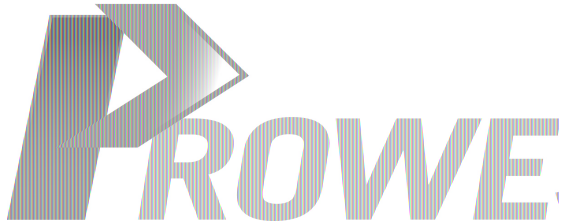
Property-based testing for Web Services

Simon Thompson
University of Kent

Thomas Arts
Quviq

Introduction

EU PROWESS project



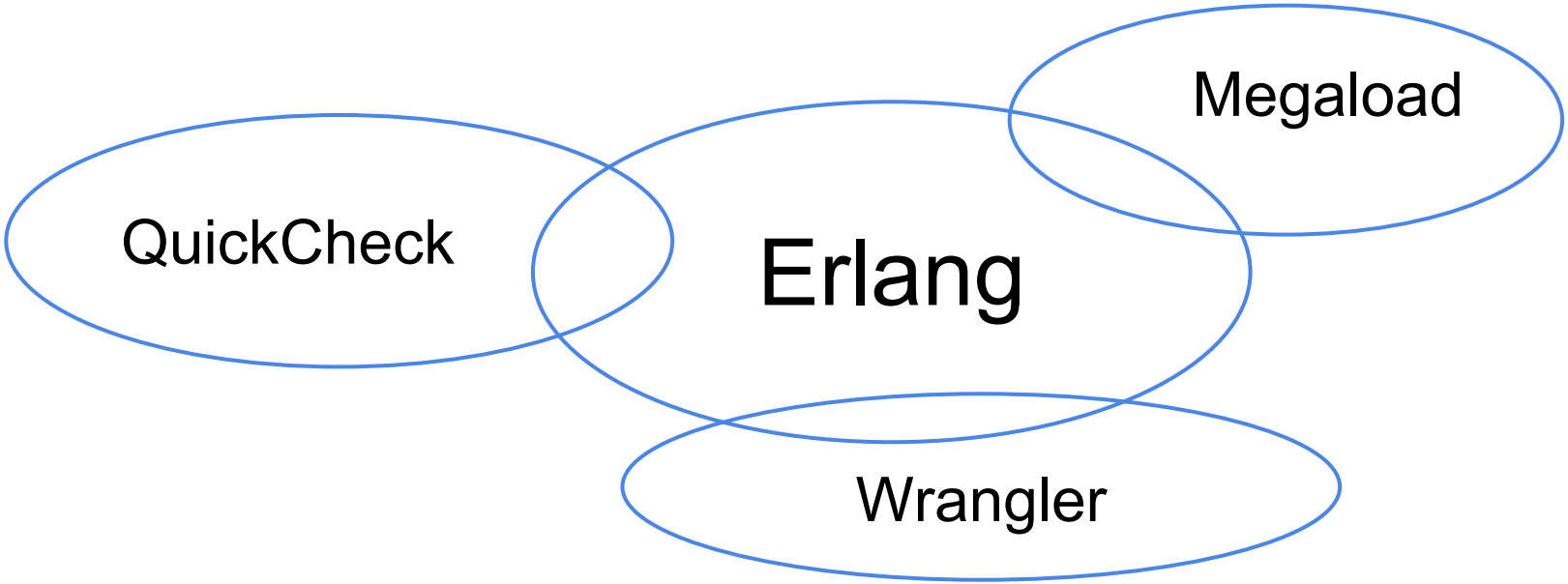
Aims to improve testing, particularly for web services, through uptake and use of property-based testing (PBT).

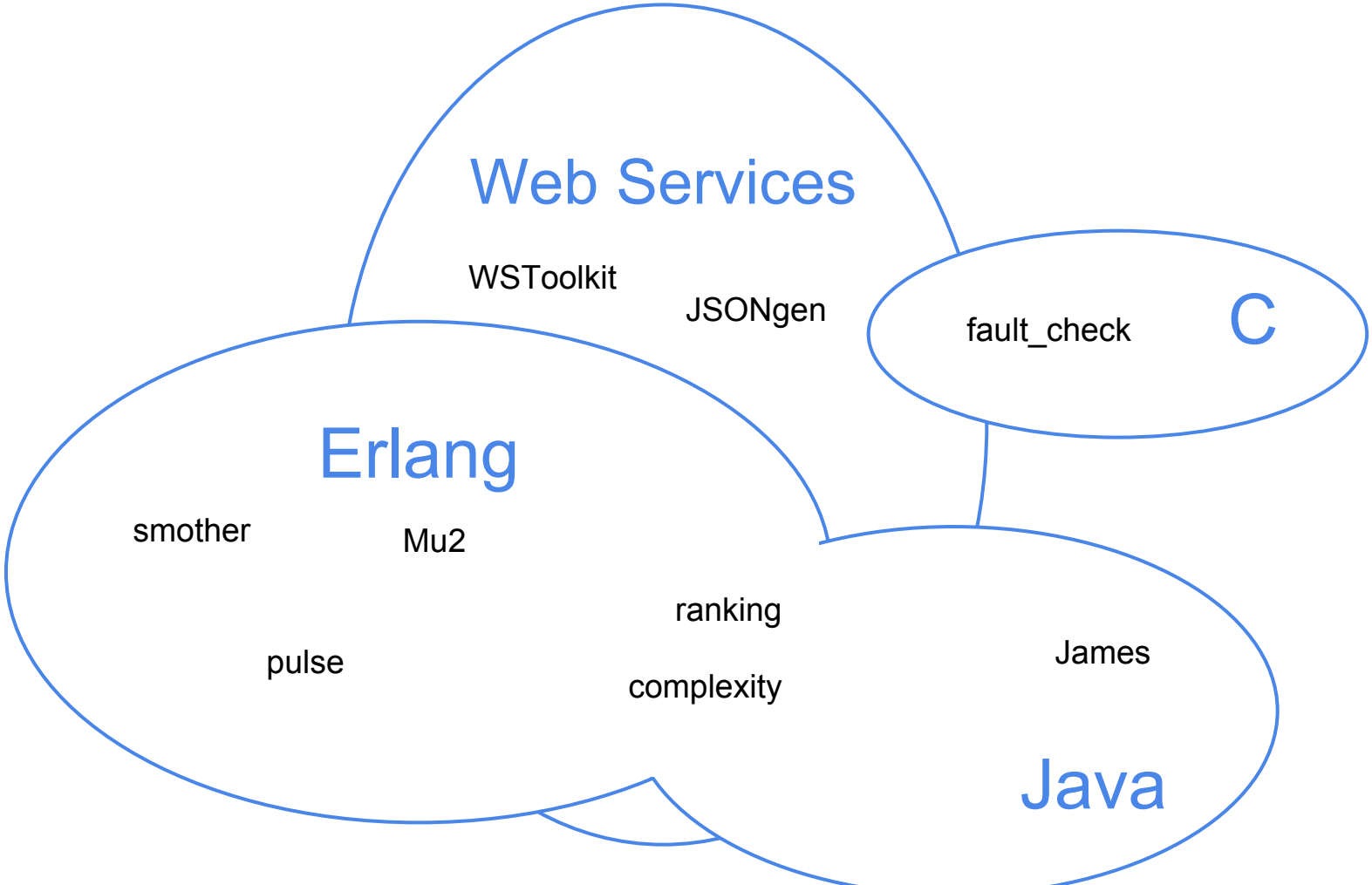
The *QuickCheck* tool for PBT can be used to test web services as well as systems built in Erlang, Java, C, …

… but system models and properties are written in Erlang.

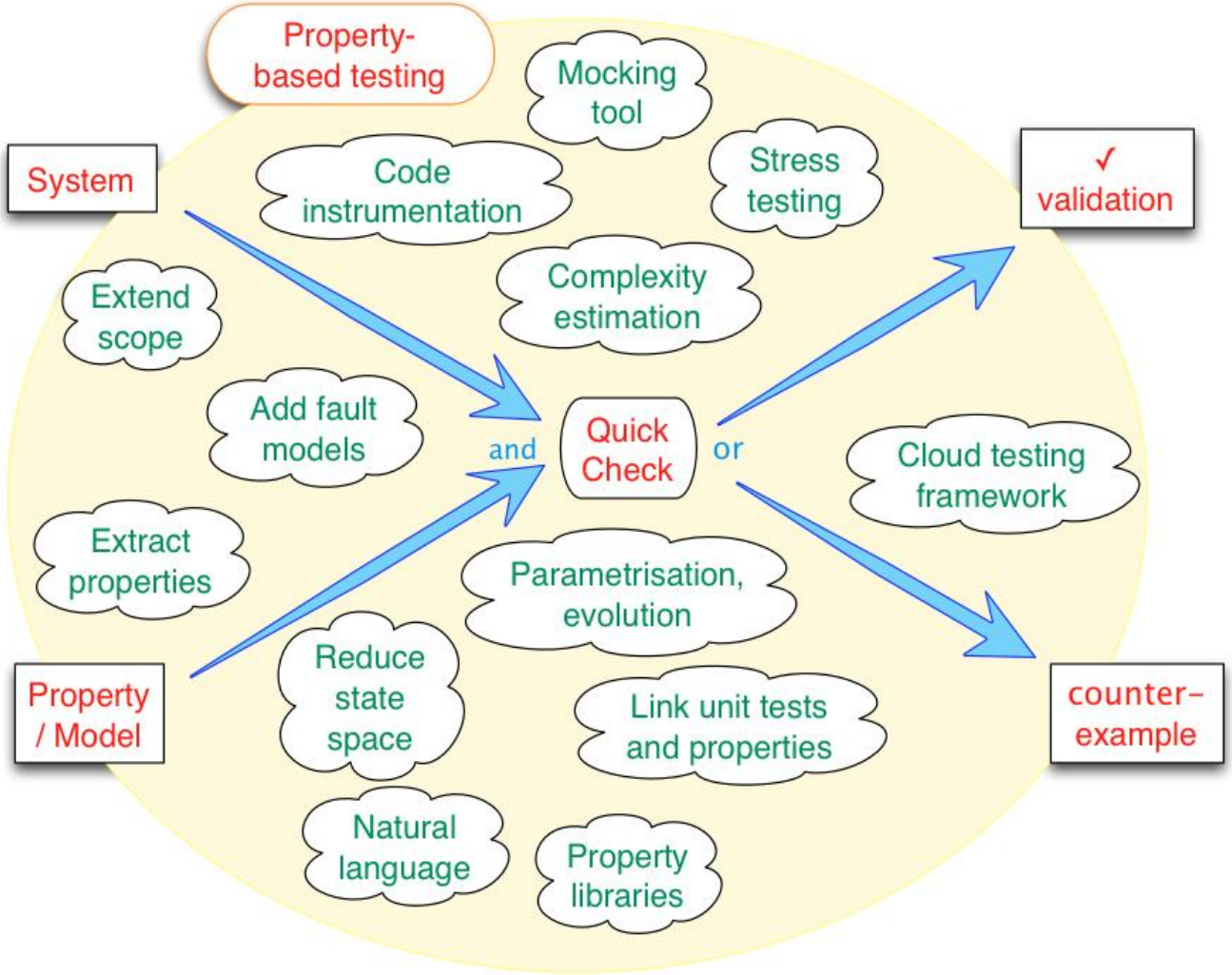
University of Sheffield	UK
University of Kent	UK
Chalmers University of Technology	Sweden
Universidad Politécnica de Madrid	Spain
University of A Coruña	Spain
Quviq AB	Sweden
Erlang Solutions Ltd	UK
Interoud Innovation S.L.	Spain
SP Technical Research Institute of Sweden	Sweden

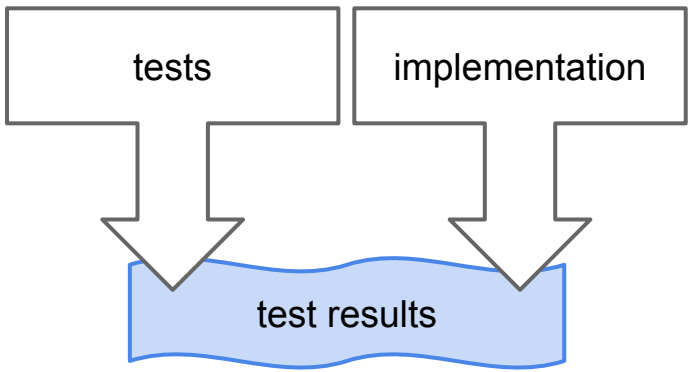
Erlang ecosystem

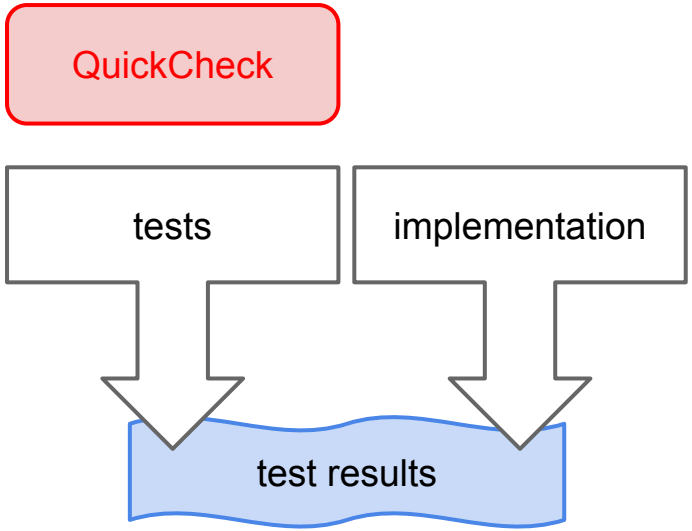


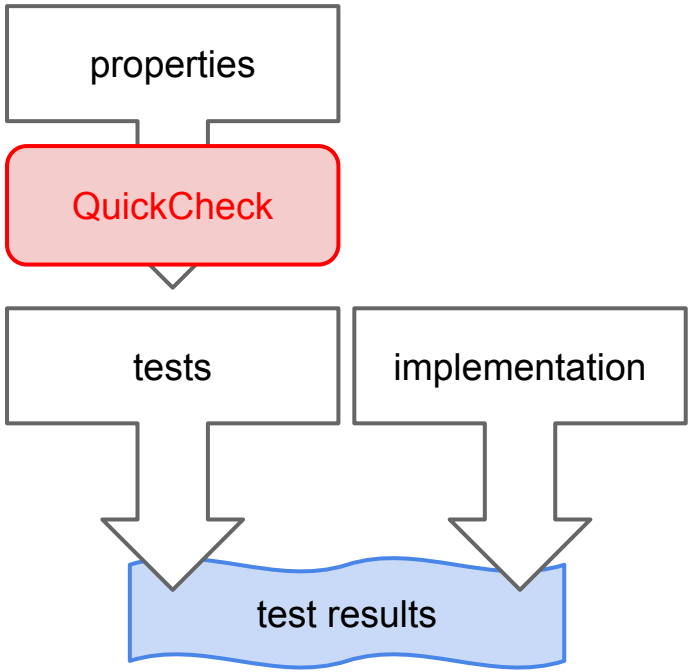

Web Services
WSToolkit
JSONgen
fault_check
C
Erlang
smother
Mu2
ranking
pulse
James
complexity
Java

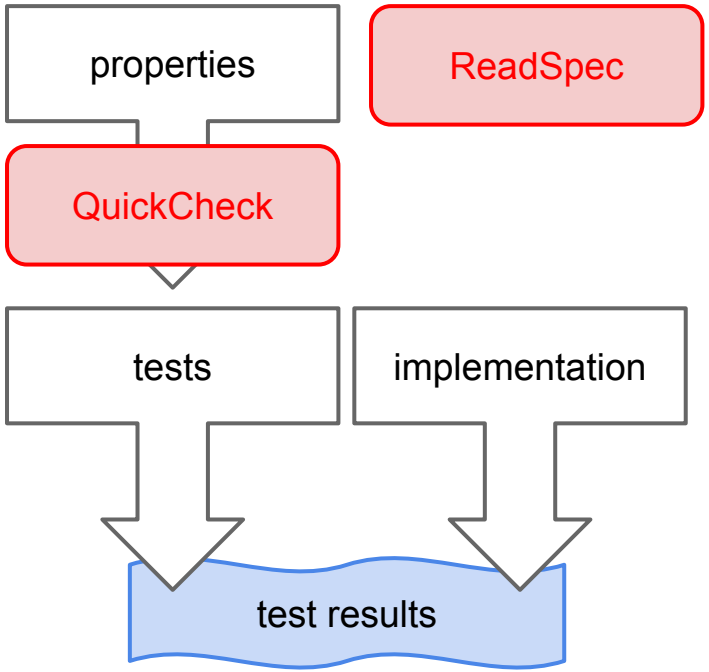
Overview

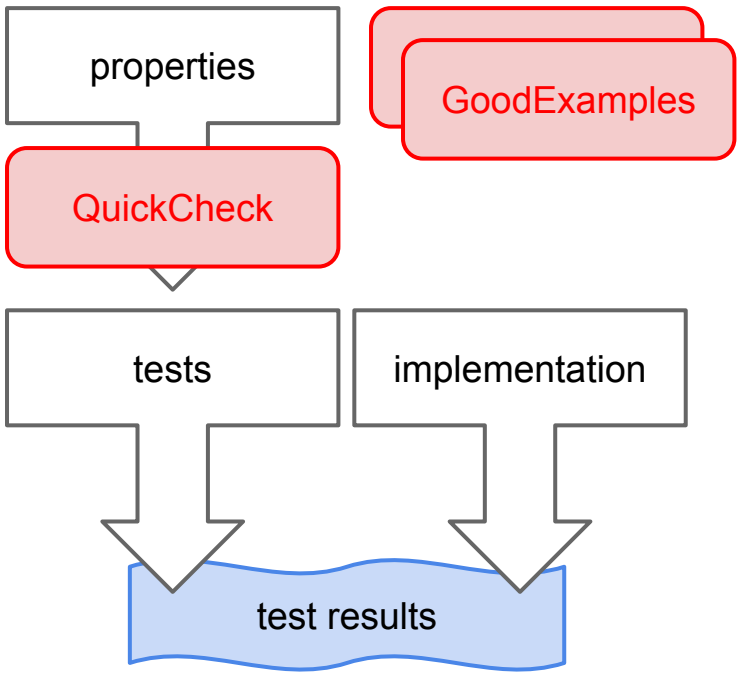

Property-based testing
Mocking tool
System
Code instrumentation
Stress testing
✓ validation
Extend scope
Complexity estimation
Add fault models
and
Quick Check
or
Cloud testing framework
Extract properties
Parametrisation, evolution
Property / Model
Reduce state space
Link unit tests and properties
counter-example
Natural language
Property libraries

