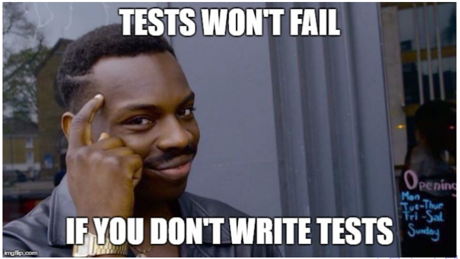
Testing Software

Wanderson Ferreira

Programmer

wanderson.ferreira@captalys.com.br



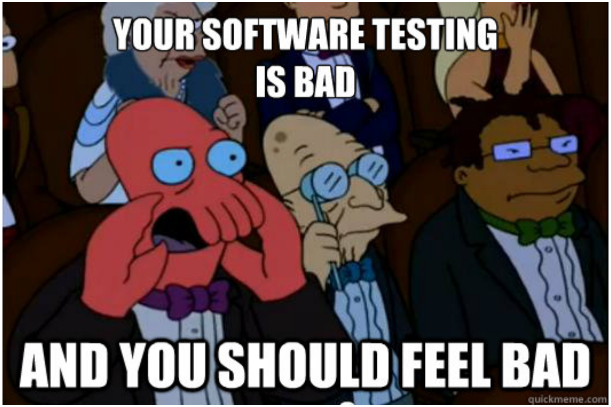
Overview

Principles

“We say, ‘I can make a change because I have tests’ Who does that? Who drives their car around banging against the guard rails?”
- **Rich Hickey**

- Reduces the *probability* of undiscovered defects
- Tests does not guarantee correctness
- Increase developers **trust index** to work on a system
- Ease to test systems, probably nailed a good **design**
- ...
- *You are the expert, figure it out!*
- *Don’t write tests!*

Cool, therefore, how are we doing?



Why is so?

Quality is never a priority

- You are paid to **ship** code! Not to build reliable solutions!
- MVPs are always production ready code
- Not good enough is good enough
- Sprint to write test or the next feature? You know.
- There are no joy in testing stuff. Puzzle solved!

They are the bad guys, right?

You cannot write good tests, even if chance was given, why not?

- You don't know your **business domain** well enough
- How about tests techniques to use?
- Your only reference to good software testing is called **TDD**

What can we do?

“Programming is not about typing... **it’s about thinking**”
- **Rich Hickey**

- Think and discuss about good methodologies for our environment
- Find ways to **measure** the impact of *having* tests
- Find ways to **measure** the impact of *not having* tests
- Study and practice
- Slowly implement and watch the impact on metrics

Flavours

1. Unit Tests
2. Integration Tests
3. e2e
4. Acceptance Tests

Unit Tests

Unit tests make usage of White Box Testing: is a method where the internal structure of the code being tested is known to the tester.

- First level of software testing
- Performed by **software developers** itself
- Do not unit test everything
- Benefits:
    - Code reuse. To make unit test possible, code must be modular
    - Development is faster
    - Helps during debugging. Look at where tests failed

Integration Tests

Integration tests make usage of White Box Testing.

- Second level of software testing
- Performed by **software developers** itself
- Individual units are combined and tested together
- You have a feature that needs to talk to 3 components to complete. Test the feature behavior

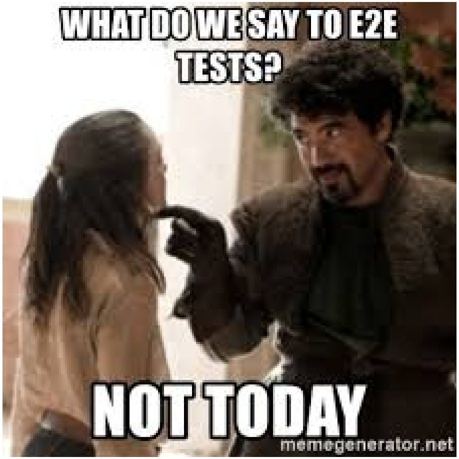
Integration Tests - Benefits

e2e Tests

Ensure that the **integrated components** of an application works as expected, the entire application is tested in a **real-world scenario**.

- Extremely complicated on Microservices architecture [MS]
- Very debate-able subject on MS
- Several companies reporting:
  - Focus on Unit test and Integration Test
  - Ensure your infrastructure can help you with **canary** deployment
  - Embrace **testing in production**

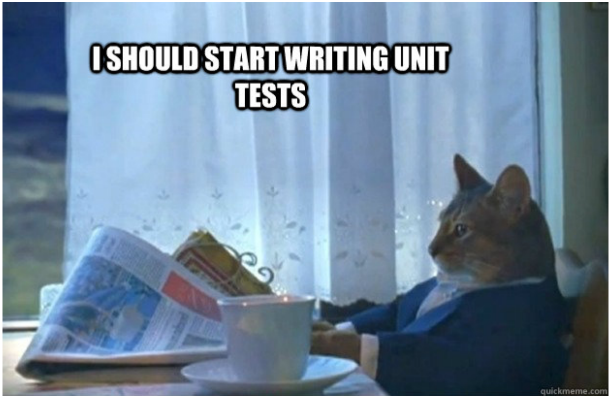
e2e Tests



Acceptance Tests

- Last level of software testing
- Performed by other members of the team e.g. Product Manager, Product Owner, QA, etc.

