
SPECIAL EDITION
CLINT EASTWOOD
THE GOOD
THE BAD
AND THE UGLY
2-DISC DVD COLLECTOR'S SET
FULLY RESTORED. ADDITIONAL FOOTAGE. HOURS OF EXTRAS. THE MASTERPIECE RETURNS WITH A VENGEANCE. OWN IT MAY 18TH.

DVD

Using Wrangler to refactor Erlang programs and tests

Simon Thompson, Huiqing Li
Adam Lindberg, Andreas Schumacher

University of Kent, Erlang Solutions, Ericsson

ProTest
property based testing

University of Kent | Computing

Overview

Refactoring Erlang in Wrangler

Clone detection and elimination

Implementation

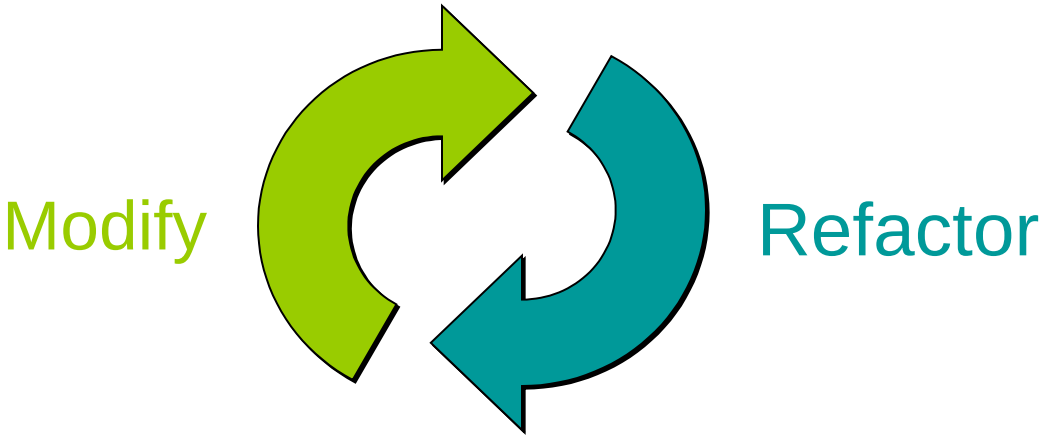
Case study: SIP message manipulation

ProTest project: property-based testing

Introduction

Refactoring

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

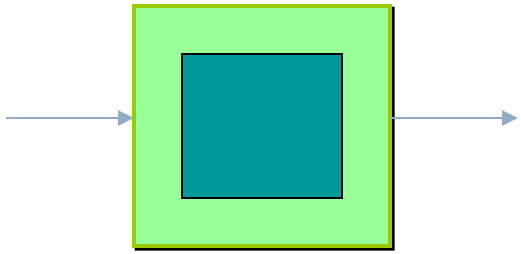


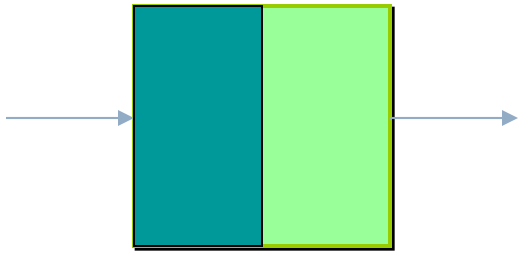
Soft-ware

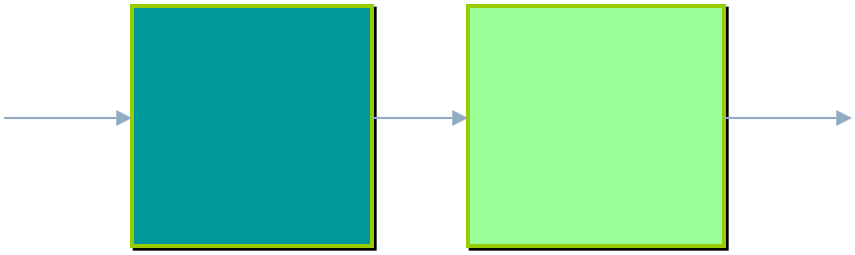
There's no single correct design …

… different options for different situations.

Maintain flexibility as the system evolves.

