

Testing Wrangler with QuickCheck



Huiqing Li
Simon Thompson

Computing Lab, University of Kent
www.cs.kent.ac.uk/projects/wrangler/

Overview

- Refactoring
- Wrangler – an Erlang Refactorer.
- Validation of Refactoring Tools.
- Testing Wrangler With QuickCheck.
- Conclusions.

Introduction

- ❑ Refactoring -- changing the structure of existing code without changing its meaning.
- ❑ Example: *generalisation* and *renaming*.

```
-module (test).  
-export ([f/1]).  
  
add_one ([]) -> [].  
add_one ([H|T]) ->  
    [H+1 | add_one (T) ].  
  
f (X) -> add_one (X) .
```



```
-module (test).  
-export ([f/1]).  
  
add_one ([], N) -> [].  
add_one ([H|T], N) ->  
    [H+N | add_one (T, N) ] ;  
  
f (X) -> add_one (X, 1) .
```

Introduction

- Refactoring -- changing the structure of existing code without changing its meaning.
- Example: *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) -> [].  
add_int ([H|T],N) ->  
    [H+N | add_int(T,N)];  
  
f(X) -> add_int(X,1).
```

Introduction

- Refactoring:
pre-conditions + program transformation + post-conditions.
- Tools support for refactoring is essential.

Wrangler – An Erlang Refactorer



- A tool for interactive refactoring of Erlang programs.
- Embedded in Emacs and Eclipse+ErlIDE.

The image shows the Emacs editor window titled "emacs@HL-LT". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Refactor", "Inspector", "Erlang", and "Help". The "Refactor" menu is open, displaying a list of options. The code in the buffer is Erlang, with the line `io:format("Hello"),` highlighted in yellow. The status bar at the bottom shows the current buffer is `--(Unix)-- test.erl` and the mode is `-1** *erl-output* All L2 (Fundamental)`. A prompt `New parameter name: A` is visible at the bottom left.

```
-module(test).  
-export([f/0]).  
  
repeat(N) when N=<0 ->  
    ok;  
repeat(N) when N>=1 ->  
    io:format("Hello"),  
    repeat(N-1).  
  