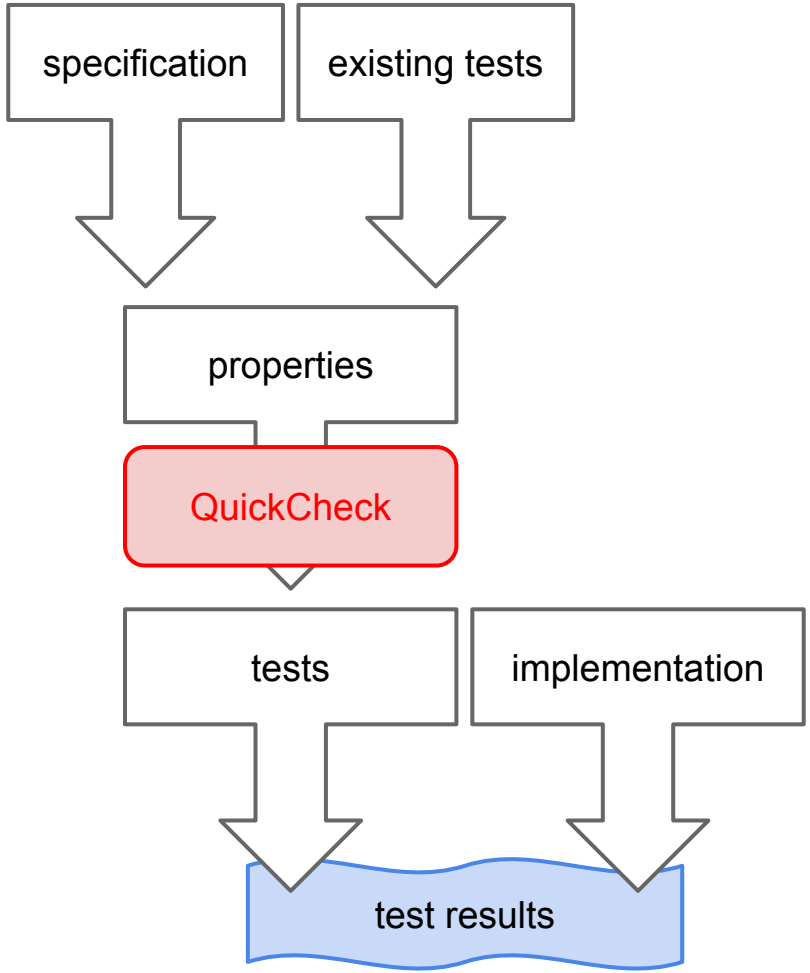

tests
implementation
test results

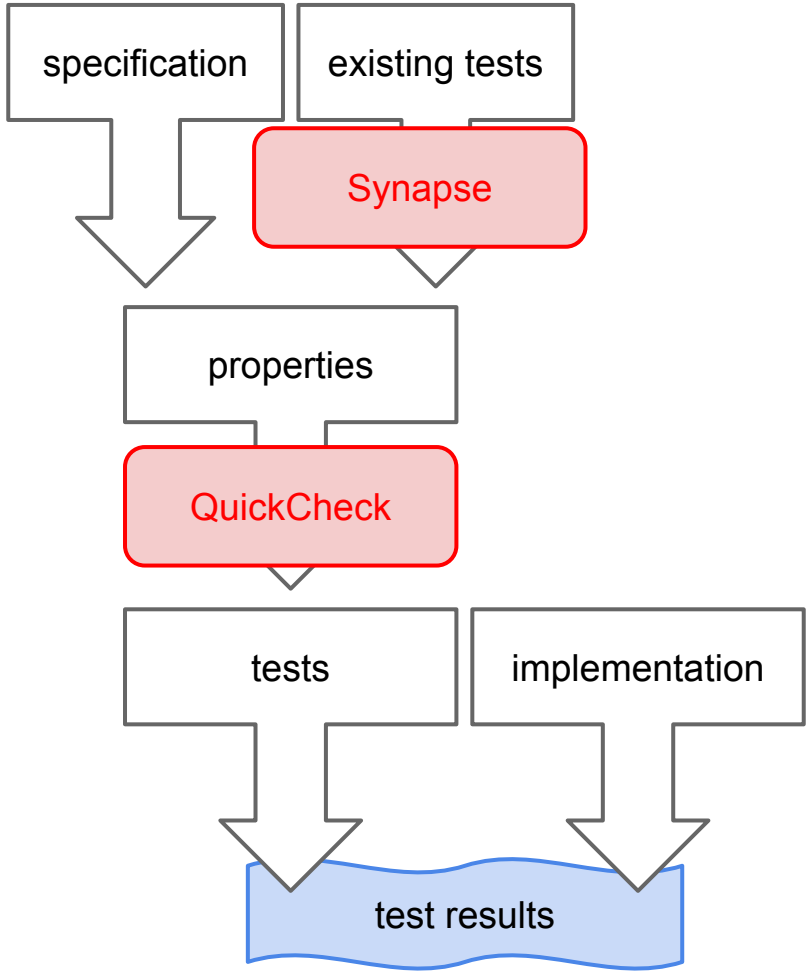

QuickCheck
tests
implementation
test results

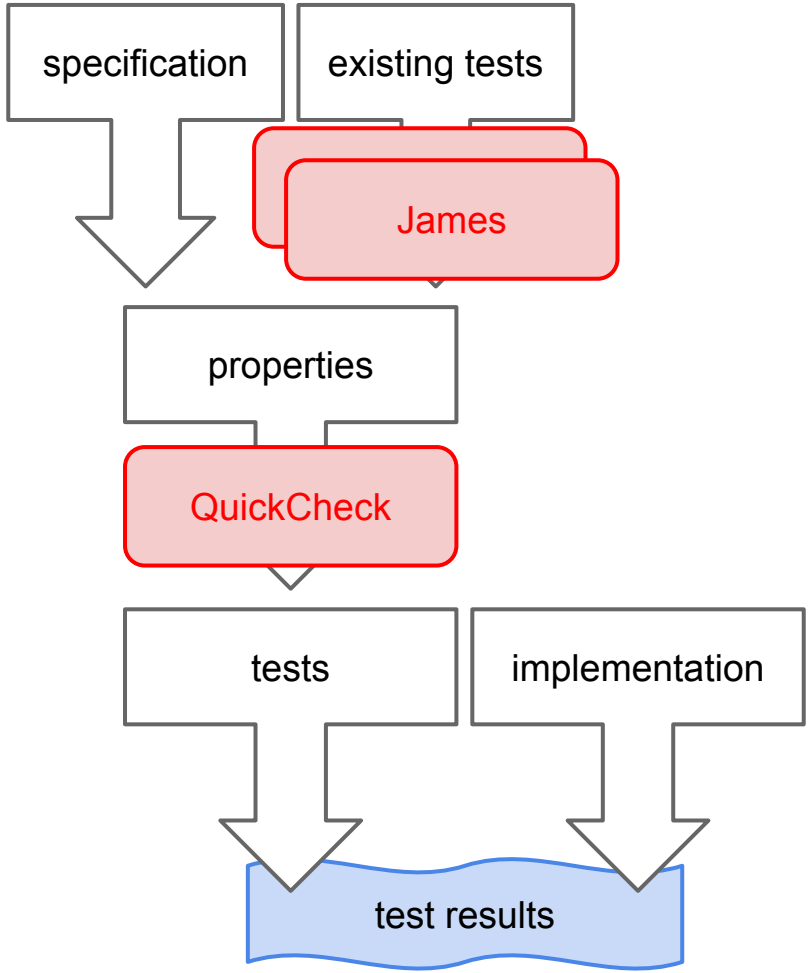

properties
QuickCheck
tests
implementation
test results

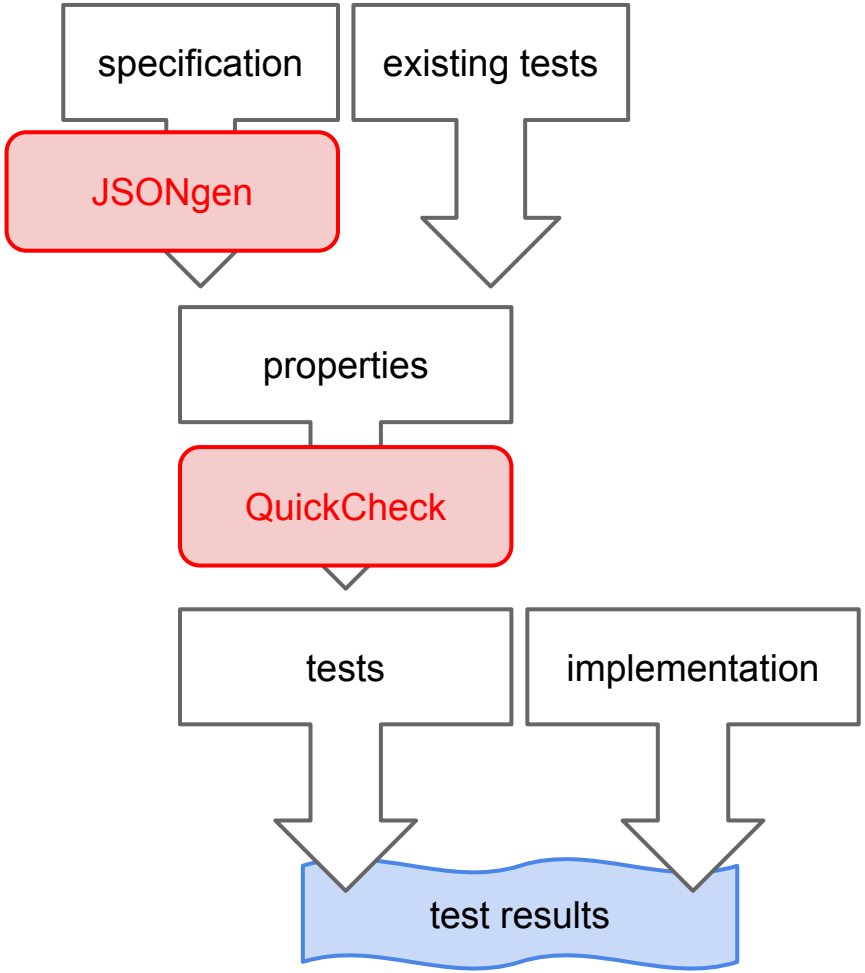

properties
ReadSpec
QuickCheck
tests
implementation
test results

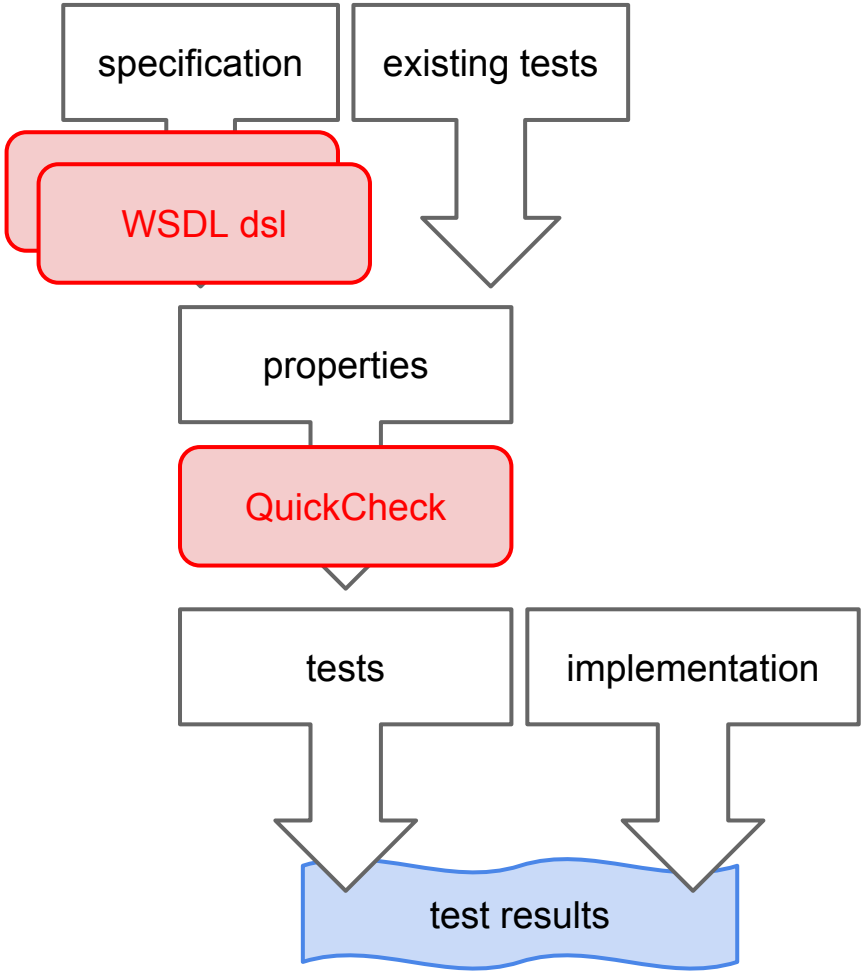

properties
GoodExamples
QuickCheck
tests
implementation
test results

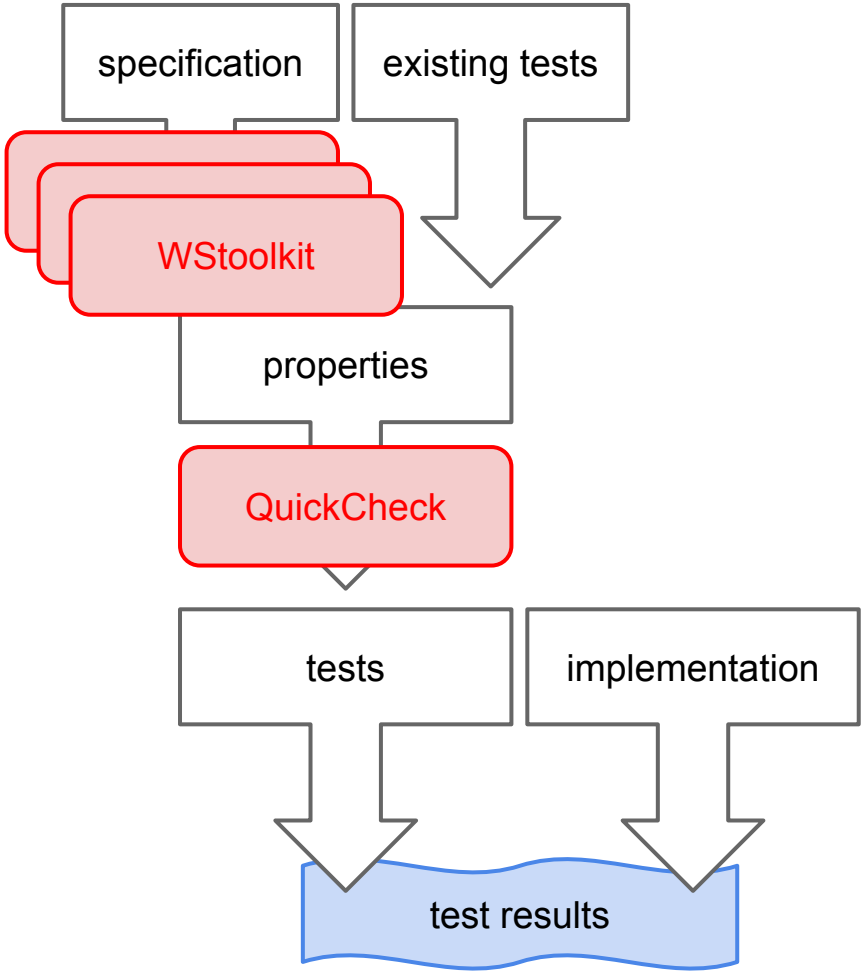

specification
existing tests
properties
QuickCheck
tests
implementation
test results

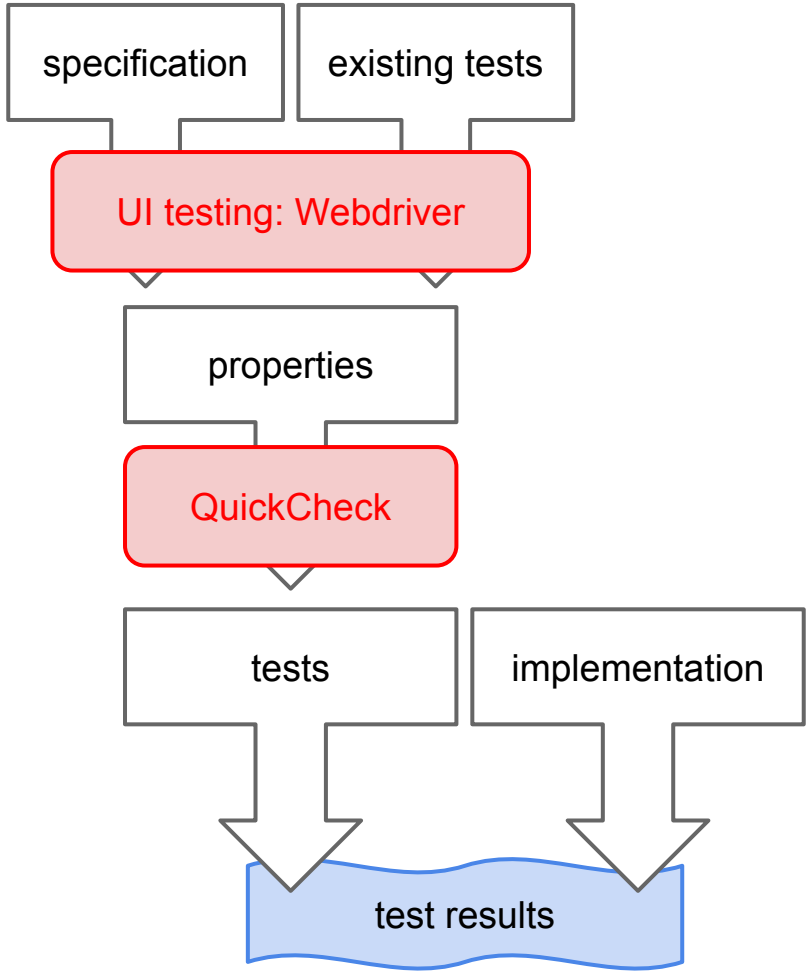

specification
existing tests
Synapse
properties
QuickCheck
tests
implementation
test results

