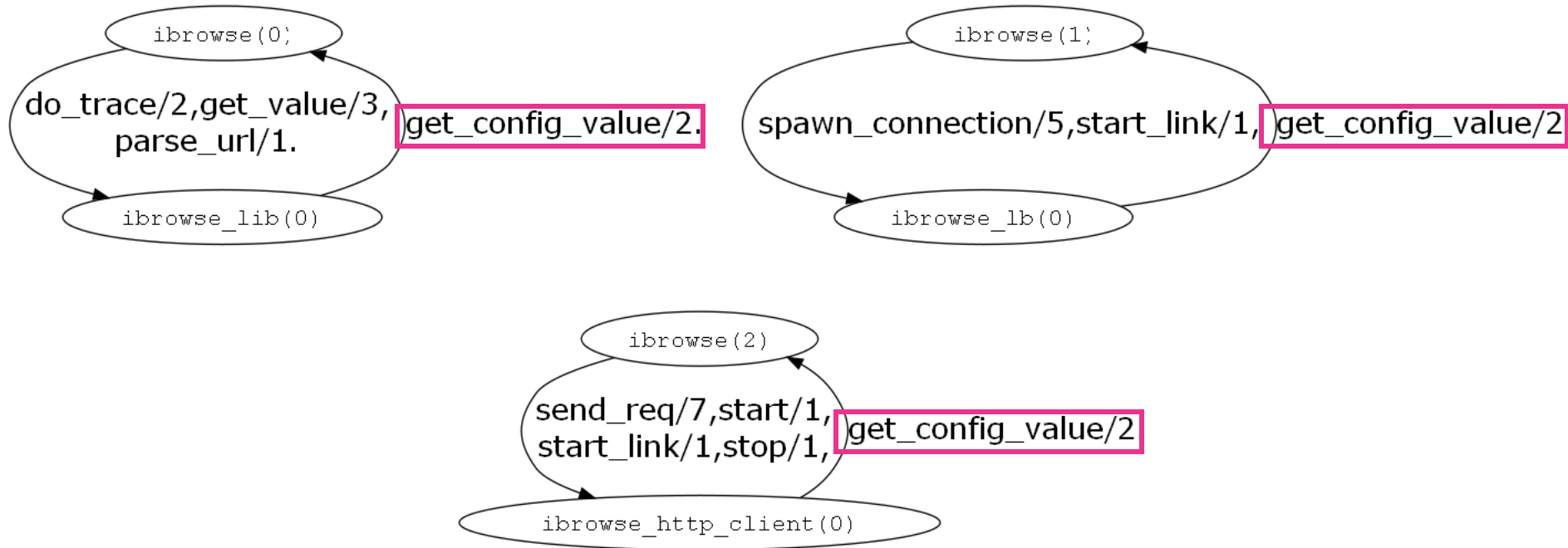


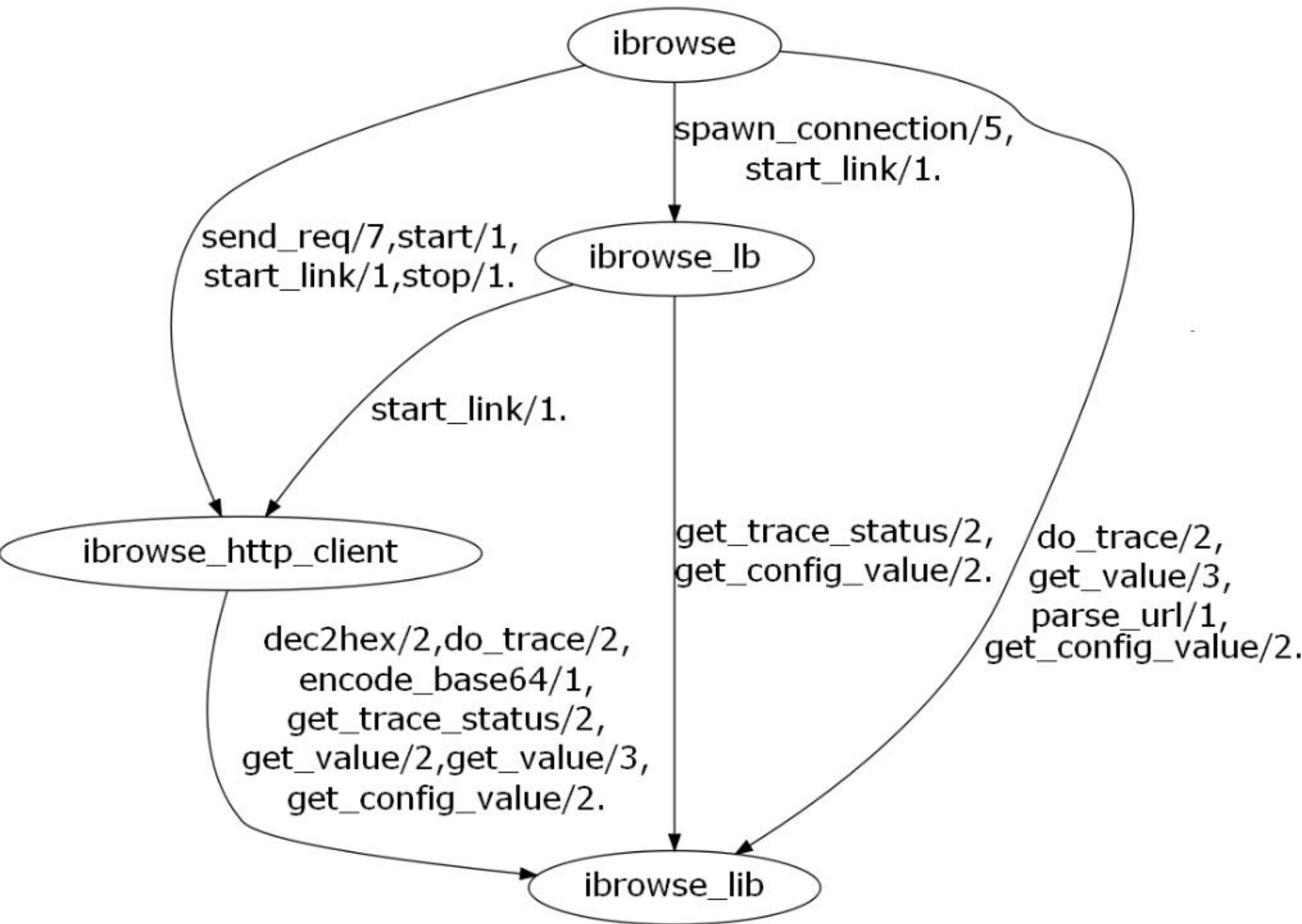
# Cyclic Module Dependency

- Reasons for cyclic module dependency:
  - Mutual recursive function definition across multiple modules.
  - API Functions from different logical layers of the system coexist in the same module.
- Some cyclic module dependencies might be legitimate.