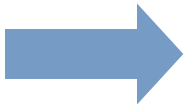






Generalisation and renaming

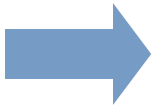
```
-module (test).
-export([f/1]).

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.


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



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

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


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

Refactoring tool support

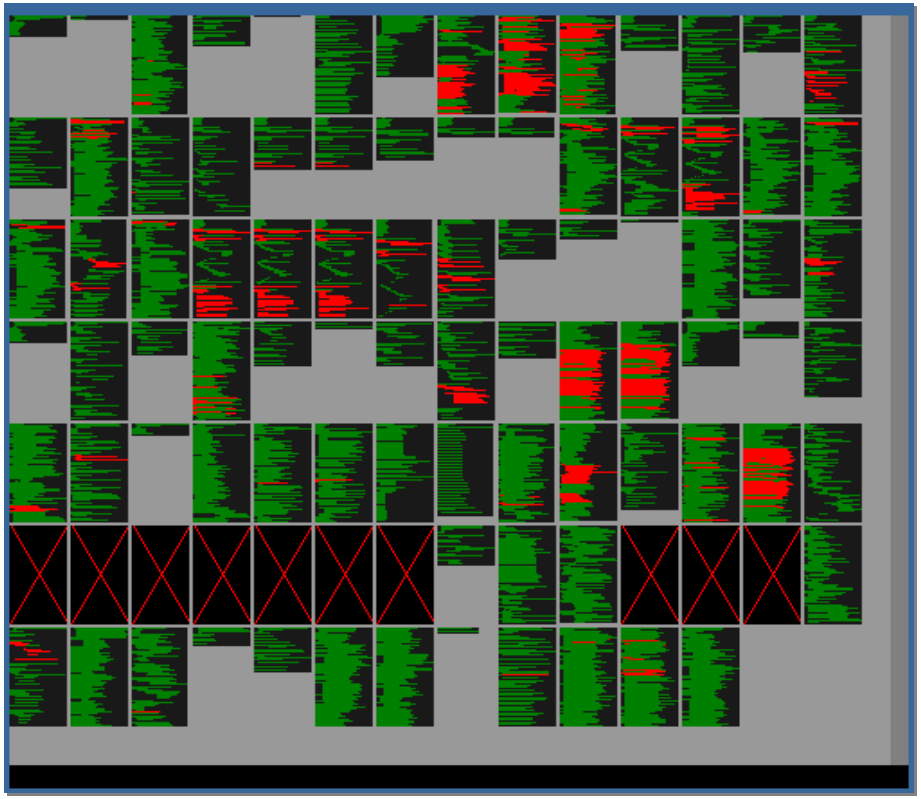
Bureaucratic and diffuse.

Tedious and error prone.

Semantics: scopes, types, modules, …

Undo/redo

Enhanced creativity



Refactoring = Transformation + Condition

Transformation

Ensure change at all those points needed.

Ensure change at only those points needed.

Condition

Is the refactoring applicable?

Will it preserve the semantics of the module? the program?

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?

Wrangler

Refactoring tool for Erlang

Integrated into Emacs and Eclipse / ErlIDE.

Multiple modules

Structural, process, macro refactorings

Duplicate code detection …

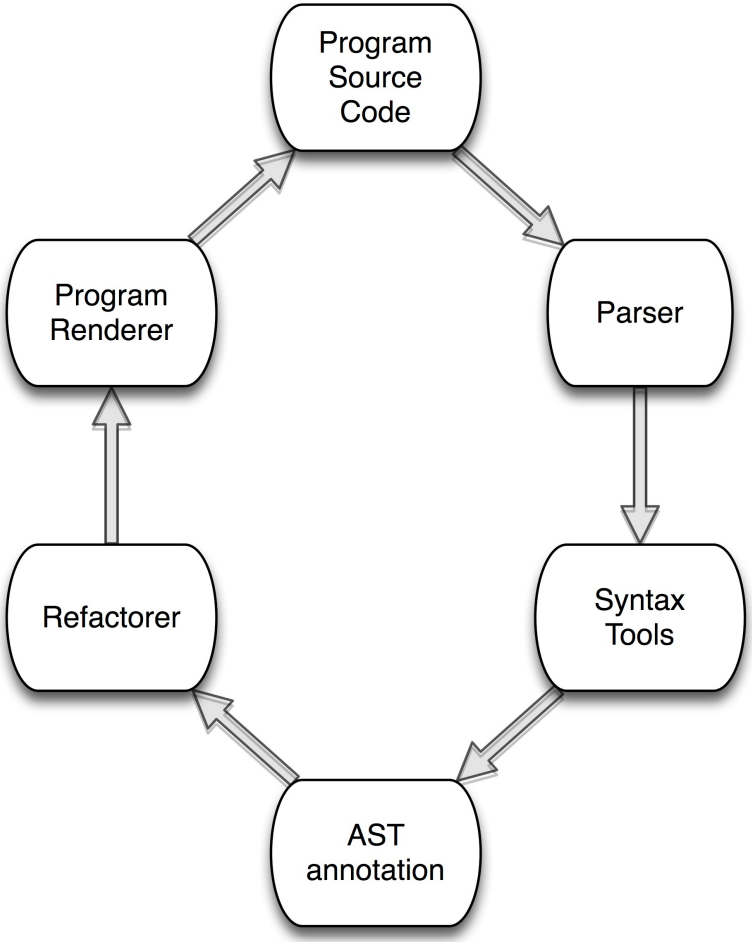
… and elimination

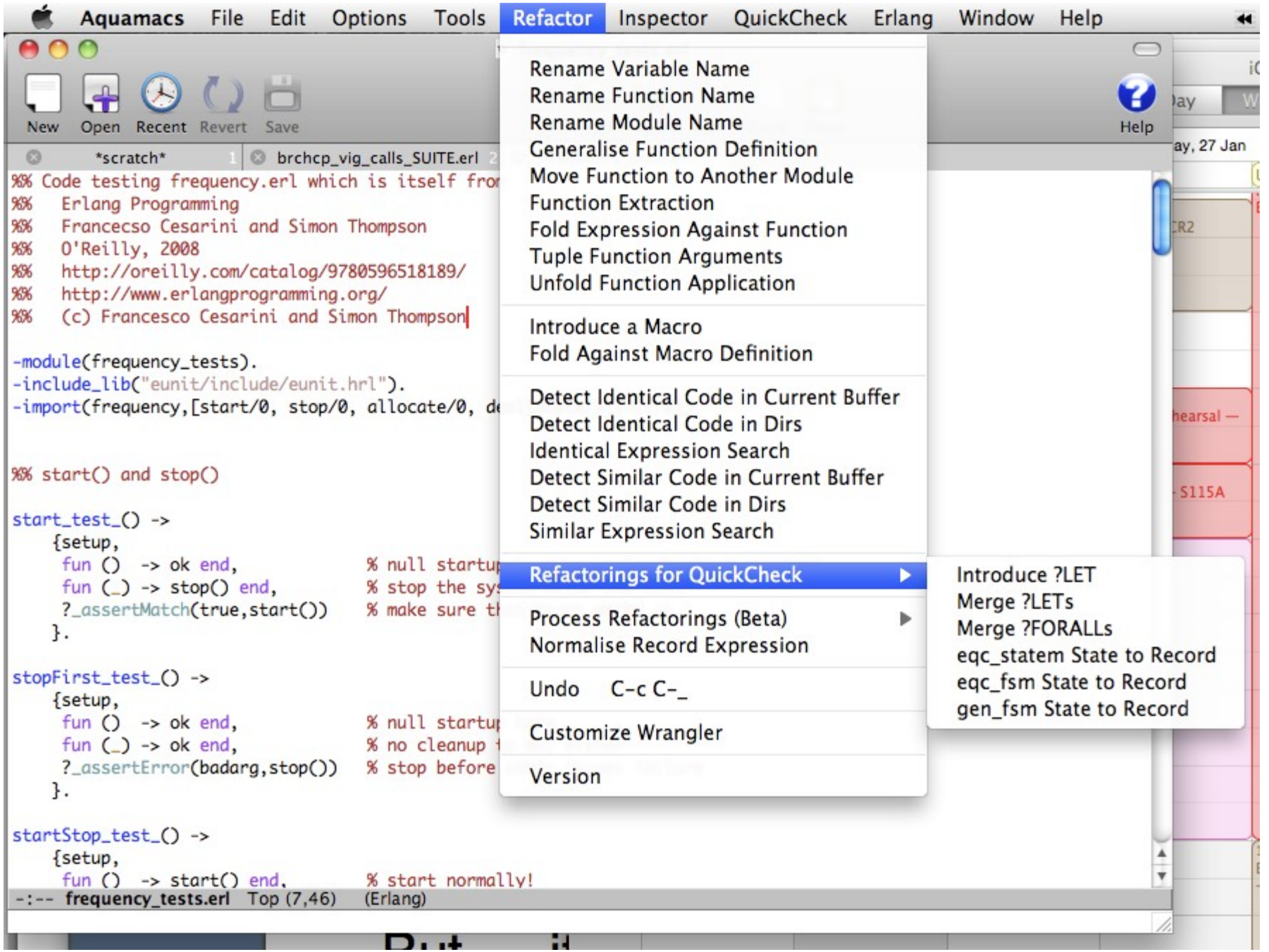
Testing / refactoring

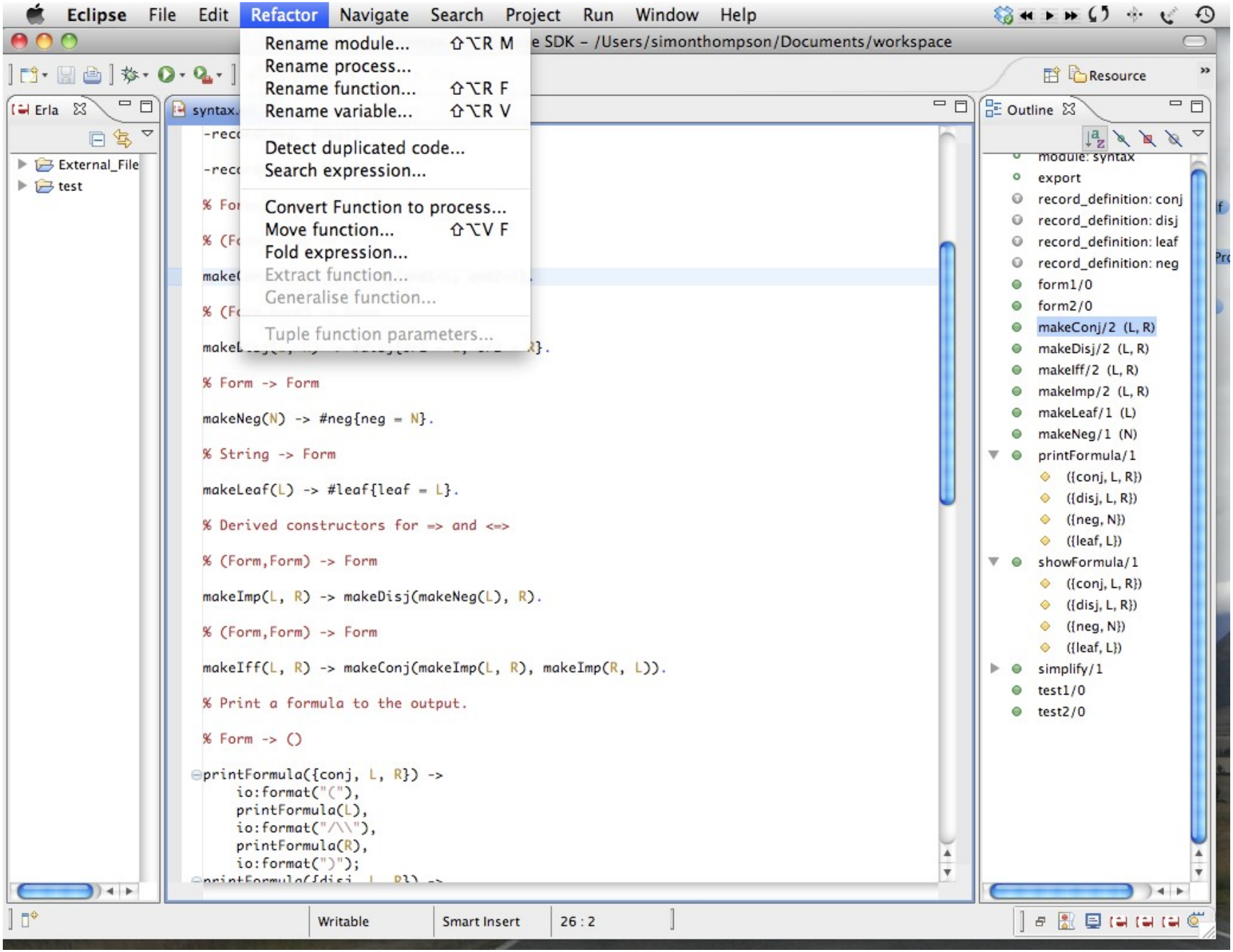
"Similar" code identification

Property discovery

Architecture of Wrangler




Aquamacs File Edit Options Tools Refactor Inspector QuickCheck Erlang Window Help
New Open Recent Revert Save
Help
scratch
brchcp_vig_calls_SUITE.erl
%% Code testing frequency.erl which is itself fro
%% Erlang Programming
%% Francecso Cesarini and Simon Thompson
%% O'Reilly, 2008
%% http://oreilly.com/catalog/9780596518189/
%% http://www.erlangprogramming.org/
%% (c) Francesco Cesarini and Simon Thompson
-module(frequency_tests).
-include_lib("eunit/include/eunit.hrl").
-import(frequency,[start/0, stop/0, allocate/0, de
%% start() and stop()
start_test_() ->
{setup,
fun () -> ok end, % null startup
fun (_) -> stop() end, % stop the sy
?_assertMatch(true,start()) % make sure th
}.
stopFirst_test_() ->
{setup,
fun () -> ok end, % null startup
fun (_) -> ok end, % no cleanup
?_assertError(badarg,stop()) % stop before
}.
startStop_test_() ->
{setup,
fun () -> start() end, % start normally!
