Refactoring Support for Modularity Maintenance in Erlang

Simon Thompson, Huiqing Li

School of Computing, University of Kent, UK

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.

ProTest property based testing

University of Kent | Computing

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

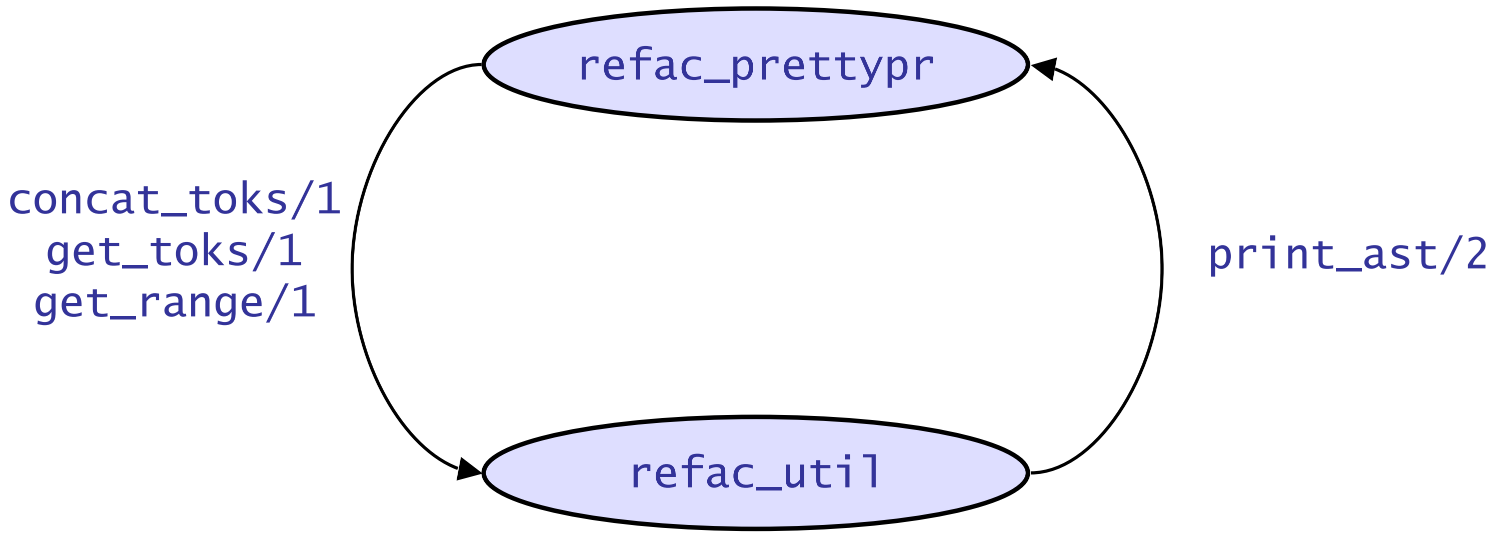
“Dogfooding” Wrangler

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.

