3rd UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

Sophia Antipolis, French Riviera
20-22 October
2015

Showing QuickCheck results to stakeholders

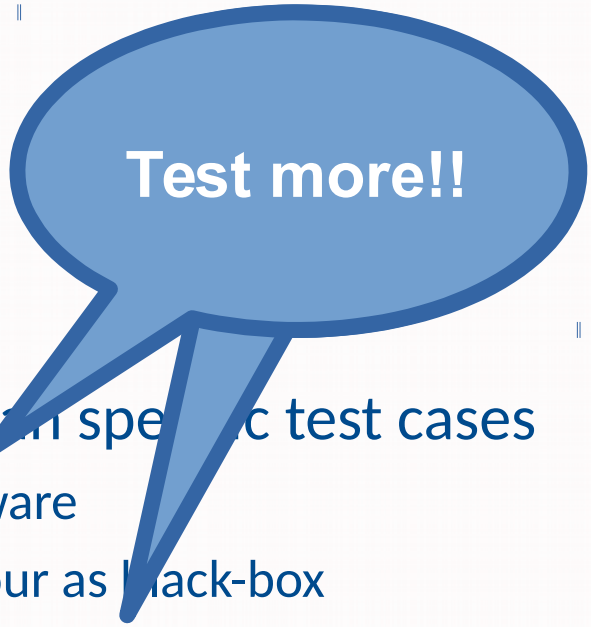
Presented by Laura M. Castro

PROWESS

What is QuickCheck?

- **Property-based testing** tool
  - **Powerful** upgrade from xUnit tools
  - Define **properties and models** rather than specific test cases
    - properties are well-suited for library-like software
    - stateful models allow to describe SUT behaviour as black-box
  - **Runs** many tests, **executes** and **evaluates** them
  - Presents **minimised counterexample**:
    - if property is found not to hold or
    - SUT exhibits behaviour that diverges from described by model

What is QuickCheck?



- **Property-based testing** tool
  - **Powerful** upgrade from xUnit tools
  - Define **properties and models** rather than specific test cases
    - properties are well-suited for library-like software
    - stateful models allow to describe SUT behaviour as black-box
  - **Runs** many tests, **executes** and **evaluates** them
  - Presents **minimised counterexample**:
    - if property is found not to hold or
    - SUT exhibits behaviour that diverges from described by model

What is the challenge in QuickCheck?

- Poses a **learning curve** to developers/testers
  - PBT artifacts (properties and models) are **more abstract** than specific test cases, thus **more difficult** to write
- It is equally **challenging** to other stakeholders
  - PBT artifacts are **not straightforward to understand**
  - Not only test results, but also *what is being tested* may be harder to grasp
  - Presenting statistics is slightly misleading
    - you do not run the same tests each time

