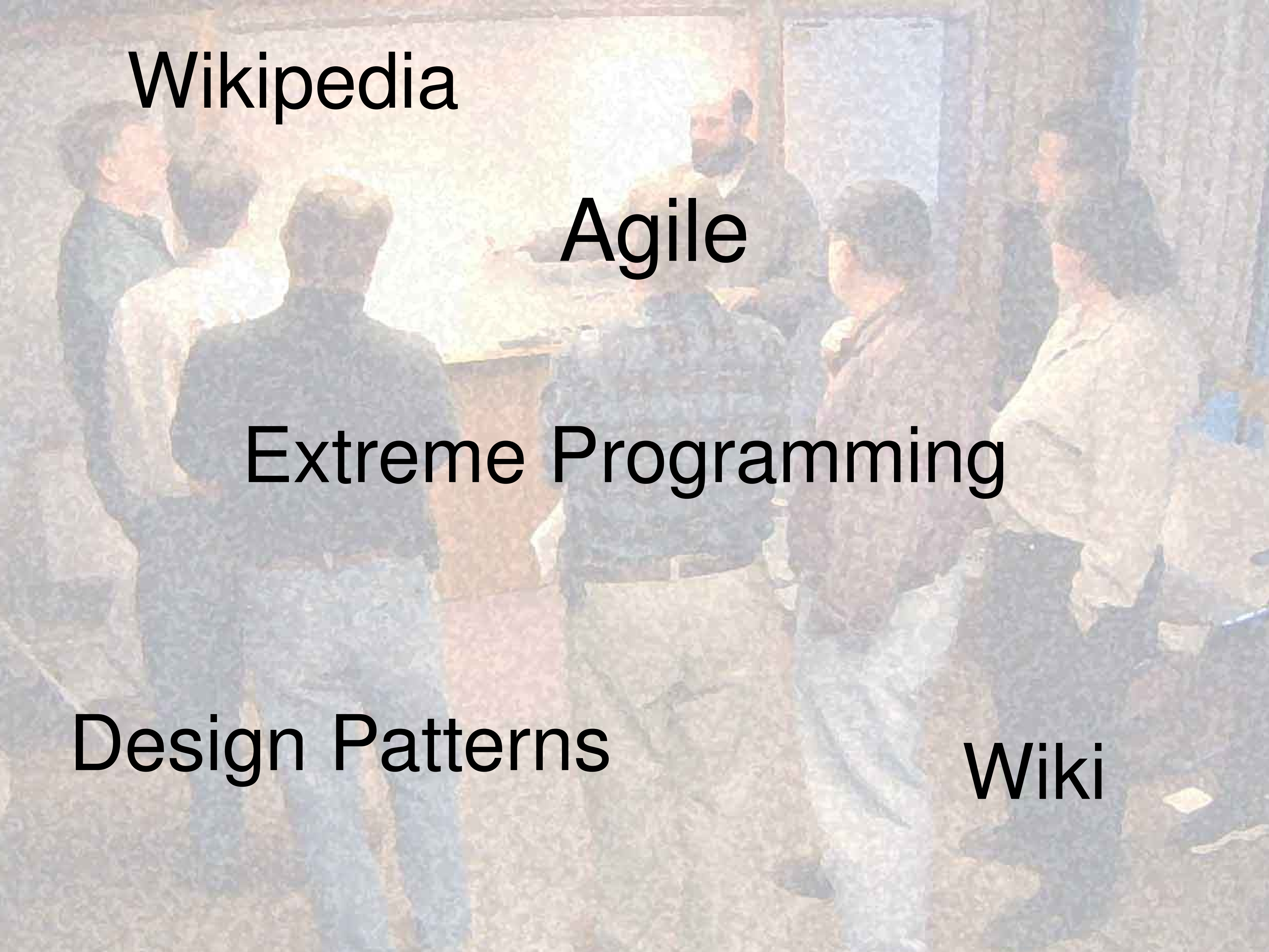
The Tao of TDD

Dominic Williams . net

Erlang Factory London 2010



Wikipedia

Agile

Extreme Programming

Design Patterns

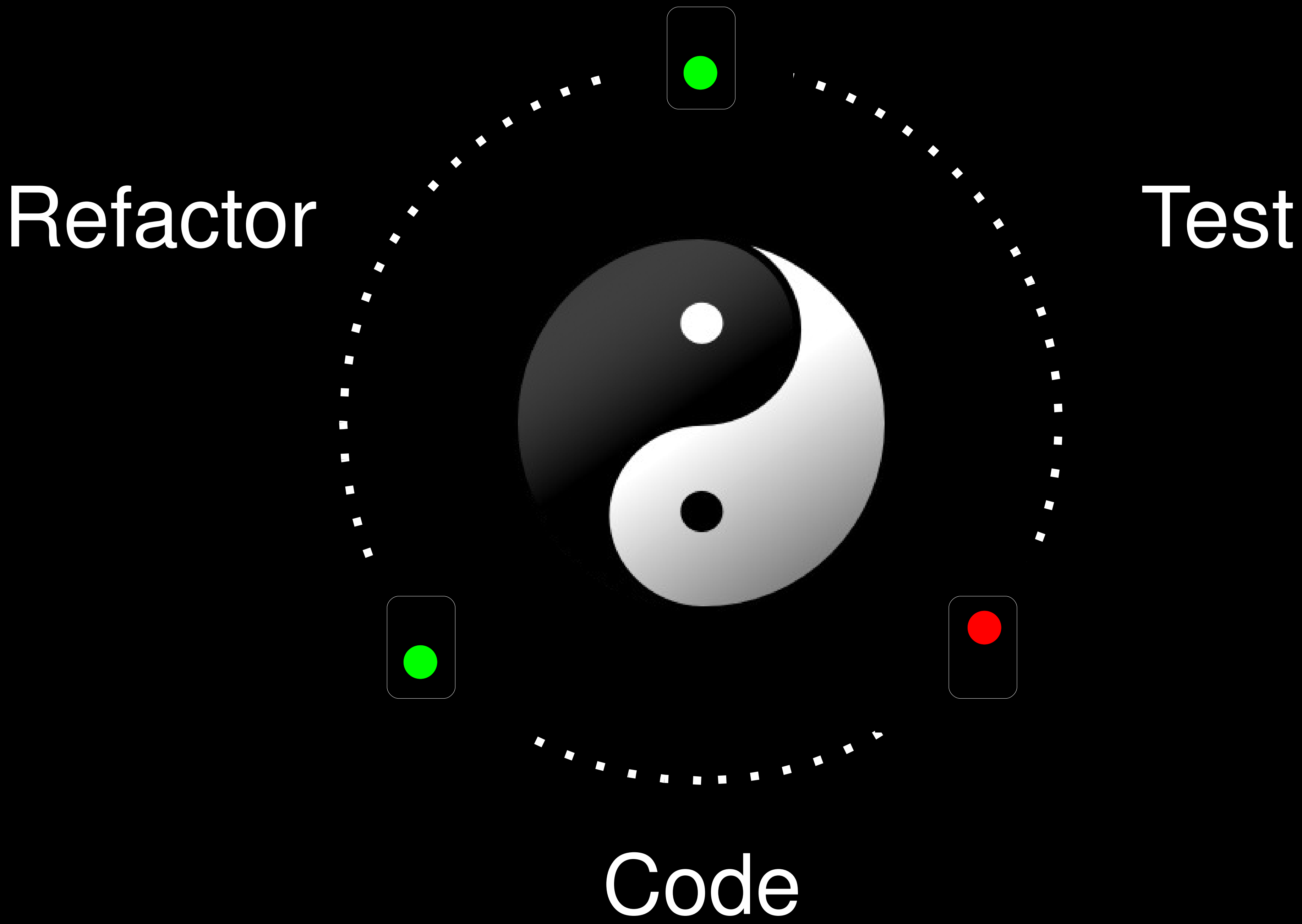
Wiki

Ward Cunningham

agile^2

erlang^2

Kent Beck


Refactor
Test
Code

Test

Driven

<u>Development</u>

Test

Driven

Development

Test

Driven

Development

© Alex Betis. Reproduced with permission.

Canonical test

```
"world" = string:substr ("Hello world", 7).
```

Characterisation tests

```
"world" = string:substr ("Hello world", 7),
"Hello" = string:substr ("Hello world", 1, 5),
“foo” = string:substr (“foo”, 1, 99),
“” = string:substr (“”, 1, 99).
```

Data-driven tests

```
Tests = [{"Hello world", 1, 99, "Hello world"},
         {"Hello world", 7, 99, "world"},
         {"Hello world", 1,  5, "Hello"},
         {"",            1,  5, ""}],
lists: foreach (
   fun ({String, Start, Len, Result}) ->
