readSpec

showing QuickCheck results to stake-holders

Laura M. Castro

University of A Coruña

Jun 10, 2015

EUC (tutorial)

Mmm. . . QuickWhat?

QuickCheck:

- *Short version*: a testing tool for Erlang

Mmm. . . QuickWhat?

QuickCheck:

- *Short version*: a testing tool for Erlang
- *Extended version*:
  - Powerful alternative to EUnit
  - You write properties/models instead of unit tests
  - Provided by Quviq, not part of OTP
  - Licenced (Quviq QuickCheck Mini runs w/out licence)
  - The original idea was first implemented in Haskell
  - A number of OS clones: PropEr, Triq

Mmm. . . QuickWhat?

QuickCheck:

- *Short version*: a testing tool for Erlang
- *Extended version*:
  - **Powerful alternative** to EUnit
  - You write properties/models instead of unit tests
  - Provided by Quviq, not part of OTP
  - Licenced (Quviq QuickCheck Mini runs w/out licence)
  - The original idea was first implemented in Haskell
  - A number of OS clones: PropEr, Triq

What do you mean by *powerful*?

- EUnit (*manual test design*):
  - As many test cases as **you can write**
  - Same test cases every time
- QuickCheck (*test property definition*):
  - As many test cases as **you have time to execute**
  - Slightly different data/test sequences every time

So, how does it look like?


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

So, how does it look like?


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

So, how does it look like?


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

So, how does it look like?


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

So, how does it look like?


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

And how does it work?


```
> eqc:quickcheck(simple_eqc:prop_simple()).
Starting Quviq QuickCheck version 1.35.0
Licence for University of A Coruna
reserved until {{2015,6,6},{18,44,22}}
.....................................................
.....................................................
....
OK, passed 100 tests
true
```

And how does it work?


```
> eqc:quickcheck(simple_eqc:prop_simple()).
Starting Quviq QuickCheck version 1.35.0
Licence for University of A Coruna
reserved until {{2015,6,6},{18,44,22}}
.....................................................
.....................................................
....
OK, passed 100 tests
true
```

Wait... what did I just test?

Wait... what did I just test?

Peek at samples of generated values:

```
> eqc_gen:sample(eqc_gen:list(eqc_gen:int())).
[-2,8,-2]                       [-7]
[]                              [-16,-10,-2,17,8]
[0,8,6]                         [11,2,0,-17,4,2]
[11,-11,-4,10]                  [-10]
[9,-6,-12]                      [9,12,-12,9,-20,-8,8]
ok
```

Wait... what did I just test?

Collect statistics from property execution:

```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      collect(lists:member(I, L),
      not lists:member(I, lists:delete(I, L))))).
```

Wait... what did I just test?

Collect statistics from property execution:

```
> eqc:quickcheck(simple_eqc:prop_simple()).
(...)
OK, passed 100 tests
87% false
13% true
```

Folk who do not speak Erlang

- This is good for technical people
- This may not be good enough for other stakeholders
- How do we communicate this to them?

Folk who do not speak Erlang

- This is good for technical people
- This may not be good enough for other stakeholders
- How do we **communicate** this to them?

What have others done?

