Update on EU project ProTest

ProTest
property based testing

University of Kent Computing · Erlang · QuviQ · Universidade da Coruña · IT University of Göteborg

http://www.protest-project.eu/

Some results from Q1 + Q2

- Testing Erlang Data Types with QuickCheck
- Refactoring with Wrangler
- Early fault detection with model-based testing
- Erlang Testing and Tools Survey

http://www.protest-project.eu/

Testing Erlang Data Types with QuickCheck

Thomas Arts
Laura Castro
John Hughes

ProTest property based testing
QuviQ
UNIVERSIDADE DA CORUÑA
IT University of Göteborg
CHALMERS | GÖTEBORG UNIVERSITY

Challenge

Erlang libraries supply a number of data types, but sometimes you want to design your own.

We presented a method that ensures full testing of an implementation of a home-made data type.

Full paper published at Erlang workshop 2008

ProTest property based testing
QuviQ
UNIVERSIDADE DA CORUÑA
IT University of Göteborg
CHALMERS | GÖTEBORG UNIVERSITY

Testing

Implementation of data type for decimals

How to test this implementation?

```
decimal() ->
  ?LET(Tuple, {int(),nat()}, new(Tuple)).

prop_sum_comm() ->
  ?FORALL({D1,D2}, {decimal(),decimal()},
          sum(D1,D2) == sum(D2,D1)).
```

QuickCheck generates thousands of tests

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Testing

Which other properties do we add?

When do we have sufficiently many properties?

Use a Model

Erlang functions

Model operations

$$[\mathrm{sum}(D1,D2)] = [D1] + [D2]$$
$$[\mathrm{subs}(D1,D2)] = [D1] - [D2]$$
$$[\mathrm{mult}(D1,D2)] = [D1] * [D2]$$
$$[\mathrm{lt}(D1,D2)]_1 = [D1] < [D2]$$

.....

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Symbolic data

Use symbolic data structures instead of real data structures in test generation:

easier to analyze errors

```
decimal() ->
    ?LET(Tuple, {int(), nat()},
         {call, decimal, new, [Tuple]}).
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Symbolic data

Translate symbolic value to real value in property

```
prop_sum() ->
  ?FORALL({SD1,SD2},{decimal(),decimal()},
          begin
            D1 = eval(SD1),
            D2 = eval(SD2),
            model(sum(D1,D2)) ==
                model(D1) + model(D2)
          end).
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Testing model equivalence

We run QuickCheck....

```
> eqc:quickcheck(decimal_eqc:prop_sum()).
........Failed! After 9 tests.
{{call,decimal,new,[{2,1}]},
 {call,decimal,new,[{2,2}]}}
Shrinking..(2 times)
{{call,decimal,new,[{0,1}]},
 {call,decimal,new,[{0,2}]}}
false
```

Thus: 0.1 + 0.2 =/= 0.3 ??

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Testing model equivalence

Indeed!

Unavoidable rounding error according to IEEE 754-1985. Our model is incorrect.

We fix it!

```
> (0.1+0.2) == 0.3.
false
> (0.1+0.2) - 0.3.
5.55112e-17
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Testing model equivalence

Property prop_sum() passes thousands of test cases.

Similarly, we can add a property prop_mult().

But... although we will obtain 100% code coverage, we miss testing combinations of mult and sum!

Recursive generators

```
decimal() ->
   ?SIZED(Size, decimal(Size)).

decimal(0) ->
   {call, decimal, new, [{int(),nat()}]};
decimal(Size) ->
   Smaller = decimal(Size div 2),
   oneof([
      decimal(0),
      ?LETSHRINK([D1, D2], [Smaller, Smaller],
                 {call, decimal, sum, [D1, D2]}),
      ?LETSHRINK([D1, D2], [Smaller, Smaller],
                 {call, decimal, mult, [D1, D2]}
   ]).
```

Testing model equivalence

Add subs and divs to generator and test **same property** again:

