Improved Testing through Refactoring

Experience from the ProTest Project

Simon Thompson, Huiqing Li

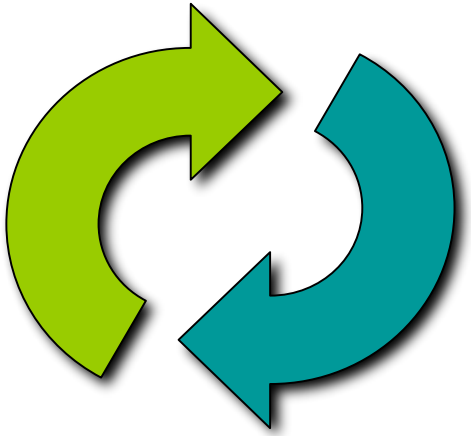
School of Computing, University of Kent

ProTest
property based testing

University of Kent | Computing

Background

Modify



Refactor

Wrangler

Interactive refactoring tool for Erlang

Integrated into Emacs and Eclipse / ErlIDE

Multiple modules

Structural, process, macro refactorings

Clone detection + removal

Improve module structure

Basic refactorings

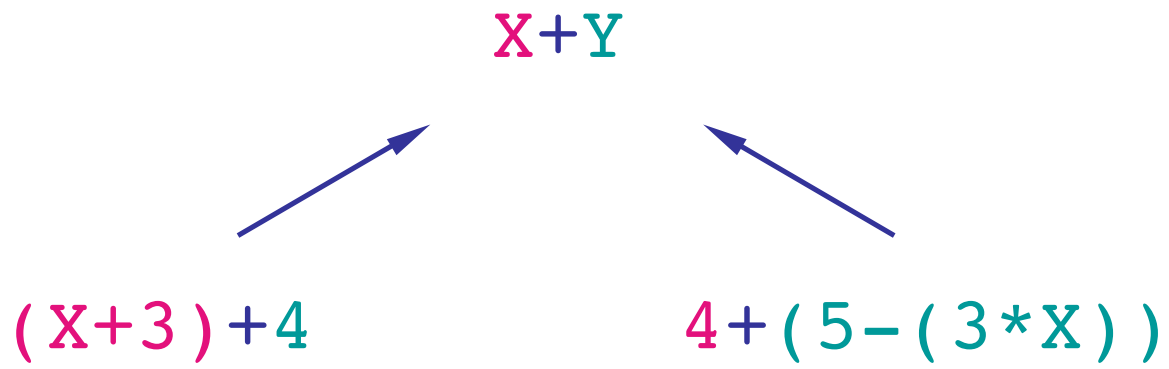
Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

What is ‘similar’ code?



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

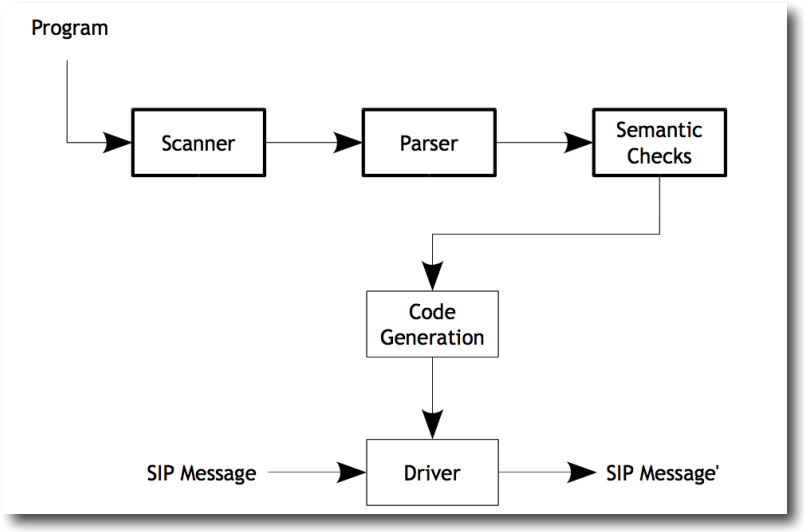
SIP case study

ERICSSON
TAKING YOU FORWARD

SIP message manipulation allows rewriting rules to transform messages.

