WebDriver:
Controlling your Web Browser

Erlang User Conference 2013

Hans Svensson, Quviq AB

hans.svensson@quviq.com

First, a confession...

I have a confession to make...

I have built a web system!

In PHP!

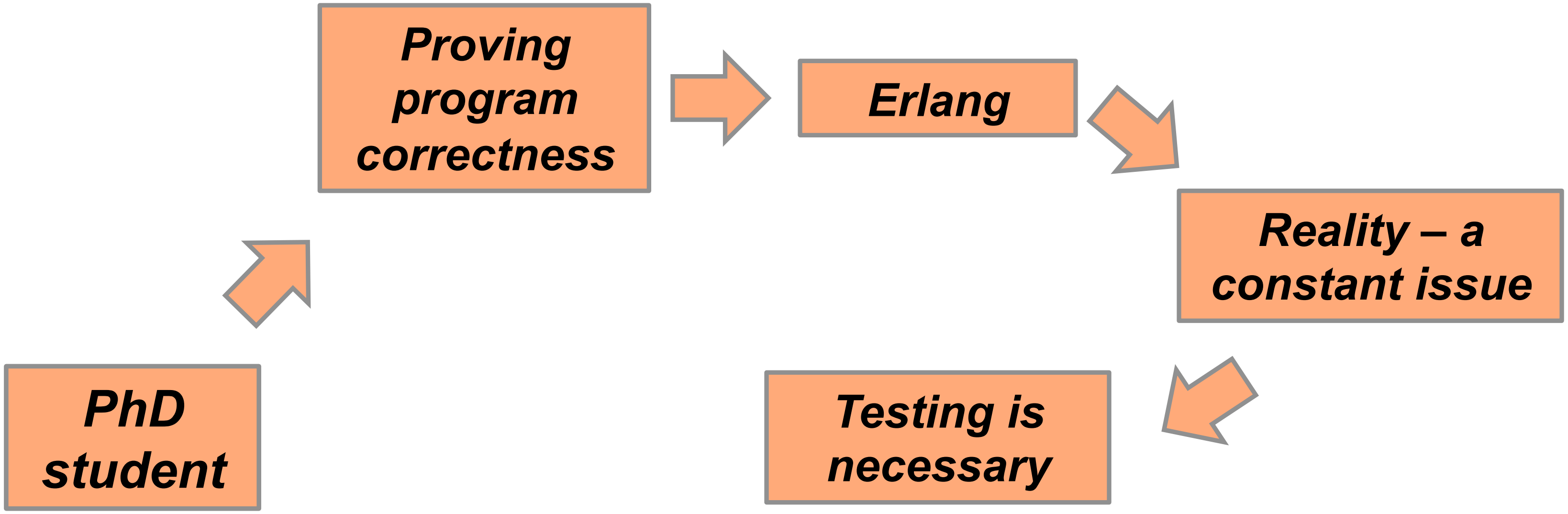
... and it was painfully mundane to test

It is all forgotten and forgiven... It was back in 2003!