```
> eqc:quickcheck(decimal_eqc:prop_sum()).
............Failed!
After 13 tests.
Shrinking....(4 times)
Reason:
{'EXIT',{{not_ok,{error,decimal_error}},
        [...]}}
{{call,decimal,divs,
   [{call,decimal,new,[{0,0}]},
    {call,decimal,new,[{0,0}]}]},
 {call,decimal,new,[{0,0}]}}
false
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Negative testing

We do want to test that division by zero results in an error... *in prop_divs, not in prop_sum*

```
prop_divs() ->
   ?FORALL({SD1, SD2}, {decimal(), decimal()},
      begin
         D1 = eval(SD1),
         D2 = eval(SD2),
         case catch (model(D1)/model(D2)) of
            {'EXIT',_} ->
                   is_error(divs(D1, D2));
            Value ->
               equiv(model(divs(D1, D2)),
                     Value)
         end
      end).
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Generate well defined values

We find the error in prop_divs and we do not want to generate decimals in which we divide by zero.

```
decimal() ->
   ?SIZED(Size, well_defined(decimal(Size))).

well_defined(G) ->
   ?SUCHTHAT(E, G, defined(E)).

defined(E) ->
   case catch {ok, eval(E)} of
      {ok, _}      -> true;
      {'EXIT', _} -> false
   end.
```

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Conclusion

Method:

1. Choose a model
2. Write symbolic (recursive) generators
3. Write one property for each operation, consider expected failing cases
4. Use a well-defined trick to avoid errors in generation

When following the proposed method, one has a guarantee that the data structure is fully tested.

ProTest property based testing QuviQ UNIVERSIDADE DA CORUÑA IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Refactoring with Wrangler

Huiqing Li
Simon Thompson
University of Kent

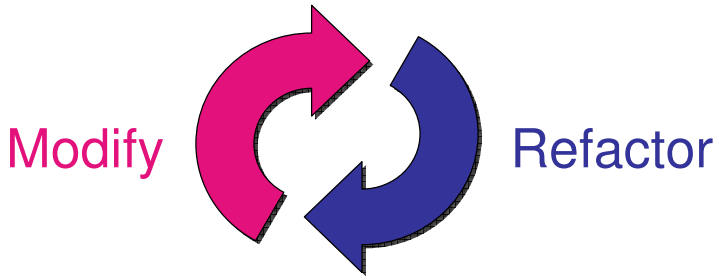
Melinda Tóth
George Orosz
Eötvos Loránd Univ

ProTest
property based testing

University of Kent Computing

Refactoring

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



ProTest
property based testing

University of Kent Computing

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).
```