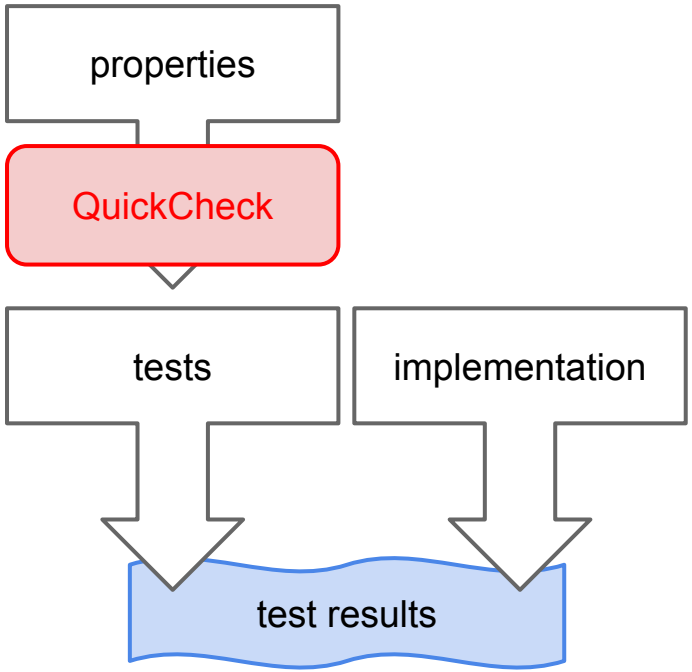

specification
existing tests
James
properties
QuickCheck
tests
implementation
test results

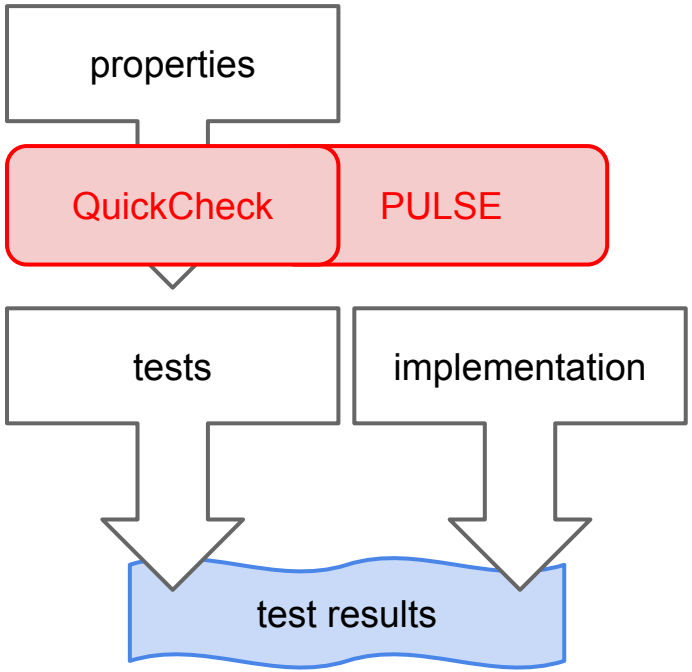

specification
existing tests
JSONgen
properties
QuickCheck
tests
implementation
test results

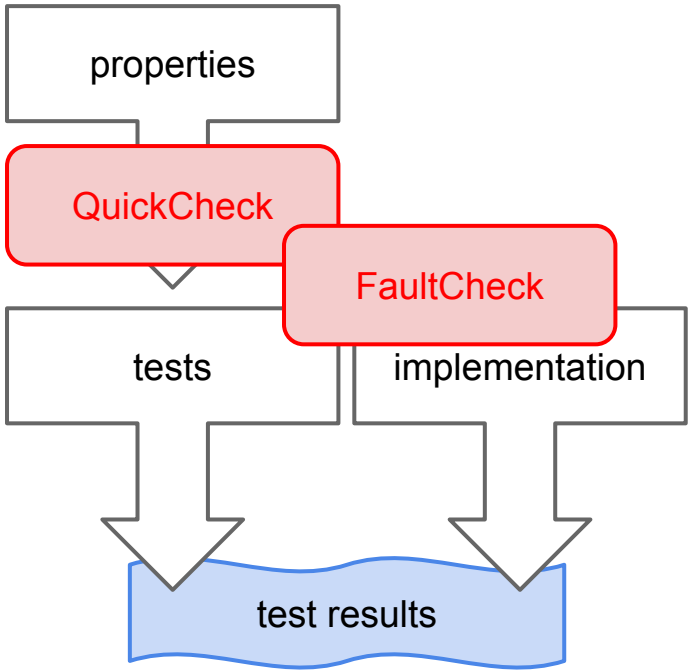

specification
existing tests
WSDL dsl
properties
QuickCheck
tests
implementation
test results

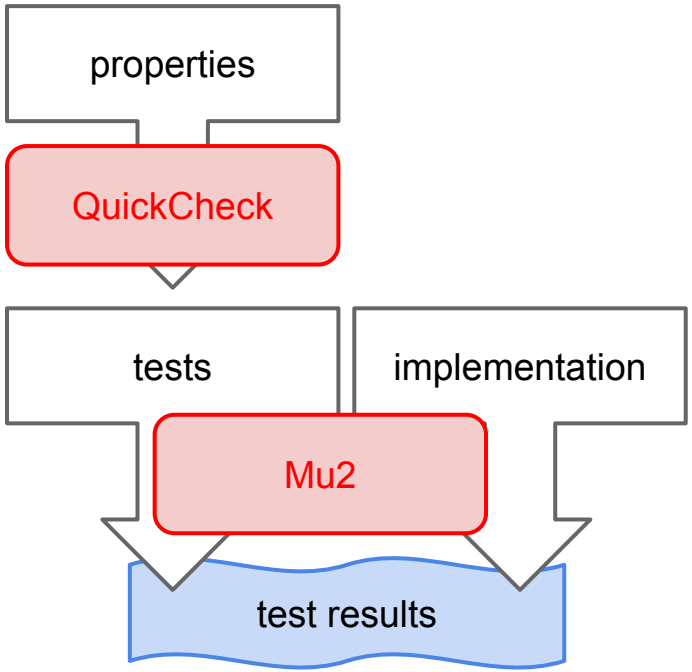

specification
existing tests
WStoolkit
properties
QuickCheck
tests
implementation
test results

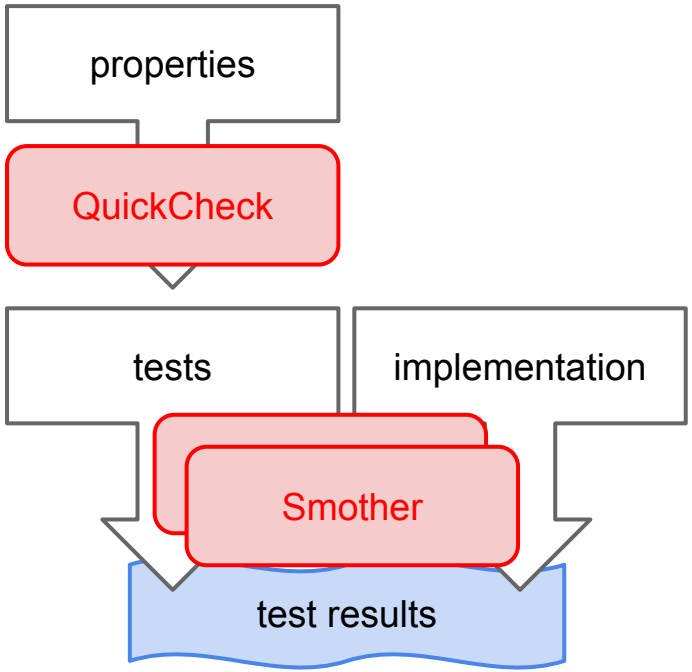

specification
existing tests
UI testing: Webdriver
properties
QuickCheck
tests
implementation
test results

