Property-based Testing for non-functional requirements

Macías López

macias.lopez@udc.es

MADS Research Group – Universidade da Coruña (Galiza, Spain)

Erlang User Conference
Stockholm, June 9th 2014

Outline

1. Introduction
2. Functional testing
3. Non-functional testing
4. Feedback

Introduction

Why should we test software?

- It increases your **confidence** in the code you write.
- Tests could be used as **documentation**.
- It helps **finding bugs earlier**, so the impact is much lower.

Introduction

Why should we test software?

- It increases your **confidence** in the code you write.
- Tests could be used as **documentation**.
- It helps **finding bugs earlier**, so the impact is much lower.
- Reduce **costs**:
  - “50% project budget”
  - “At least 1/3 and probably more than 1/2 of the project budget”

Introduction

Remembering some concepts...

- **Sucessful** tests are those that **find bugs**.
- Testing **cannot prove** that the software **has not bugs**.
- Testing **cannot prove** that the software **fulfill its specification**.
- Sometimes, **more testing** implies finding **less bugs**.

Introduction

Good practices

- Tests should be **independent** from each other.
- Tests should be **repeatable**.
- Tests should be **guided by the specification**.
- After testing, the system should **remain as it was**.
- Test code should be **separated from the code** itself.

- **Static vs dynamic**
  - Software must be running or not
- **White-box vs black-box**
  - We need access to software internals or not
- **Positive vs negative**
  - Software testing in normal conditions or not

Introduction

Testing levels

Unit testing: isolate each part of the program and show that the individual parts fit the specification.

Integration testing: individual software modules are combined and tested as a group.

System testing: the whole software system is evaluated

Acceptance testing: the software we built fits bussiness requirements.

Introduction

Depending on what we want to test:

Introduction

Depending on what we want to test:

- **Functional testing:** *what* the software will do.
- **Non-functional testing:** related to requirements that describe *not* *what* the software will do, but *how* the software will do it.

Introduction

Depending on what we want to test:

- **Functional testing:** *what* the software will do.
- **Non-functional testing:** related to requirements that describe *not what* the software will do, but *how* the software will do it.

But in general, testing is identified by its functional side...

Functional testing

In the Erlang world

Functional testing

In the Erlang world

- **EUnit**

Functional testing

In the Erlang world

- **EUnit**
  - The classic xUnit approach
  - Test cases are implemented manually as part of test functions
  - Integrated with *rebar*.
  - Typical assertions
    - `assert(BoolExpr)`
    - `assertNot(BoolExpr)`
    - `assertMatch(Pattern, Expr)`
    - `assertEqual(Expected, Expr)`

Functional testing

In the Erlang world

- **EUnit**
  - The classic xUnit approach
  - Test cases are implemented manually as part of test functions
  - Integrated with *rebar*.
  - Typical assertions
    - `assert(BoolExpr)`
    - `assertNot(BoolExpr)`
    - `assertMatch(Pattern, Expr)`
    - `assertEqual(Expected, Expr)`

Example!

Functional testing

In the Erlang world

- **CommonTest**

Functional testing

In the Erlang world

- **CommonTest**
  - Automates the execution of test functions.
  - We can analyse past execution of test functions.
  - Includes coverage data.
  - Setting it up can be difficult.
  - Ideal for **integration testing**.

Functional testing

In the Erlang world

- **CommonTest**
  - Functions to setup and teardown the complete suite or test case
  - We can avoid certain testcases
  - Test suites can affect to different Erlang nodes.

Functional testing

In the Erlang world

- **CommonTest**
  - Functions to setup and teardown the complete suite or test case
  - We can avoid certain testcases
  - Test suites can affect to different Erlang nodes.

Example!

Functional testing

Property-based testing

Uses declarative statements to specify **properties** that the software needs to satisfy according to its **specification**.

Functional testing

Property-based testing

Uses declarative statements to specify **properties** that the software needs to satisfy according to its **specification**.

Using this approach:

- Test cases can be automatically derived from those properties.
- Test cases can be automatically run and diagnosed.

Functional testing

Property-based testing

Uses declarative statements to specify **properties** that the software needs to satisfy according to its **specification**.

Using this approach:

- Test cases can be automatically derived from those properties.
- Test cases can be automatically run and diagnosed.

The tools we use to perform PBT in Erlang:

- **QuickCheck** / **PropEr**

Property-based testing

The process

- **Define properties** for our code.
- **Run test cases** using QuickCheck/PropEr generators.
- Check whether the defined properties **hold or not**.

Property-based testing

The process

- **Define properties** for our code.
- **Run test cases** using QuickCheck/PropEr generators.
- Check whether the defined properties **hold or not**.
- One **interesting feature** of Quickcheck/PropEr:
  - When a failing test case is found, QuickCheck/PropEr automatically shrinks it to the smallest equivalent counterexample.

Property-based testing

The process

- **Define properties** for our code.
- **Run test cases** using QuickCheck/PropEr generators.
- Check whether the defined properties **hold or not**.
- One **interesting feature** of Quickcheck/PropEr:
  - When a failing test case is found, QuickCheck/PropEr automatically shrinks it to the smallest equivalent counterexample.
- Properties themselves are also **written in Erlang**.

Property-based testing

The process

- **Define properties** for our code.
- **Run test cases** using QuickCheck/PropEr generators.
- Check whether the defined properties **hold or not**.
- One **interesting feature** of Quickcheck/PropEr:
  - When a failing test case is found, QuickCheck/PropEr automatically shrinks it to the smallest equivalent counterexample.
- Properties themselves are also **written in Erlang**.
- **State-machine** based testing for complex systems.