Test by `smm_SUITE.erl`, 2658 LOC.

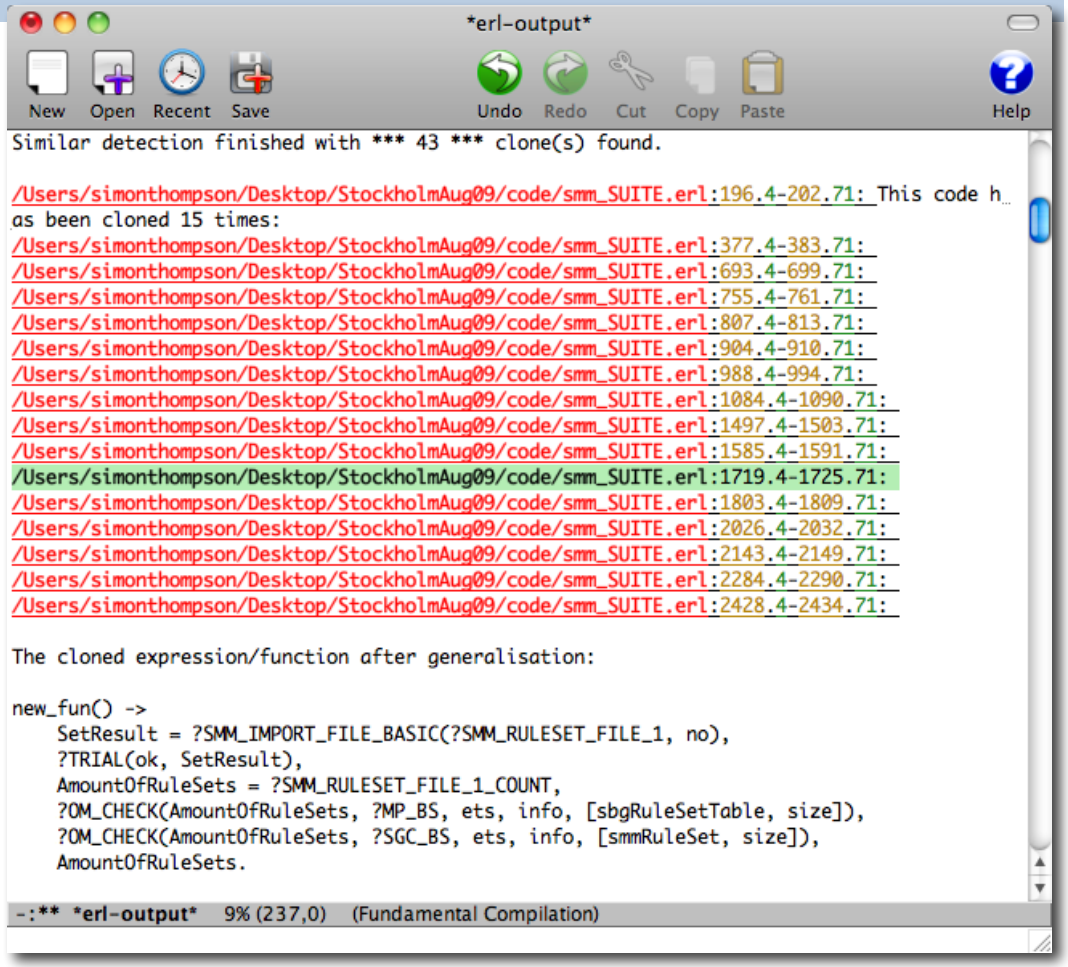
2658 to 2042 in twelve steps.



Step 1

The largest clone class has 15 members.