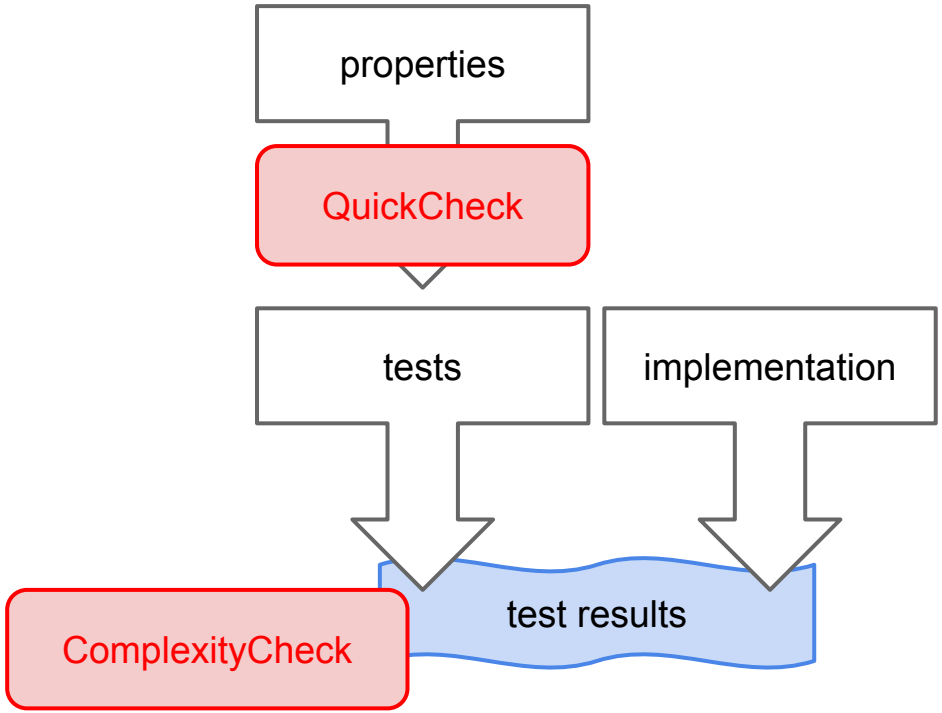

properties
QuickCheck
tests
implementation
test results

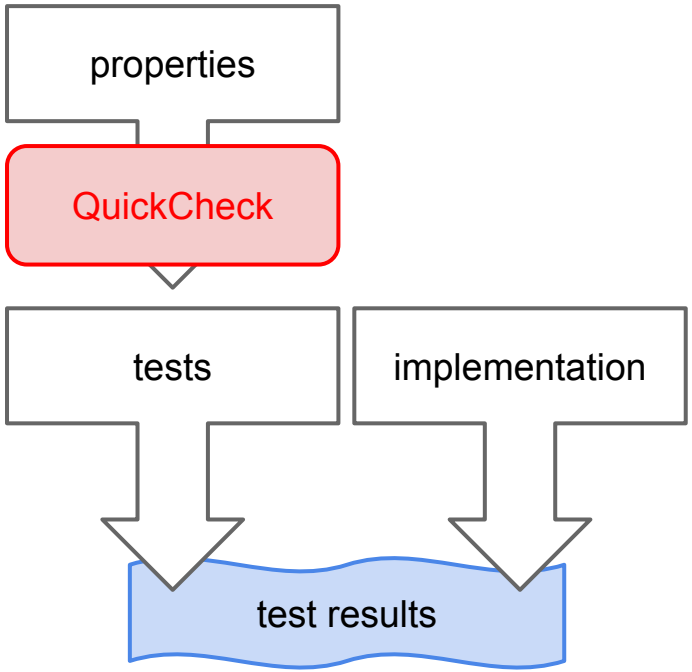

properties
QuickCheck
PULSE
tests
implementation
test results

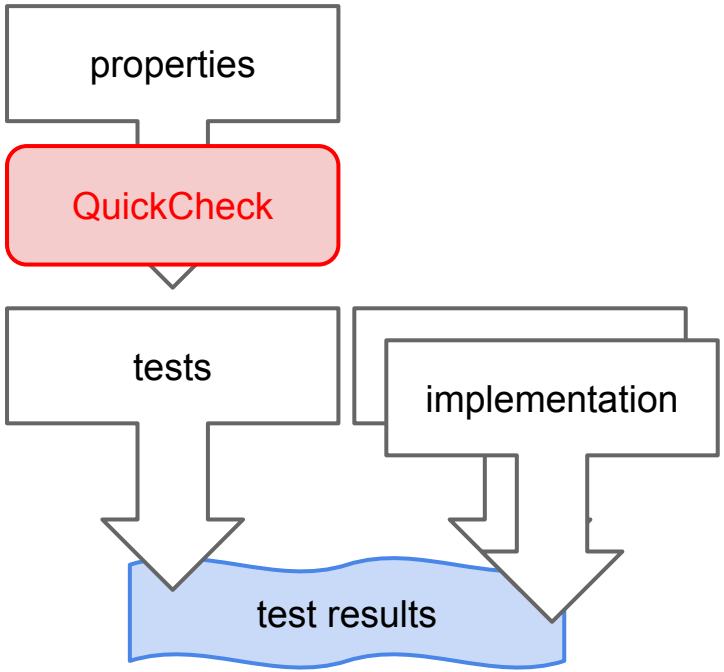

properties
QuickCheck
FaultCheck
tests
implementation
test results

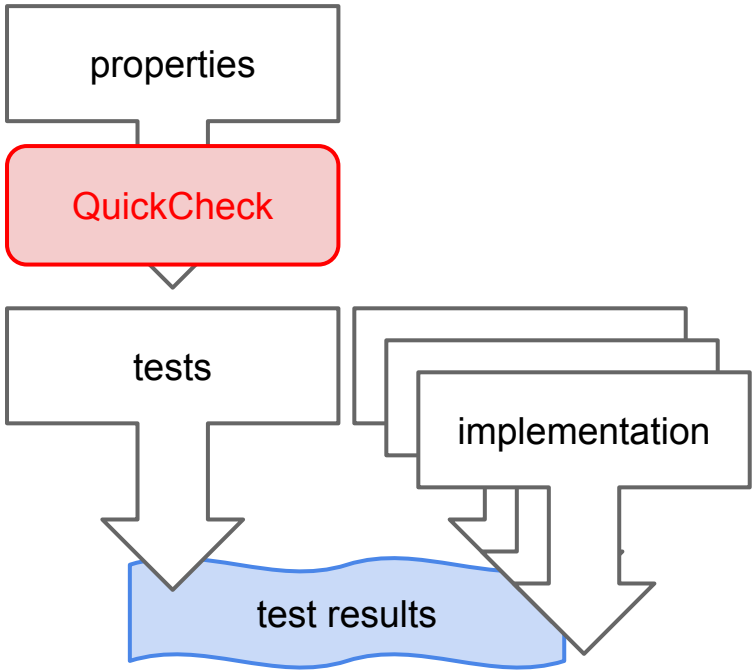

properties
QuickCheck
tests
implementation
Mu2
test results

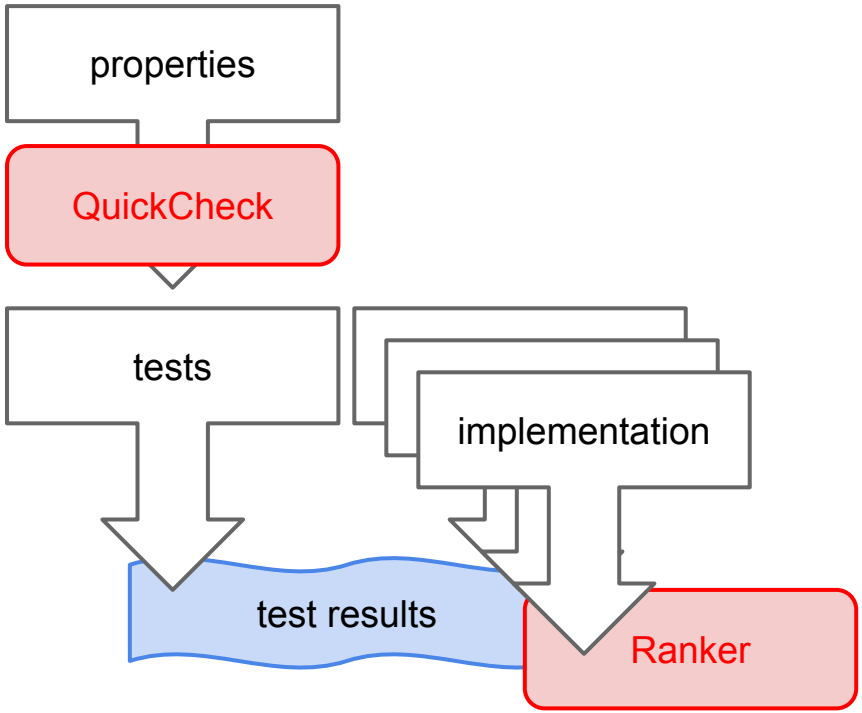

properties
QuickCheck
tests
implementation
Smother
test results

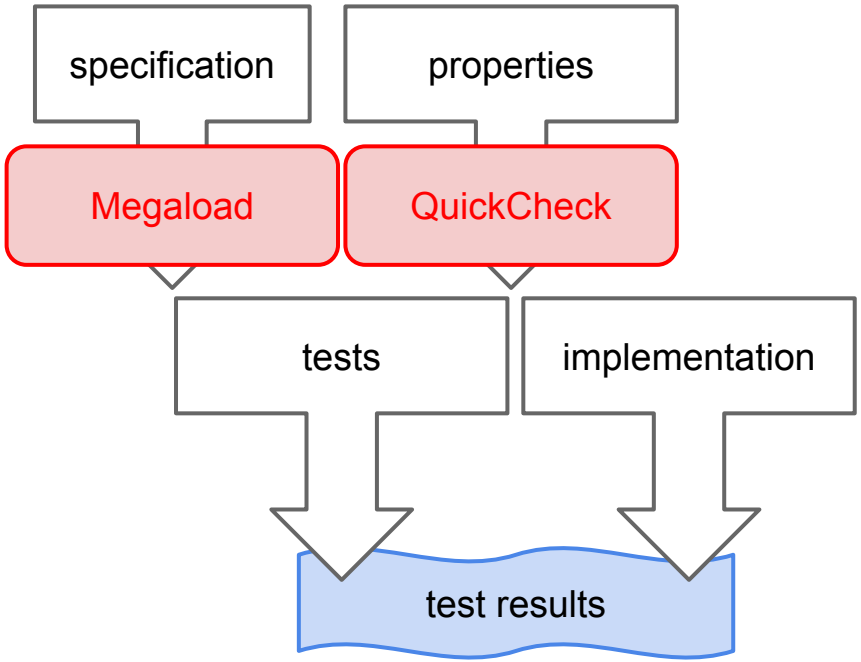

properties
QuickCheck
tests
implementation
test results
ComplexityCheck


properties
QuickCheck
tests
implementation
test results


properties
QuickCheck
tests
implementation
test results


properties
QuickCheck
tests
implementation
test results


properties
QuickCheck
tests
implementation
test results
Ranker


specification
properties
Megaload
QuickCheck
tests
implementation
test results

Case study

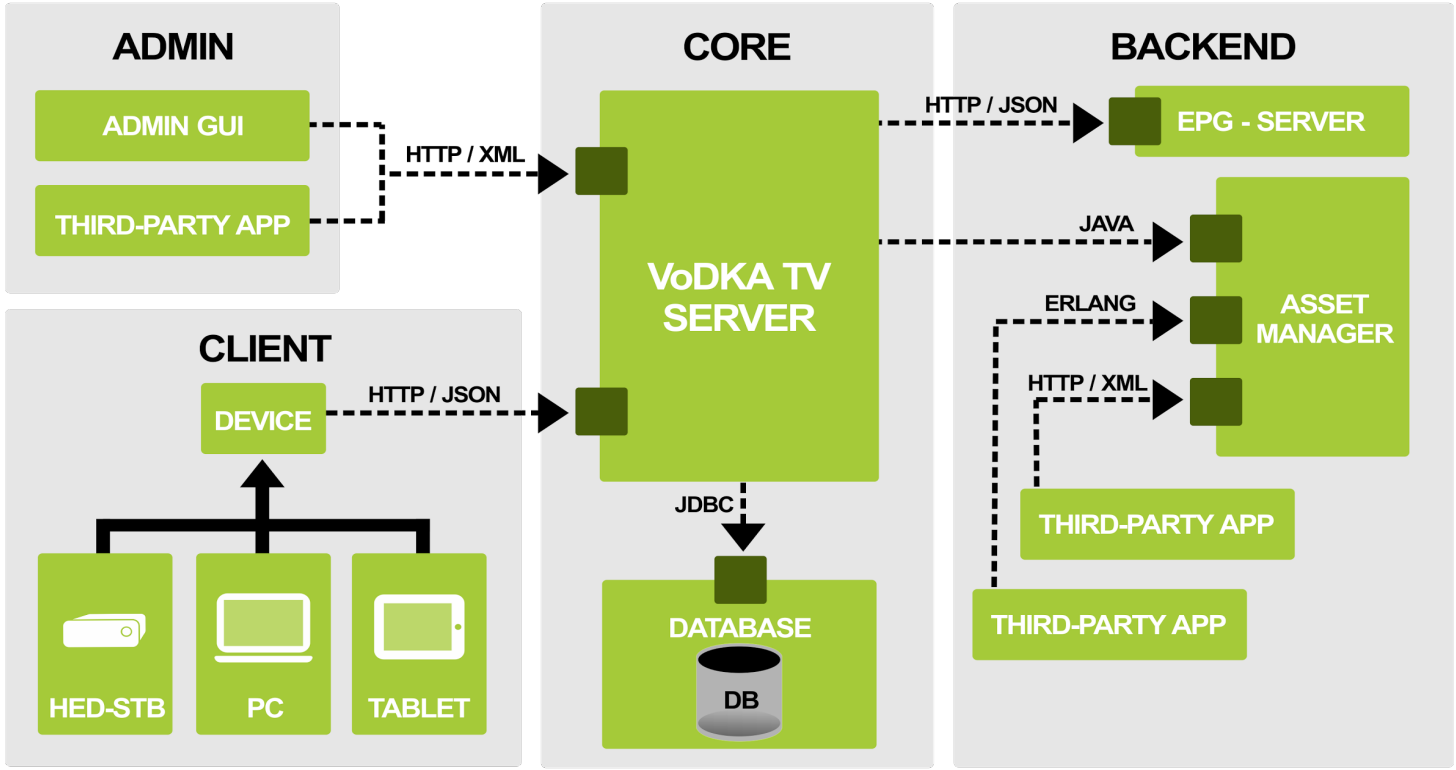
VoDKATV

Internet-Protocol TV (IPTV) / “Over the top” content (OTT) Cloud Middleware Architecture.

Interactive services for IPTV/OTT environments, eg, hotels.

Runs on a set-top-box (STB) , connected to a TV + remote.

Component-based; on client side: STB, tablet, PC, phone, …


ADMIN
ADMIN GUI
THIRD-PARTY APP
HTTP / XML
CORE
VoDKA TV SERVER
JDBC
DATABASE
DB
BACKEND
HTTP / JSON
EPG - SERVER
JAVA
ERLANG
HTTP / XML
ASSET MANAGER
THIRD-PARTY APP
THIRD-PARTY APP
CLIENT
DEVICE
HTTP / JSON
HED-STB
PC
TABLET

Set-top box

The STB includes

- a portable middleware layer implemented in Erlang,
- a UI layer developed in HTML, JavaScript and CSS (Webkit browser);
- communication between the UI layer and the middleware via a WebSocket-based protocol.

Web services for interactions

Some APIs respond in XML, others in JSON

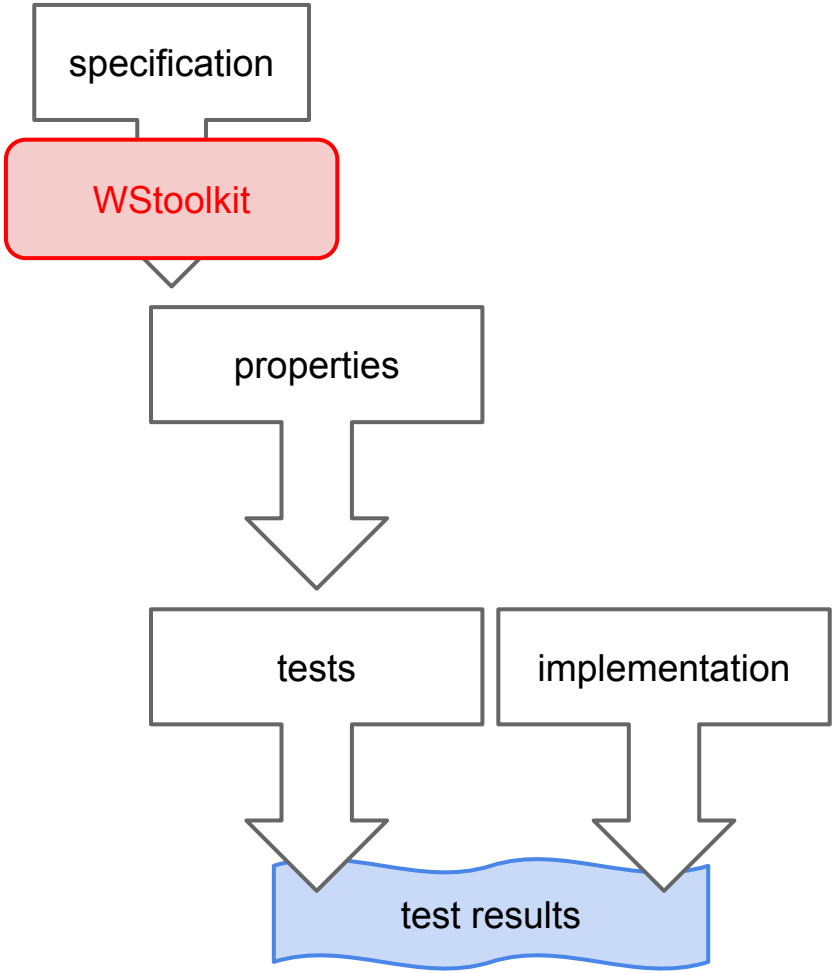
Different kinds of authentication for access to the APIs:

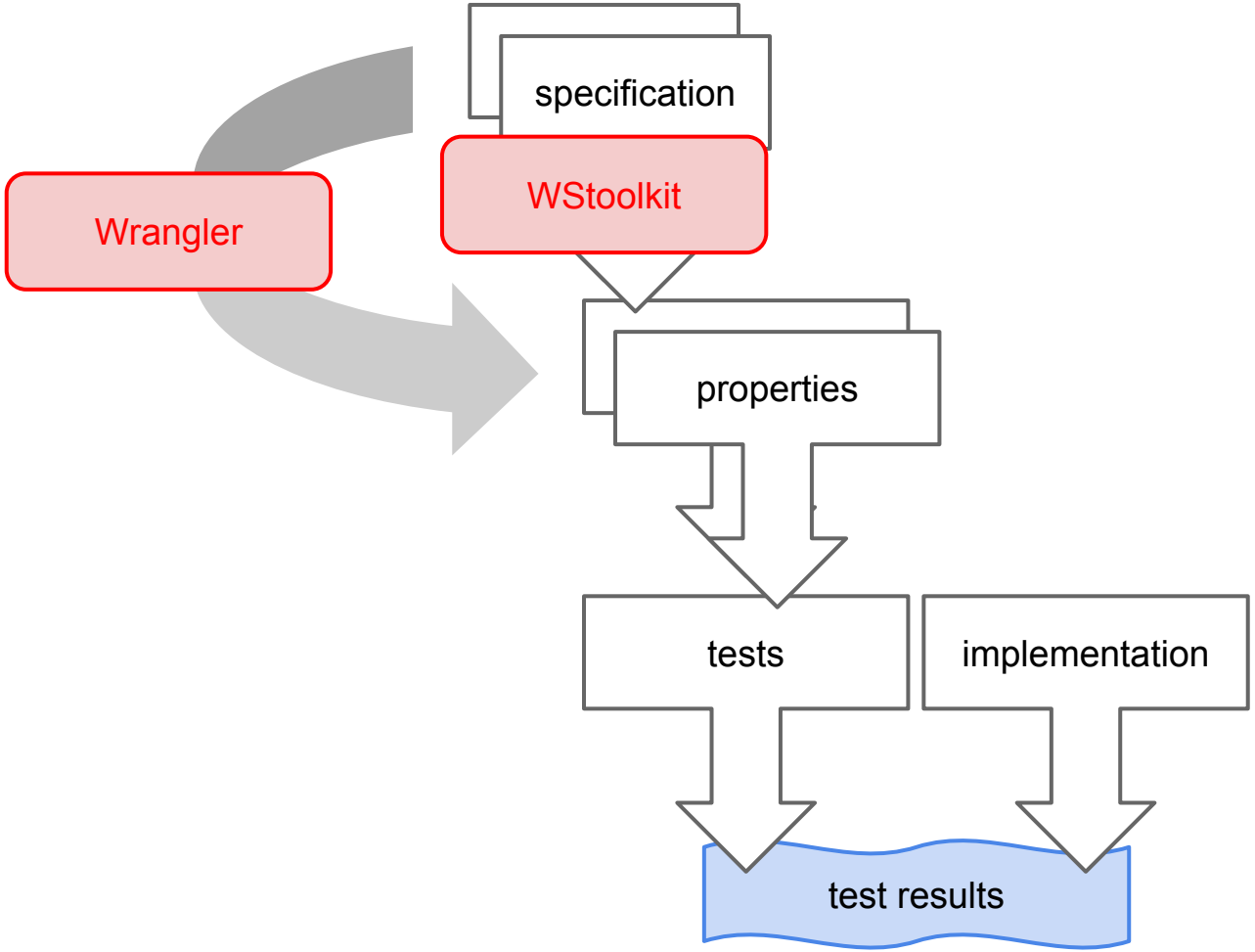
- none required,
- authentication with cookies
- authentication with tokens, e.g. expiration time, max # logins per user, …

Property-based testing for VoDKATV

This is where the demo by Thomas fits …

The toolset


specification
WStoolkit
properties
tests
implementation
test results


specification
WStoolkit
Wrangler
properties
tests
implementation
test results

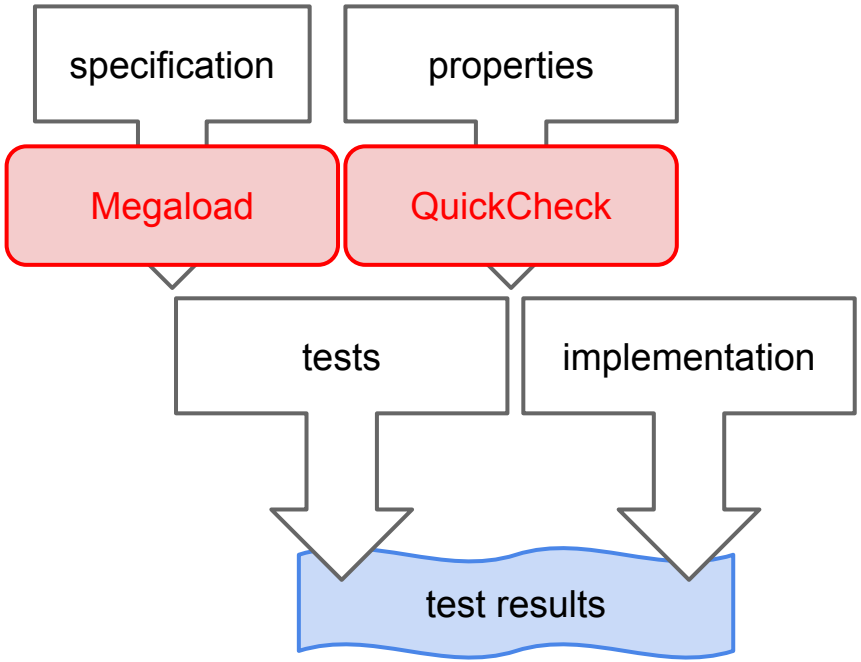
Evolution in PBT with WStoolkit

Using Wrangler, Kent's tool for refactoring Erlang systems.

Infer of changes between WSDL descriptions …

… from these generate refactoring scripts …

… which automate model evolution as much as possible.


specification
properties
Megaload
QuickCheck
tests
implementation
test results

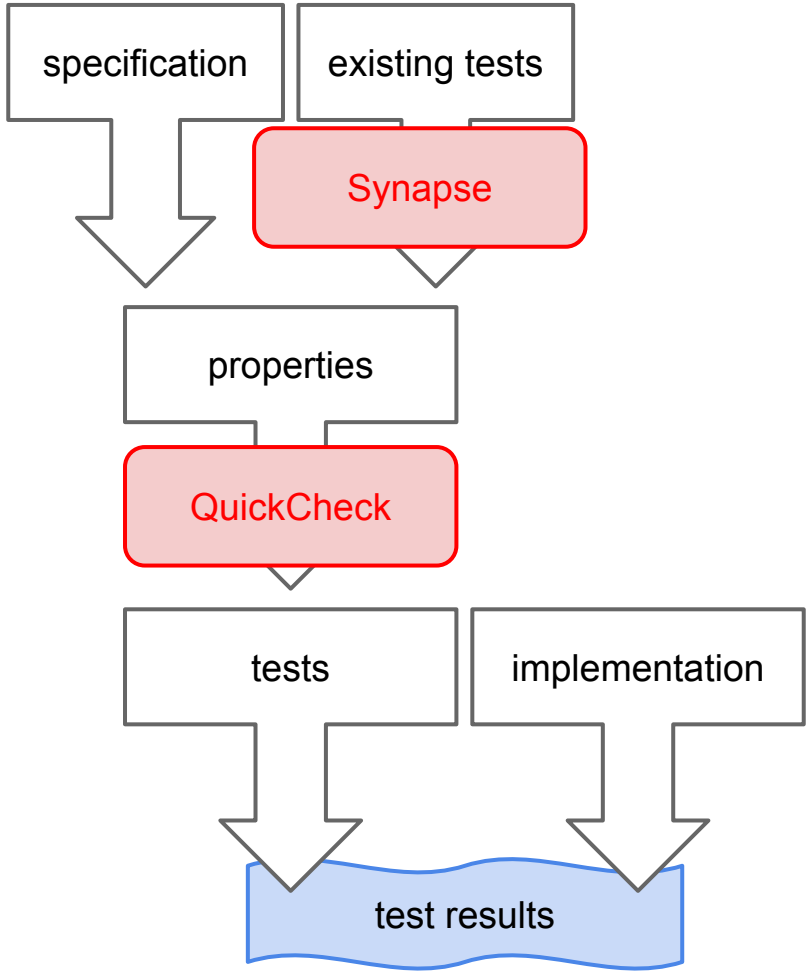
Megaload – Load testing VoDKA

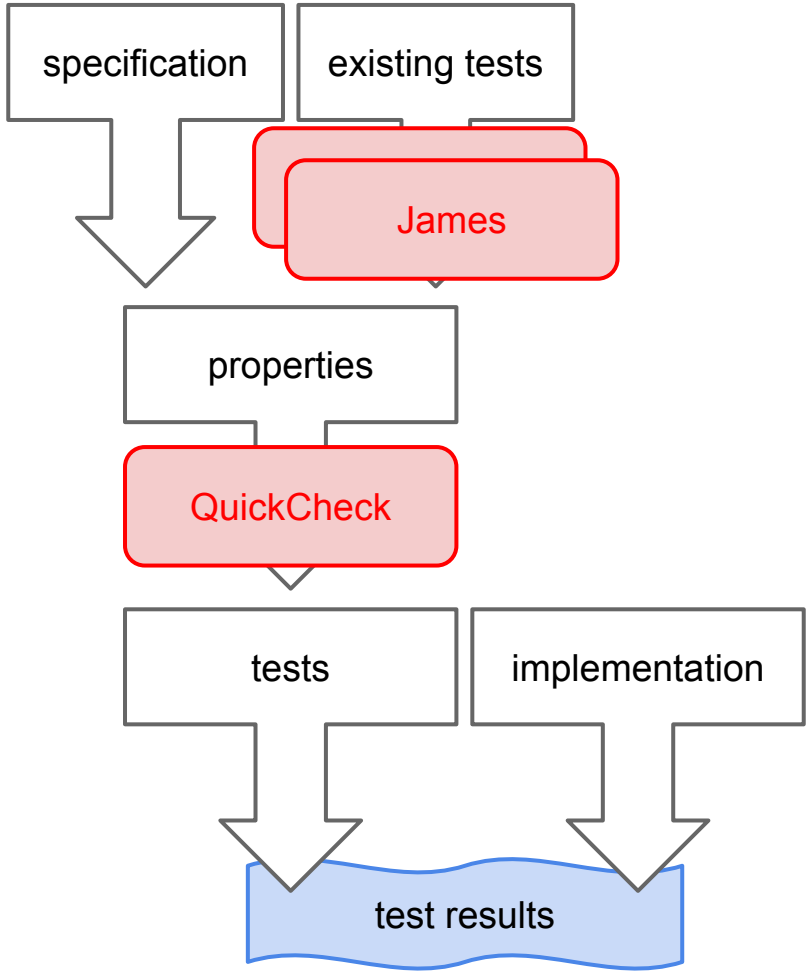
Cloud-based load testing of systems.