f() ->  
    repeat(5).
```

--(Unix)-- test.erl

New parameter name: A

Refactor menu items:
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
Rename a Process (beta)
Add a Tag to Messages (beta)
Register a Process (beta)
From Function to Process
Detect Duplicated Code in Current Buffer
Detect Duplicated Code in Dirs
Identical Expression Search
Introduce a Macro
Fold Against Macro Definition
Undo (C-c C-)
Customize Wrangler
Version

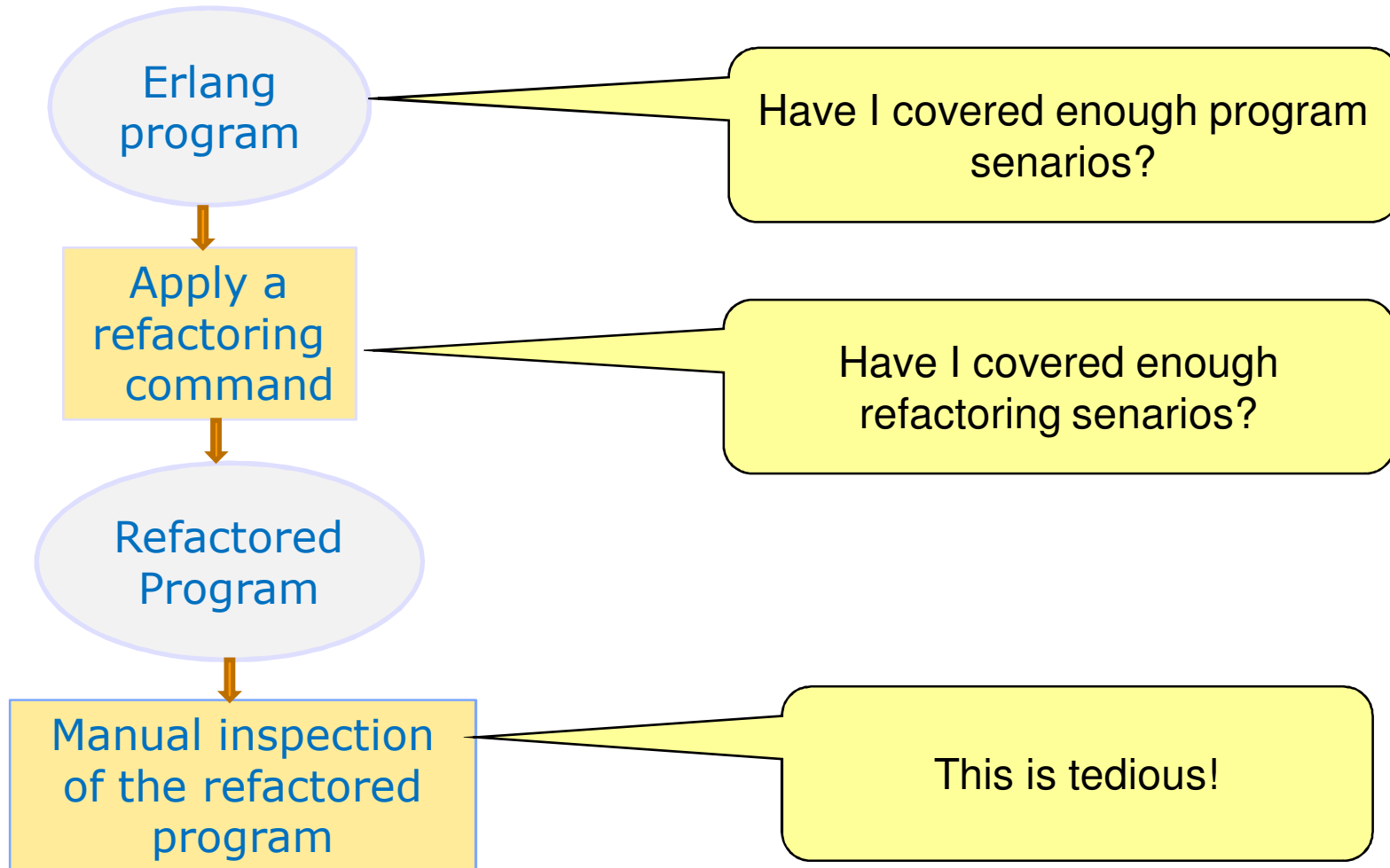
Validation of Refactoring Engines

- ❑ Refactoring tools should be reliable.
- ❑ A bug in a refactoring tool could introduce bugs into the programs refactored.
- ❑ Validation of refactoring tools is hard.

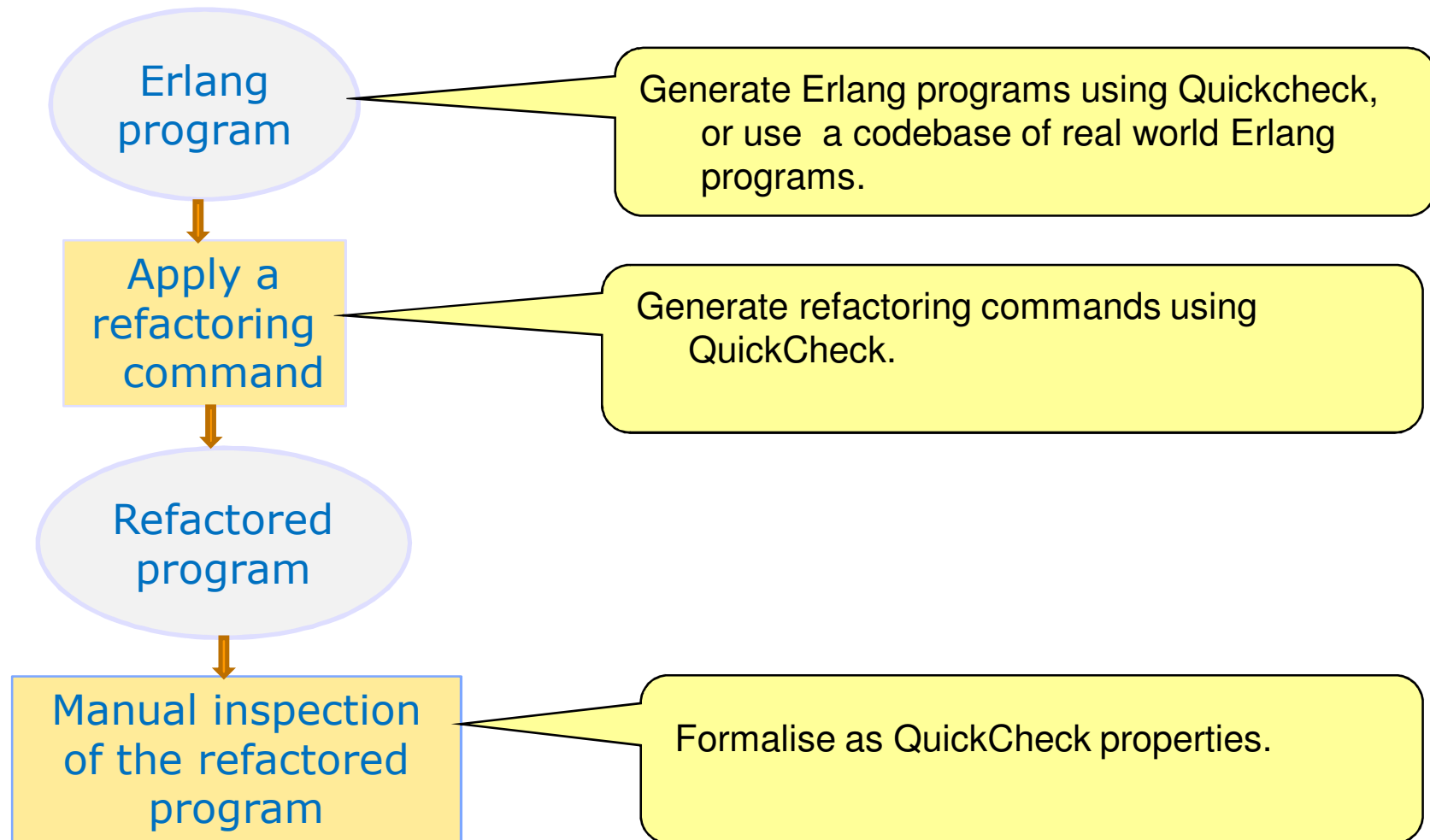
Validation of Refactoring Engines

- Regression testing of refactored programs.
- Programs as data
 - Abstract Syntax Tree (AST).
 - Source code, when layout is also important.
- Program verification.
- Property-based testing.

Manual Validation of Refactoring Engines



Testing Wrangler with QuickCheck



Testing Wrangler with QuickCheck.

- Example: testing *renaming a function name*.
 - Refactoring command generation.