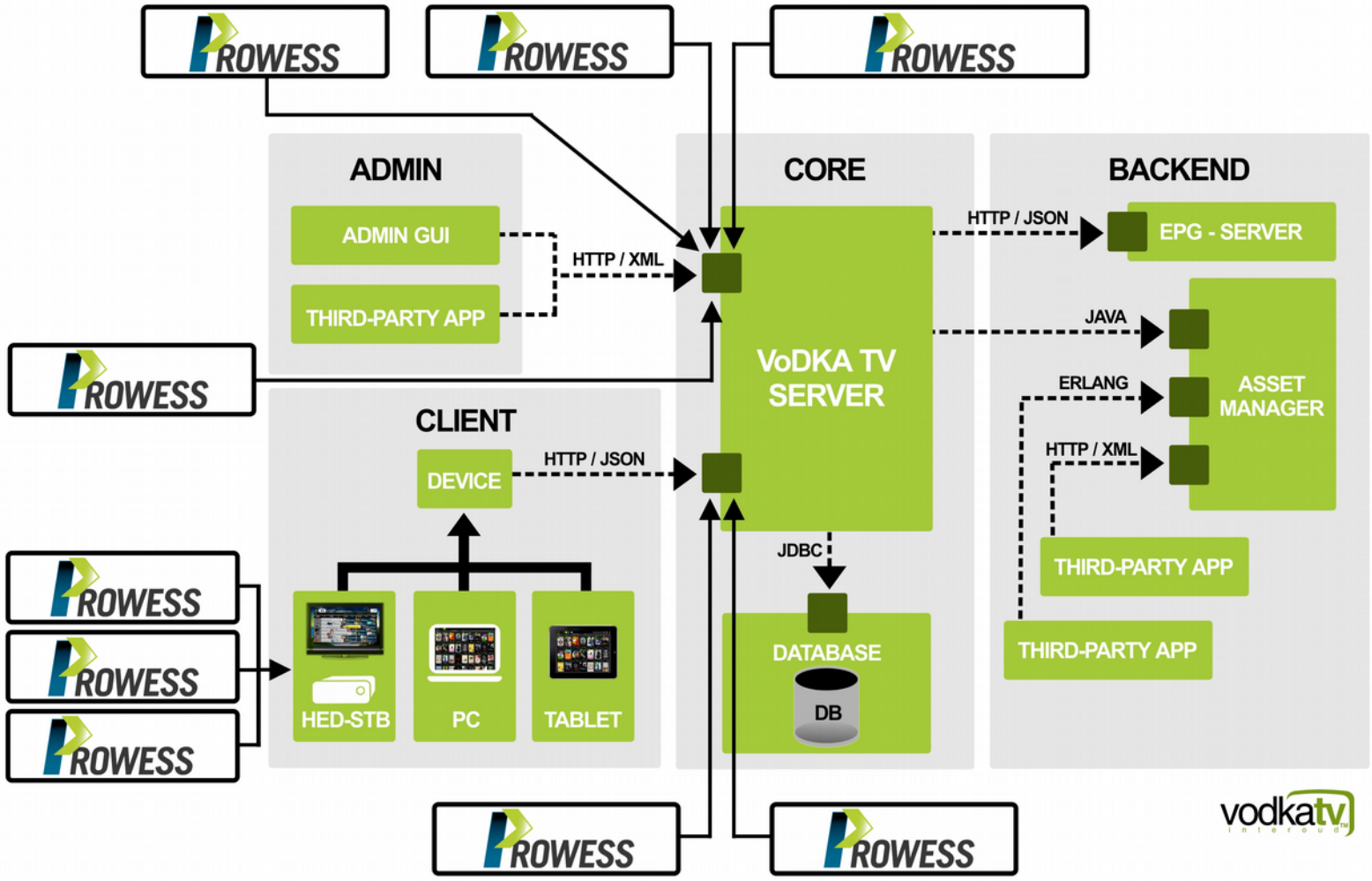
What is the challenge in QuickCheck?

- Poses a **learning curve** to developers/testers
  - PBT artifacts (properties and models) are **more abstract** than specific test cases, thus **more difficult** to write
- It is equally **challenging** to other stakeholders
  - PBT artifacts are **not straightforward to understand**
  - Not only test results, but also *what is being tested* may be harder to grasp
  - Presenting statistics is slightly misleading
    - you do not run the same tests each time

Addressing the challenge: PROWESS

- EU FP7 ICT project (2012-2015)
- Total budget 4.4M€ (3.3M€ EU contribution)
- 9 partners (3 SMEs, 1 research centre, 5 universities), 3 countries (Spain, Sweden, United Kingdom)
- Specific work package devoted to **dealing with the complexity of creating and understanding PBT artifacts**, featuring:
  - Alternative representation of test results
  - Alternative edition (graphical) of test models
  - Alternative representation (using semi-natural language) of test artifacts