      Result = string:substr(String, Start, Len)
   end,
   Tests).
```

Tabular tests

String	***Start***	***Len***	***string: substr***
“Hello world”	7		“world”
“Hello world”	1	5	“Hello”
“foo”	1	99	“foo”
“”foo	1	1	“f”
“”	1	99	“”

http://fit.c2.com

The problem(s) with objects

- Construction and initialisation
- Results are hidden behind queries (“getters”)
- Effects propagate to included objects
- Inheritance and interfaces

➔ Canonical tests are rare

```
public void testMixedAddition () {
  Expression fiveBucks = Money.dollar(5);
  Expression tenFrancs = Money.franc(10);
  Bank bank = new Bank();
  bank.addRate("CHF", "USD", 2);
  Money result =
    bank.reduce(fiveBucks.plus(tenFrancs), "USD");
  asserEquals(Money.dollar(10), result);
}
```

The same in Erlang

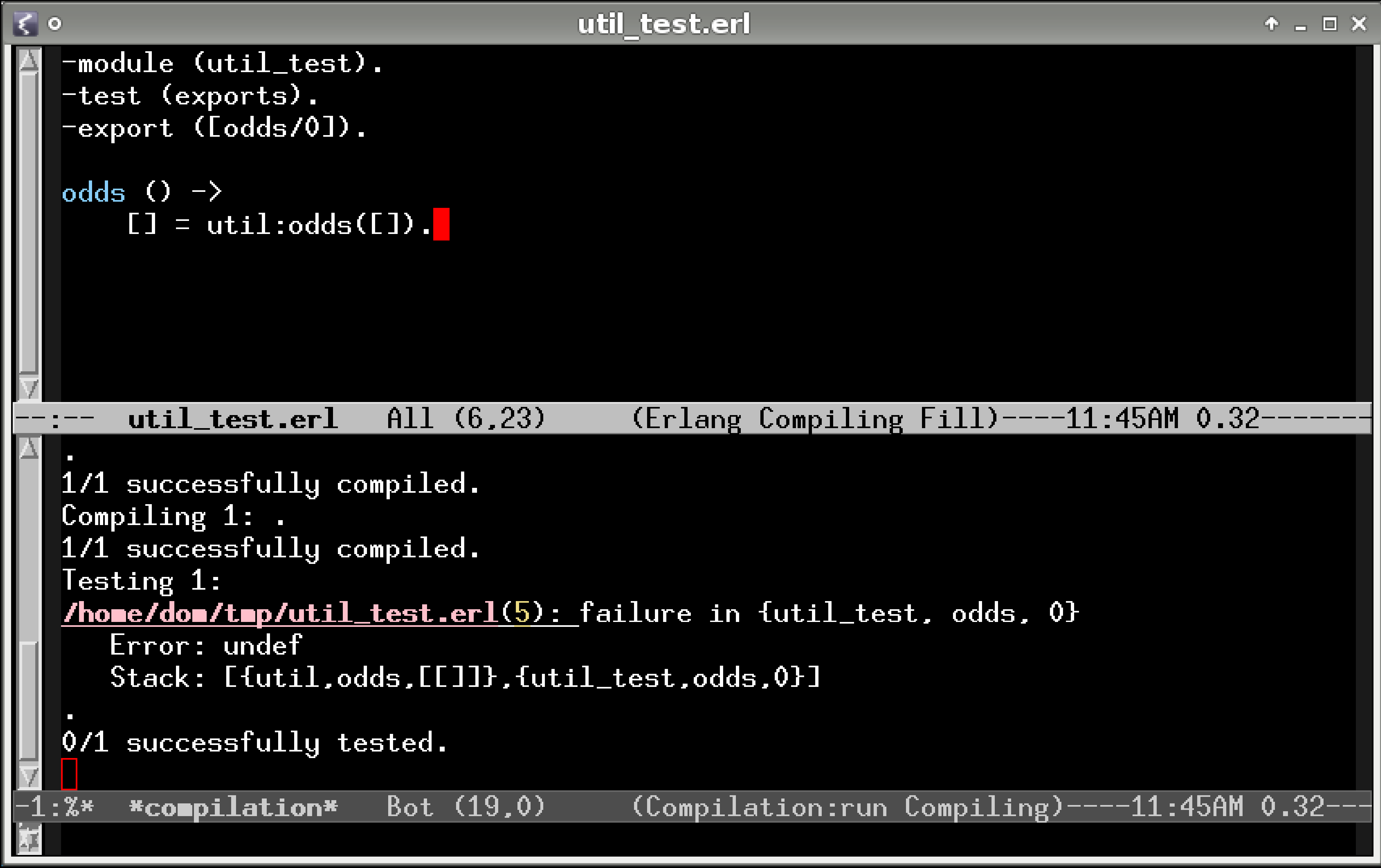
```
Rates = [{“CHF”, “USD”, 2}],
{10,“USD”} = money:add({5,”USD”}, {10,“CHF”}, Rates).
```