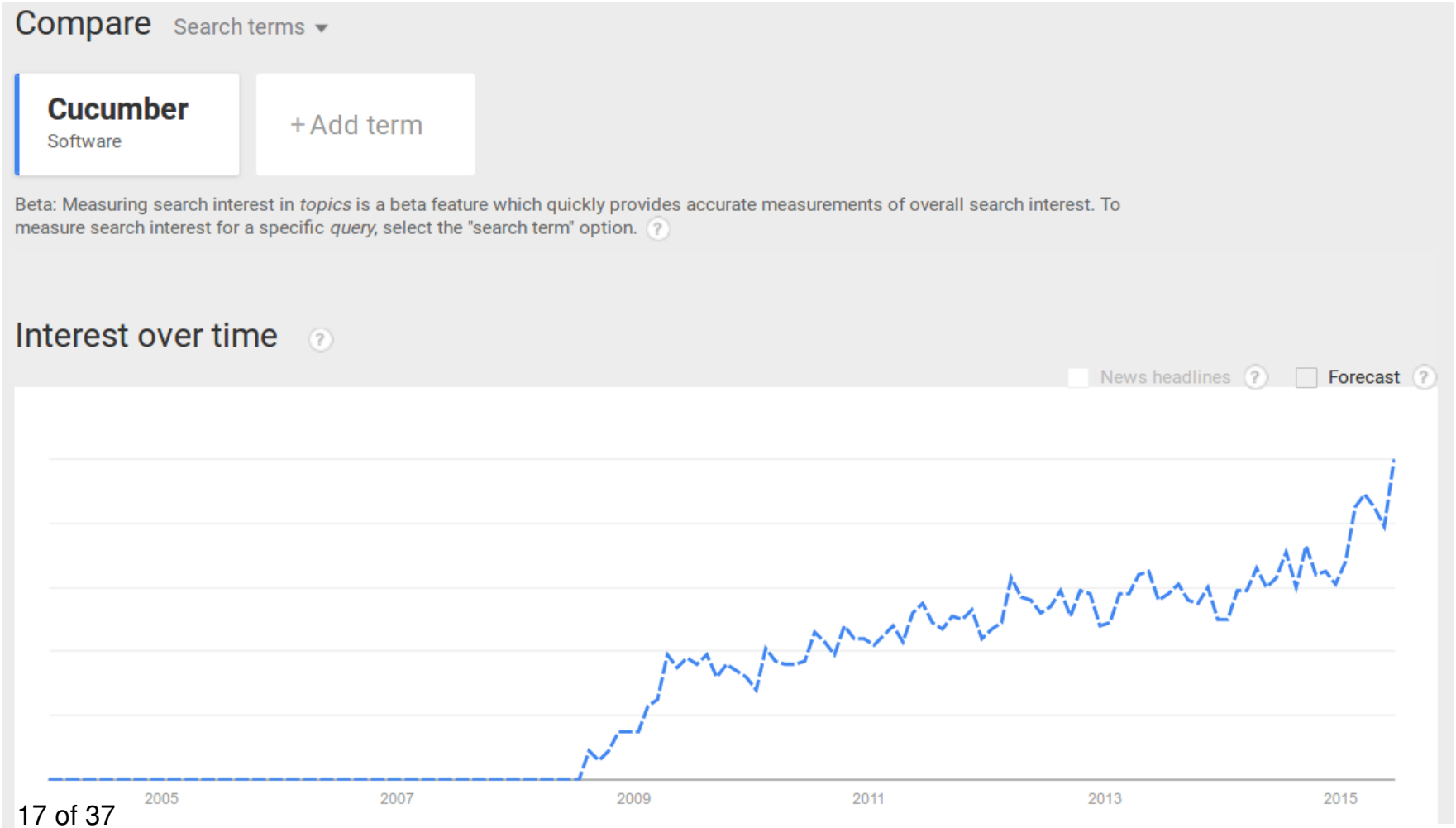
Simple, human collaboration

What have others done?

```
Feature: Deletion of element from list
  In order to operate lists
  As a user of the lists module
  I want to delete an element from a list

  Scenario: Delete integer from list of integers
    Given I have the integer 3
    And I have the list of integers [0,8,6]
    When I use the function lists:delete/2
    Then the resulting list should not contain 3
```

What have others done?



What have others done?

- Original Ruby implementation
- Implementations in many other languages
  - Java, JavaScript, Clojure, Lua,.NET, PHP, C++
- Several Erlang implementations
  - BravoDelta, CucumberI, KucumberI

How does it work?


```
Feature: Deletion of element from list
  In order to operate lists
  As a user of the lists module
  I want to delete an element from a list

  Scenario: Delete integer from list of integers
    Given I have the integer 3
    And I have the list of integers [0,8,6]
    When I use the function lists:delete/2
    Then the resulting list should not contain 3
```

How does it work?


```
-export([setup/0, teardown/0,
         given/3, 'when'/3, then/3]).

setup()    -> [].
teardown() -> ok.

given("I have the integer (\\d+)", State, [Num]) ->
  {ok, State ++ [erlang:list_to_integer(Num)]};
...
```