-:-- frequency_tests.erl Top (7,46) (Erlang)
Rename Variable Name
Rename Function Name
Rename Module Name
Generalise Function Definition
Move Function to Another Module
Function Extraction
Fold Expression Against Function
Tuple Function Arguments
Unfold Function Application
Introduce a Macro
Fold Against Macro Definition
Detect Identical Code in Current Buffer
Detect Identical Code in Dirs
Identical Expression Search
Detect Similar Code in Current Buffer
Detect Similar Code in Dirs
Similar Expression Search
Refactorings for QuickCheck
Process Refactorings (Beta)
Normalise Record Expression
Undo C-c C-_
Customize Wrangler
Version
Introduce ?LET
Merge ?LETs
Merge ?FORALLs
eqc_statem State to Record
eqc_fsm State to Record
gen_fsm State to Record


Eclipse File Edit Refactor Navigate Search Project Run Window Help
Rename module... ⇧⌥R M
Rename process...
Rename function... ⇧⌥R F
Rename variable... ⇧⌥R V
Detect duplicated code...
Search expression...
Convert Function to process...
Move function... ⇧⌥V F
Fold expression...
Extract function...
Generalise function...
Tuple function parameters...
e SDK - /Users/simonthompson/Documents/workspace
Resource
Erla
External_File
test
syntax.
% Form -> Form
makeNeg(N) -> #neg{neg = N}.
% String -> Form
makeLeaf(L) -> #leaf{leaf = L}.
% Derived constructors for => and <=>
% (Form,Form) -> Form
makeImp(L, R) -> makeDisj(makeNeg(L), R).
% (Form,Form) -> Form
makeIff(L, R) -> makeConj(makeImp(L, R), makeImp(R, L)).
% Print a formula to the output.
% Form -> ()
printFormula({conj, L, R}) ->
io:format("("),
printFormula(L),
io:format("/\\"),
printFormula(R),
io:format(")");
Outline
module: syntax
export
record_definition: conj
record_definition: disj
record_definition: leaf
record_definition: neg
form1/0
form2/0
makeConj/2 (L, R)
makeDisj/2 (L, R)
makeIff/2 (L, R)
makeImp/2 (L, R)
makeLeaf/1 (L)
makeNeg/1 (N)
printFormula/1
{conj, L, R}
{disj, L, R}
{neg, N}
{leaf, L}
showFormula/1
{conj, L, R}
{disj, L, R}
{neg, N}
{leaf, L}
simplify/1
test1/0
test2/0
Writable
Smart Insert
26 : 2

Clone detection

Duplicate code considered harmful

It's a *bad smell* …

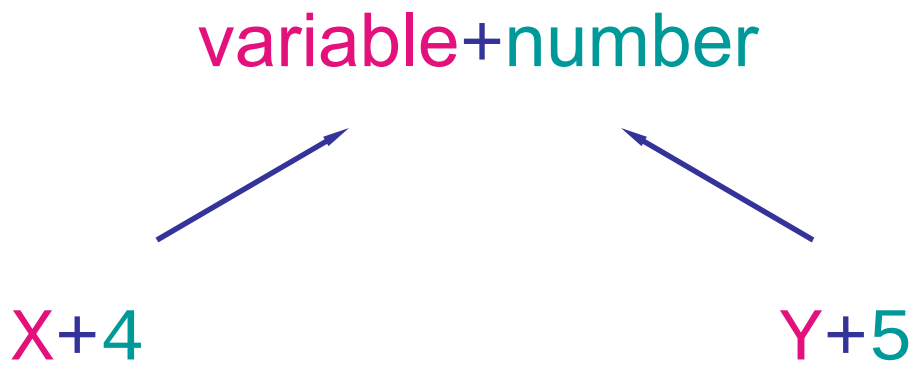
- increases chance of bug propagation,
- increases size of the code,
- increases compile time, and,