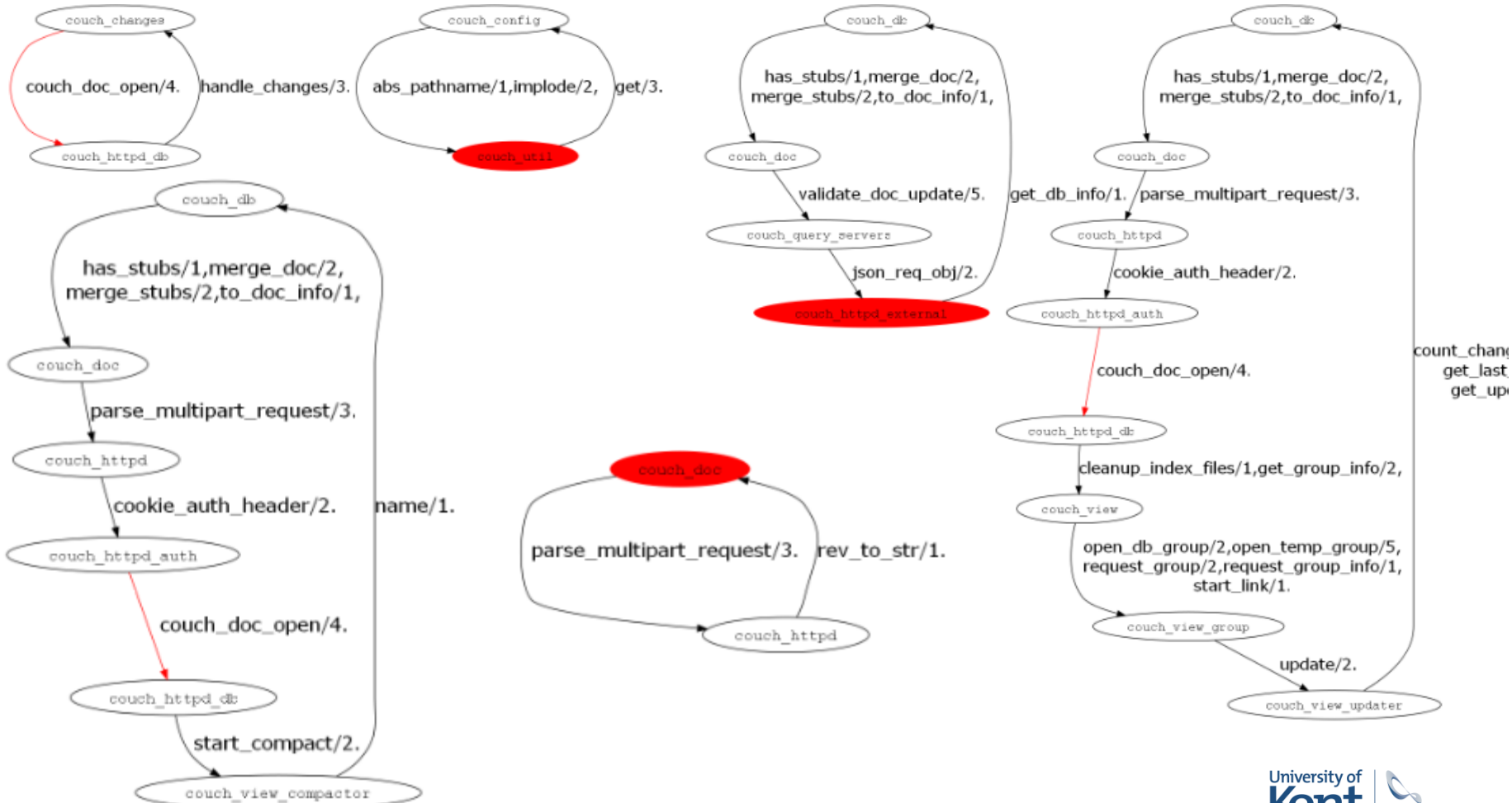


# Ibrowse cycles





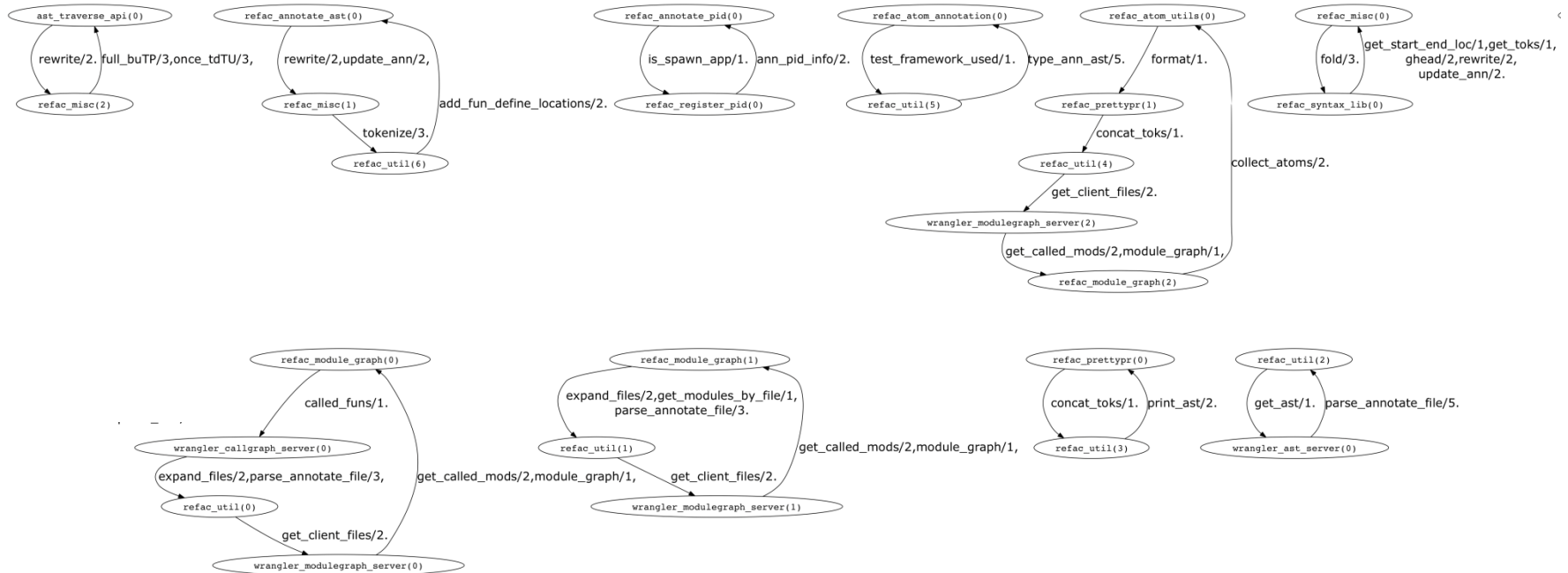
# Some CouchDB cycles



# Some terminology

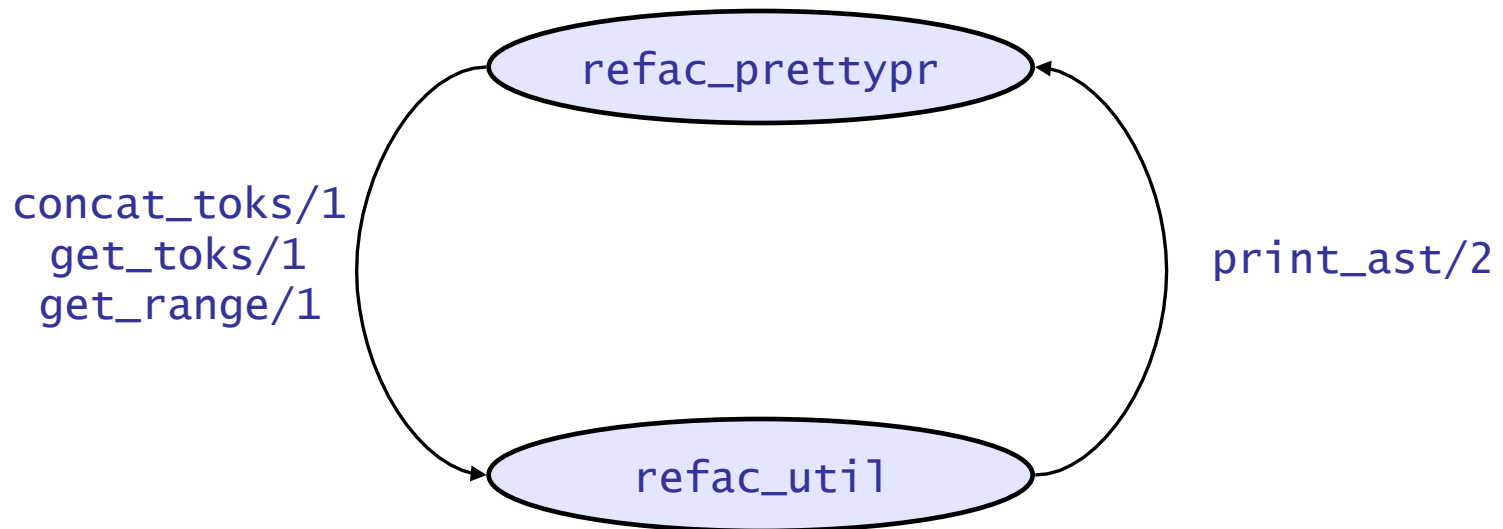
- **Intra-layer dependency:** mutually recursive functions across multiple modules.
- **Inter-layer dependency:** mutually recursive modules, but not mutually recursive functions.