ProTest property based testing

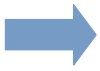
University of Kent Computing

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]).
```

ProTest property based testing

University of Kent Computing

Refactoring tool support

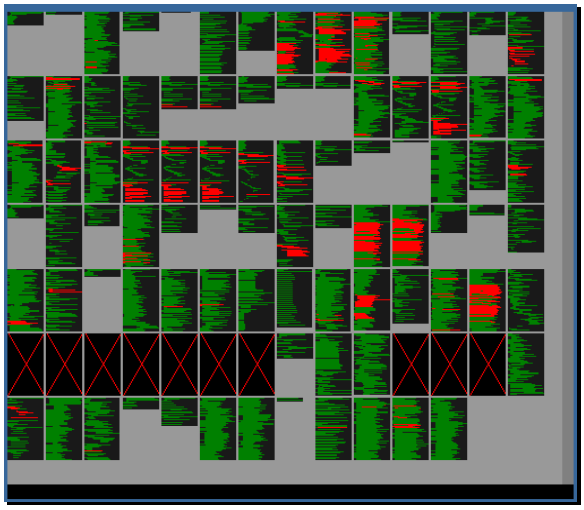
Bureaucratic and diffuse.

Tedious and error prone.

Semantics: scopes, types, modules, …

Undo/redo

Enhanced creativity



ProTest property based testing

University of Kent Computing

Wrangler

Embedded in Emacs and Eclipse.

Structural, data type and module refactorings.

AAST-based analysis and transformation.

Works with multiple modules.

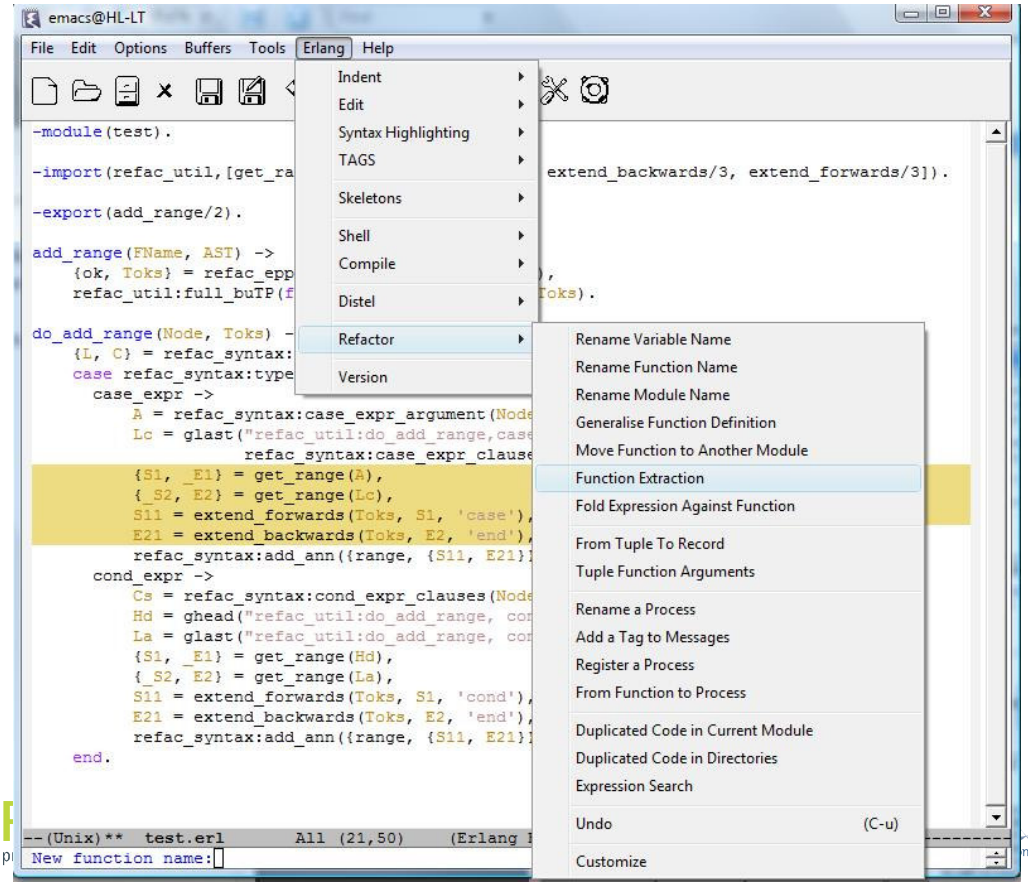
Supports undo of refactorings

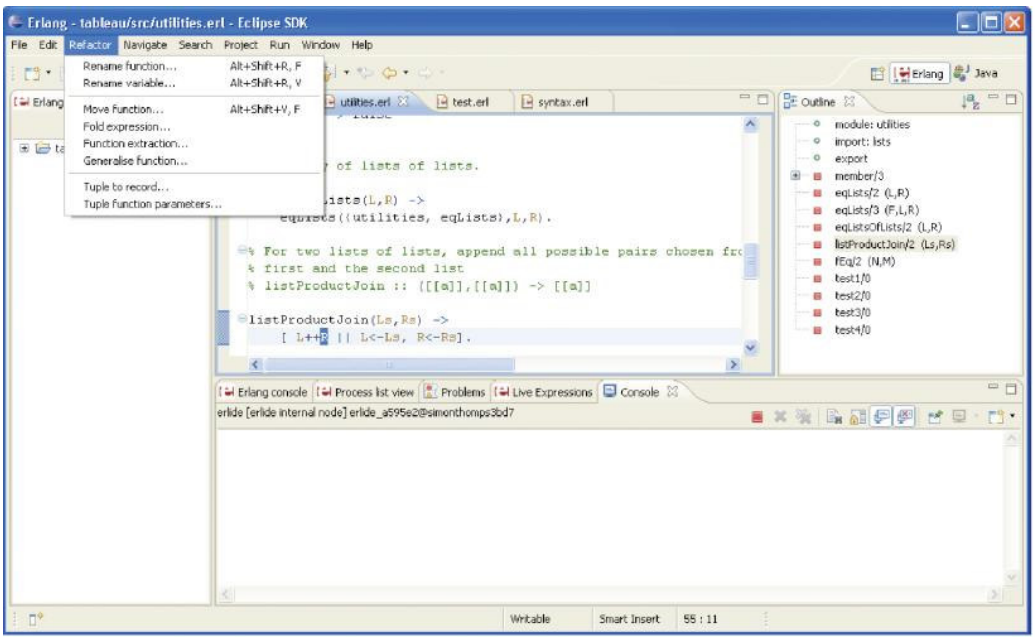
Preserves layout and comments as much as possible.

Respects aspects of the macro system.

ProTest property based testing

University of Kent Computing





ProTest
property based testing

University of Kent Computing

Refactorings in Wrangler

- Renaming variable, function, module, process
- Function generalisation
- Move function
- Function extraction
- Fold against defn.
- Tuple function arguments
- Register a process
- From function to process
- Add a tag to messages

ProTest property based testing

University of Kent Computing

Duplicate Code Detection

Especially for Erlang/OTP programs.