Megaload: loads, monitors and presents results.

Generating load profiles …

… and shrinking to minimal (counter-) examples in the most load-effective way.


specification
existing tests
Synapse
properties
QuickCheck
tests
implementation
test results


specification
existing tests
James
properties
QuickCheck
tests
implementation
test results

Inference and PBT

How to develop properties for a system. Two tools:

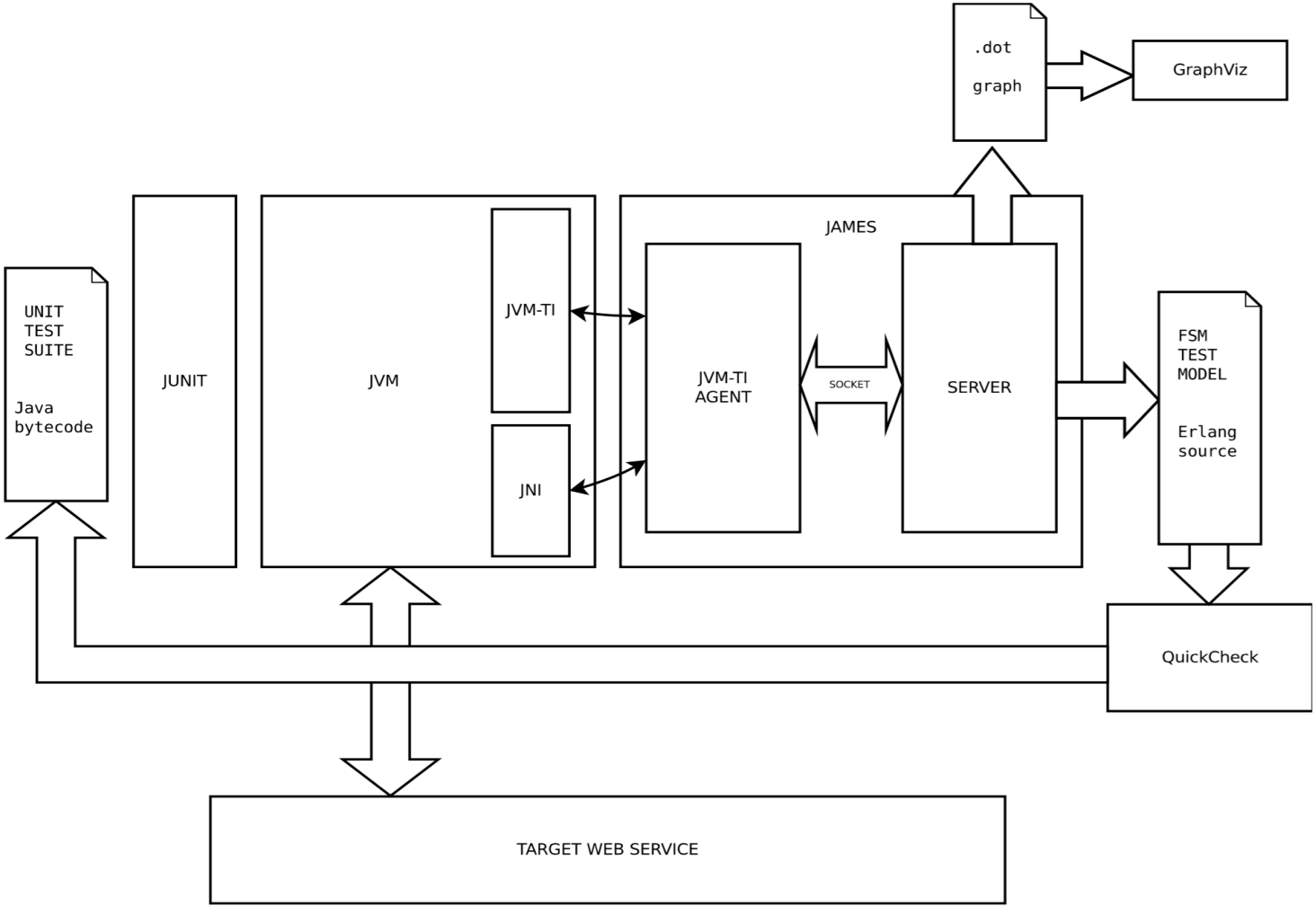
- *James* – infer models for web services from unit tests written in Java, using JUnit.

- *Synapse* - infer FSM from systems, and visualise the difference between models / systems.

James

New JUnit tests from existing tests, by model inference.

Track a combination of data- / control-flow information …
… extracted from running the test suite on the SUT
… run the tests on the Java VM
… track information using C++ agent and JVM-TI API


.dot
graph
GraphViz
JAMES
UNIT
TEST
SUITE
Java
bytecode
JUNIT
JVM
JVM-TI
JNI
JVM-TI
AGENT
SOCKET
SERVER
FSM
TEST
MODEL
Erlang
source
QuickCheck
TARGET WEB SERVICE

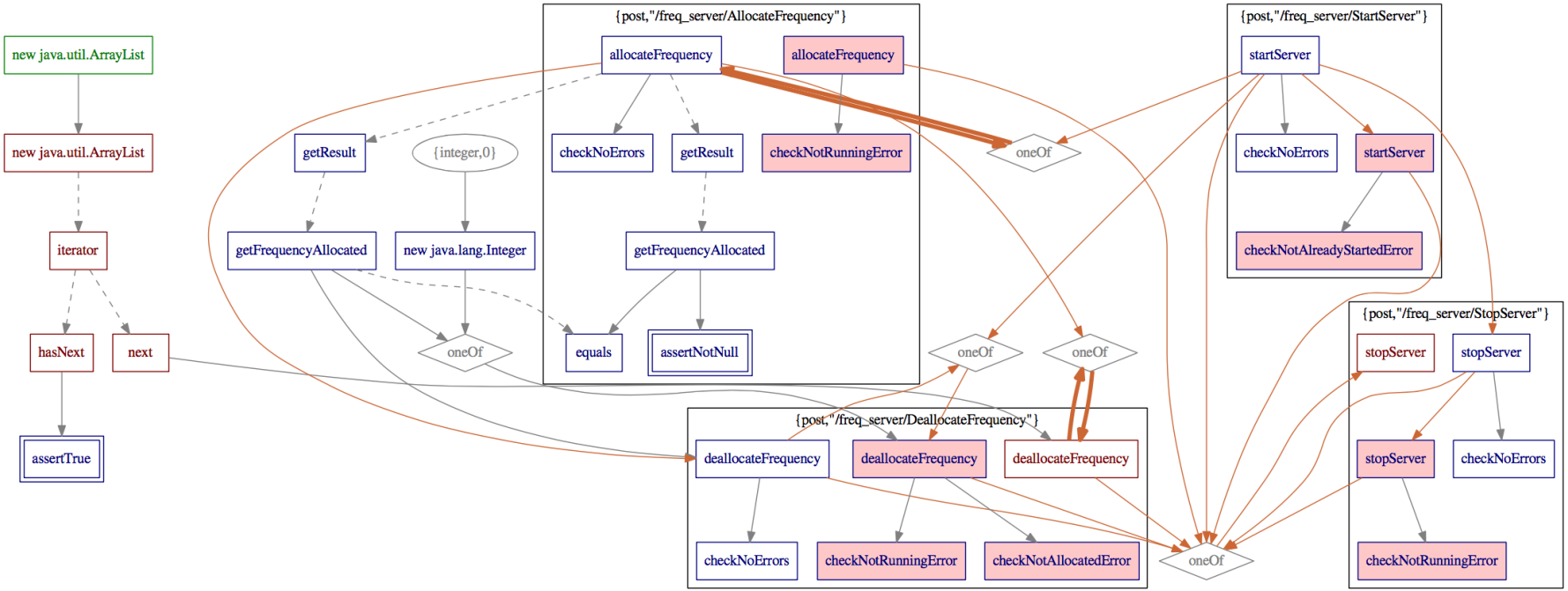
James

Track and send to an Erlang server:

- the execution order of the calls in the JUnit tests, and
- how objects are reused.

Server generates a model … visualised through GraphViz.

Translate model into QuickCheck … then generate new tests, that can be added to the original test suite.



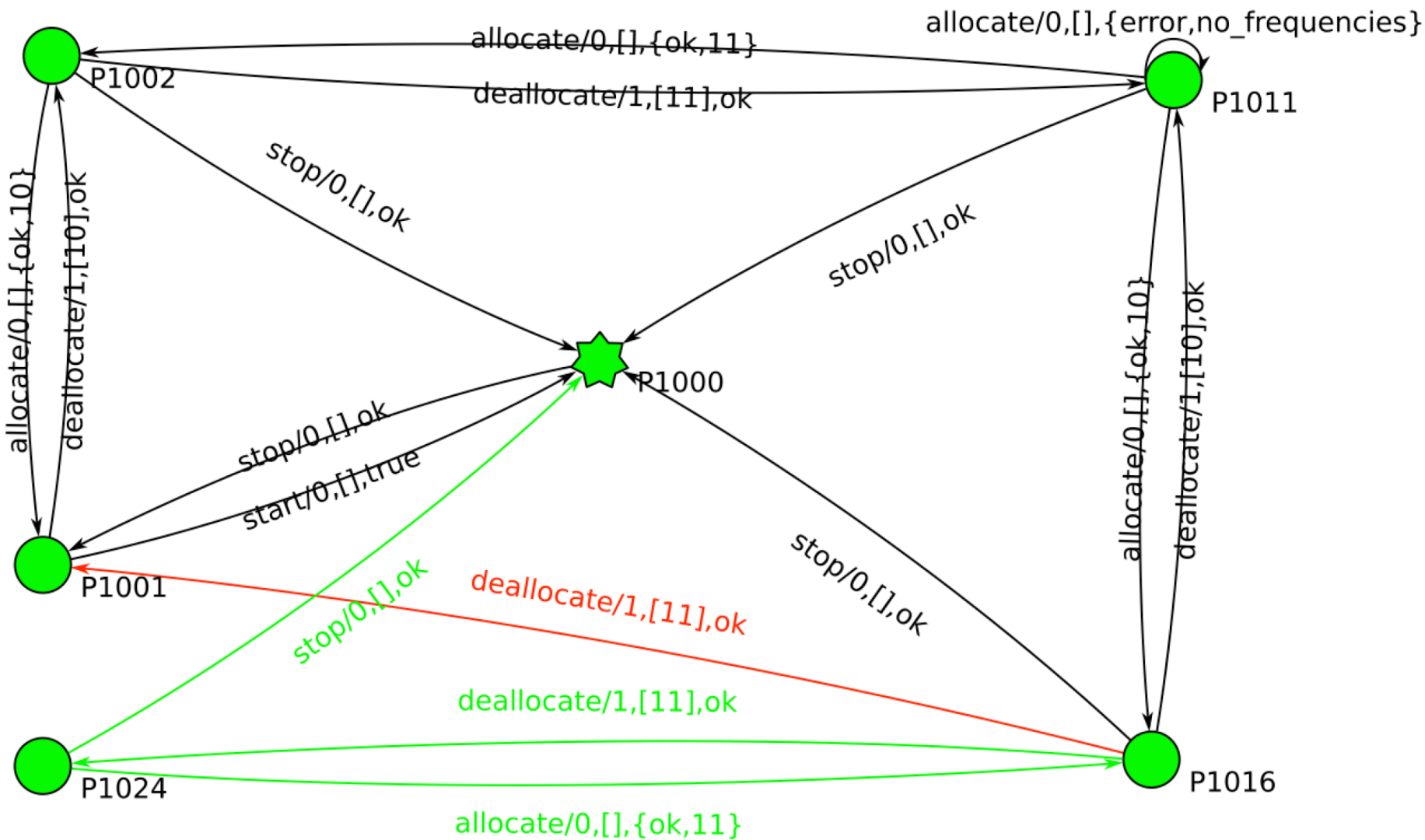
Java Interactions

Synapse

An Erlang interface to grammar inference tools.

Synapse interfaces to the *StateChum* tool for passive and active inference of FSM models, as well as:

- active and passive learning,
- model differencing, and
- FSM and difference visualisation.

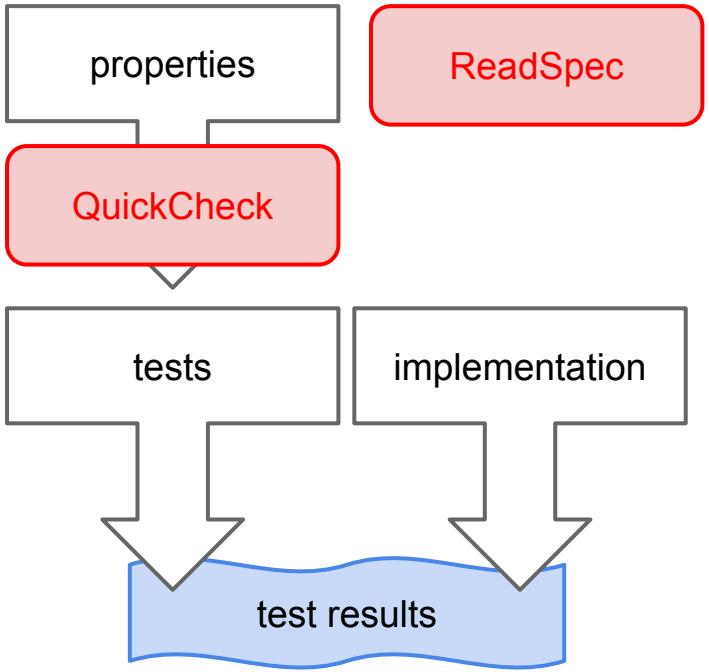

allocate/0,[],{error,no_frequencies}
allocate/0,[],{ok,11}
deallocate/1,[11],ok
P1002
P1011
stop/0,[],ok
stop/0,[],ok
allocate/0,[],{ok,10}
deallocate/1,[10],ok
P1000
allocate/0,[],{ok,10}
deallocate/1,[10],ok
stop/0,[],ok
start/0,[],true
P1001
stop/0,[],ok
stop/0,[],ok
deallocate/1,[11],ok
deallocate/1,[11],ok
P1024
P1016
allocate/0,[],{ok,11}

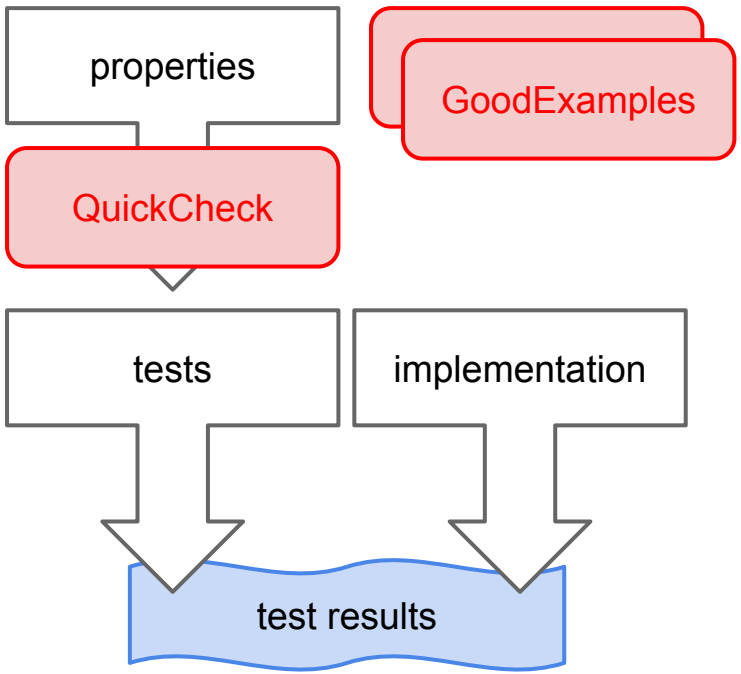
Understanding properties and models

ReadSpec to render QuickCheck models in (semi-)natural language.

Synapse tool allows users to visualise differences between variants of models / systems as FSMs.

GoodExamples tool to make the meaning of a property more concrete by viewing it as a set of unit tests.


properties
ReadSpec
QuickCheck
tests
implementation
test results


properties
GoodExamples
QuickCheck
tests
implementation
test results

ReadSpec

ReadSpec uses QuickCheck to automatically generate semi-natural language descriptions of QuickCheck properties and QuickCheck state machine models.

Example: `simple_eqc.erl` contains a property to test the delete operation of the lists module:

```
?FORALL({I,L}, {int(), list(int())},
                  not lists:member(I,lists:delete(I, L)))
```

FEATURE: 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 19

AND I have the list [7, -24, -18, 17, -8, -9, -8]

THEN lists:member(19, lists:delete(19, [7,-24,-18,17,-8,-9,-8]))

IS FALSE.

Good Examples tool

It can be hard to tell what a property tests…
properties - powerful and general;
unit tests - easy to understand but specific.

Good Examples - makes the meaning of a property more concrete by viewing it as a set of unit tests.

Scenario 1: From a test suite, which of our test cases the property captures?

Solution: Our technique can say with high probability whether a property captures a given test case.

Scenario 2: what does a property test?