The advantage(s) of FP

- Everything is geared towards making pure functions possible
- Data is explicit
- Complex data structures (lists, tuples) can be constructed on the fly
- Dynamic typing

The match with matching

- TDD is often performed using techniques called “fake it” and “triangulation”



util_test.erl

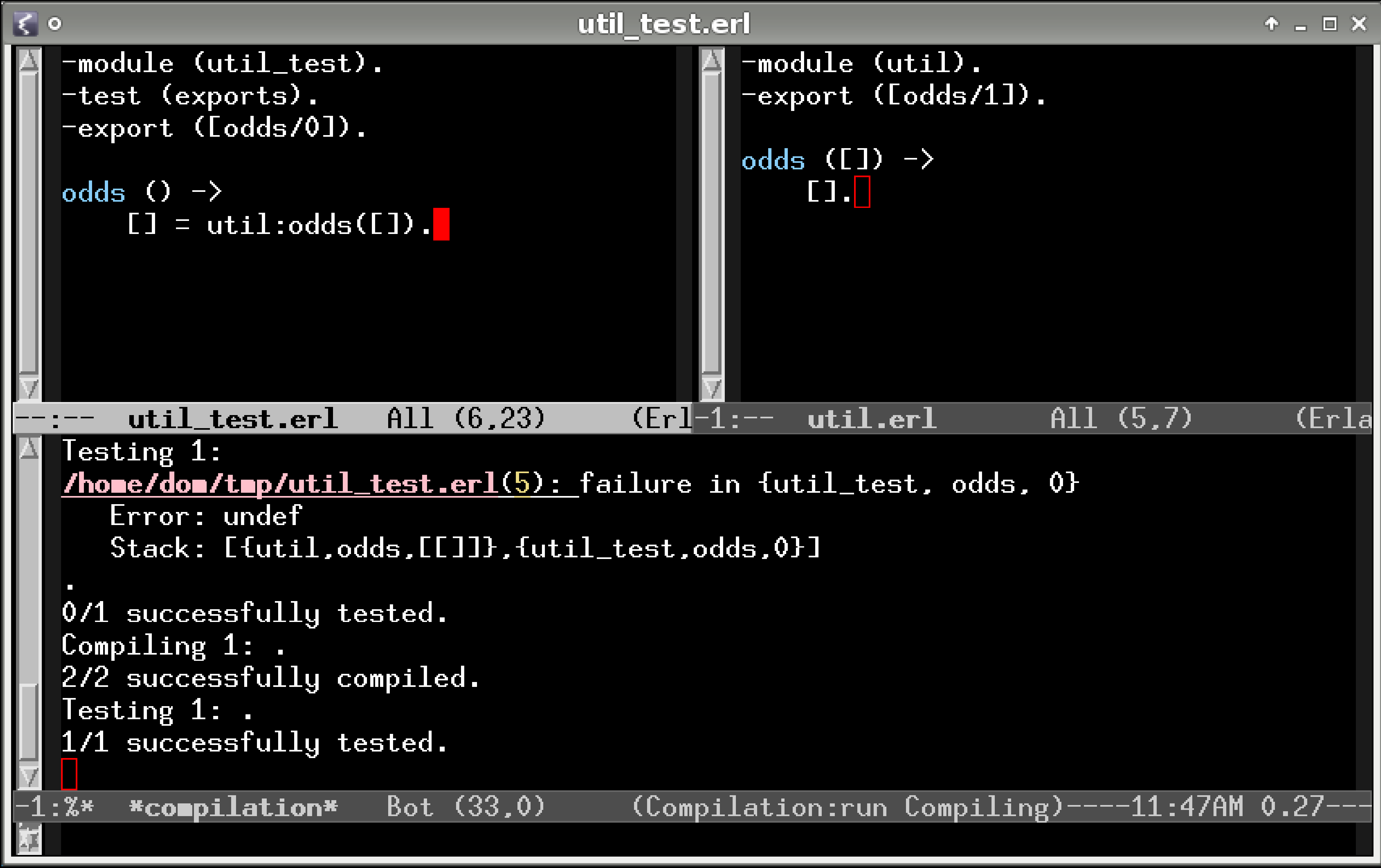
```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]).
```

```
--:--  util_test.erl   All (6,23)     (Erlang Compiling Fill)----11:45AM 0.32-------
```

```
.
1/1 successfully compiled.
Compiling 1: .
1/1 successfully compiled.
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: undef
   Stack: [{util,odds,[[]]},{util_test,odds,0}]
.
0/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (19,0)     (Compilation:run Compiling)----11:45AM 0.32---
```


util_test.erl

```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]).
```

```
--:--  util_test.erl   All (6,23)     (Erl
```