Report syntactically well-formed code fragments that are identical after consistent renaming of variables …

… ignoring differences in literals and layout.

Integrated with the refactoring environment.

ProTest property based testing

University of Kent Computing

Code Inspection Support

- Variable use/binding information.
- Caller functions.
- Caller/callee modules.
- Case/if/receive expressions nested more than a specified level.
- Long function/modules.
- Non tail-recursive servers.
- Non-flushed unknown messages
- . . .

ProTest property based testing

University of Kent Computing

Ongoing and Future work

- Continue the integration of Wrangler with Eclipse + Erlide
- More refactorings are being added including introduce macros, from module to process, etc.
- To investigate the use of trace information to help the refactoring process, especially process-related refactorings.

ProTest property based testing

University of Kent Computing

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

ProTest
property based testing

University of Kent | Computing

Protest Survey on Erlang Testing Tools

Aniko Nagyné Víg
Tamás Nagy
Francesco Cesarini
Erlang Training and Consulting

ProTest
property based testing

Erlang

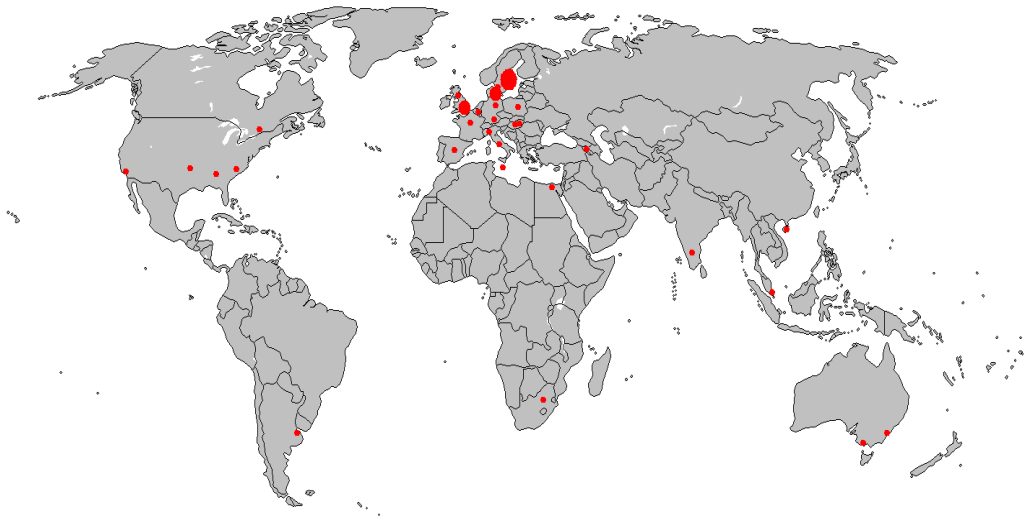
Research Method

- Published an online survey
- Advertised it by email:
  - Erlang Questions, approx. 1000 users