```
rename_fun_commands(Dir) ->
  ?LET(FileName, gen_filename(Dir),
    {FileName,
      oneof(collect_fun_locs(FileName)),
      oneof(collect_names(FileName)),
      Dir}).
```

- Some sample commands generated.


```
1% {"refac_rename_fun.erl", {243, 64}, halt, "c:/wrangler/test"}
1% {"refac_qc.erl", {184, 48}, ordsets, "c:/wrangler-0.1/test"}
1% {"test.erl", {5, 39}, "DDD", "c:/wrangler-0.1/test"}
```

Testing Wrangler with QuickCheck.

- Post-conditions as properties.
 - General conditions.
 - The refactored code meets all the tests that the original version met.
 - The refactoring engine should not crash.
 - The new program should compile.
 - Refactoring-specific conditions, e.g.
 - Renaming a variable name should not change the binding structure of the program.
 - Inversibility
 - Generalising a function, f/n say, turns occurrences of f/n to $f/(n+1)$.

Conclusions

- ❑ The correctness of refactoring is tested against specifications written in Erlang.
- ❑ The development of refactorings and their testing are very closely integrated.
- ❑ Able to run many test cases in a very short time, and find bugs more quickly.
- ❑ A lot easier to test the refactoring tool on large systems.



Thank you

Questions?