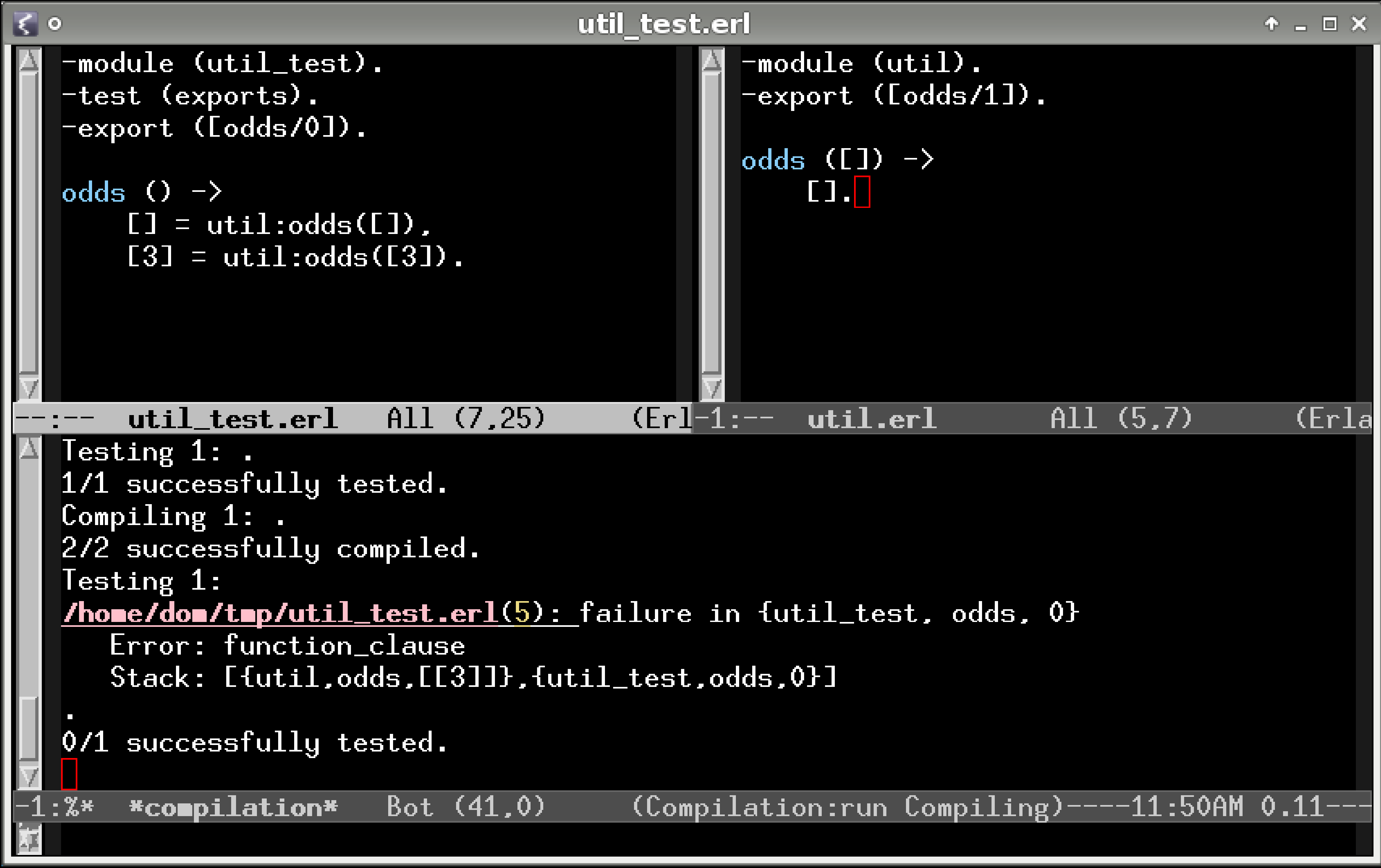
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [].
```

```
-1:--  util.erl        All (5,7)      (Erla
```

```
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: undef
   Stack: [{util,odds,[[]]},{util_test,odds,0}]
.
0/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1: .
1/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (33,0)     (Compilation:run Compiling)----11:47AM 0.27---
```



util_test.erl