  - Erlang Training and Consulting Newsletter list, approx. 1000 users
  - Smaller Erlang related mailing lists
    - Trapexit User Group, 500 users
    - ProTest Mailing List, 50 users
    - London / Stockholm Erlang User Groups, 100 users
- 200 direct emails to relevant contacts at ETC
  - Merged with the main survey after the results were similar
- 40-45% of total(200) responses were from developers

ProTest property based testing

Erlang

Geographical Diversion



ProTest property based testing

Erlang

Survey Structure

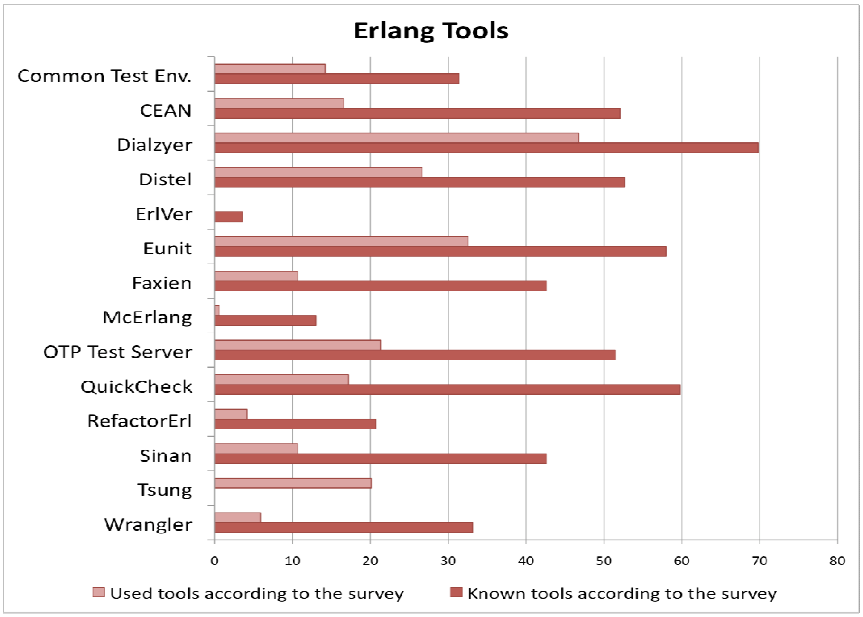
We asked 20 questions about

- The Erlang development environment
- Usage and Knowledge of existing tools and open source applications
- Submitter's job role and Erlang background
- Identify common processes to improve tools support

ProTest property based testing

Erlang

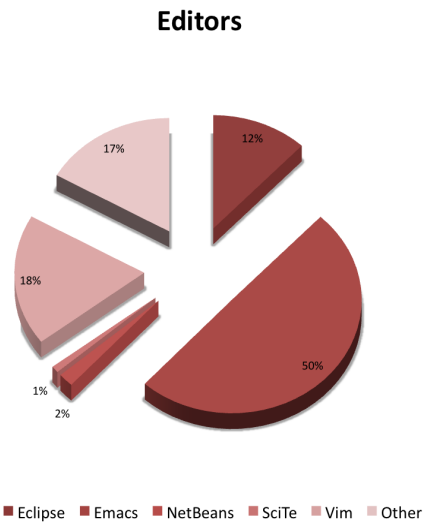
Erlang Tools Knowledge / Usage

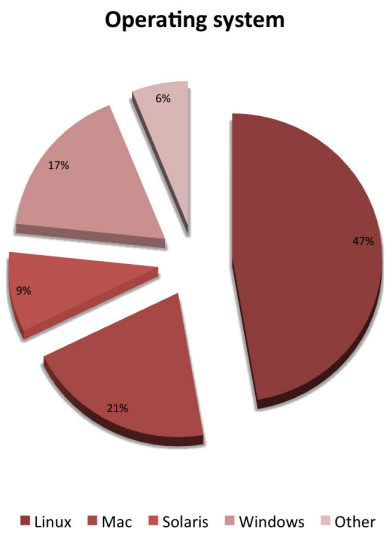


ProTest property based testing

Erlang

Editors and Operating Systems





ProTest
property based testing

Erlang

Problems Identified by the Survey

- Weaknesses of most Erlang tools and projects were found to be
  - Lack of documentation
  - Lack of examples and tutorials
  - Incomplete and untested tools
- Design issues included
  - Badly layered software
  - Not extensible and not structured
- Doubts about sustainability & support