How does it work?

```
'when' ("I use the function (\\w+)
         from the module (\\w+)", State, [F, M]) ->
  {ok, erlang:apply(erlang:list_to_atom(M),
                    erlang:list_to_atom(F), State)}.
```

How does it work?

```
then("the resulting list
      should not contain (\\d+)", State, [Num]) ->
  Value = erlang:list_to_integer(Num),
  case not lists:member(Value, State) of
    true  -> {ok, State};
    false -> {failed, State}
  end.
```

How does it work?


```
1> kucumberl_cli:main([]).
Feature: Deletion of element from list
  Scenario: Delete integer from list of integers
    Given I have the integer 3                                    OK
    And I have the list of integers 0,8,6                         OK
    When I use the function delete from the module lists          OK
    Then the resulting list should not contain 3                  OK

1 Scenarios (0 failed, 1 passed)
4 Steps (0 failed, 4 passed, 0 skipped, 0 not implemented)
ok
```

So... Cucumber or QuickCheck?

- For testing, definitely QuickCheck!
- But to **communicate** in a friendly manner to **non-technical stakeholders**, why not follow Cucumber's example?

readSpec

- **Cucumber-like descriptions for the test cases your properties produce**
  - **you** write QC properties
  - **readSpec** translates them into Cucumber-like features
    - QC's coverage-based suite generation (`eqc_suite`)
- Available at:

  `https://github.com/prowessproject/readspec`

readSpec (ii)


```
prop_simple() ->
  ?FORALL(I, int(),
    ?FORALL(L, list(int()),
      not lists:member(I, lists:delete(I, L)))).
```

readSpec (ii)

```
> readspec:suite(simple_eqc, prop_simple).
Generating feature based test suite...
[line,simple_eqc,19,line,simple_eqc,20]
1 test cases generated.
Generating feature based test suite...
[line,simple_eqc,19,line,simple_eqc,20]
1 test cases generated.
...
ok
```

readSpec (ii)

File: *simple.feature*

```
FEATURE: simple
 Simple QuickCheck properties

SCENARIO: Deleting an integer from a list should
          result in a list that does not
          contain that integer.
GIVEN I have the integer 0
AND I have the list []
THEN lists:member(0, lists:delete(0, [])) IS FALSE.
```

readSpec (ii)

```
...

SCENARIO: Deleting an integer from a list should
          result in a list that does not
          contain that integer.
GIVEN I have the integer 6
AND I have the list [-1, 2, 13, 0, 5]
THEN lists:member(6, lists:delete(6, [-1,2,13,0,5]))
...
```

readSpec (iii)

- As in test execution, specific values change every time
- Representativity of the examples does not change (empty list, list with one element, etc.)
- Uses edoc comments if present in source

readSpec (iii)

- As in test execution, specific values change every time
- Representativity of the examples does not change (empty list, list with one element, etc.)
- Uses edoc comments if present in source
- Works for **counterexamples** as well

readSpec (iv)


```
> eqc:quickcheck(simple_eqc:prop_simple()).
.....................................................
....................Failed! After 69 tests.
15
[-13,-15,-2,-17,-20,15,15]
Shrinking xxxxxxxx.xx.xxxxxxxx(2 times)
15
[15,15]
false
```

readSpec (iv)

```
> C = eqc:counterexample().
[15,[15,15]]

> readspec:counterexample(simple_eqc,
                          prop_simple,[C]).
ok
```

readSpec (iv)