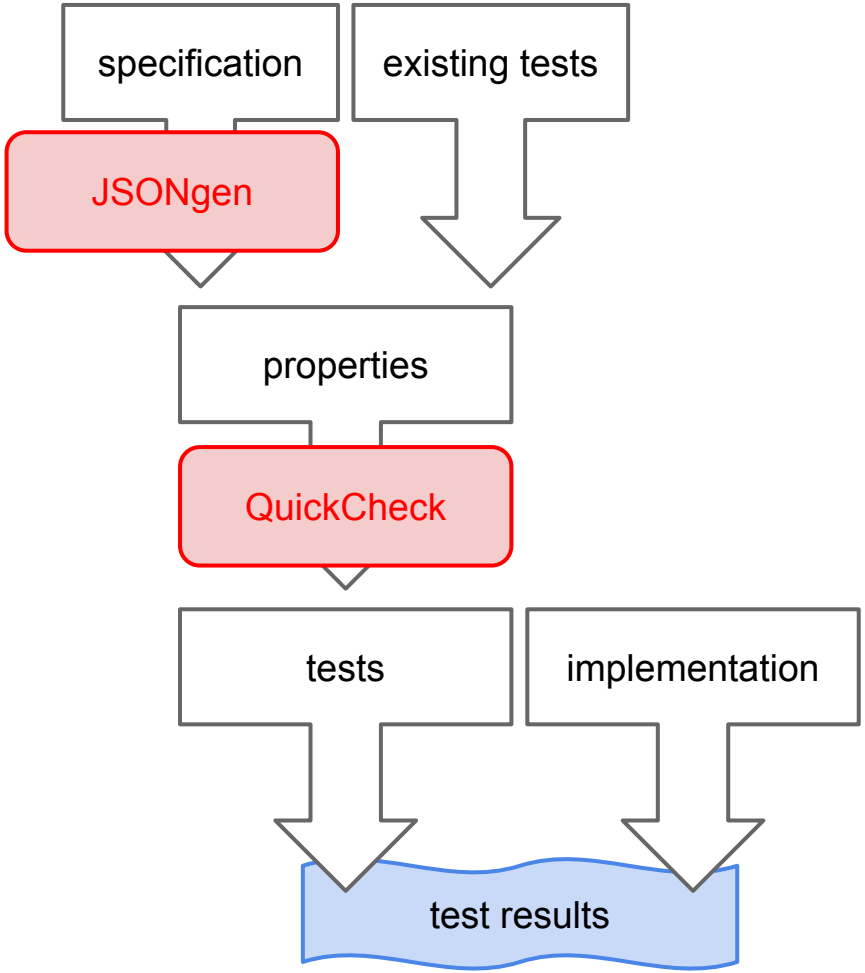
Solution: Our technique to generates representative examples of what a QuickCheck property tests.

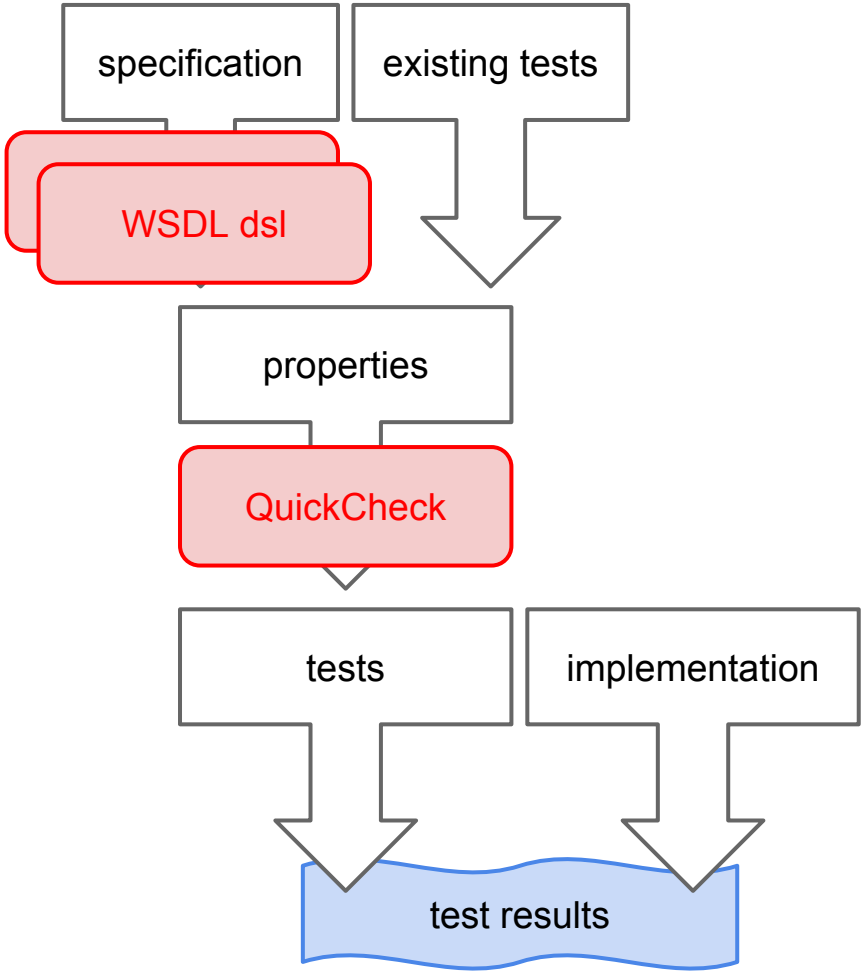
Support for Web Services

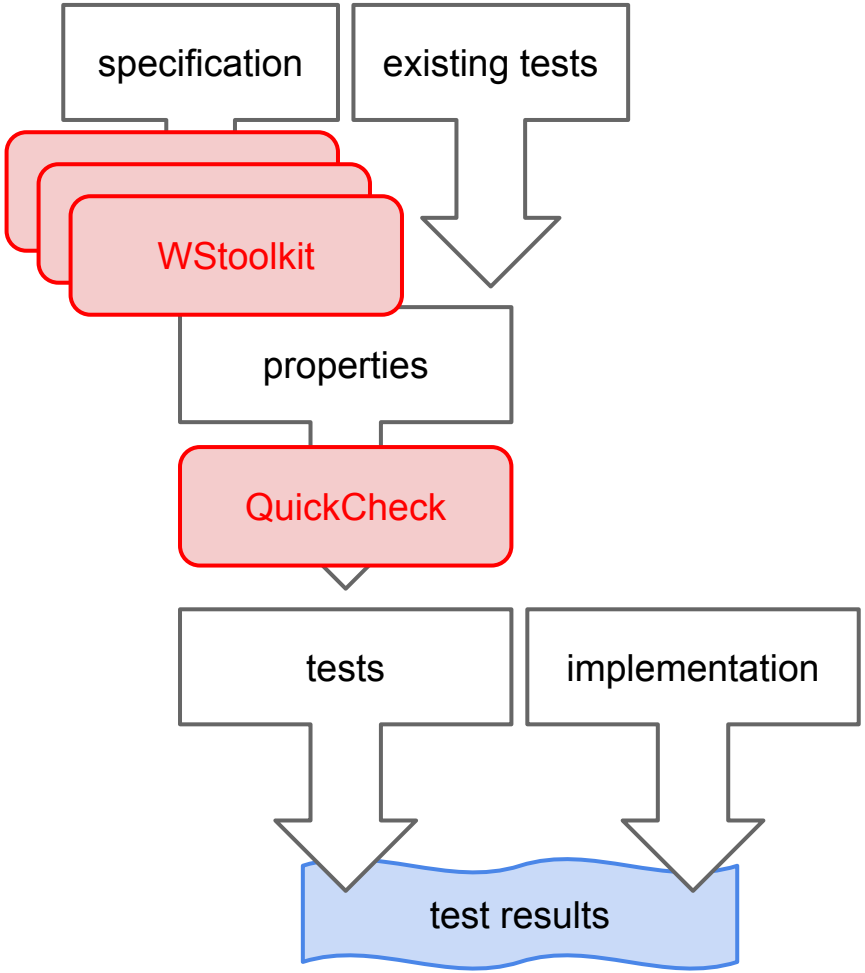
Tools to support data generation for web services models:

JSONgen is a library for generating QuickCheck generators from descriptions of JSON data using JSON schemas, and for automatically exploring and testing JSON web services.

wsdl_dsl is a QuickCheck library that implements a domain specific language which re-uses the WSDL syntax to allow users to express WSDL types as QuickCheck generators.


specification
existing tests
JSONgen
properties
QuickCheck
tests
implementation
test results


specification
existing tests
WSDL dsl
properties
QuickCheck
tests
implementation
test results


specification
existing tests
WStoolkit
properties
QuickCheck
tests
implementation
test results

JSONgen – convert and explore

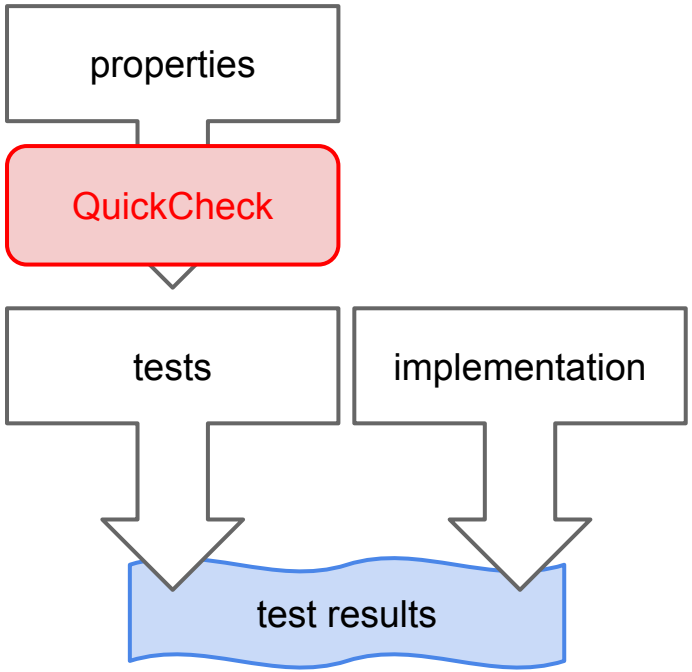
Convert JSON schema to mochijson2 Erlang term.

Convert JSON schema into a QuickCheck generator.

Convert JSON data value in mochijson2 format to text

Explore and test a JSON based web service using the web links / data types embedded in the JSON schema args.

Can tailor the actions with a QuickCheck state machine.

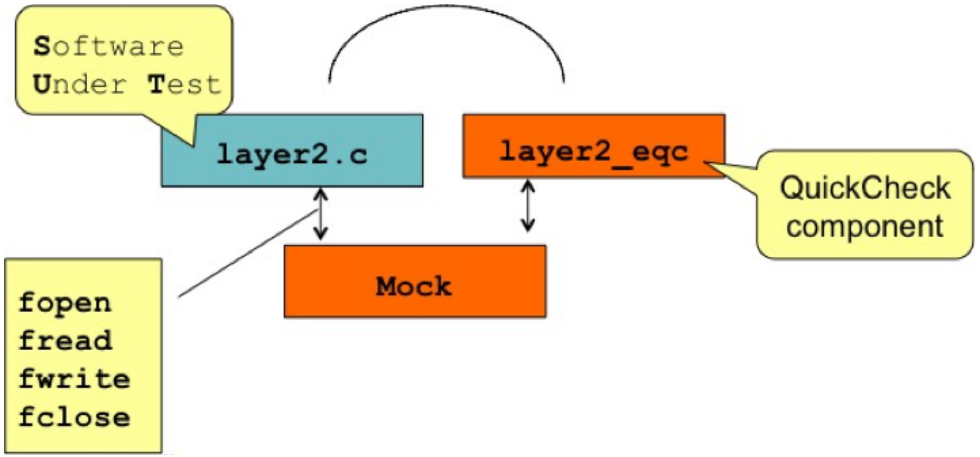

properties
QuickCheck
tests
implementation
test results

Scaling PBT

Model using *components* instead of a single model.

Library for *mocking* the behaviour of callout components.

Clustered system resulting from the component models.



MoreBugs

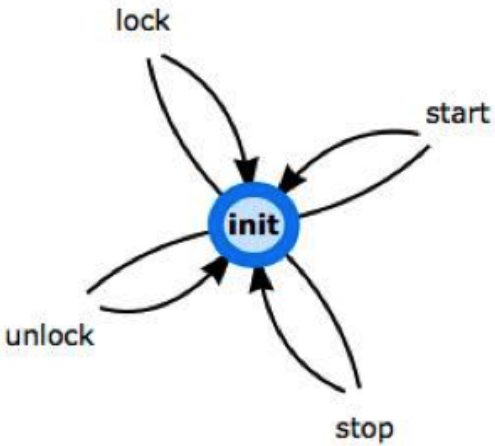
QuickCheck “by hand”: run QC, fix bug, repeat …

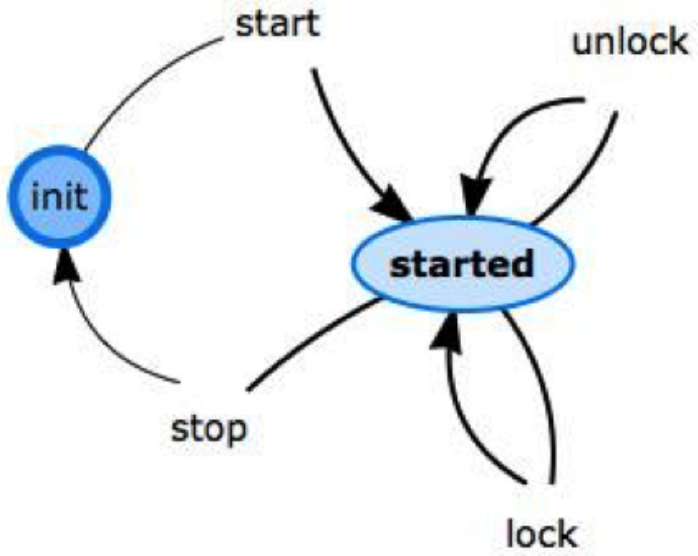
With MoreBugs, can find “all” bugs at once, through

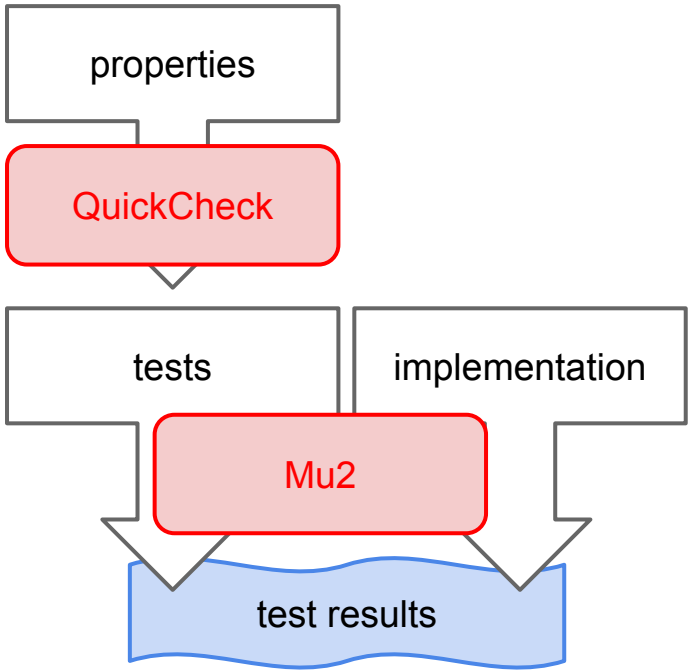
- find bug,
- generalise
- modify generator to avoid it

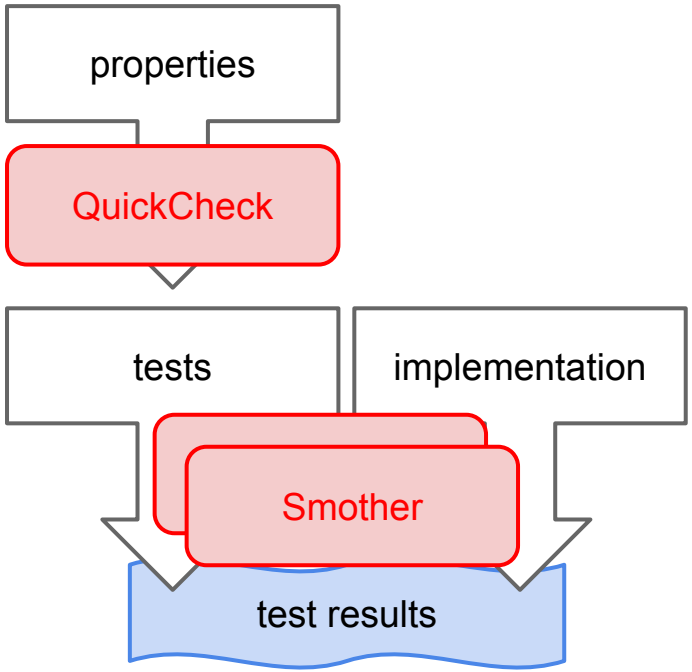
and repeat …

Graphical editing






properties
QuickCheck
tests
implementation
Mu2
test results


properties
QuickCheck
tests
implementation
Smother
test results

Validating quality of test suites

Smother used to assess the MC/DC coverage of a test suite.