```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]).
```

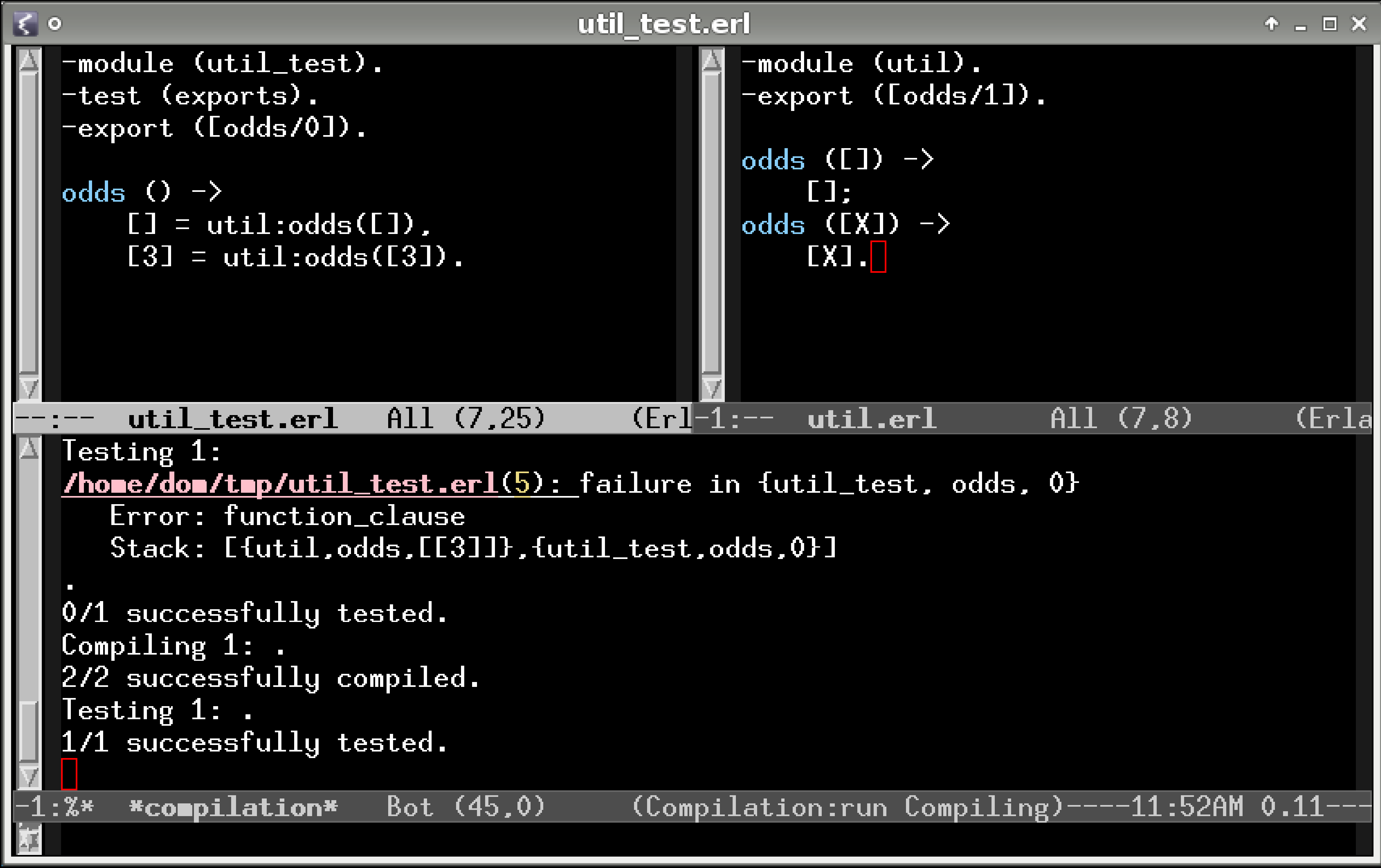
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [].
```

```
--:--  util_test.erl   All (7,25)    (Erl-1:--  util.erl      All (5,7)      (Erla
```

```
Testing 1: .
1/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: function_clause
   Stack: [{util,odds,[[3]]},{util_test,odds,0}]
.
0/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (41,0)    (Compilation:run Compiling)----11:50AM 0.11---
```



```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]).
```

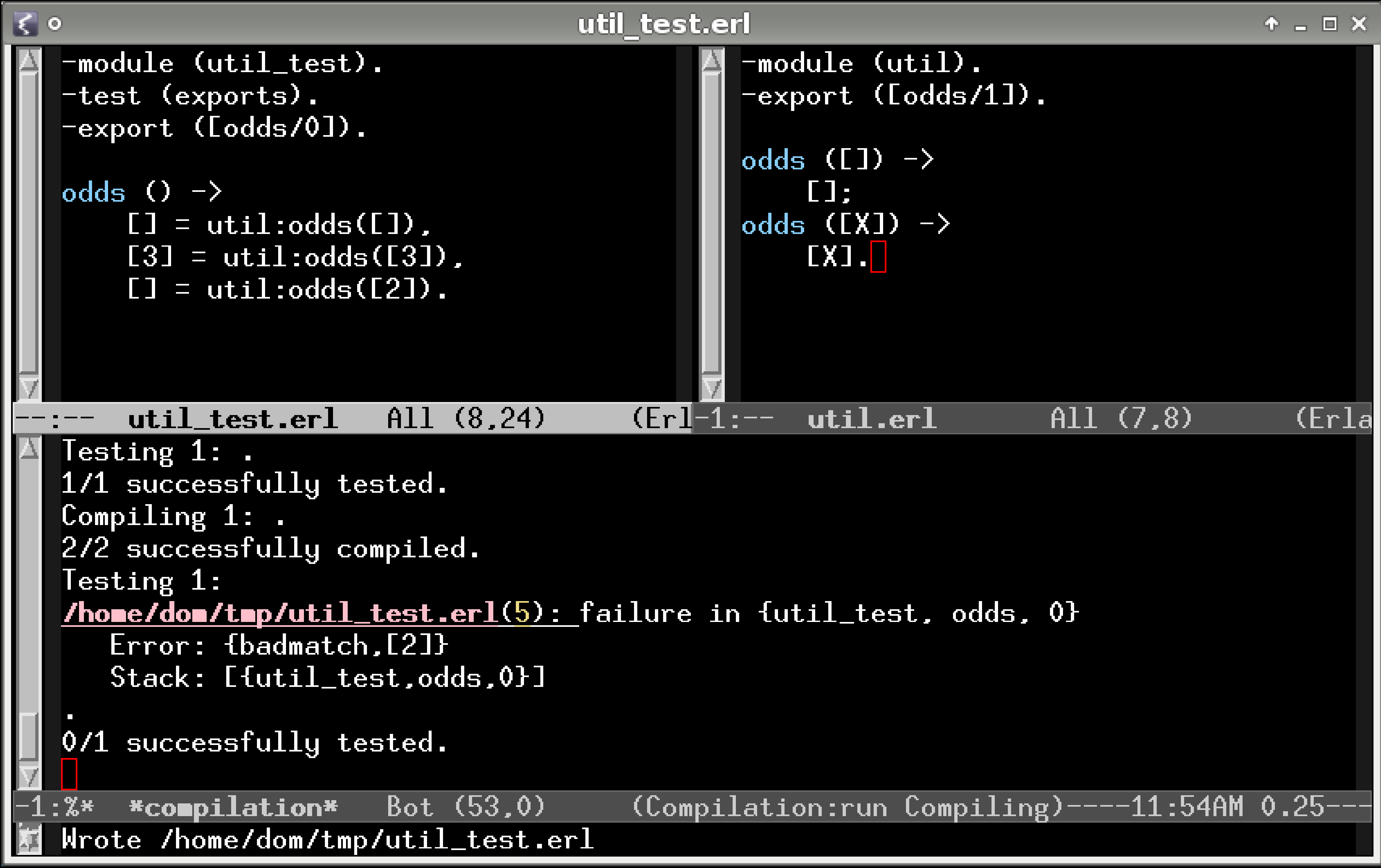
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [];
odds ([X]) ->
    [X].
```