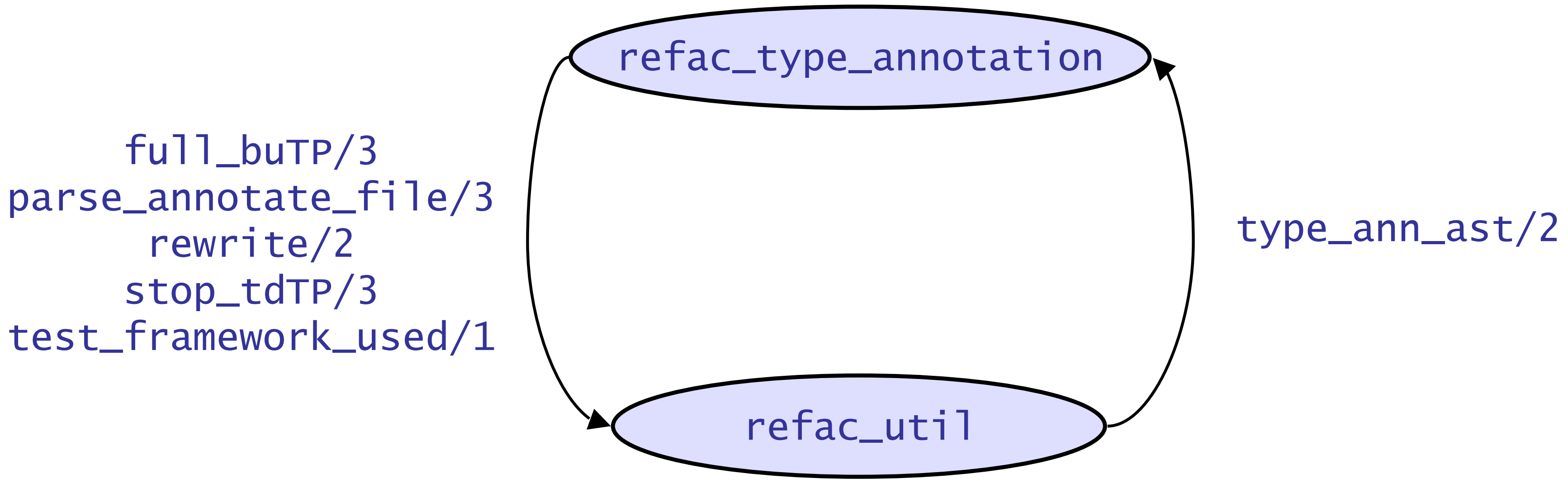
Inter-layer dependency


$$\begin{tikzcd}
\text{refac\_prettypr} \arrow[d, bend right, "\substack{\text{concat\_toks/1}\\ \text{get\_toks/1}\\ \text{get\_range/1}}"'] \\
\text{refac\_util} \arrow[u, bend right, "\text{print\_ast/2}"']
\end{tikzcd}$$

```
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).
```

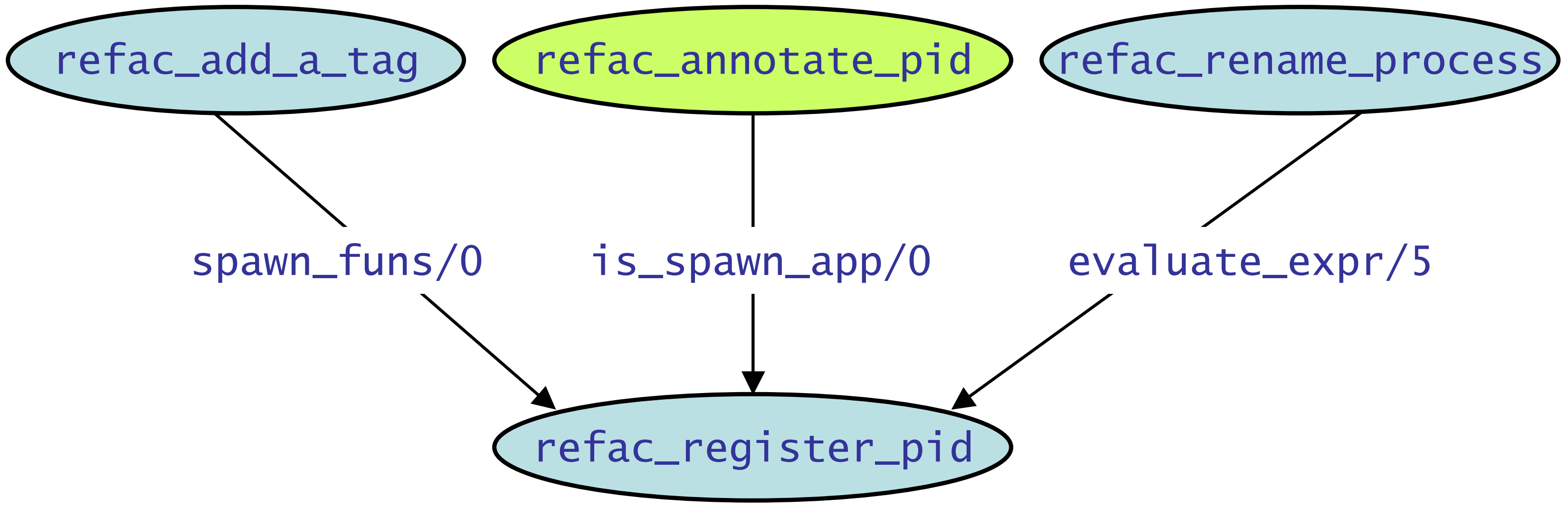
Intra-layer dependency



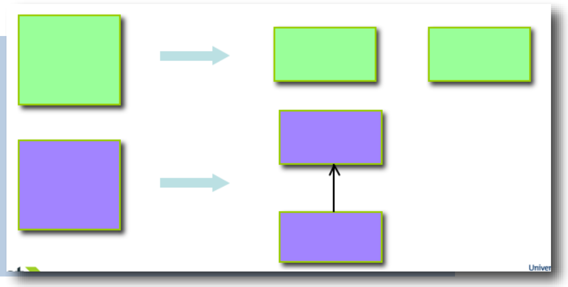
Identifying "API" functions

- 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.

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.

Questions?

Statement

100% automation of source code analysis **and manipulation** is unlikely ever to deliver anything useful.