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



ProTest
property based testing

University of Kent | Computing

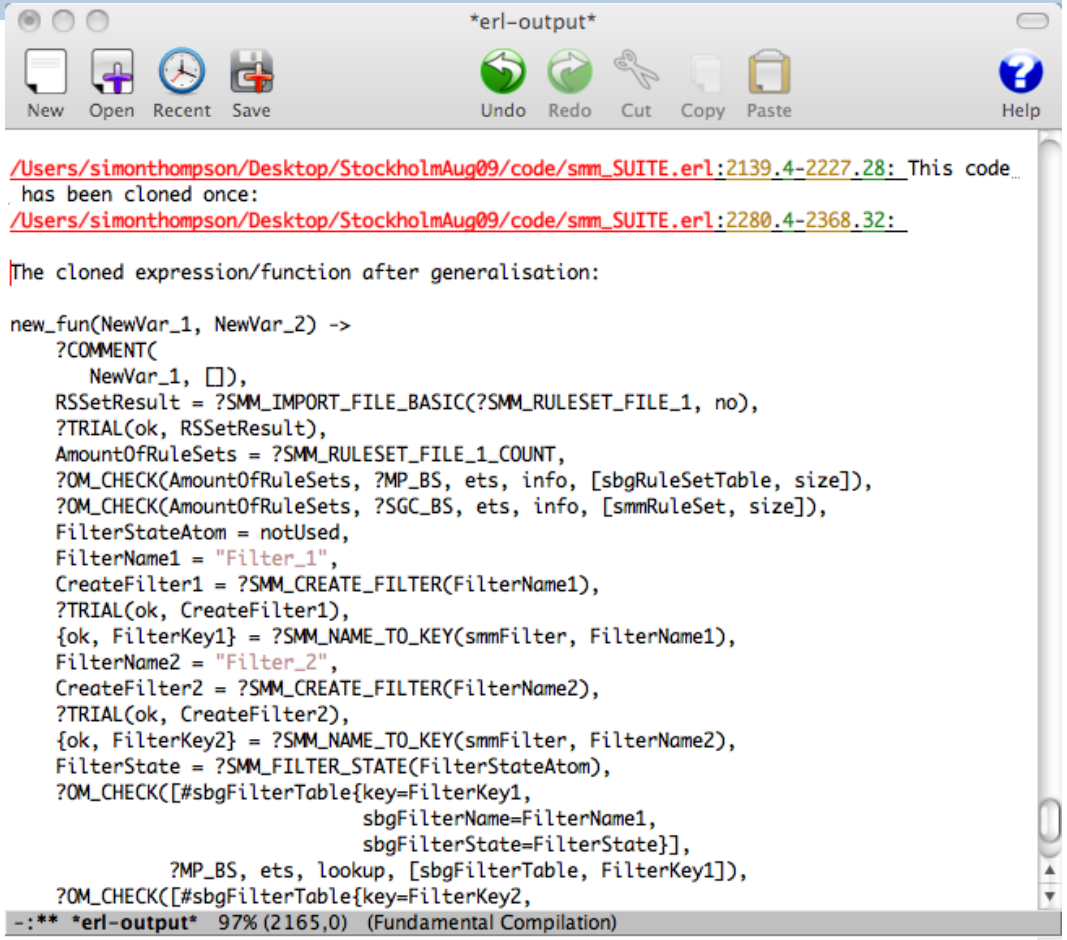
Not step 1

The largest clone has 88 lines, and 2 parameters.

But what does it represent?

What to call it?

Best to work bottom up.



ProTest
property based testing

University of Kent | Computing

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'

Step 3

23 line clone occurs;
choose to replace a smaller clone.

Rename function and parameters, and reorder them.

```
new_fun() ->
  {FilterKey1, FilterName1, FilterState, FilterKey2,
   FilterName2} = create_filter_12(),
  ?OM_CHECK([#smmFilter{key=FilterKey1,
                filterName=FilterName1,
                filterState=FilterState,
                module=undefined}],
      ?SGC_BS, ets, lookup, [smmFilter, FilterKey1]),
  ?OM_CHECK([#smmFilter{key=FilterKey2,
                filterName=FilterName2,
                filterState=FilterState,
                module=undefined}],
      ?SGC_BS, ets, lookup, [smmFilter, FilterKey2]),
  ?OM_CHECK([#sbgFilterTable{key=FilterKey1,
                sbgFilterName=FilterName1,
                sbgFilterState=FilterState}],
      ?MP_BS, ets, lookup, [sbgFilterTable, FilterKey1]),
  ?OM_CHECK([#sbgFilterTable{key=FilterKey2,
                sbgFilterName=FilterName2,
```

```
check_filter_exists_in_sbgFilterTable(FilterKey, FilterName, FilterState) ->
  ?OM_CHECK([#sbgFilterTable{key=FilterKey,
                sbgFilterName=FilterName,
                sbgFilterState=FilterState}],
     ?MP_BS, ets, lookup, [sbgFilterTable, FilterKey]).
```