```
--:--  util_test.erl   All (7,25)    (Erl-1:--  util.erl      All (7,8)     (Erla
```

```
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: function_clause
   Stack: [{util,odds,[[3]]},{util_test,odds,0}]
.
0/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1: .
1/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (45,0)    (Compilation:run Compiling)----11:52AM 0.11---
```



```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]),
    [] = util:odds([2]).
```

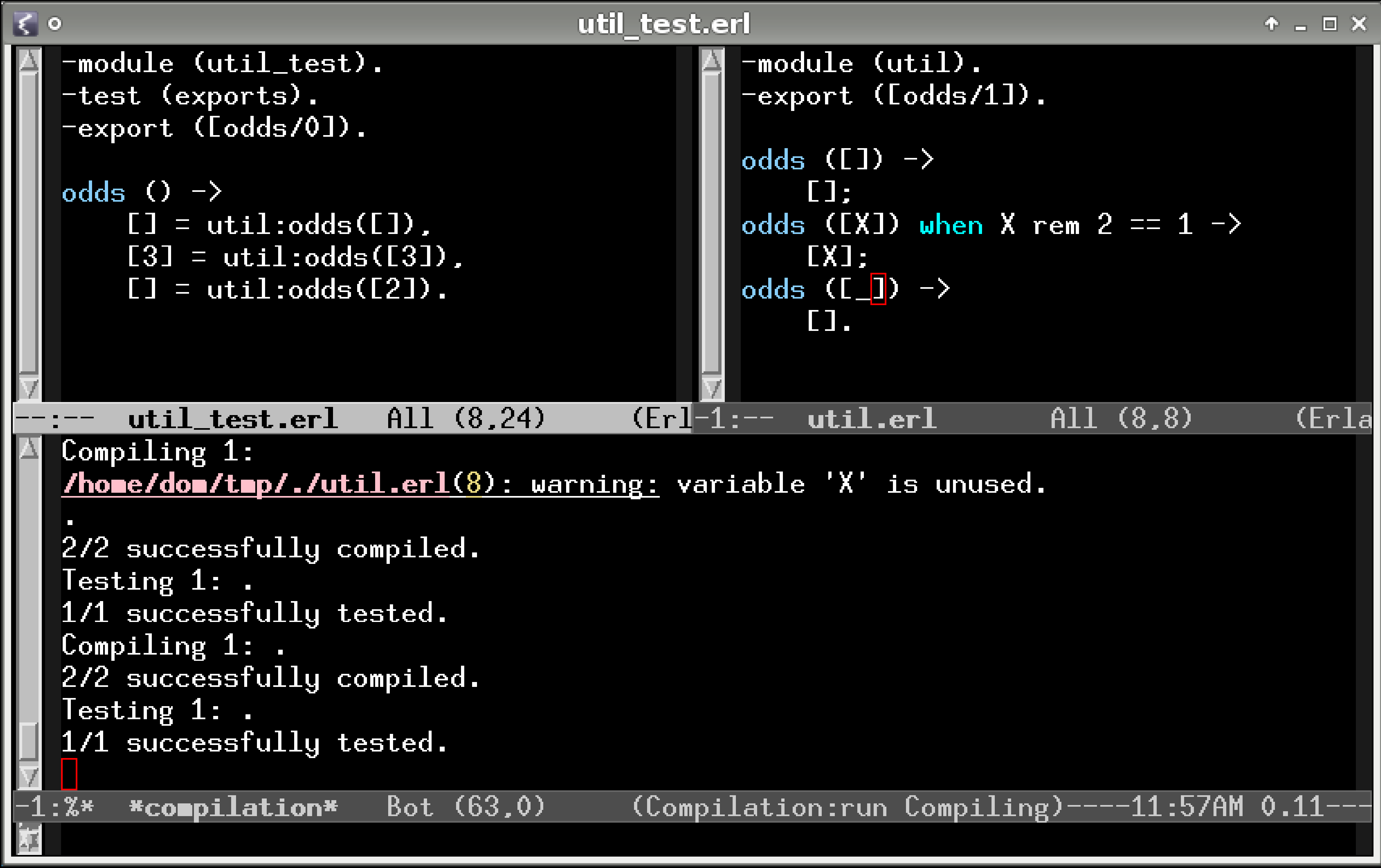
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [];
odds ([X]) ->
    [X].
```

```
--:--  util_test.erl   All (8,24)     (Erl-1:--  util.erl      All (7,8)      (Erla
```

```
Testing 1: .
1/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: {badmatch,[2]}
   Stack: [{util_test,odds,0}]
.
0/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (53,0)     (Compilation:run Compiling)----11:54AM 0.25---
Wrote /home/dom/tmp/util_test.erl
```



util_test.erl

```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]),
    [] = util:odds([2]).
```

```
--:--  util_test.erl   All (8,24)    (Erl
```