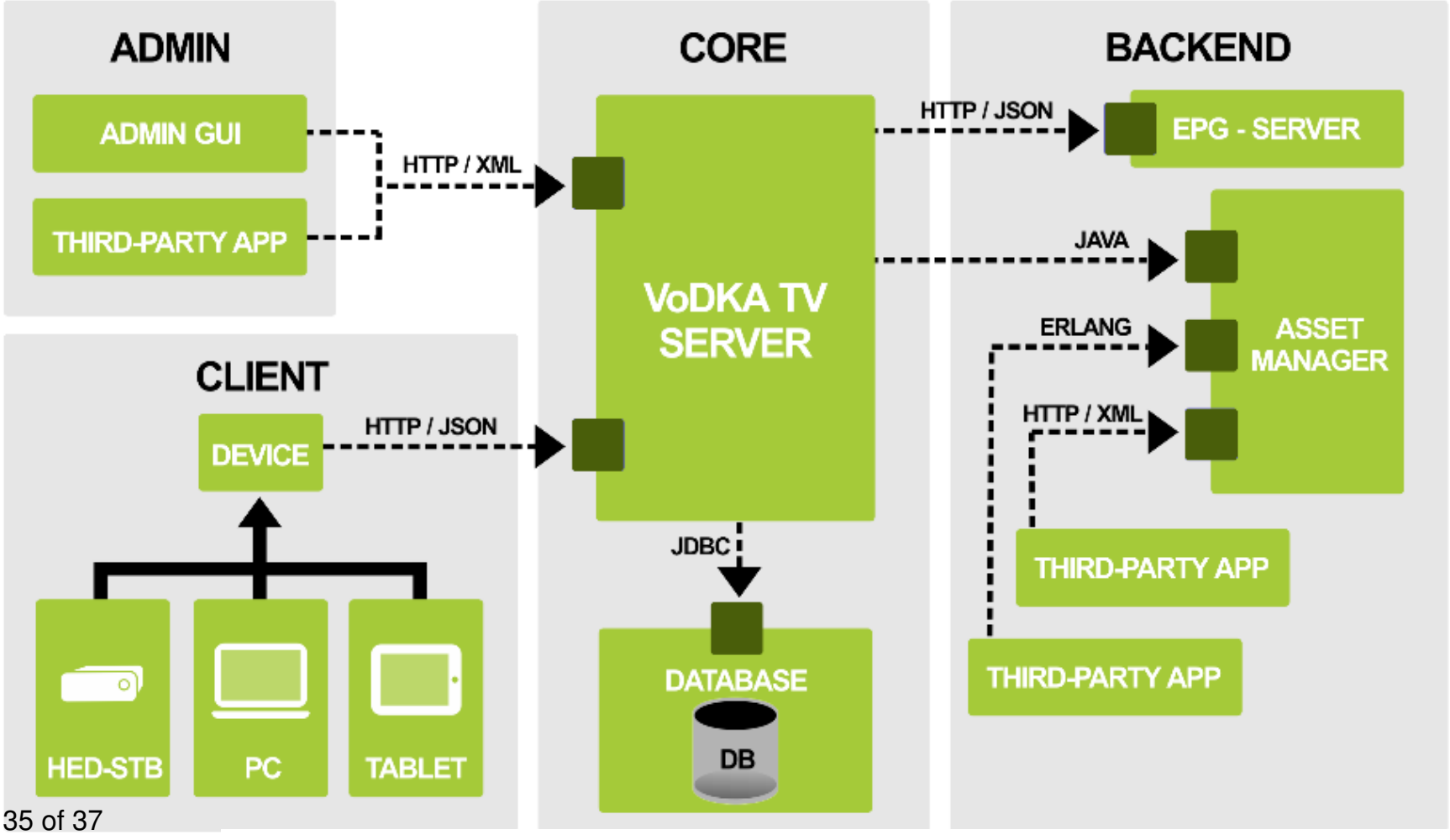
File: *prop_simple.counterexample.feature*

```
GIVEN I have the integer 15
AND I have the list [15, 15]
THEN lists:member(15,lists:delete(15,[15,15]))
IS TRUE
```