- increases the cost of maintenance.

But … it's not always a problem.

Clone detection

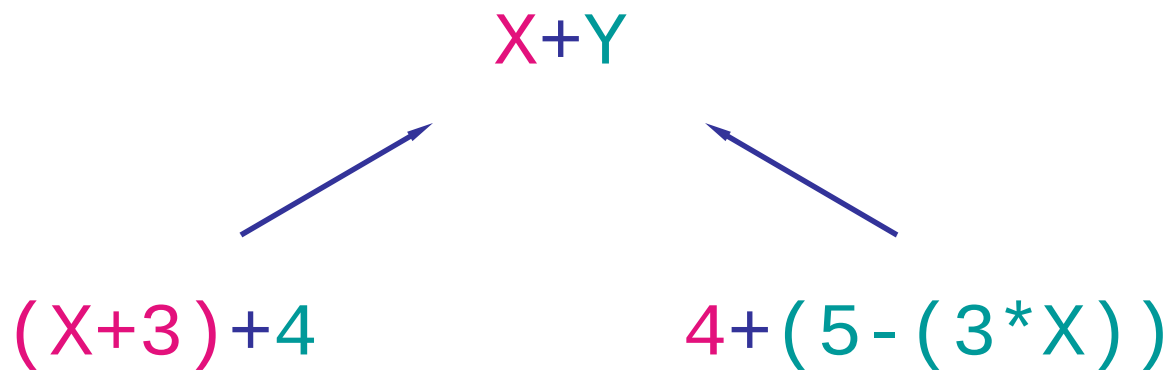
- The Wrangler clone detector
  - relatively efficient
  - no false positives
- User-guided interactive removal of clones.
- Integrated into development environments.

What is ‘identical’ code?



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.

Detection

All clones in a project meeting the threshold parameters …

… and their common generalisations.

Default threshold:
≥ 5 expressions and similarity of ≥ 0.8.

Expression search

All instances of expressions similar to this expression …

… and their common generalisation.

Default threshold:
≥ 20 tokens.

Similarity

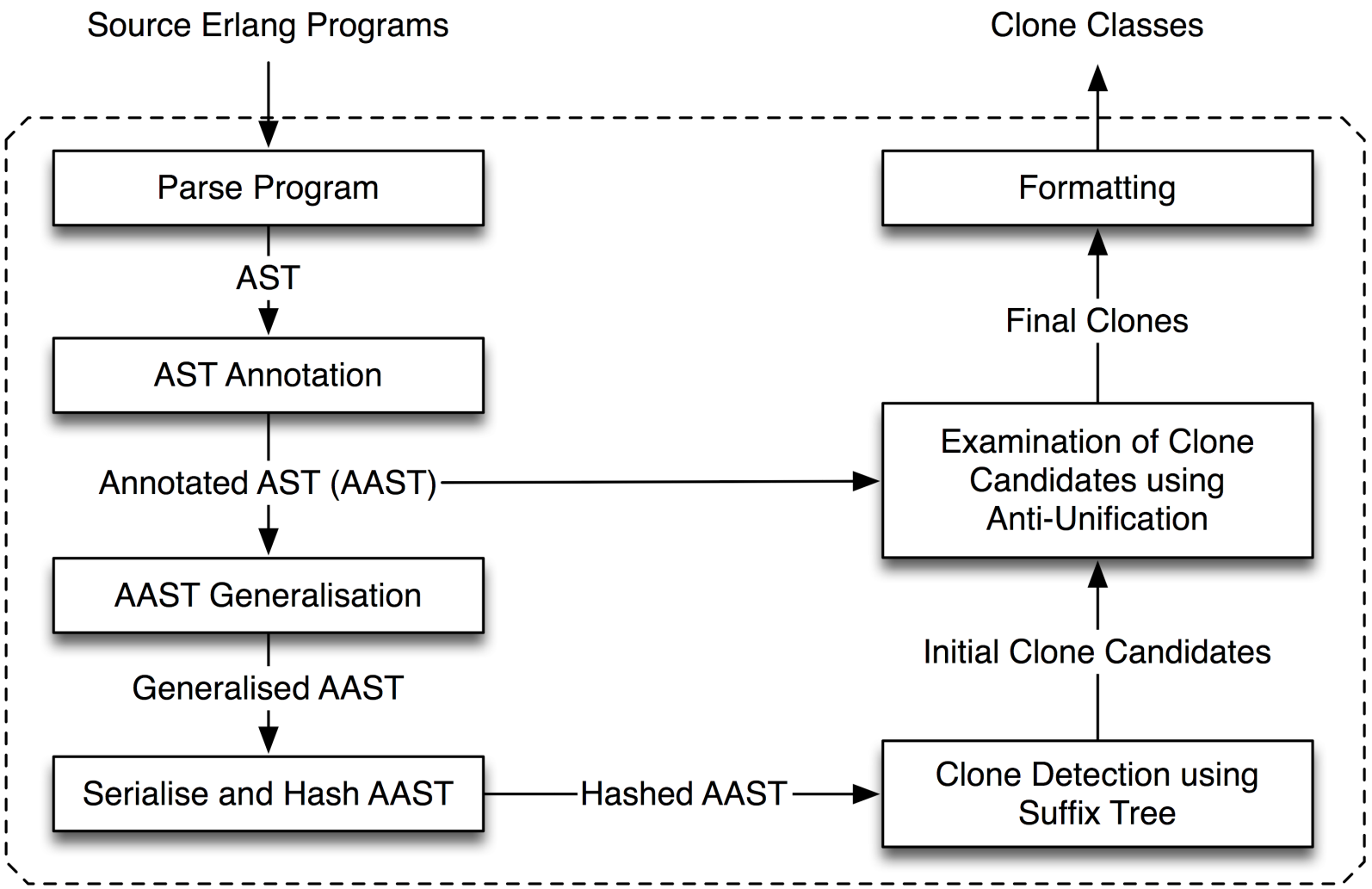
Threshold: anti-unifier should be big enough relative to the class members:

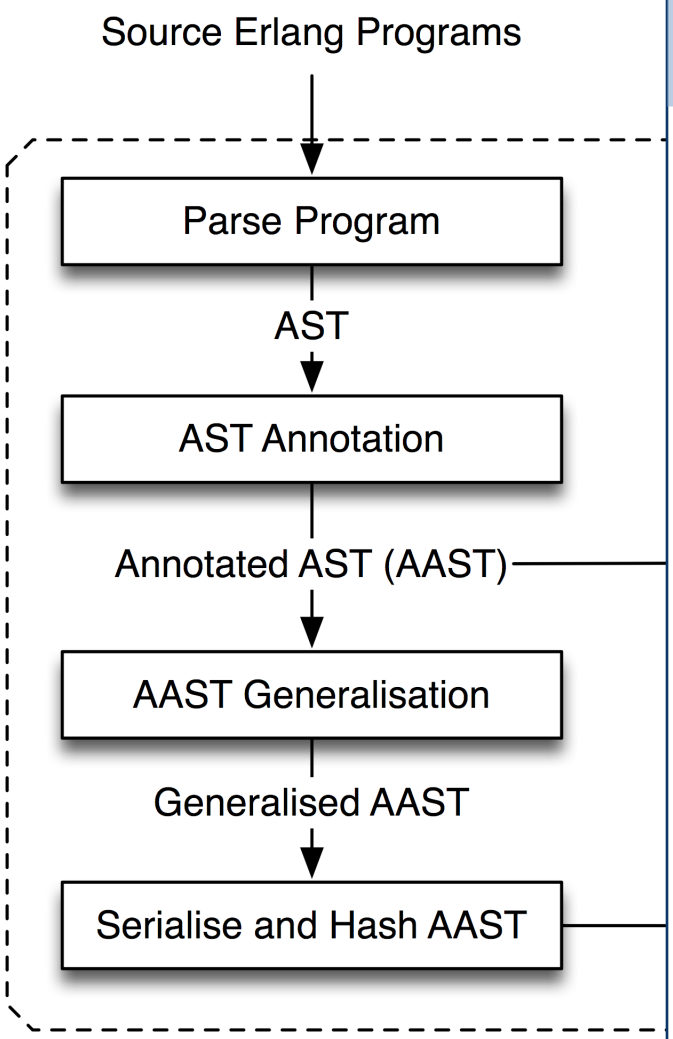
$$similarity = \min_{i=1..n} (size(AU)/size(E_i))$$

where $AU = anti\text{-}unifier(E_1, \ldots, E_n)$.

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

Implementation


Source Erlang Programs
Clone Classes
Parse Program
AST
AST Annotation
Annotated AST (AAST)
AAST Generalisation
Generalised AAST
Serialise and Hash AAST
Hashed AAST
Formatting
Final Clones
Examination of Clone Candidates using Anti-Unification
Initial Clone Candidates
Clone Detection using Suffix Tree

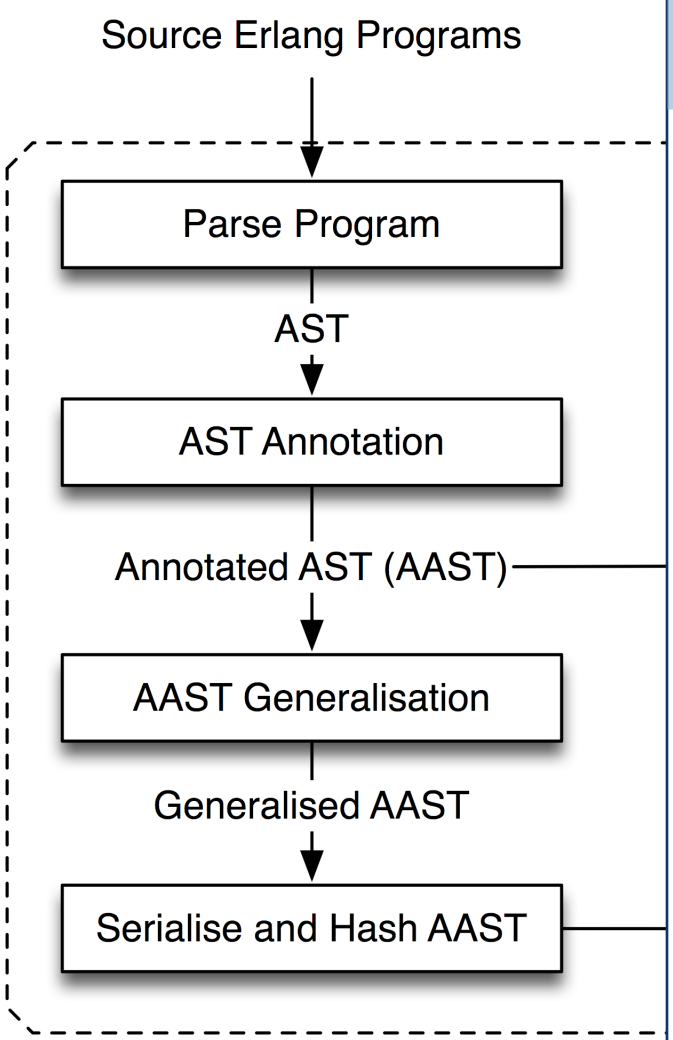


Parse program

Parse the program with modified parser to ensure that location information (line, column) is included.

This ensures that can map between different program representations.

Bypasses the Erlang pre-processor.

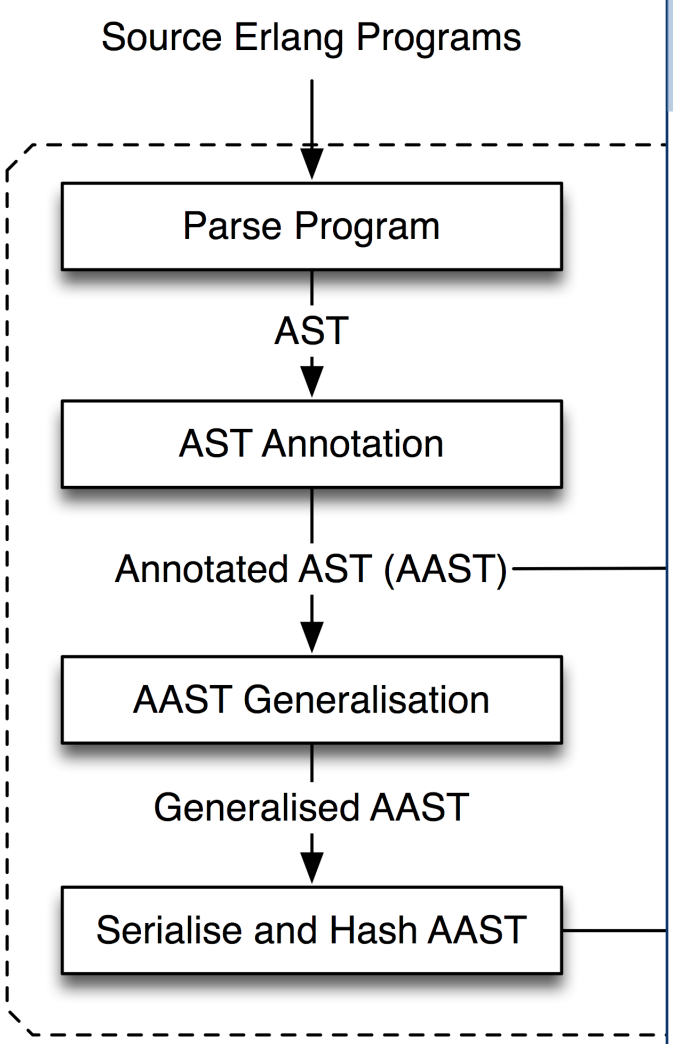


Annotate AST

Resolve the use of identifiers to their binding occurrences.

Use location information to identify occurrences.

Erlang allows a variable to have multiple binding occurrences, e.g. in different arms of a `case` expression.



Generalise AST

Capture structural similarity between expressions while keeping a structural *skeleton* of the original.

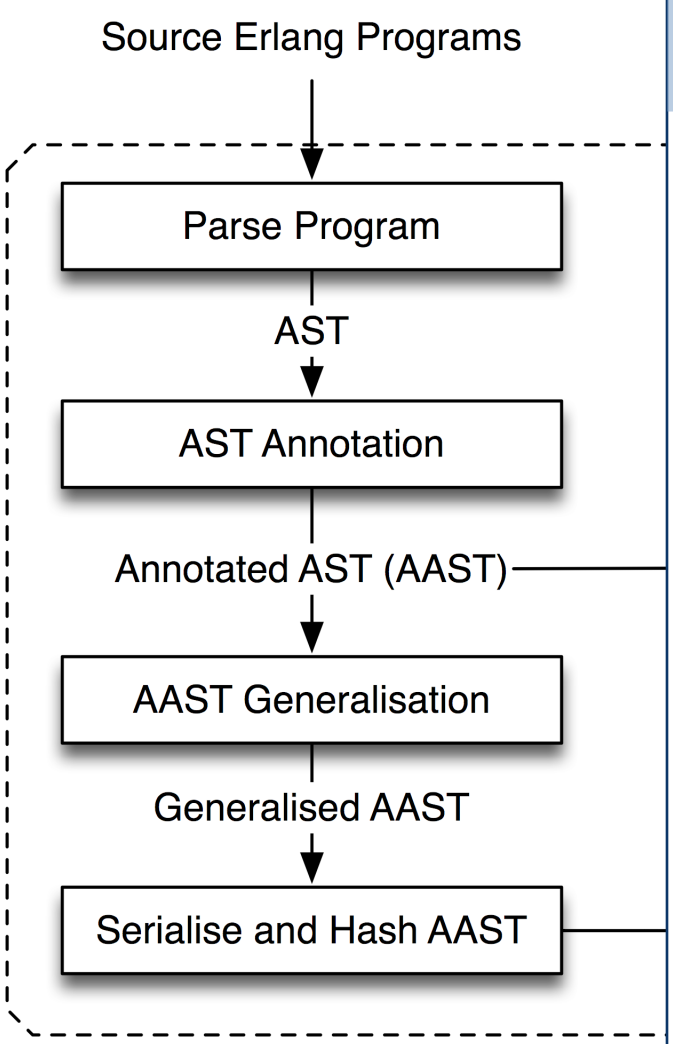
Replace certain subtrees with a placeholder …

… but only if sensible to do this, e.g. expressions including `funs` but *not* conditionals, patterns, `try … catch …`, `receive`, etc.

Example of generalised code