ProTest
property based testing

University of Kent | Computing

Steps 4, 5

2 variants of `check_filter_exists_in_sbgFilterTable` …

- Check for the filter occurring uniquely in the table: call to `ets:tab2list` instead of `ets:lookup`.
- Check a different table, replace `sbgFilterTable` by `smmFilter`.
- Don't generalise: too many parameters, how to name?

```
check_filter_exists_in_sbgFilterTable(FilterKey, FilterName, FilterState) ->
  ?OM_CHECK([#sbgFilterTable{key=FilterKey,
                sbgFilterName=FilterName,
                sbgFilterState=FilterState}],
     ?MP_BS, ets, lookup, [sbgFilterTable, FilterKey]).
```

Step 10

‘Widows’ and ‘orphans’ in clone identification.

Avoid passing commands as parameters?

Also at step 11.

```
new_fun(FilterName, NewVar_1) ->
  FilterKey = ?SMM_CREATE_FILTER_CHECK(FilterName),
  %%Add rulests to filter
  RuleSetNameA = "a",
  RuleSetNameB = "b",
  RuleSetNameC = "c",
  RuleSetNameD = "d",
  ... 16 lines which handle the rules sets are elided ...
  %%Remove rulesets
  NewVar_1,
{RuleSetNameA, RuleSetNameB, RuleSetNameC, RuleSetNameD, FilterKey}.
```

```
new_fun(FilterName, FilterKey) ->
  %%Add rulests to filter
  RuleSetNameA = "a",
  RuleSetNameB = "b",
  RuleSetNameC = "c",
  RuleSetNameD = "d",
  ... 16 lines which handle the rules sets are elided ...
  %%Remove rulesets

{RuleSetNameA, RuleSetNameB, RuleSetNameC, RuleSetNameD}.
```

Clone elimination and testing

Copy and paste … many hands.

Shorter, more comprehensible and better structured code.

Emphatically not “push button” …

Need domain expert involvement.

Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

Property discovery in Wrangler

Find (test) code that is similar …

… build a common abstraction

… accumulate the instances

… and generalise the instances.

Example:

Test code from Ericsson: different media and codecs.

Generalisation to all medium/codec combinations.

Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

Testing frameworks

EUnit, Common Test and Quick Check each give a template for writing tests and a platform for performing them.

Want to refactor code and test code in step.

Extend refactorings while observing

- Naming conventions
- Macros
- Callbacks
- Meta-programming
- Coding patterns

Quick Check example

Callbacks, macros and meta-programming.

```
-export( …, command/1, postcondition/3, … ,prop/0]).

command({N}) when N<10 ->
     frequency([{3,{call,nat_gen,next,[]}},
                {1,{call,nat_gen,stop,[]}}]); …

postcondition({N},{call,nat_gen,next,_},R)-> R == N; …

prop() ->
  ?FORALL(Commands,commands(?MODULE),
    begin {_H,_S,Result} = run_commands(?MODULE,Commands),
          Result == ok end).
```

Quick Check example

Callbacks, macros and meta-programming.

```
-export( …, command/1, postcondition/3, … ,prop/0]).

command({N}) when N<10 ->
     frequency([{3,{call,nat_gen,next,[]}},
                {1,{call,nat_gen,stop,[]}}]); …

postcondition({N},{call,nat_gen,next,_},R)-> R == N; …

prop() ->
  ?FORALL(Commands,commands(?MODULE),
    begin {_H,_S,Result} = run_commands(?MODULE,Commands),
          Result == ok end).
```

Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

Refactoring within QuickCheck

FSM-based testing: transform state variable from simple value to record.

Stylised usage supports robust transformation.

Spinoff to OTP libs.

Property refactorings:

Introduce local definitions (LET)

Merge local definitions and quantifiers (FORALL).

[EUnit too …]

Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

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