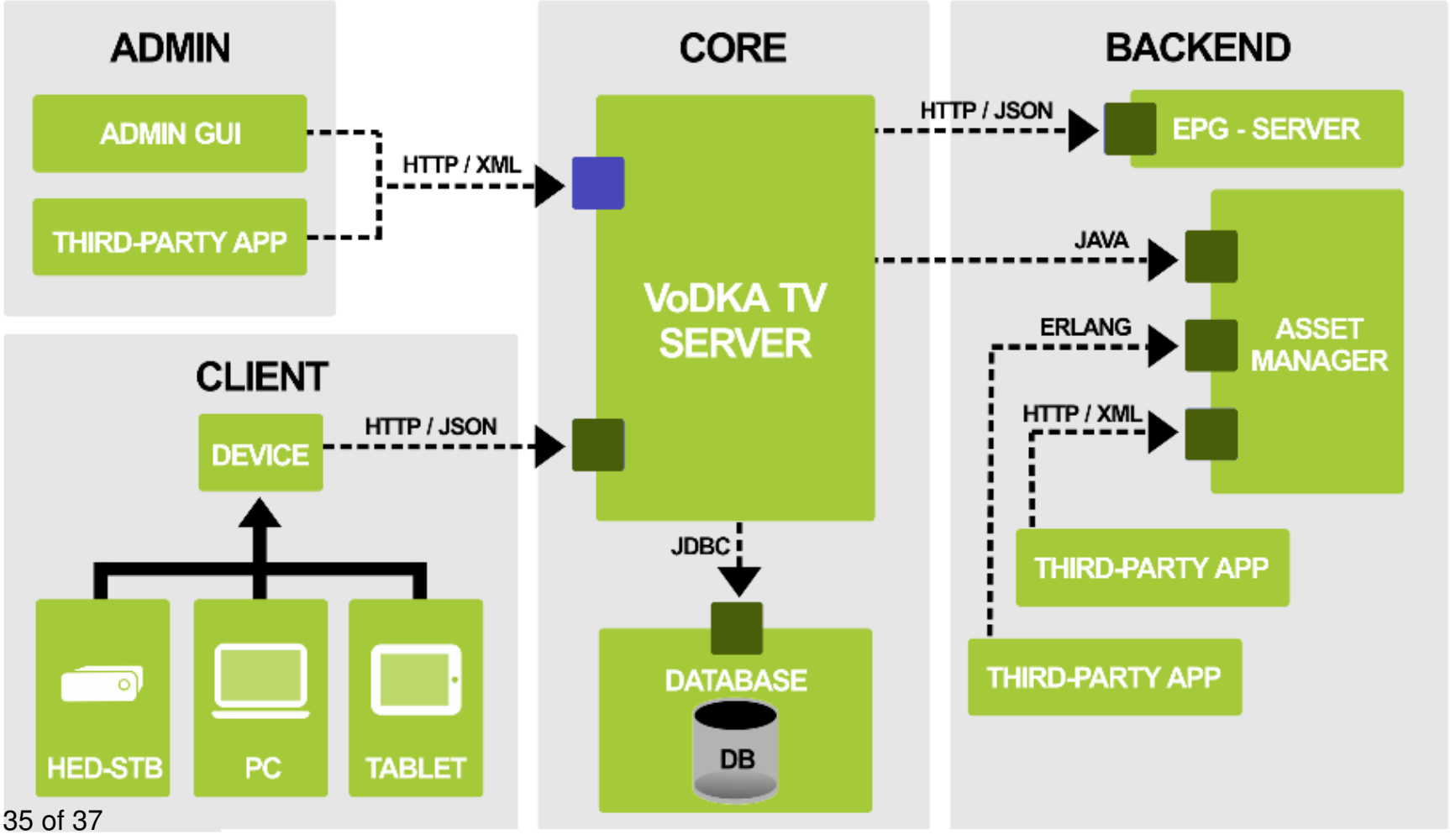
readSpec (v)

- As in test execution, specific values change every time
- Representativity of the examples does not change (empty list, list with one element, etc.)
- Uses edoc comments if present in source
- Works for counterexamples as well
- Works for **stateful models** as well

VoDKATV: pilot study



VoDKATV: pilot study



VoDKATV: properties

- What we found about properties written by VoDKATV developers:
  - Heavy use of MACROS and funs (unclear descriptions)
  - Heavy re-use of generators (slower suite generation)
  - Ambiguities in tuples of arguments, arguments as tuples

Conclusions

- **readSpec** offers a different way to present QC artifacts
- use it, report your issues, contribute!

Conclusions

- **readSpec** offers a different way to present QC artifacts
- use it, report your issues, contribute!
- and **thanks** for listening!! :-)