```
foo(X) ->
 Y =
  case X of
   one    -> 12;
   Others -> 196
  end,
 X+Y,
 g(X,Y).
```

```
foo(X) ->
 ? =
  case ? of
   ?      -> ?;
   ?      -> ?
  end,
 ?,
 ?.
```



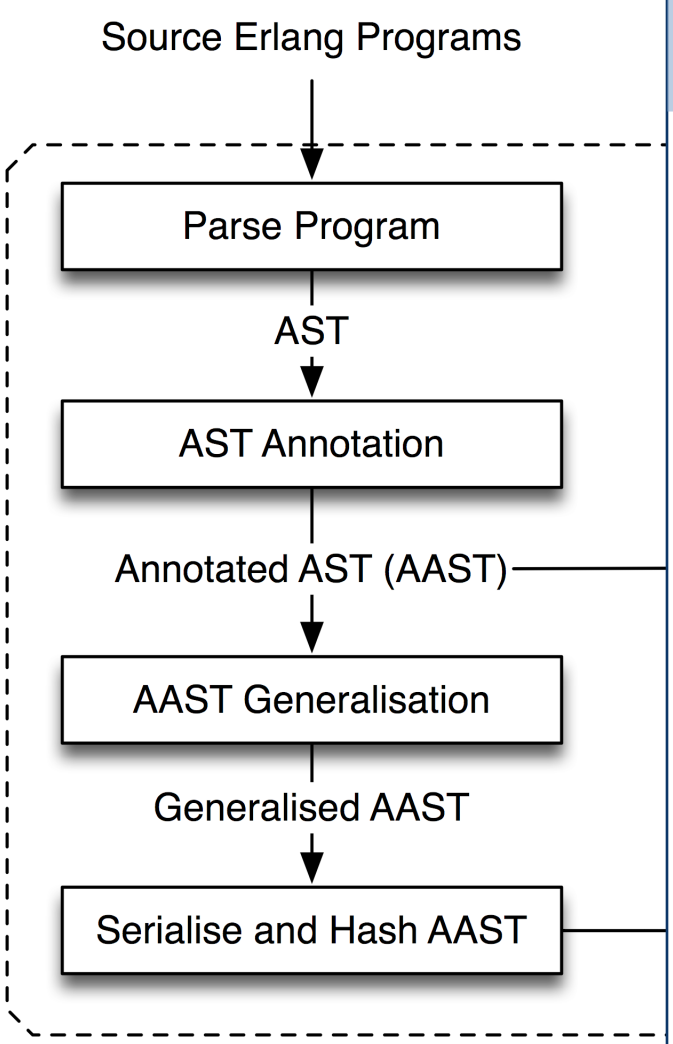
Serialise the AST

Pretty print generalised sub-expression sequences and then serialise into a single sequence.

A delimiter separates each sub-expression sequence.