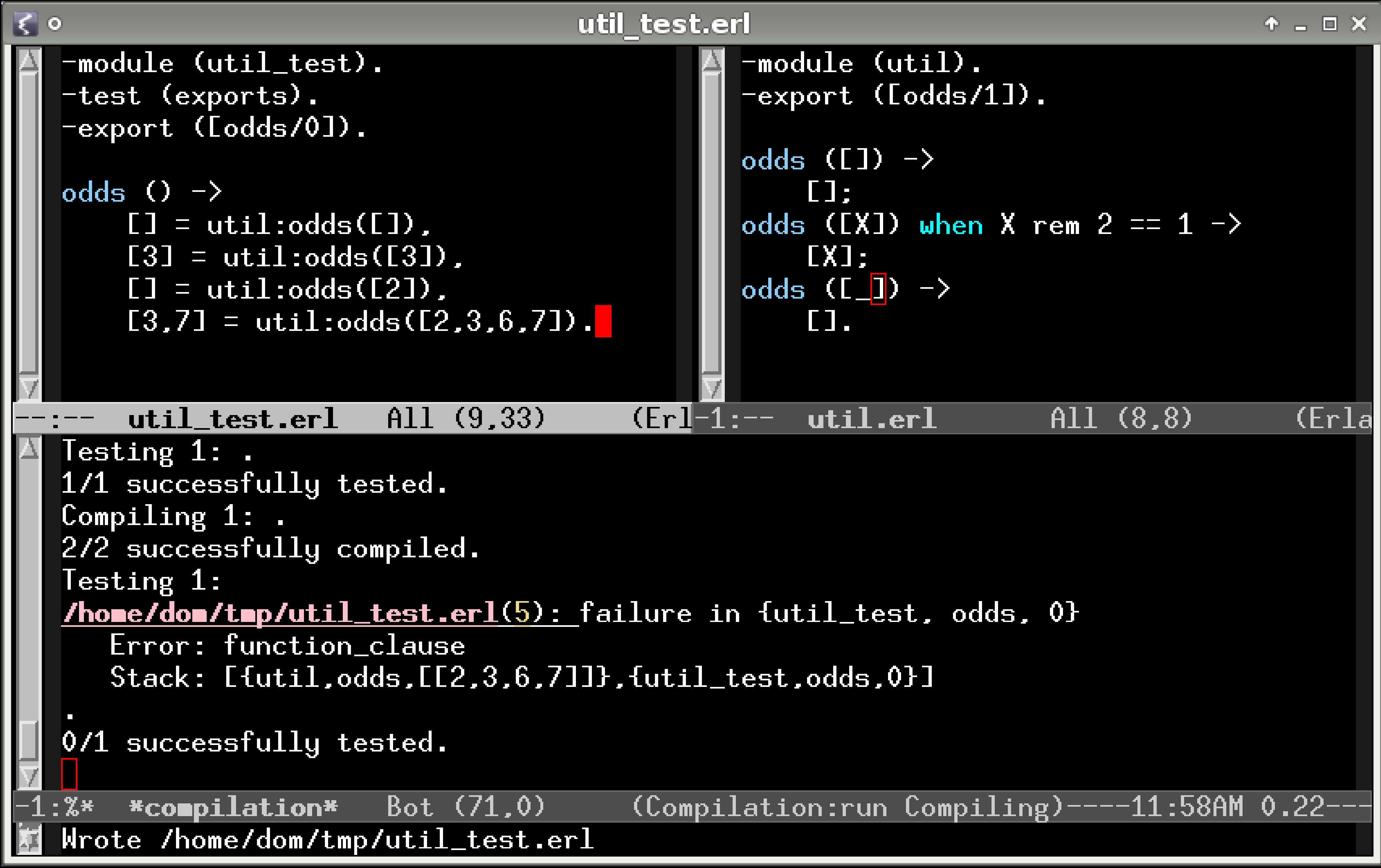
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [];
odds ([X]) when X rem 2 == 1 ->
    [X];
odds ([_]) ->
    [].
```

```
-1:--  util.erl       All (8,8)      (Erla
```

```
Compiling 1:
/home/dom/tmp/./util.erl(8): warning: variable 'X' is unused.
.
2/2 successfully compiled.
Testing 1: .
1/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1: .
1/1 successfully tested.

-1:%*  *compilation*   Bot (63,0)    (Compilation:run Compiling)----11:57AM 0.11---
```



util_test.erl

```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]),
    [] = util:odds([2]),
    [3,7] = util:odds([2,3,6,7]).
```