Mu2 supports mutation testing

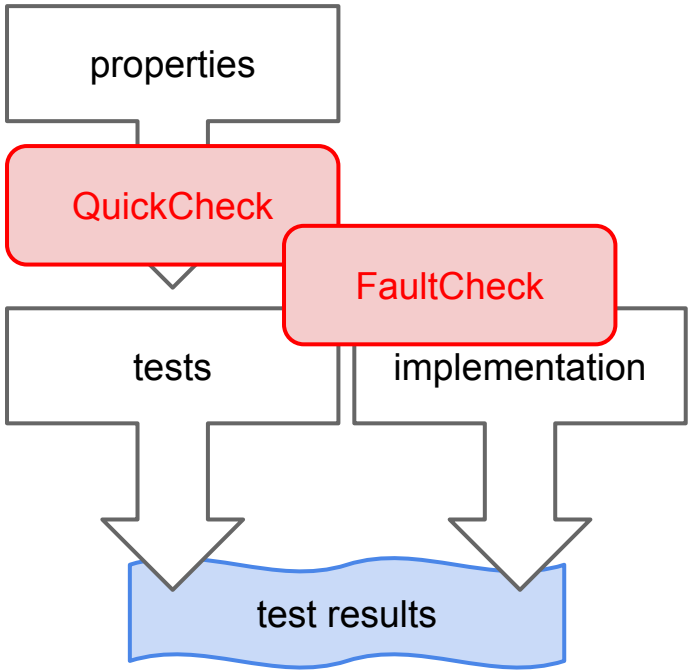
```
dv(A,B) ->
    if (A == 0) and (B > 4) ->
           B / 1;
       true ->
           B / A
    end.
```

(A == 0) and (B > 4)

- Matched: 1 times
- Non-Matched: 2 times

When non-matched: 75.0% sub-component coverage

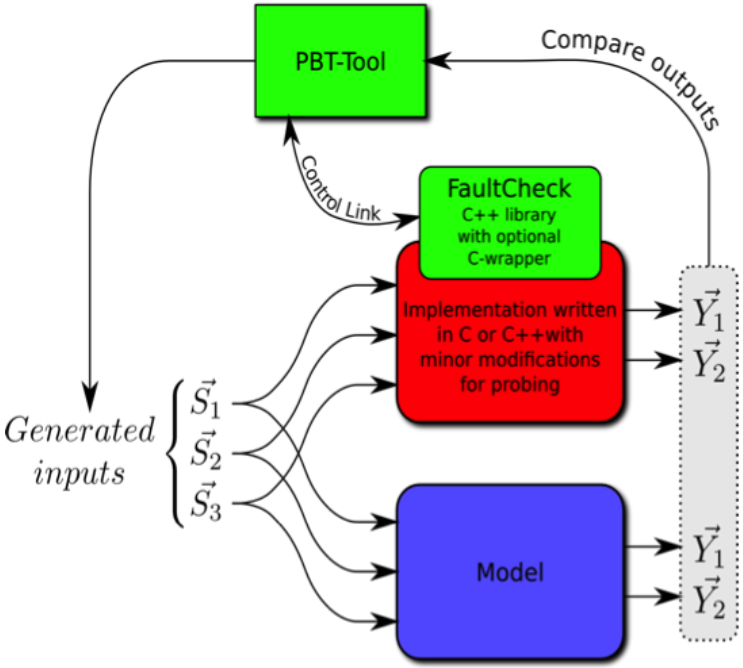
	matched	non-matched
A == 0	0	2
B > 4	1	1


properties
QuickCheck
FaultCheck
tests
implementation
test results

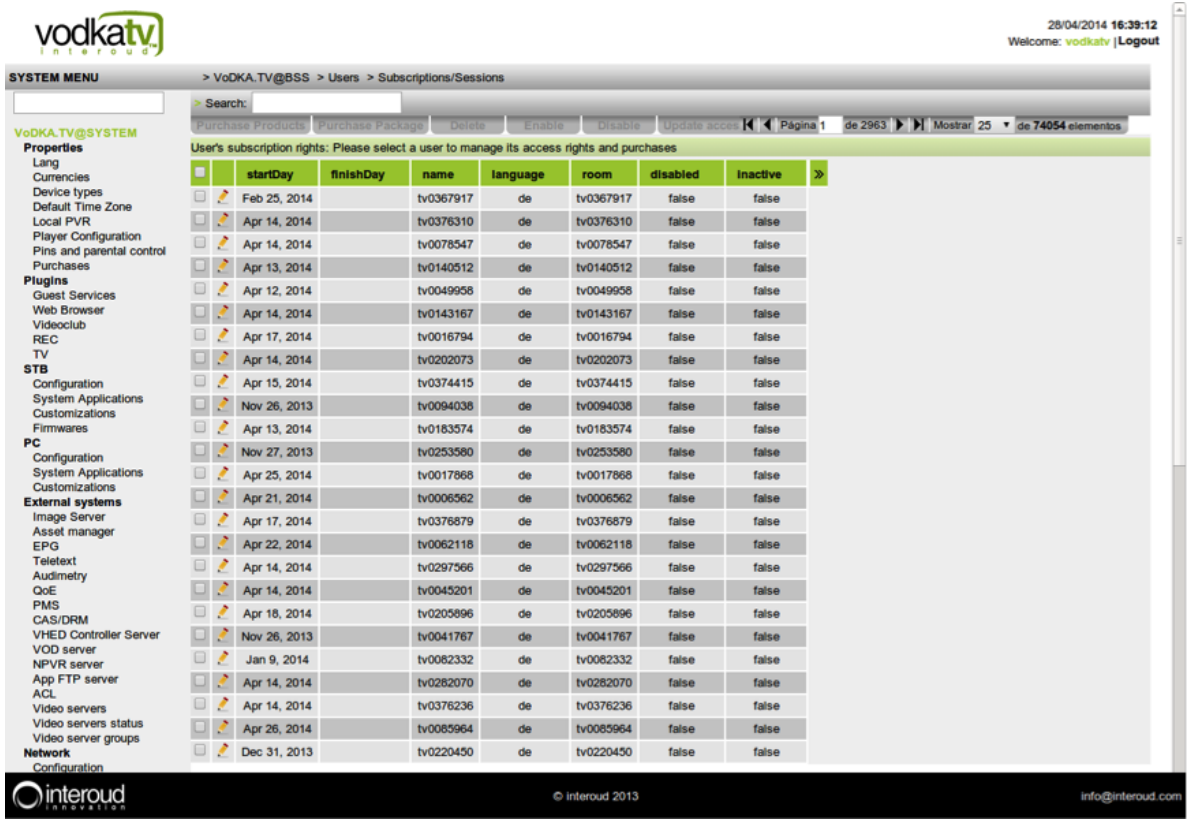
Testing non-functional requirements

FaultCheck …

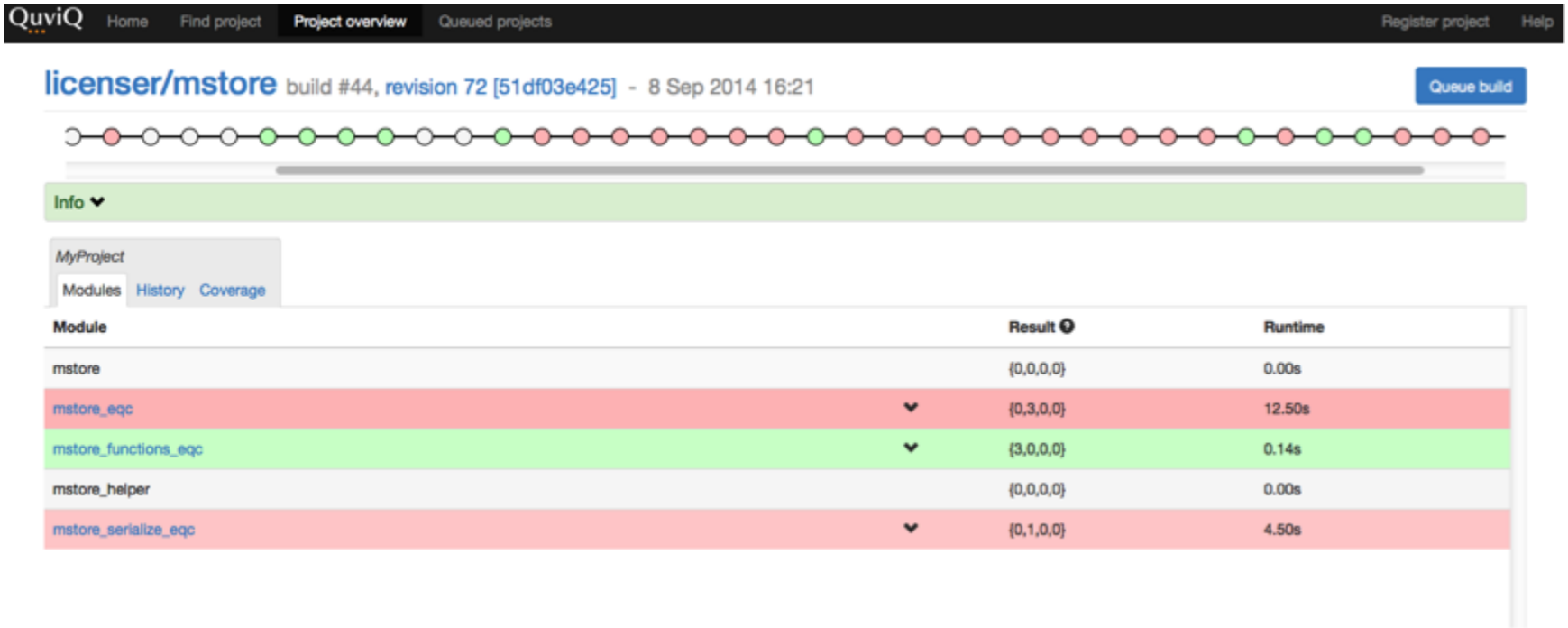
… a fault-injection tool for C code that combines fault-injection and property based testing using QuickCheck.



WebDriver


vodkatv
interoud
28/04/2014 16:39:12
Welcome: vodkatv | Logout
SYSTEM MENU
> VoDKA.TV@BSS > Users > Subscriptions/Sessions
Search:
Purchase Products
Purchase Package
Delete
Enable
Disable
Update acces
Página 1 de 2963
Mostrar 25 de 74054 elementos
VoDKA.TV@SYSTEM
Properties
Lang
Currencies
Device types
Default Time Zone
Local PVR
Player Configuration
Pins and parental control
Purchases
Plugins
Guest Services
Web Browser
Videoclub
REC
TV
STB
Configuration
System Applications
Customizations
Firmwares
PC
Configuration
System Applications
Customizations
External systems
Image Server
Asset manager
EPG
Teletext
Audimetry
QoE
PMS
CAS/DRM
VHED Controller Server
VOD server
NPVR server
App FTP server
ACL
Video servers
Video servers status
Video server groups
Network
Configuration
User's subscription rights: Please select a user to manage its access rights and purchases
startDay
finishDay
name
language
room
disabled
inactive
Feb 25, 2014 tv0367917 de tv0367917 false false
Apr 14, 2014 tv0376310 de tv0376310 false false
Apr 14, 2014 tv0078547 de tv0078547 false false
Apr 13, 2014 tv0140512 de tv0140512 false false
Apr 12, 2014 tv0049958 de tv0049958 false false
Apr 14, 2014 tv0143167 de tv0143167 false false
Apr 17, 2014 tv0016794 de tv0016794 false false
Apr 14, 2014 tv0202073 de tv0202073 false false
Apr 15, 2014 tv0374415 de tv0374415 false false
Nov 26, 2013 tv0094038 de tv0094038 false false
Apr 13, 2014 tv0183574 de tv0183574 false false
Nov 27, 2013 tv0253580 de tv0253580 false false
Apr 25, 2014 tv0017868 de tv0017868 false false
Apr 21, 2014 tv0006562 de tv0006562 false false
Apr 17, 2014 tv0376879 de tv0376879 false false
Apr 22, 2014 tv0062118 de tv0062118 false false
Apr 14, 2014 tv0297566 de tv0297566 false false
Apr 14, 2014 tv0045201 de tv0045201 false false
Apr 18, 2014 tv0205896 de tv0205896 false false
Nov 26, 2013 tv0041767 de tv0041767 false false
Jan 9, 2014 tv0082332 de tv0082332 false false
Apr 14, 2014 tv0282070 de tv0282070 false false
Apr 14, 2014 tv0376236 de tv0376236 false false
Apr 26, 2014 tv0085964 de tv0085964 false false
Dec 31, 2013 tv0220450 de tv0220450 false false
interoud
innovation
© interoud 2013
info@interoud.com

Continuous integration

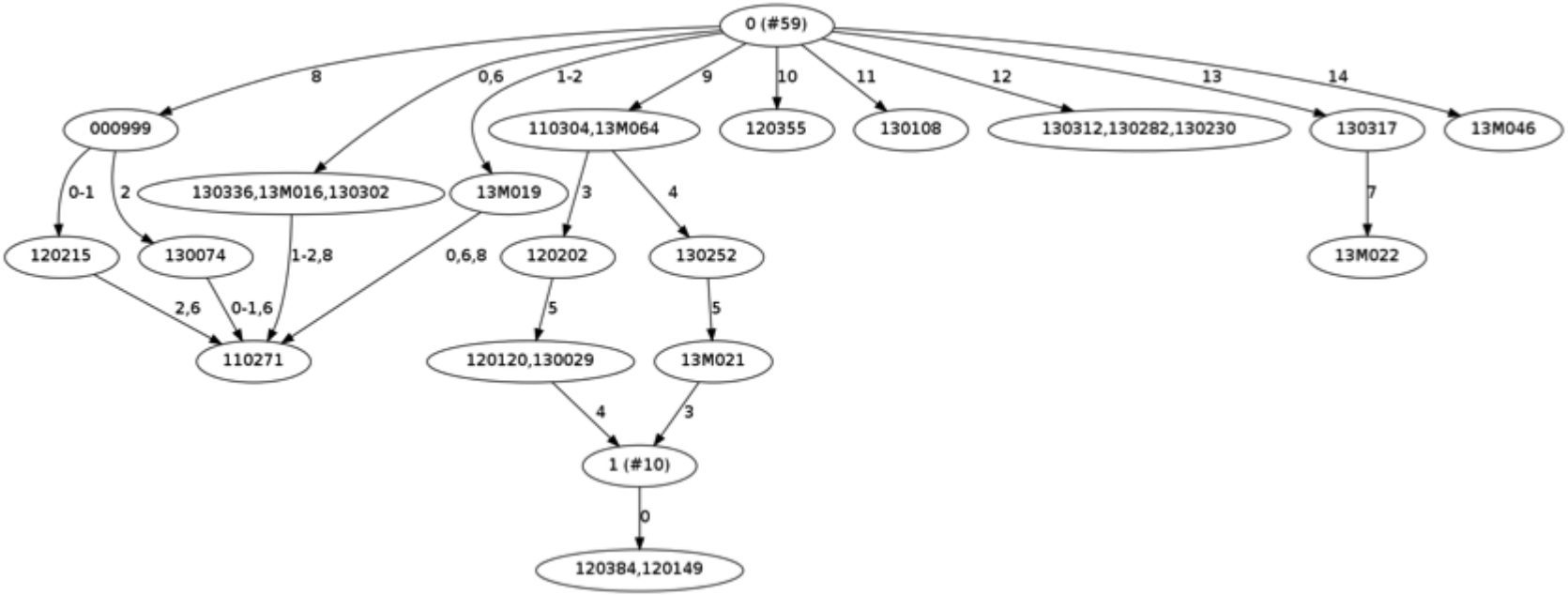


```
-ifdef(TEST).
-include_lib("eunit/include/eunit.hrl").
-compile(export_all).
-endif.

-record(mstore, {name, file, offset, size, index=gb_trees:empty(), next=0}).
-record(mset, {size, files=[], dir, metrics=gb_sets:new()}).

-define(OPTS, [raw, binary]).
-export([put/4, get/4, new/2, delete/1, close/1, open/1, metrics/1,
         fold/3]).

%% @doc Opens an existing mstore.

delete(MSet = #mset{dir=Dir}) ->
    close(MSet),
    {ok, Files} = file:list_dir(Dir),
    Files1 = [[Dir, 47 | File] || File <- Files],
    [file:delete(F) || F <- Files1],
    file:del_dir(Dir).

-spec open(Dir :: string()) -> {ok, #mset{}} | {error, not_found}.

open(Dir) ->
    case open_mstore([Dir | "/mstore"]) of
        {ok, FileSize, Metrics} ->
            {ok, #mset{size=FileSize, dir=Dir, metrics=Metrics}};
        _ ->
            {error, not_found}
    end.

new(FileSize, Dir) when is_binary(Dir) ->
```