```
foo(X, Y) ->
   A = case X>Y of
          true -> Z=1,
                  X + Y + Z;
          false ->
                  Z = 2,
                  X + Y -2
       end,
   A + 37.
```

```
A = case …
A + 37
--
Z=1
X + Y + Z
--
Z = 2
X + Y -2
```



Hash expressions

Hash each expression, mapping it to an 128 bit value, using non-clashing hash function.

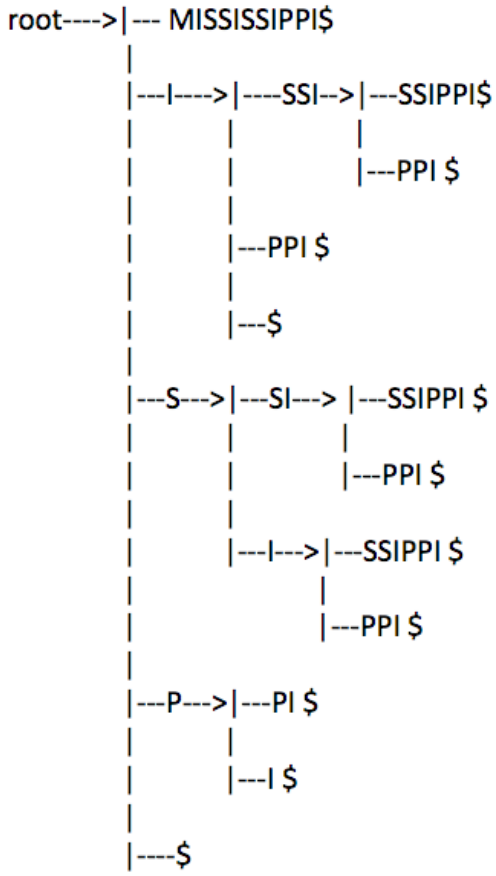
Expressions represented by start / end positions in the source code.

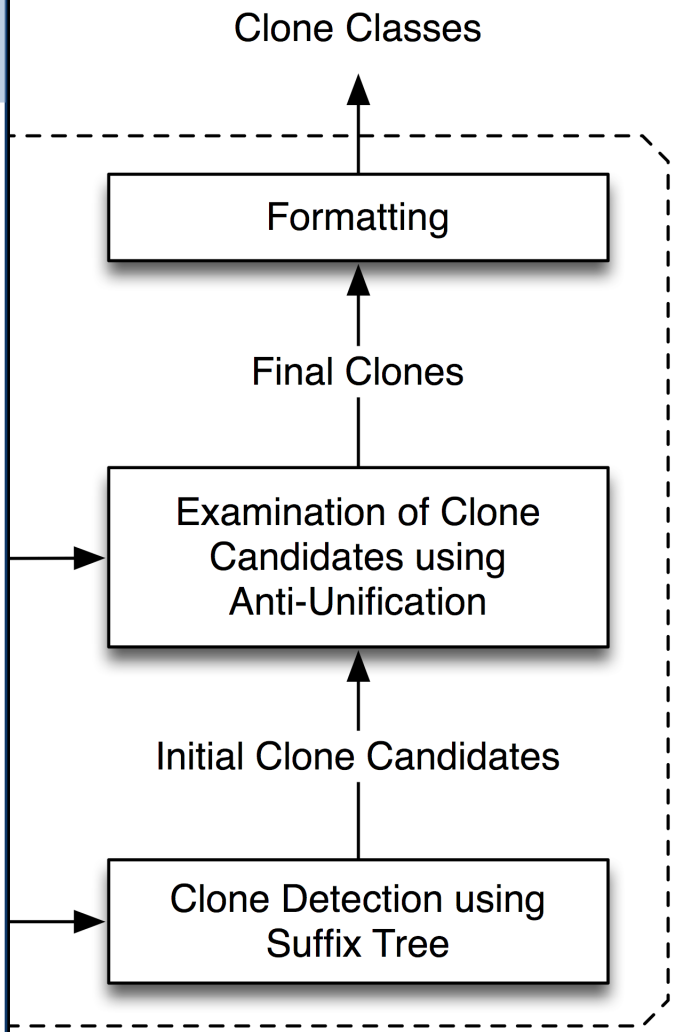
Hash values stored in indexed table - indexes smaller than hash values.

Build suffix tree

Build a suffix tree from the expression sequence.

Clones are given by paths that branch.





ProTest property based testing

University of Kent Computing

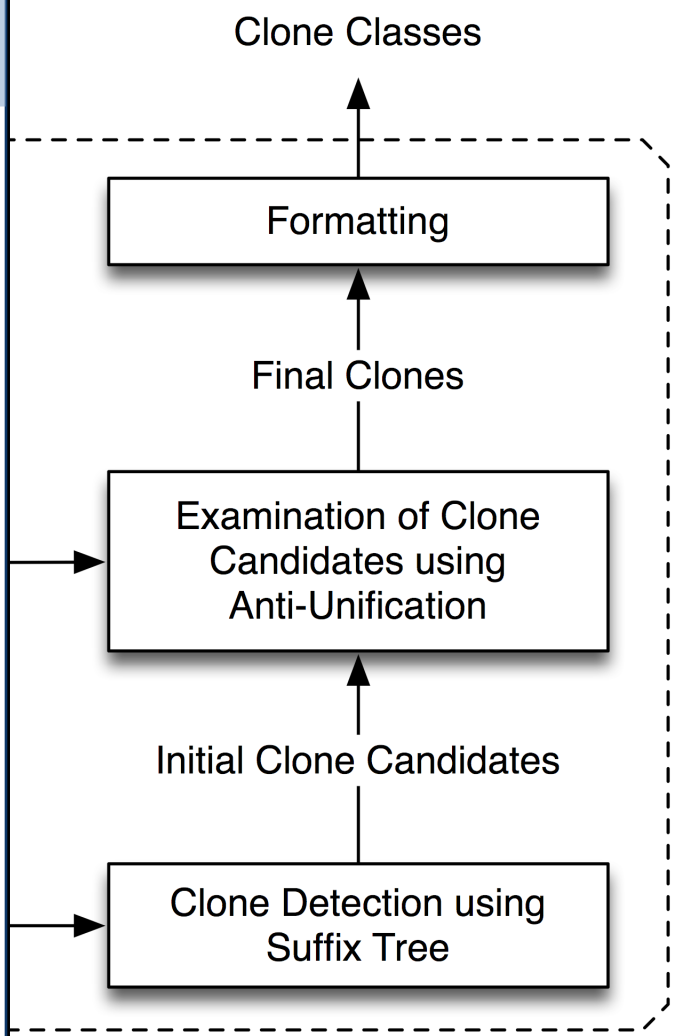
Check clone classes

Check a clone class for anti-unification. Will return

- no classes,
- one class, or
- multiple sub-classes

each with the corresponding anti-unification function.

Results depend on the threshold parameters.



Example: clone candidate

```
S1 = "This",          S1 = "This",          D1 = [1],     D1 = [X+1],
S2 = " is a ",        S2 = "is another ",   D2 = [2],     D2 = [5],
S3 = "string",        S3 = "String",        D3 = [3],     D3 = [6],
[S1,S2,S3]            [S3,S2,S1]            [D1,D2,D3]    [D3,D2,D1]
```