First DEMO

DEMO

A bit of history



PhD student

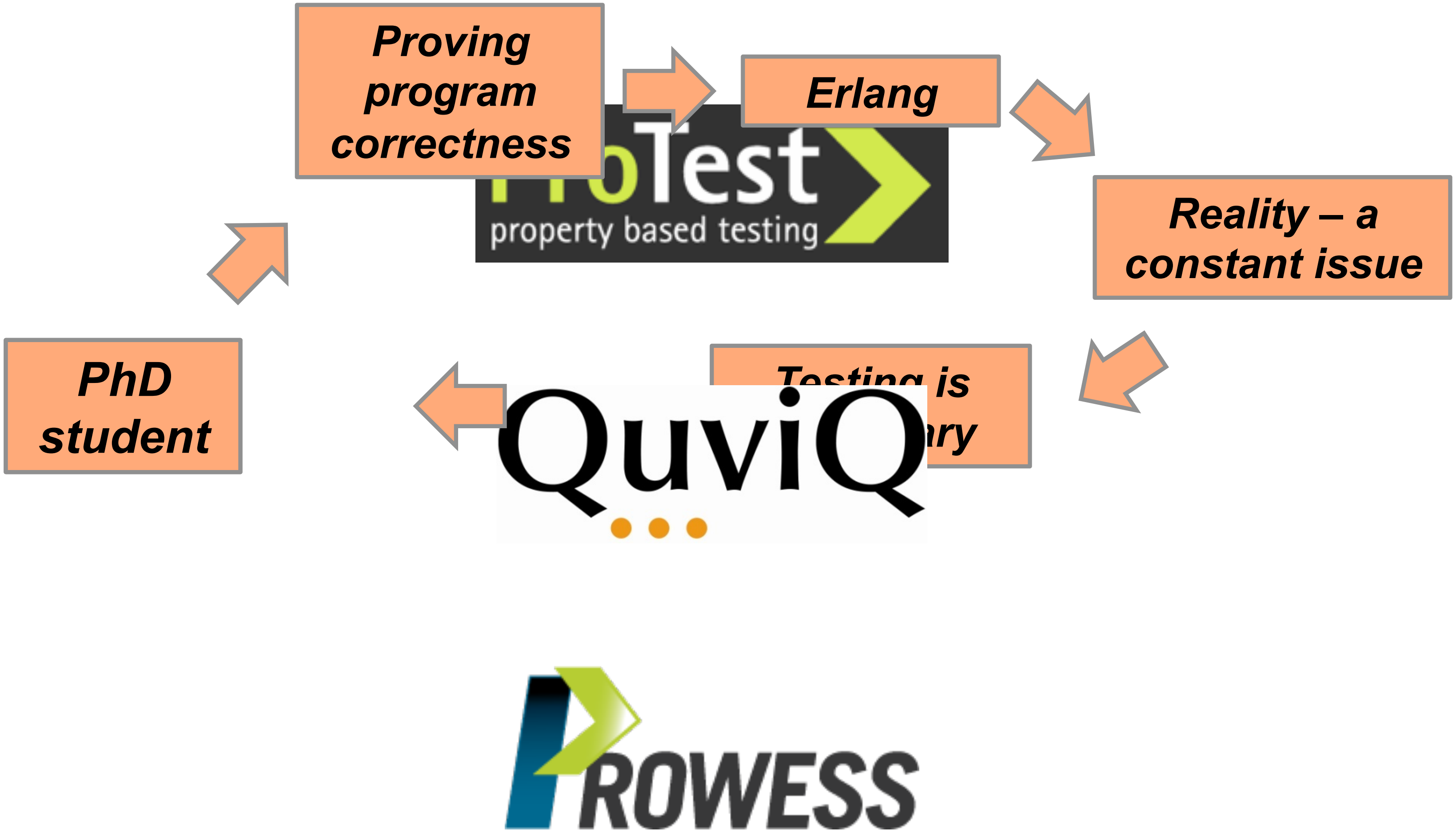
Proving program correctness

Erlang

Reality – a constant issue

Testing is necessary

A bit of history



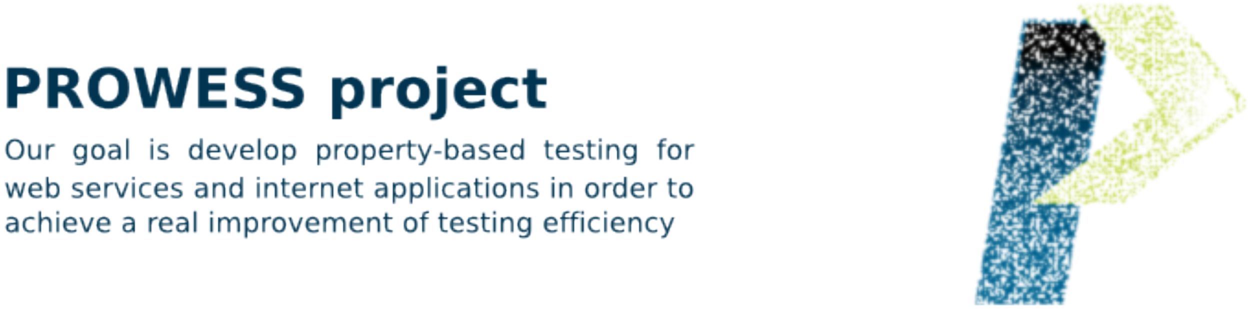
PhD student

Proving program correctness

Erlang

Reality – a constant issue

Testing is
ary



PROWESS project

Our goal is develop property-based testing for web services and internet applications in order to achieve a real improvement of testing efficiency

- We like to write our properties in QuickCheck
- How can we control ‘a browser’ from Erlang?
- Doing it all from scratch seems hard, unnecessary, stupid, ...