```
--:--  util_test.erl   All (9,33)     (Erl
```

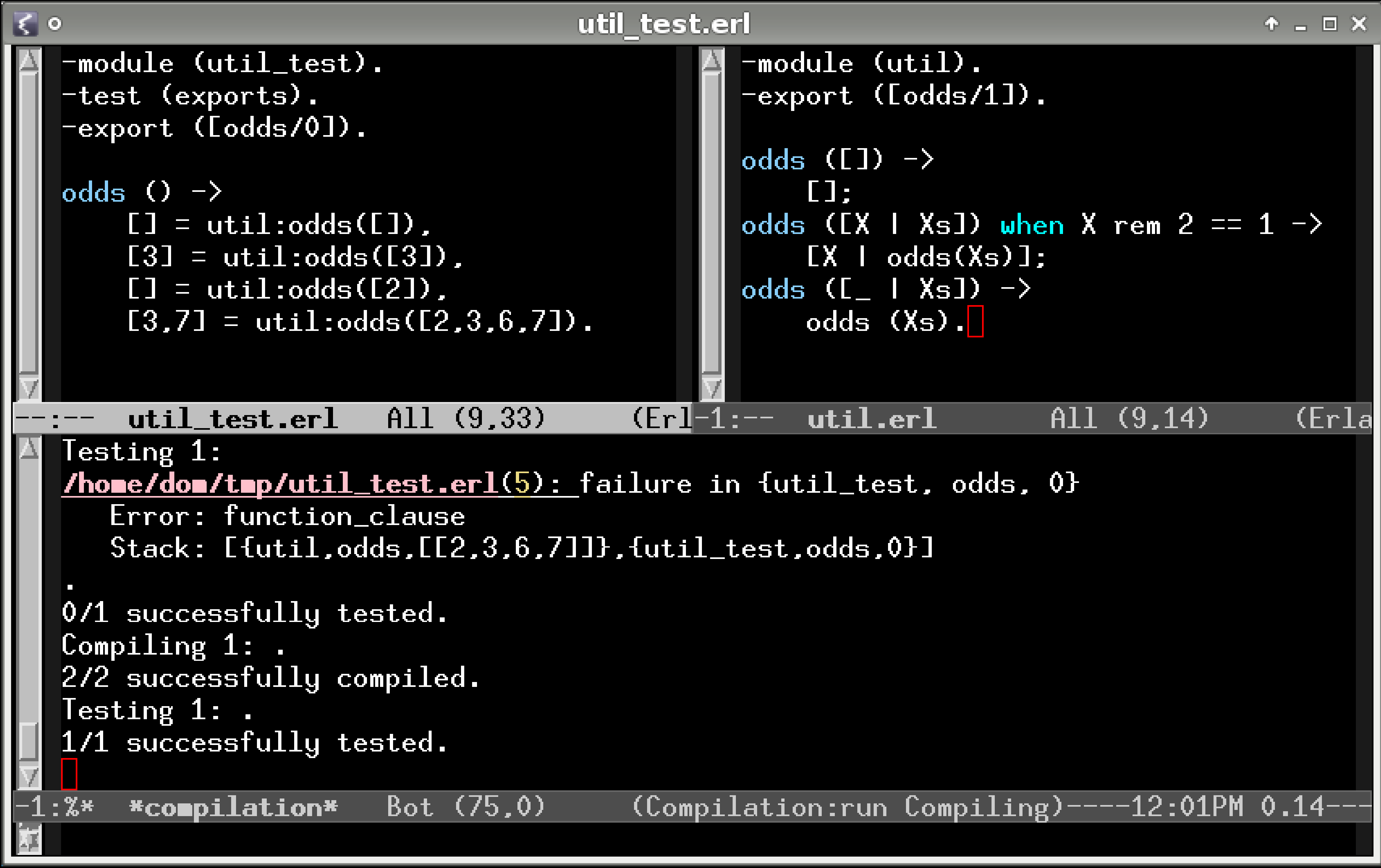
```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [];
odds ([X]) when X rem 2 == 1 ->
    [X];
odds ([_]) ->
    [].
```

```
-1:--  util.erl        All (8,8)      (Erla
```

```
Testing 1: .
1/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: function_clause
   Stack: [{util,odds,[[2,3,6,7]]},{util_test,odds,0}]
.
0/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (71,0)     (Compilation:run Compiling)----11:58AM 0.22---
Wrote /home/dom/tmp/util_test.erl
```


util_test.erl

```
-module (util_test).
-test (exports).
-export ([odds/0]).

odds () ->
    [] = util:odds([]),
    [3] = util:odds([3]),
    [] = util:odds([2]),
    [3,7] = util:odds([2,3,6,7]).
```

```
-module (util).
-export ([odds/1]).

odds ([]) ->
    [];
odds ([X | Xs]) when X rem 2 == 1 ->
    [X | odds(Xs)];
odds ([_ | Xs]) ->
    odds (Xs).
```

```
--:--  util_test.erl   All (9,33)     (Erl-1:--  util.erl       All (9,14)      (Erla
```

```
Testing 1:
/home/dom/tmp/util_test.erl(5): failure in {util_test, odds, 0}
   Error: function_clause
   Stack: [{util,odds,[[2,3,6,7]]},{util_test,odds,0}]
.
0/1 successfully tested.
Compiling 1: .
2/2 successfully compiled.
Testing 1: .
1/1 successfully tested.
```

```
-1:%*  *compilation*   Bot (75,0)     (Compilation:run Compiling)----12:01PM 0.14---
```

Lambda, the ultimate mock

- Inheritance, interfaces, delegation:
  - Test classes or subclasses
  - Mock objects
  - Full implementation
- Lambda:
  - On-the-fly creation
  - Minimal

CSP vs. threads

- Threads:
  - Impossible to “unit test”
  - Can wreak havoc in execution of other tests
- CSP:
  - Easy to unit test
  - Deterministic
  - Easy to fake collaborating processes

Smalltalk

- The original playground of Design patterns, XP and TDD
- Dynamic typing
- Closures
- Duck typing

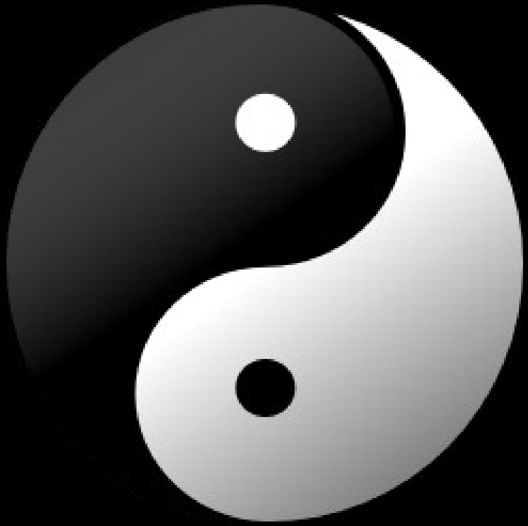
I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea.

The big idea is "messaging".

Alan Kay

The Tao of TDD

erlang^2



agile^2

Thank you !