```
? = ?,
? = ?,
? = ?,
[?,?,?]
```

Example: clone from sub-sequence

```
S1 = "This",        S1 = "This",           D1 = [1],     D1 = [X+1],
S2 = " is a ",      S2 = "is another ",    D2 = [2],     D2 = [5],
S3 = "string",      S3 = "String",         D3 = [3],     D3 = [6],
[S1,S2,S3]          [S3,S2,S1]             [D1,D2,D3]    [D3,D2,D1]
```

```
new_fun(NewVar_1,
        NewVar_2,
        NewVar_3) ->
  S1 = NewVar_1,
  S2 = NewVar_2,
  S3 = NewVar_3,
  {S1,S2,S3}.
```

Example: sub-clones

```
S1 = "This",          S1 = "This",          D1 = [1],     D1 = [X+1],
S2 = " is a ",        S2 = "is another ",   D2 = [2],     D2 = [5],
S3 = "string",        S3 = "String",        D3 = [3],     D3 = [6],
[S1,S2,S3]            [S3,S2,S1]            [D1,D2,D3]    [D3,D2,D1]
```

```
new_fun(NewVar_1,
        NewVar_2,
        NewVar_3) ->
  S1 = NewVar_1,
  S2 = NewVar_2,
  S3 = NewVar_3,
  [S1,S2,S3].
```

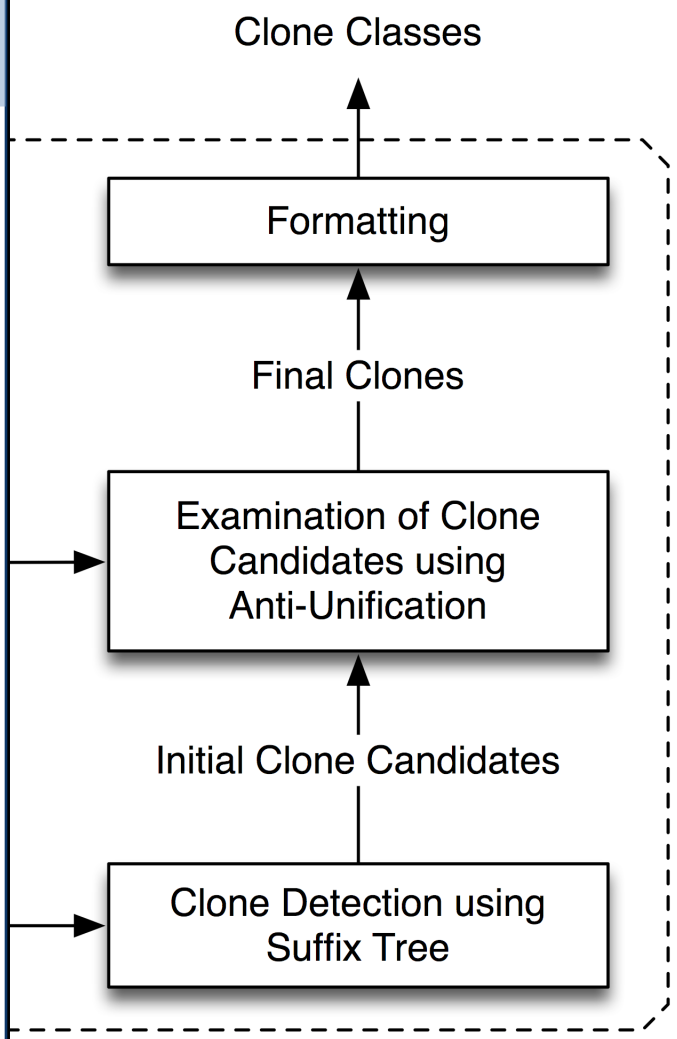
```
new_fun(NewVar_1,
        NewVar_2,
        NewVar_3) ->
  S1 = NewVar_1,
  S2 = NewVar_2,
  S3 = NewVar_3,
  [S3,S2,S1].
```

Clone class output

Clone classes are reported in two different orders

- the size of the *clone class*, and
- the size of the *members* of the clone.

Together with each class is the *anti-unifier*, rendered as an Erlang function definition to cut and paste into the program.



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.

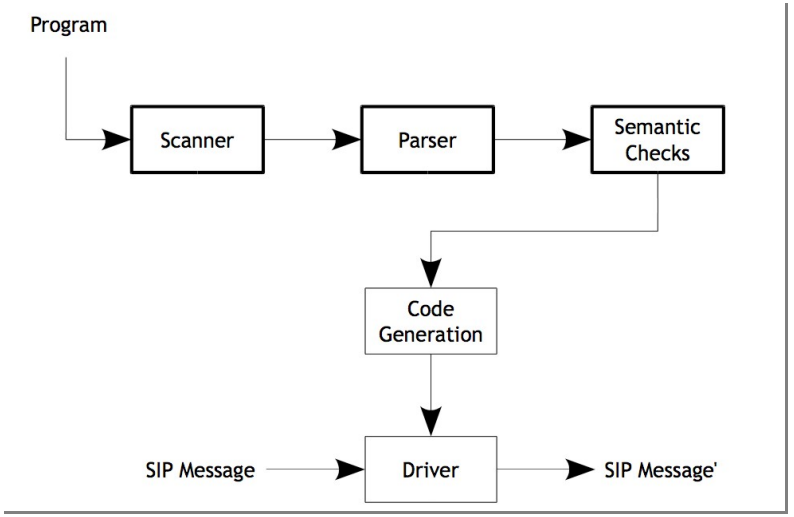
SIP case study

ERICSSON
TAKING YOU FORWARD

Session Initiation Protocol

SIP message manipulation allows rewriting rules to transform messages.

Test by `smm_SUITE.erl`, 2658 LOC.



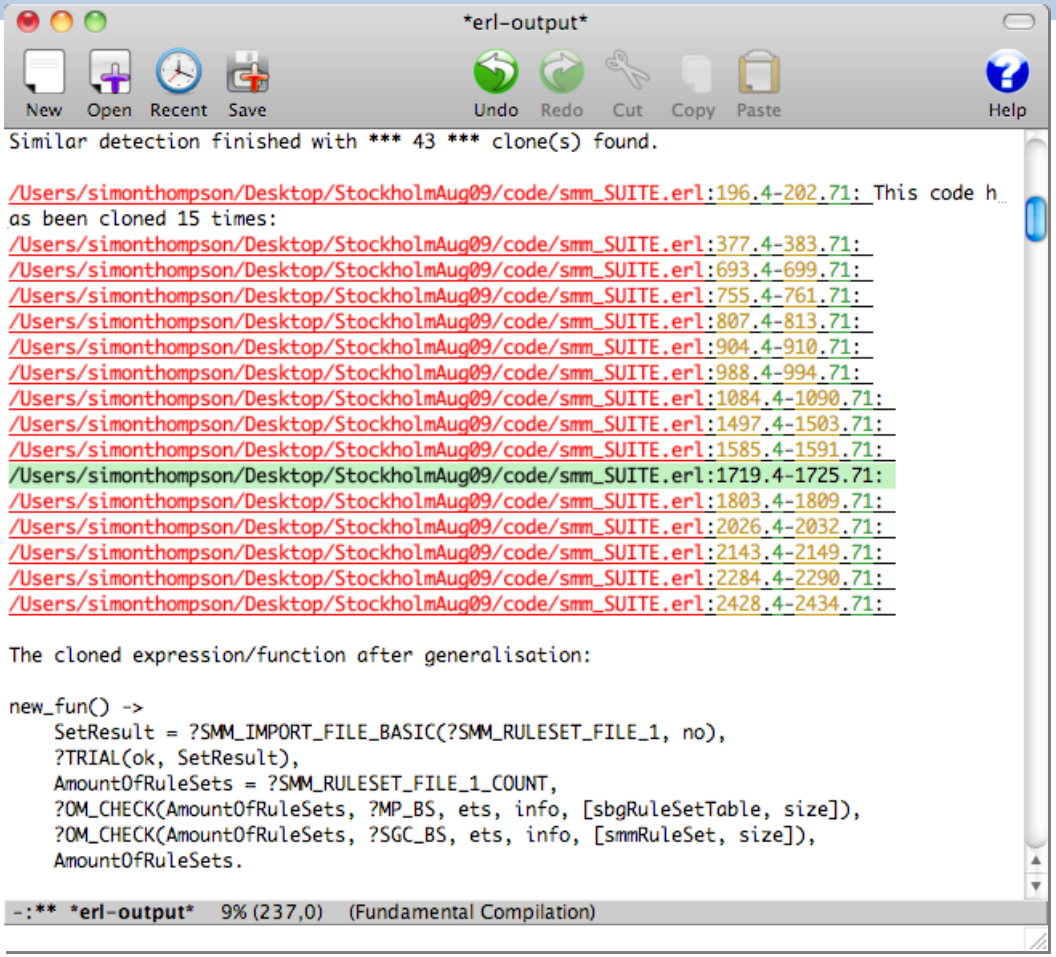
Reducing the case study

1	2658	6	2218	11	2131
2	2342	7	2203	12	2097
3	2231	8	2201	13	2042
4	2217	9	2183	…	…
5	2216	10	2149		

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

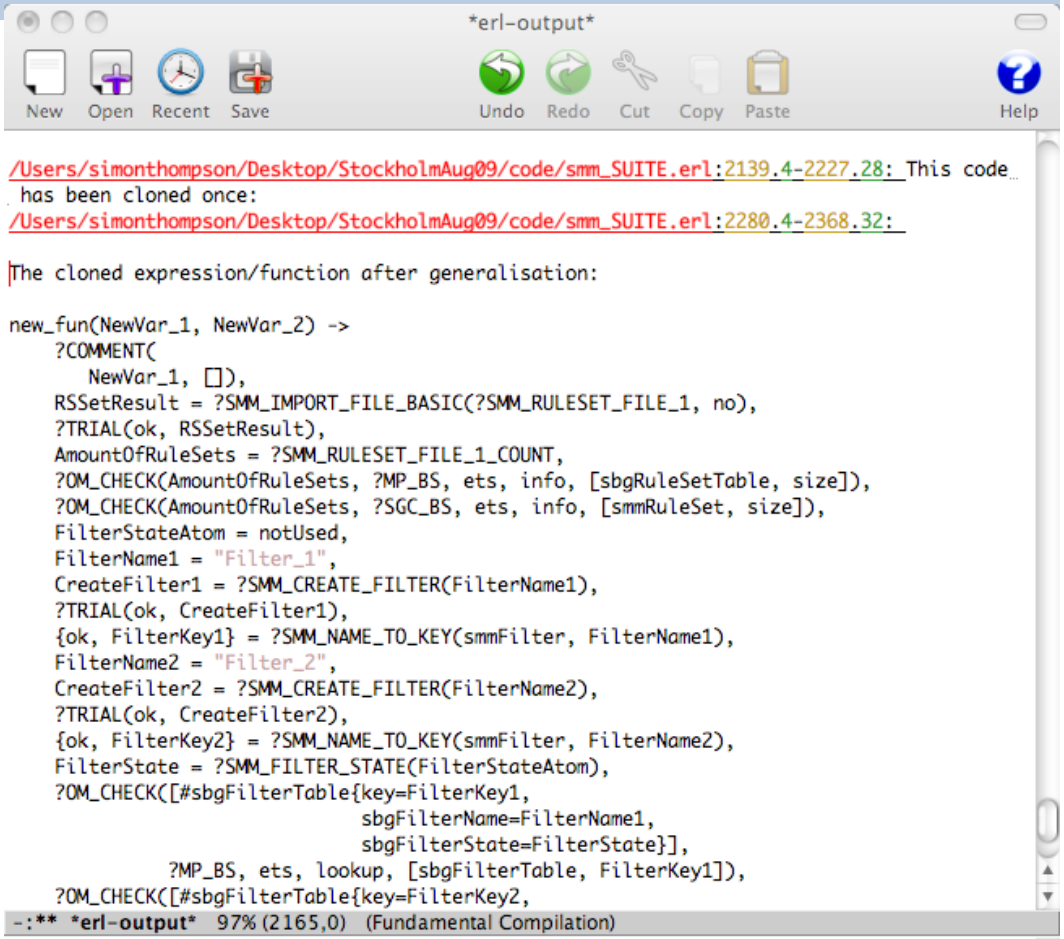
Not step 1

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?

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 7

Different checks: ?OM_CHECK vs ?CH_CHECK