Selenium

> *“**Selenium automates browsers**. That's it. What you do with that power is entirely up to you. Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) also be automated as well.”*

Selenium

- Basically you can create and run scripts
- Supported by a wide range of browsers and operating systems
  - Run tests in: Chrome, IE, Opera, Firefox, and partial support also for other browsers.
  - Runs on: Windows, OS X, Linux, Solaris, and others.
  - Script recording using Selenium IDE (Firefox plugin).
- Language support for: C#, Java, Python, Ruby, and partial support for Perl and PHP.

- Widely used to create Unit-tests and regression testing suites for web services.

Selenium 2 - WebDriver

- In version 2, Selenium introduced the WebDriver API
- Via WebDriver it is possible to drive the browser natively
- The browser can be local or remote – possible to use a grid test
- It is a compact Object Oriented API

- Supports Chrome, Firefox, HtmlUnit, Opera, IE, and IPhone and Android
- Languages implementing driver: C#, Java, Python, and Ruby.

WebDriver Wire Protocol

- What if your preferred language is not in the list?
- All WebDriver drivers communicates via HTTP using the WebDriver Wire Protocol
- It is a RESTful web service using JSON over HTTP

- The Wire Protocol consists of ~80 different commands controlling different aspects such as:
  - Windows/Tabs – open, close, resize, set_position, maximize, ...
  - Page elements – find, find relative, click, send_keys, submit, ...
  - Navigation – back, forward, refresh, set_url, ...
  - Timeouts – script timeout, page load timeout, ...
  - Cookies – set, get, delete, ...

WebDriver in Erlang

- Implementation consists of
  - webdrv_cap – handle web browser capabilities
  - webdrv_wire – purely functional implementation of the wire protocol
  - webdrv_session – wrapper module for WebDriver sessions
  - json – JSON library, written by Tony Garnock-Jones http://github.com/tonyg/erlang-rfc4627

webdrv_wire

```
77> webdrv_wire:get_status(
     #webdrv_opts{url = "http://localhost:9515/"}).
{ok,<<>>,{obj,[{"build",{obj,[{"version",<<"alpha">>}]}},
               {"os",{obj,[{"arch",<<"x86">>},
                           {"name",<<"Linux">>},
                           {"version",<<"3.2.0-43-generic">>}]}}]}}
78> webdrv_wire:start_session(
     #webdrv_opts{url = "http://localhost:9515/"},
     webdrv_
{ok,<<"8189
    {obj,[{"ac
          {"appl
          {"browserConnectionEnabled",false},
          {"browserName",<<"chrome">>},
                  ... ]}}
79> webdrv_wire:set_url(
     #webdrv_opts{url = "http://localhost:9515/",
                  session_id = "81892cb363f177b6b7a90d40e646c924" },
     <<"http://google.se">>).
{ok,<<"81892cb363f177b6b7a90d40e646c924">>,null}
```

Ouch! That is a lot of typing...

webdrv_session

```
93> webdrv_session:start_session(
      test, "http://localhost:9515/", webdrv_cap:default_chrome()).
{ok,<0.3045.0>}
94> webdrv_session:set_url(test, "http://google.se").
ok
95> {ok, E} = webdrv_session:find_element(test, "name", "q").
{ok,"0.4722895685117692:1"}
96> webdrv_session:send_value(
      test, E, "Erlang User Conference 2013").
ok
97> webdrv_session:submit(test, E).
ok
98> webdrv_session:get_page_title(test).
{ok,"erlang user conference 2013 - Sök på Google"}
```

It is on GitHub

https://github.com/Quviq/webdrv

- Makefile
- make release – creates webdrv-1.0, copy it to <erlang>/lib to install