# Wrangler cycles





# Inter-layer dependency

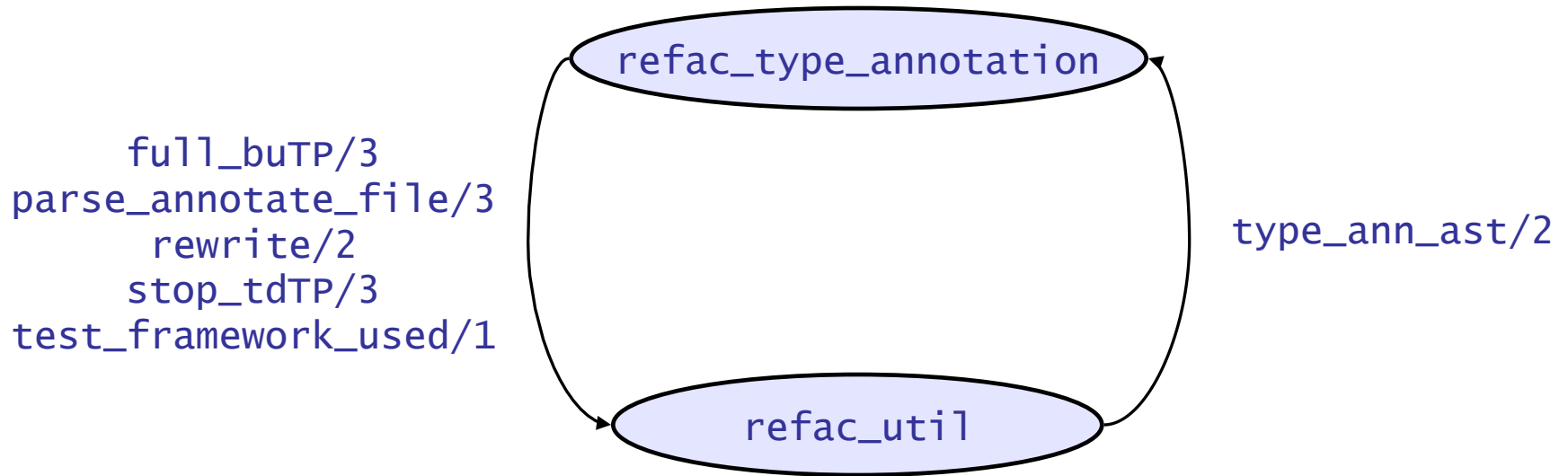


Inter-layer cyclic module dependency found:  
[refac\_prettypr, refac\_util, refac\_prettypr]

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).
```

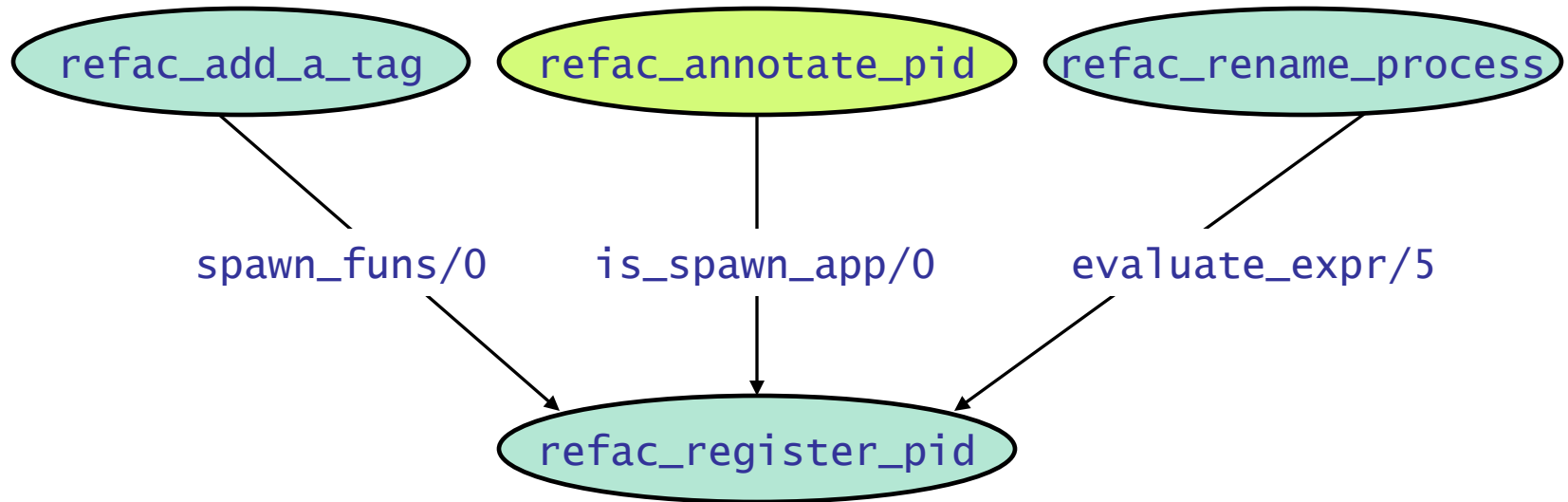
# Intra-layer dependency



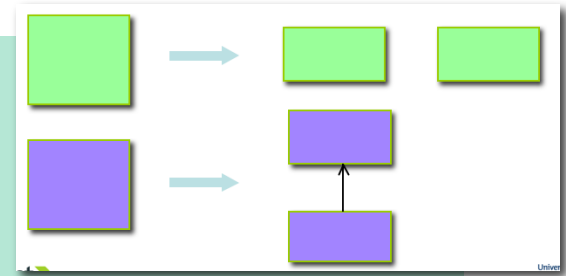
# Identifying "API" functions

- 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}]
```

...

# Demo

# Future work

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

Partition module exports according to client modules.

Case studies.

# Conclusions

Identify and solve existing modularity flaws in an incremental way.

Code smell detection and refactoring suggestions help to improve the usability of refactoring tools.



[www.cs.kent.ac.uk/projects/wrangler/](http://www.cs.kent.ac.uk/projects/wrangler/)