```
code_is_loaded(BS, om, ModuleName, false) ->
 ?OM_CHECK(false, BS, code, is_loaded, [ModuleName]).
code_is_loaded(BS, om, ModuleName, true) ->
 ?OM_CHECK({file, atom_to_list(ModuleName)}, BS, code,
           is_loaded, [ModuleName]).
```

But the calls to ?OM_CHECK have disappeared at step 6 …

… a case of premature generalisation!

Need to inline code_is_loaded/3 to be able to use this …

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}.
```

Steps 14+

Similar code detection (default params):

16 clones, each duplicated once.

193 lines in total: get 145 line reduction.

Reduce similarity to 0.5 rather than the default of 0.8: 47 clones.

Other refactorings: data etc.

Going further


ProTest
property based testing

Property-based testing

Property-based testing will deliver more effective tests, more efficiently.

- Property discovery
- Test and property evolution
- Property monitoring
- Analysing concurrent systems

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.

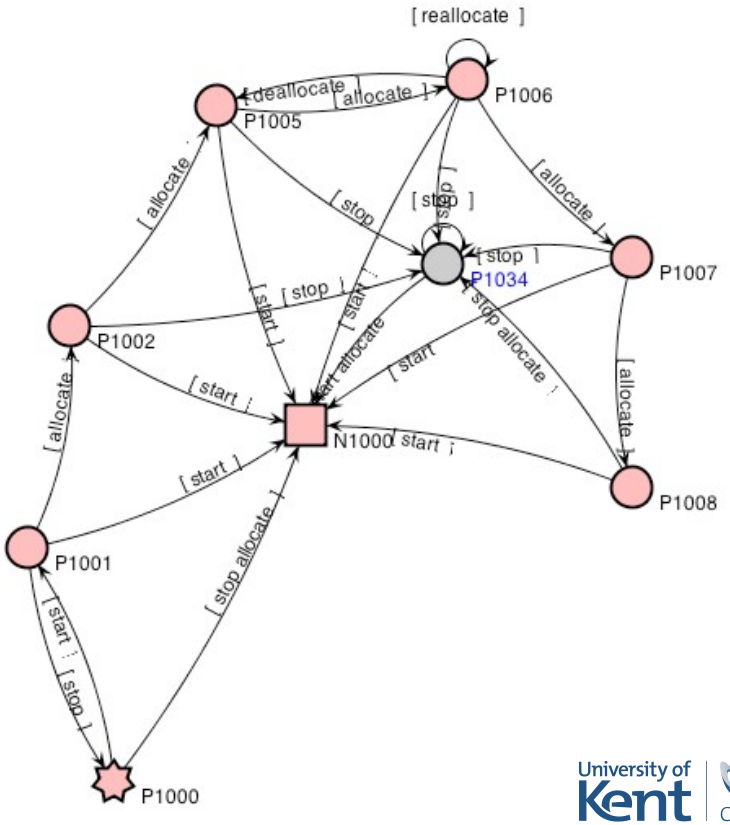
Systems test: FSM discovery

Use FSM to model expected behaviour.

Test random paths through the FSM to test system function.

Extract the FSM from sets of existing test cases.

Use +ve and -ve cases.



ProTest
property based testing

University of Kent Computing

Refactoring and testing

Refactor tests e.g.

- Tests into EUnit tests.
- Group EUnit tests into a single test generator.
- Move EUnit tests into a separate test module.
- Normalise EUnit tests.
- Extract setup and tear-down into EUnit fixtures.

Respect test code in EUnit, QuickCheck and Common Test …

… and refactor tests along with refactoring the code itself.

Next steps

Refine the notion of similarity …

… to take account of insert / delete in command seqs.

Scaling up: look for incremental version; check vs. libraries …

Refactorings of tests and properties themselves.

Extracting FSMs from sets of tests.

Support property extraction from 'free' and EUnit tests.

Conclusions

Efficient clone detection possible on medium-medium sized projects.

This supports improved testing …

… but only with expert involvement.

There's a useful interaction between refactoring and testing.

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