- MIT license

QuickCheck

- Originally developed by John Hughes and Koen Claessen
- Property based testing
- Controlled randomness
- Shrinking
- QuickCheck libraries in many languages

Quviq - QuickCheck

- Written in Erlang
- Many extensions:
  - Statem: testing using finite state machines
  - PULSE: finding race conditions
  - Symbolic test case generation
  - Mocking (C and Erlang)
  - Testing C-code
  - Composable components

Testing WebDriver with QuickCheck

- We can combine QuickCheck and WebDriver
- Insert data generators
- Use QuickCheck (finite) state machine to
  - store system state
  - generate valid sequences of WebDriver calls
- Rely on shrinking to find minimal counterexample
- Play back a counterexample

- The tests used: Chrome, Firefox, and HtmlUnit
- The tests used: Selenium Server 2.33.0, and Chromedriver 2 (r202239)

What did we find?

- How hard can it be to follow a specification?

Answer: Very hard!

Problem I – Selenium 302 redirect

Problem:

When creating a new session (doing a POST request), Selenium answers with a 302 redirection.

From specification:

Returns: A 303 See Other redirect to /session/:sessionId, where :sessionId is the ID of the newly created session.

Prob. II – Chromedriver case sensitive headers

Problem:

Certain parameters in a HTTP POST request are ignored by Chromedriver. If they are passed with the expected capitalization it works.

RFC 2616:

Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive.

Prob. III – httpc_request is broken in OTP/R15

Problem:

When getting a **303 See Other**, from Chromedriver, {autoredirect, true} fails to automatically redirect.

RFC 2616, 10.3.4:

The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.

Fixed in R16, but more problems arose...

Prob. IV – Maximizing a window twice hangs

Problem:

When using Chromedriver, maximizing a window twice hangs the WebDriver session.

DEMO

Prob. IV – Maximizing a window twice hangs

```
set_window_maximize({Session, Window}) ->
  webdrv_session:set_window_maximize(Session, Window).

set_window_maximize_args(S) ->[session_and_window(S)].

set_window_maximize_pre(S, [{Sess, _Window}]) ->
  #session{ browser = B, driver = D, maximized = M }
    = get_session(S, Sess),
  %% Bug in chromedriver used by selenium.
  not M orelse B /= chrome orelse D /= selenium.

set_window_maximize_next(S, _V, [{Sess, _Win}]) ->
  Win = get_curr_window(S, Sess),
  set_window(S, Sess, Win#window{ size = undefined,
                                  position = undefined,
                                  maximized = true}).

set_window_maximize_post(_S, [{_Sess, _Win}], Res) ->
  eq(Res, ok).
```

Prob. IV – Maximizing a window twice hangs

Problem:

When using Chromedriver, maximizing a window twice hangs the WebDriver session.

Fixed in Chromedriver 2

Prob. V – Chromedriver 2 screenshot fails

Problem:

If you open a second window/tab, and then focus back on the first window, grabbing a screenshot of that window fails. If we close the second window (in the background) a screenshot can be taken!

The power of QuickCheck:

No (sane) tester would write this (unit) test case. (In fact the first non-shrunk test case involved three windows and several re-focusing operations...)

Other problems and quirks

- Timeouts/2 not implemented - Fixed
- Get window size/position fails if multiple tabs (=windows) are open.
- Not reporting errors consistently (get_window_size/ postition + others)
- Inconsistent error reporting 'UnknownError'/'NoSuchElement'

The way forward

- Write traditional QuickCheck models of web service API
- Write utility libraries on top of webdrv_session
- Integrate with Selenium IDE – converting test cases into Erlang EUnit tests
- Automatically derive a state machine specification from a set of EUnit tests

Using recorded tests in Erlang

- Test cases can be recorded in Selenium IDE (only available for Firefox)
- Reasonably straightforward to translate a test case into a sequence of webdrv_session calls
- Wrapping it as an EUnit test or a simple QuickCheck property

Deriving state machines from EUnit tests

- Earlier research have looked into automatic generation/derivation of state machines
- Input is a set of tests, output is a state machine describing the set of tests
- Could be a cheap way to get started with a QuickCheck model of a web service

Try it! Break it! Fix it!

- Try using it
- If it breaks, fix it!

https://github.com/Quviq/webdrv