PROWESS industrial pilot: VoDKATV



moreBugs

- **Goal**: reveal as ***many bugs*** present in SUT as possible
- **Why**: the random component of PBT may ***hit the same bug once and again*** when there are others yet unrevealed; bug reports in consultancy-like work are expected to inform of as many defects as possible
- **How**: test execution is ***automatically*** steered, so that instead of stopping on the first specification violation, ***new tests are executed that do not include the interactions that already failed*** in a previous run

moreBugs

Normal QC output:

```
..Failed! After 3 tests.

erlang:whereis(b) -> undefined
erlang:whereis(c) -> undefined
reg_eqc:spawn() -> <0.291.0>
reg_eqc:spawn() -> <0.292.0>
erlang:whereis(b) -> undefined
erlang:register(a, <0.292.0>) -> true
reg_eqc:spawn() -> <0.293.0>
erlang:register(b, <0.292.0>) -> !!! {exception, {'EXIT', {badarg, ...}}}

Shrinking xxxxx.xx...xx.xx(5 times)

reg_eqc:spawn() -> <0.319.0>
erlang:register(a, <0.319.0>) -> true
erlang:register(a, <0.319.0>) -> !!! {exception, {'EXIT', {badarg, ...}}}
```

moreBugs

Normal QC output with `moreBugs`:

```
Bug 1:
  V1 = reg_eqc:spawn(),
  erlang:register(b, V1) | V3 = reg_eqc:spawn(),
  erlang:register(b, V3)

Bug 2:
  V1 = reg_eqc:spawn(),
  erlang:register(a, V1),
  erlang:unregister(a),
  erlang:unregister(a)

Bug 3:
  V1 = reg_eqc:spawn(),
  erlang:register(a, V1) | erlang:register(b, V1)
```

Graphical edition

- **Goal**: make QC test ***models easier to manipulate***
- **Why**: QC stateful models require the developer to ***implement*** a number of callbacks (***pre/post conditions***, ***test state update***, etc.) which is challenging for new adopters, especially if not familiar with Erlang
- **How**: mouse-based ***manipulation of QC models using the browser***, supporting the most important edition operations (state & state transition addition/removal, transition weight edition, failure visualization, etc.)