- Hard to install and use
  - Especially for non Erlang users
  - Extensive manual configuration required

ProTest
property based testing

Erlang

Missing Functionality

- No tools for stub generation
- Testing tools lack high quality results display
  - Web interface or dashbord
- Load testers are not available for all requirements
  - Especially state based protocols
- Continous integration
  - Hooks towards version control systems
  - Integrated into a general framework
- A complete framework that integrates different tools

ProTest
property based testing

Erlang

Conclusion

What are the key factors for building a successful Erlang tool?

- Reliable software
- User friendliness
- Good documentation
- Support
- Well promoted!!

ProTest
property based testing

Erlang

Further Reading

Paper from the ACM SIGPLAN Erlang Workshop and Complete Survey Results are available at

www.protest-project.eu/publications.html

ProTest property based testing

Erlang

Questions?

ProTest property based testing

ProTest property based testing

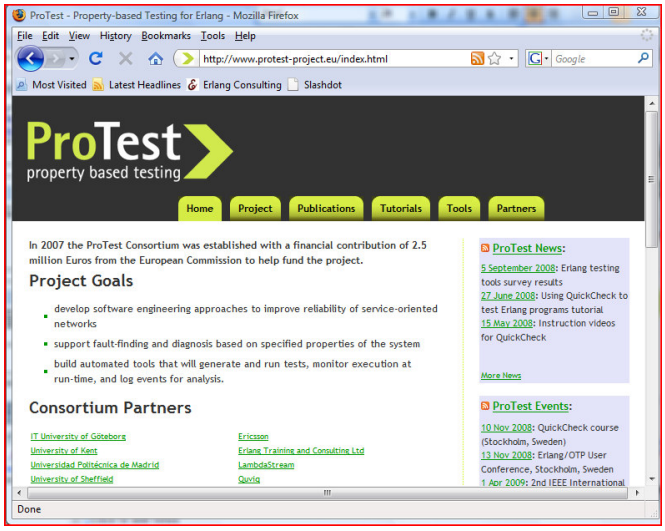
University of Kent Computing

Erlang

QuviQ

IT University of Göteborg CHALMERS | GÖTEBORG UNIVERSITY

Subscribe to our RSS feeds and newsletter @ www.protest-project.eu



ProTest property based testing

University of Kent Computing

Erlang

QuviQ

IT University of Göteborg
CHALMERS | GÖTEBORG UNIVERSITY