Property-based testing

Simple example

"After we have applied the delete function to a list of numbers and an specific number, such number should not appear in the resulting list."

Property-based testing

Simple example

"After we have applied the delete function to a list of numbers and an specific number, such number should not appear in the resulting list."

Property in QuickCheck:

```
prop_lists_delete() ->
  ?FORALL(I, eqc_gen:int(),
    ?FORALL(List, eqc_gen:list(eqc_gen:int()),
      not lists:member(I, lists:delete(I, List)))).
```

Property-based testing

Simple example

"After we have applied the delete function to a list of numbers and an specific number, such number should not appear in the resulting list."

Property in QuickCheck:

```
prop_lists_delete() ->
  ?FORALL(I, eqc_gen:int(),
    ?FORALL(List, eqc_gen:list(eqc_gen:int()),
      not lists:member(I, lists:delete(I, List)))).
```

Property-based testing

Simple example

"After we have applied the delete function to a list of numbers and an specific number, such number should not appear in the resulting list."

Property in QuickCheck:

```
prop_lists_delete() ->
  ?FORALL(I, eqc_gen:int(),
    ?FORALL(List, eqc_gen:list(eqc_gen:int()),
      not lists:member(I, lists:delete(I, List)))).
```

Property-based testing

Simple example

Sample of generator output:

```
2> eqc_gen:sample(eqc_gen:list(eqc_gen:int())).

[10,-2,-9,6]

[-8,6,-11]

[-7,-3,7]

[3]

[]

[11,8,14,12,3]

[-4]

...
```

Property-based testing

Simple example

Running test cases means running the property.

Property-based testing

Simple example

Running test cases means running the property.

Running the property:

```
3> eqc:quickcheck(test:prop_lists_delete()).

.....................................................

OK, passed 100 tests

true
```

Property-based testing

Simple example

Running test cases means running the property.

Running the property:

```
3> eqc:quickcheck(test:prop_lists_delete()).

.....................................................

OK, passed 100 tests

true
```

QuickCheck tutorial by Thomas Arts

Functional testing

Functional testing **is not enough**...

Functional testing

Functional testing **is not enough**...

- In specific domains **when** the results are produced or **how long** they take to be available can be decisive.

Functional testing

Functional testing **is not enough**...

- In specific domains **when** the results are produced or **how long** they take to be available can be decisive.
- Normally these requirements **arise at very later stages** of development process.

Functional testing

Functional testing **is not enough**...

- In specific domains **when** the results are produced or **how long** they take to be available can be decisive.
- Normally these requirements **arise at very later stages** of development process.
- Or even when the system is deployed!

Non-functional testing

Many requirements can be considered as **non-functional** (also known as **extra-functional**):

Non-functional testing

Many requirements can be considered as **non-functional** (also known as **extra-functional**):

- Performance
- Dependability
- Security
- Reliability
- ...

Non-functional testing

Many requirements can be considered as **non-functional** (also known as **extra-functional**):

- Performance
- Dependability
- Security
- Reliability
- ...

These requirements are generally informally stated, they are often contradictory. It is difficult to keep traces of what we have tested.

Non-functional testing

The question

Can we property-based test any of these non-functional requirements?

Non-functional testing

The question

Can we property-based test any of these non-functional requirements?

Work in progress

Library for testing non-functional requirements to be used in combination

with property-based testing tools

Non-functional testing

Among all possible non-functional requirements, select one and implement a prototype of properties.

Non-functional testing

Among all possible non-functional requirements, select one and implement a prototype of properties.

Performance

Non-functional testing

Among all possible non-functional requirements, select one and implement a prototype of properties.

Performance

- From a **black-box** approach

Non-functional testing

Among all possible non-functional requirements, select one and implement a prototype of properties.

Performance

- From a **black-box** approach
- Possible properties
  - “The response time is less than a value T”
  - “The average response time of N requests is less than a value T”

Non-functional testing

Among all possible non-functional requirements, select one and implement a prototype of properties.

Performance

- From a **black-box** approach
- Possible properties
  - “The response time is less than a value T”
  - “The average response time of N requests is less than a value T”

Example!

Performance testing

Before running the tests, set a specific **workload** in the system. Possible integration with:

- Tsung
- Megaload

Performance testing

Before running the tests, set a specific **workload** in the system. Possible integration with:

- Tsung
- Megaload

Use `?SETUP` macro.

PBT is partially integrated in Megaload. Diana Corbacho's tutorial.

Performance testing

Queue example

Due to the internal implementation of queues in Erlang, the response time of removing the first element of the queue after a lot of insertions, should be slightly high.

Performance testing

Queue example

Due to the internal implementation of queues in Erlang, the response time of removing the first element of the queue after a lot of insertions, should be slightly high.

- We can build a **state-machine in combination with our library** to check that.

Performance testing

Queue example

Due to the internal implementation of queues in Erlang, the response time of removing the first element of the queue after a lot of insertions, should be slightly high.

- We can build a **state-machine in combination with our library** to check that.

Example!

Feedback

Feedback

- More properties related to performance?

Feedback

- More properties related to performance?
  - Simulate incresing memory, disk or network usage

Feedback

- More properties related to performance?
  - Simulate incresing memory, disk or network usage
- Move to grey-box testing to get info at earlier stages?

Feedback

- More properties related to performance?
  - Simulate incresing memory, disk or network usage
- Move to grey-box testing to get info at earlier stages?
- What non-functional requirement would you like to have tools for testing it?

Feedback

- More properties related to performance?
  - Simulate incresing memory, disk or network usage
- Move to grey-box testing to get info at earlier stages?
- What non-functional requirement would you like to have tools for testing it?
- How could PBT help when testing that requirements?

Audience ! thanks