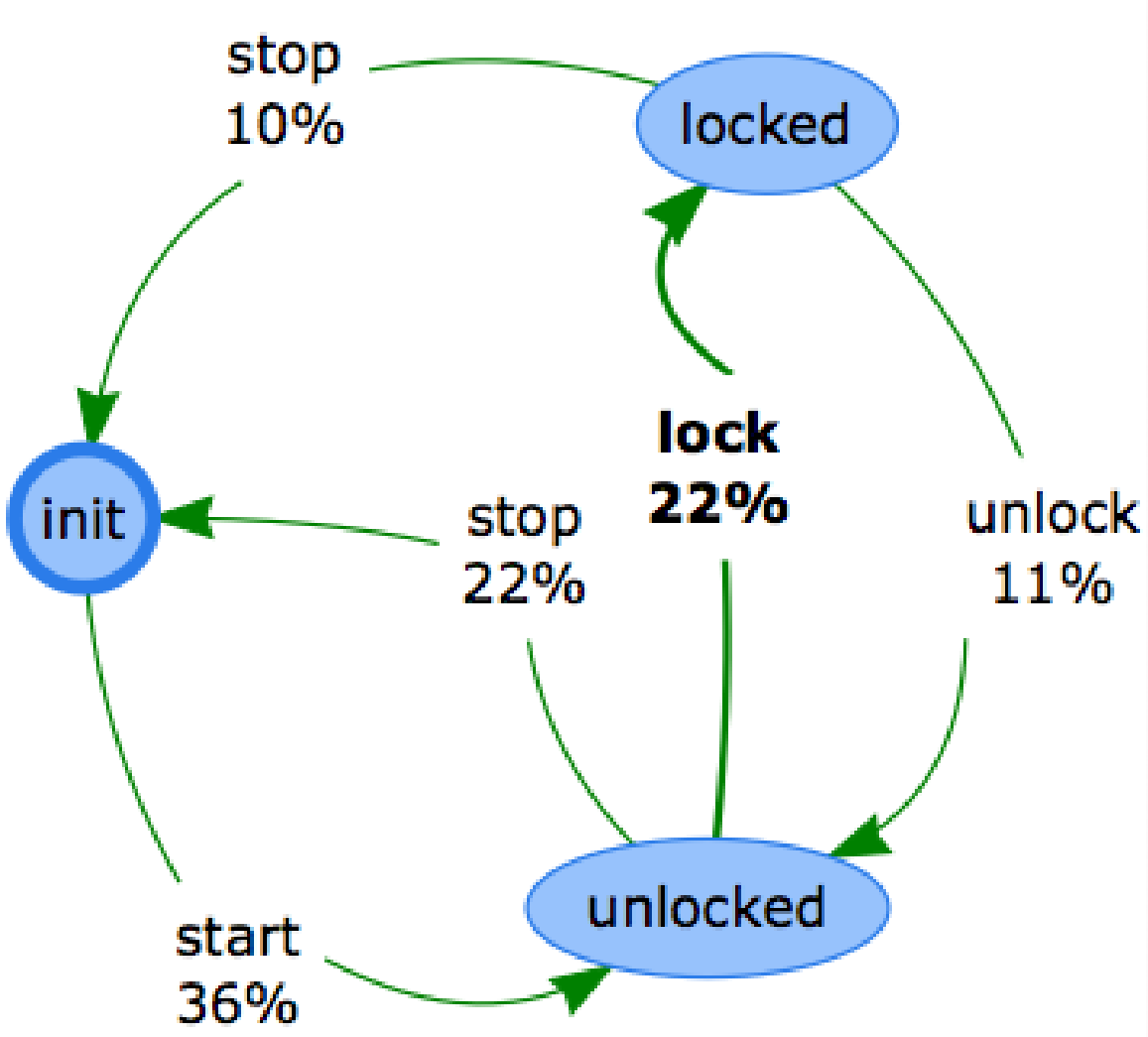
Graphical edition

Sample QC stateful model:

```
-record(state,{started}).
initial_state()     -> #state{started = false}.
start_pre(S)        -> not S#state.started.
start_args(_S)      -> [].
start_next(S,_,[])  -> S#state{started = true}.
stop_pre(S)         -> S#state.started.
stop_args(_S)       -> [].
stop_next(S,_,[])   -> S#state{started = false}.
lock_pre(S)         -> S#state.started andalso not S#state.locked.
lock_args_S)        -> [].
lock_next(S,_,[])   -> S#state{locked=true}.
unlock_pre(S)       -> S#state.started andalso S#state.locked.
unlock_args(_S)     -> [].
unlock_next(S,_,[]) -> S#state{locked=false}.
```

Graphical edition

Sample editable QC stateful model:



readSpec

- **Goal**: make PBT artifacts ***readable*** for stakeholders
- **Why**: stakeholders need to assess what is being tested, but cannot read PBT artifacts and/or understand what they mean in terms of what is being tested with them
- **How**: takes PBT ***artifacts as input***, produces semi-natural English ***text as output***
  - For properties, `readSpec` produces Cucumber-compliant text
  - For stateful models, `readSpec` produces own text explanation

readSpec

Sample input:

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

Sample output:

```
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

validation showed most value when complexity increases

Sample input:

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

Sample output:

```
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]))
```

To take home

- **Property-based testing** keeps proving itself a very valuable strategy in terms of **efficiency** and **effectiveness**
- Property-based testing imposes a **steeper learning curve** not only for developers, but for all stakeholders
- **PROWESS project** has studied several angles to these issues, and produced tools that **can help**
  - We have seen here three of them, but check out our project website `www.prowess-project.eu` and our project GitHub page `github.com/prowessproject` for more
  - … and a few other talks during this conference!

Thanks!

Questions?

Contact me: lcastro@udc.es