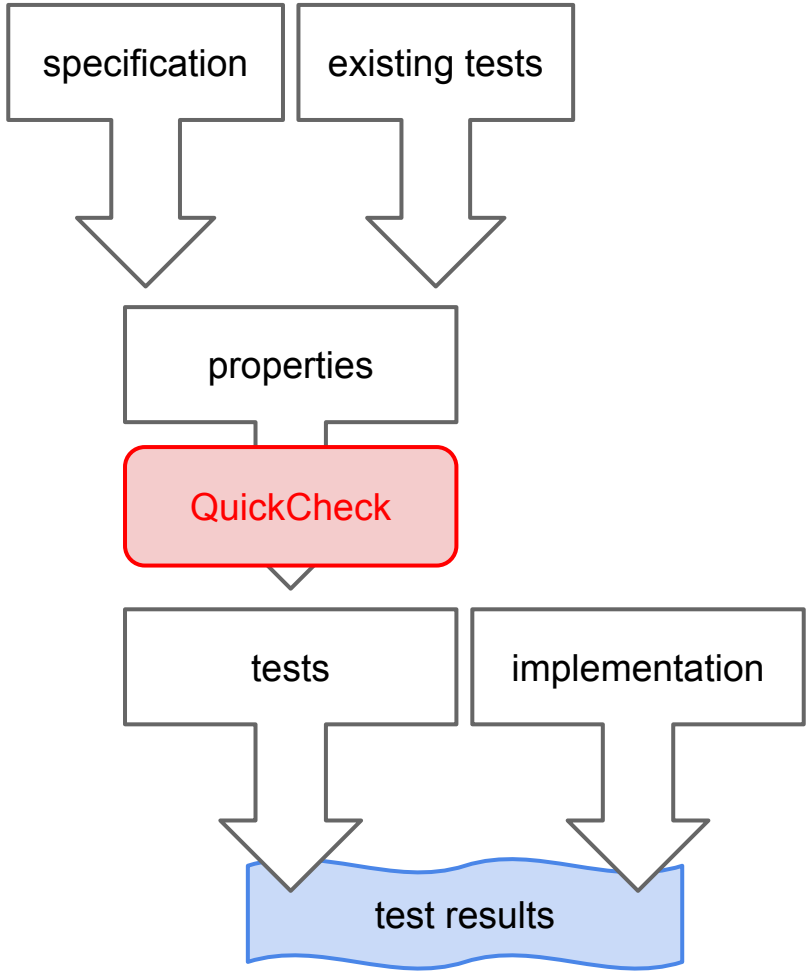
mstore_eqc:prop_gb_comp() : 60 calls

quickcheck-ci.com/cover/build/408/mstore.erl.html#

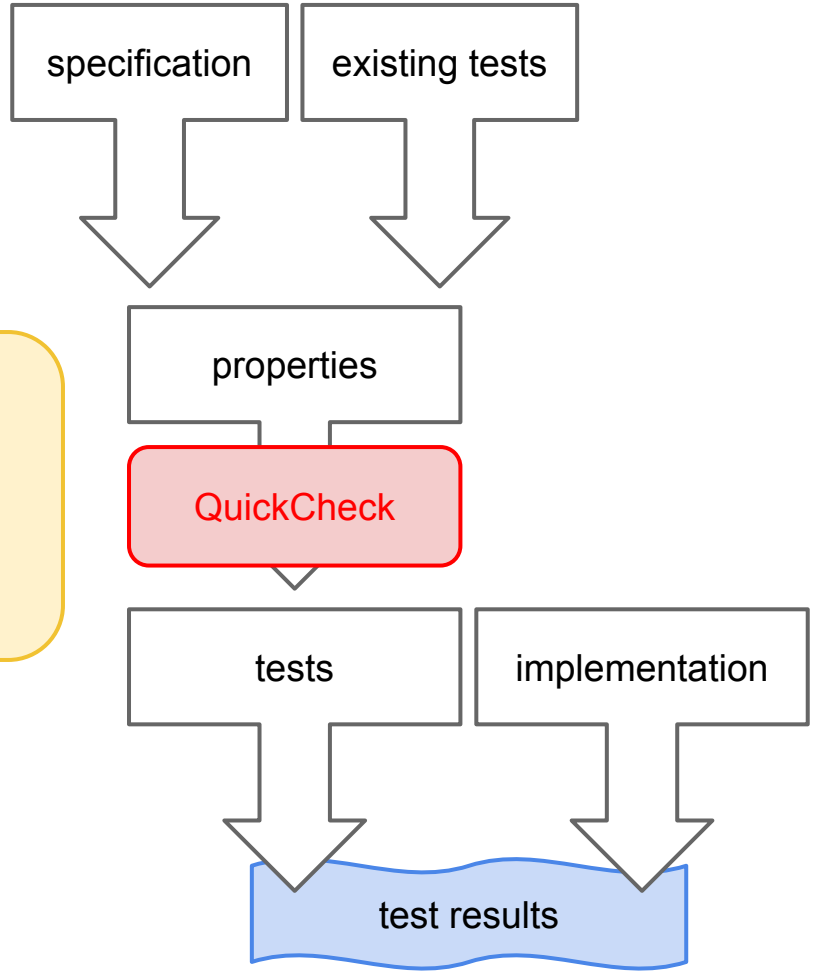
Comparing different implementations



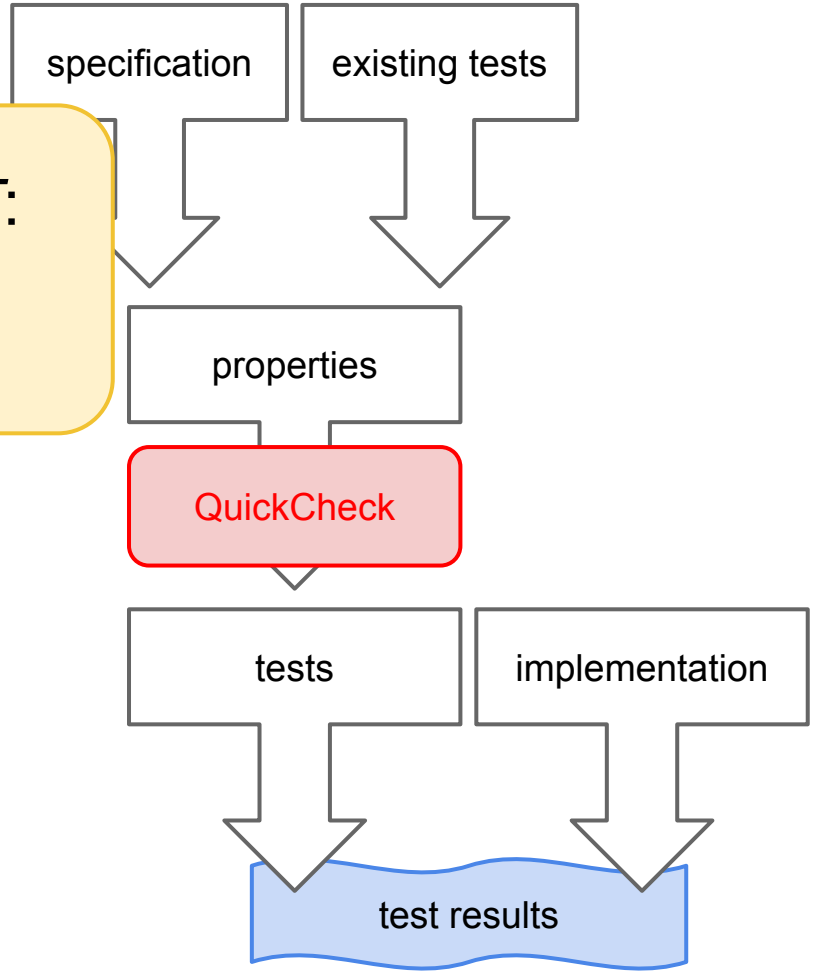
Results

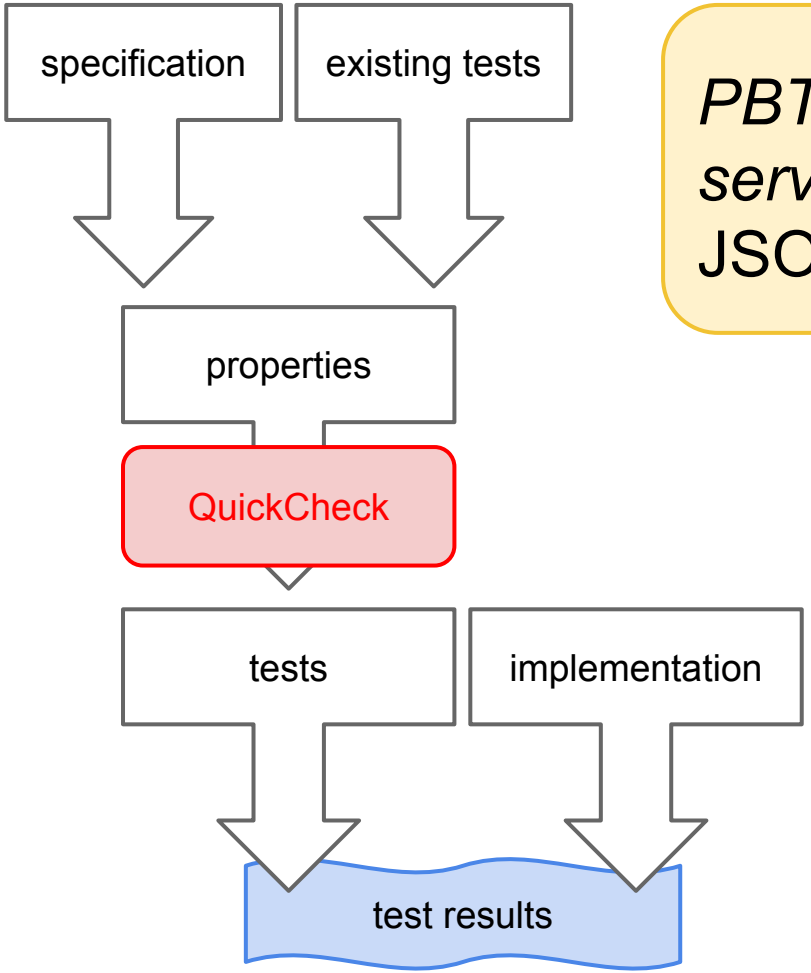

specification
existing tests
properties
QuickCheck
tests
implementation
test results

Scalable PBT: components and mocking

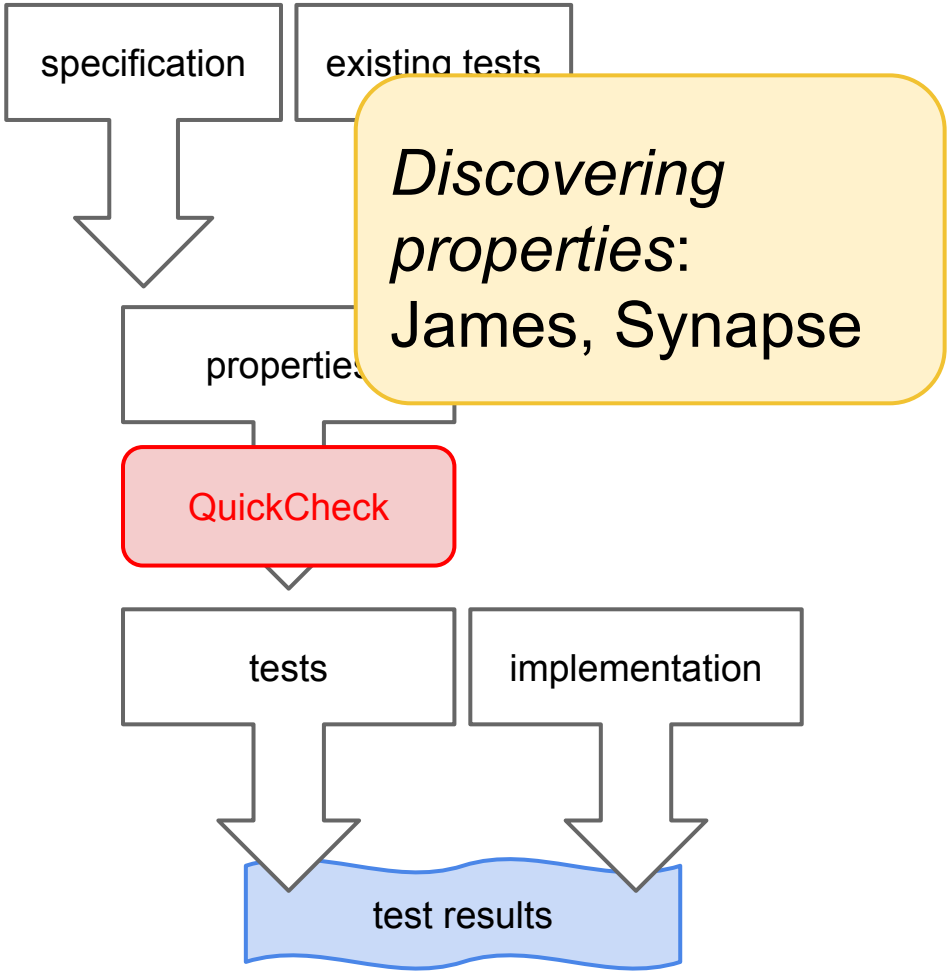


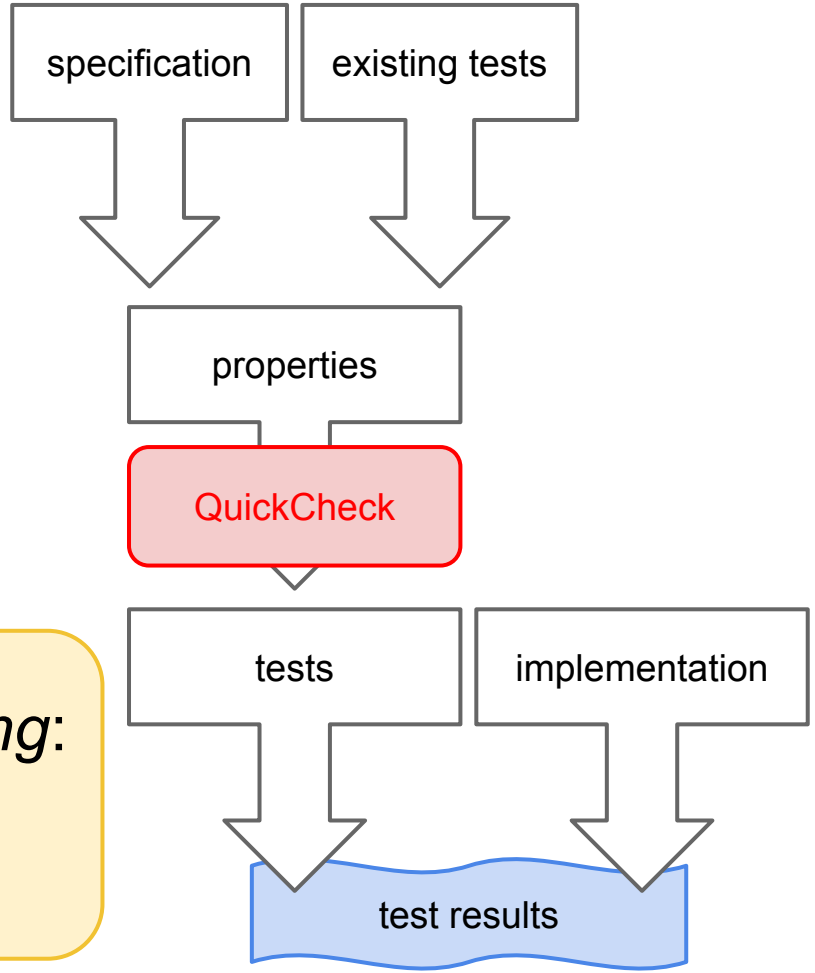
Accessible PBT:
ReadSpec,
GoodExamples



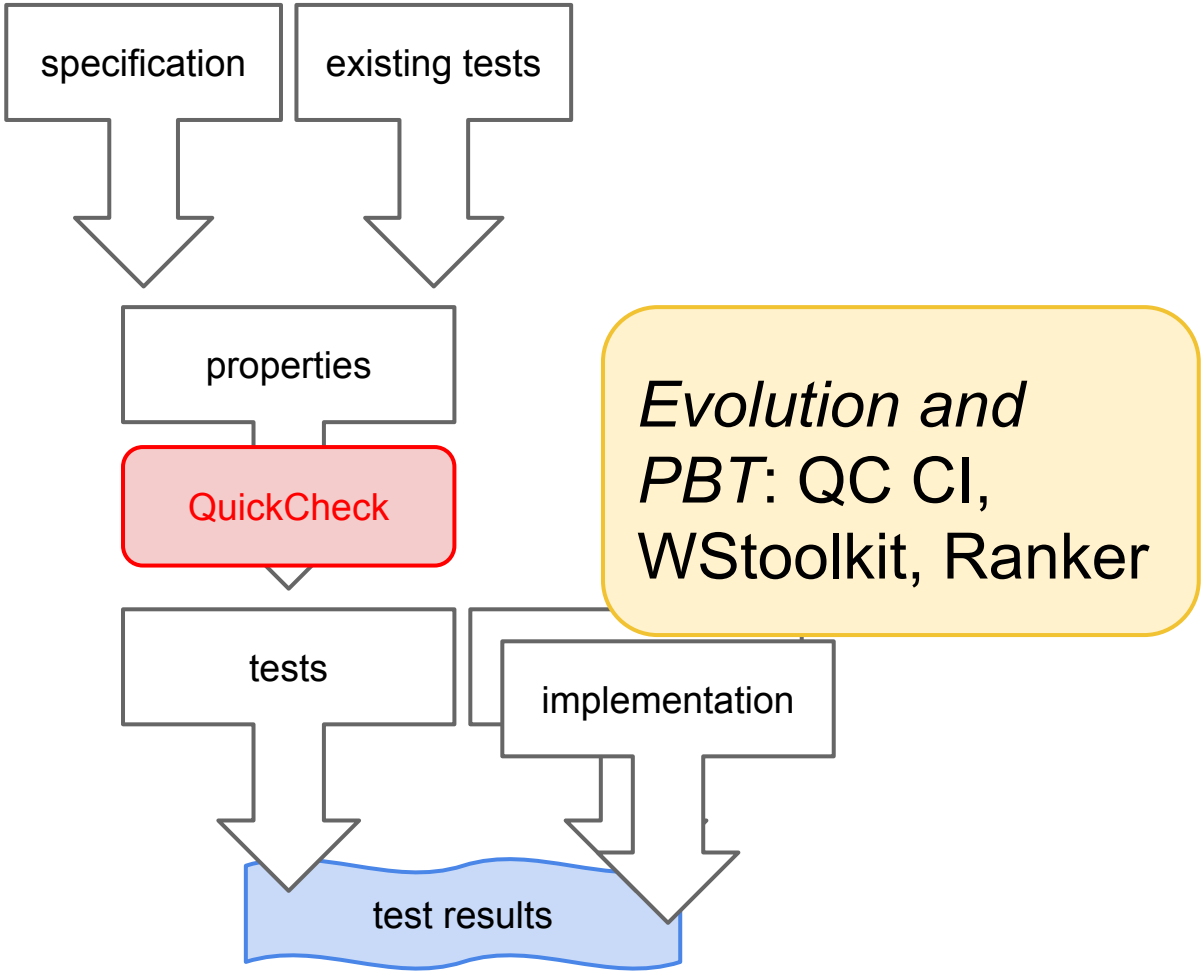


PBT for web services: WStoolkit, JSONgen


specification
existing tests
properties
QuickCheck
tests
implementation
test results
Discovering properties:
James, Synapse



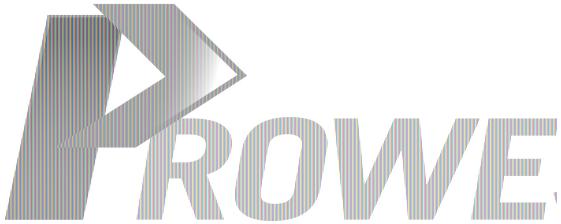
Improved testing:
Smother, Mu2,
FaultCheck


specification
existing tests
properties
QuickCheck
tests
implementation
test results
Evolution and PBT: QC CI, WStoolkit, Ranker

Acknowledgement

The Universities of Sheffield, Kent, A Coruña, Chalmers Technical University and the Polytechnic University of Madrid; Quviq AB, Interoud, Erlang Solutions Ltd and SP gratefully acknowledge the support of the European Commission for the PROWESS project, funded under Framework Programme 7.

Results



Scalable PBT: components, mocking

Accessible PBT: ReadSpec, GoodExamples

PBT for web services: WStoolkit, JSONgen

Discovering properties: James, Synapse

Improved testing: Smother, Mu2, FaultCheck

Evolution and PBT: QC CI, WStoolkit, Ranker

www.prowess-project.eu