Ok, let's face it. And now what?



Study case - What if, we worked with Credit?

```clojure
(defn valid?
  [credit]
  (let [blacklist #{"Luis" "Bruno"}]
    (and
     (> (:credit/value credit) 0)
     (not (contains? blacklist (:credit/owner credit)))
     (= (:credit/buyer credit) "Captalys"))))

(defn save! [credit]
  (when (valid? credit)
    (-> (d/transact (:connection database/server) [credit])
        deref
        :tempids
        first
        second)))

(defn update-payment [credit amount-paid]
  (let [adding (fnil + 0)]
    (update credit :credit/paid-value adding amount-paid)))

(defn payment! [credit-owner amount-paid]
  (let [db (d/db (:connection database/server))
        credit (d/q '[:find (pull ?e [*])
                      :in $ ?owner
                      :where
                      [?e :credit/owner ?owner]]
                    db credit-owner)]
    (-> credit
        ffirst
        (update-payment amount-paid)
        vector
        (as-> upd-credit (d/transact (:connection database/server) upd-credit))
        deref)))
```

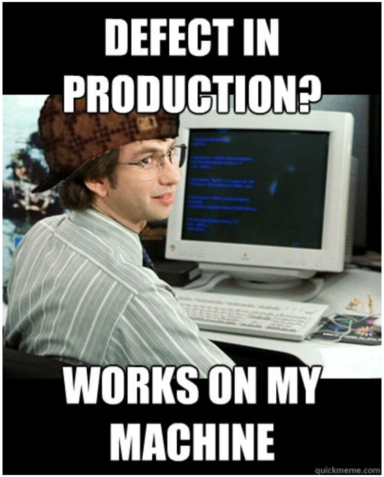
Study case - How to test?

```
(comment
  (def db (d/db (:connection database/server)))

  (def credit {:credit/value 1000.0
               :credit/owner "Wand"
               :credit/buyer "Captalys"
               :credit/future-value 1500.00})

  (save! credit)

  ;; verify if credit is inside the database: query for owner
  (d/q '[:find (pull ?e [*])
         :where
         [?e :credit/owner "Wand"]]
       db)

  (payment! "Wand" 200.00))
```

Study case - What we just did?

1. Developed business logic functions *~~[with no docs, shame]~~*
2. Picked singular example to test *~~[hope to choose wisely]~~*
3. Connected to a real database *~~[hope not to be prod]~~*
4. Left database in bad state *~~[hope no one uses this afterwards]~~*

How are you feeling?



Study case - there must be a better way!

1. Developed business logic functions ~~*[with no docs, shame]*~~

Study case - there must be a better way!

1. Developed business logic functions ~~[with no docs, shame]~~

```
(ns study-test.credit
  (:require [clojure.spec.alpha :as s]))

(s/def :credit/owner string?)
(s/def :credit/value float?)
(s/def :credit/buyer string?)
(s/def :credit/future-value float?)

(s/def ::credit-spec
  (s/keys :req [:credit/owner
                :credit/value
                :credit/buyer
                :credit/future-value]))
```

Study case - there must be a better way!

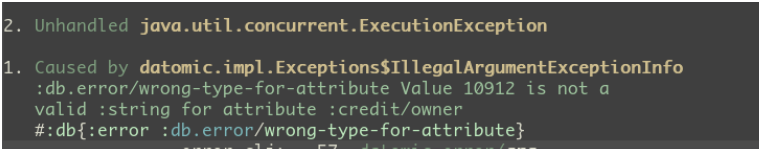
1. Developed business logic functions ~~*[with no docs, shame]*~~

```clojure
(defn valid?
  [credit]
  {:pre (s/valid? ::credit-spec)}
  (let [blacklist #{"Luis" "Bruno"}]
    (and
     (> (:credit/value credit) 0)
     (not (contains? blacklist (:credit/owner credit)))
     (= (:credit/buyer credit) "Captalys"))))
```

Study case - there must be a better way!

1. Developed business logic functions *~~[with no docs, shame]~~*

```clojure
(def credit {:credit/value 1000.0
             :credit/owner 10912
             :credit/buyer "Captalys"
             :credit/future-value 1500.00})

(valid? credit)
```



```
2. Unhandled java.util.concurrent.ExecutionException

1. Caused by datomic.impl.Exceptions$IllegalArgumentExceptionInfo
   :db.error/wrong-type-for-attribute Value 10912 is not a
   valid :string for attribute :credit/owner
   #:db{:error :db.error/wrong-type-for-attribute}
```

Study case - Next one!

1. Developed business logic functions
2. Picked singular example to test ~~*[hope to choose wisely]*~~

Study case - there must be a better way!

1. Developed business logic functions
2. Picked singular example to test ~~[hope to choose wisely]~~

```
(s/fdef valid?
  :args (s/cat :credit ::credit-spec)
  :ret boolean?)

(s/exercise-fn `valid?)
([(#:credit{:owner "", :value -2.0, :buyer "", :future-value -1.0}) false] [(#:
 [(#:credit{:owner "81", :value 0.75, :buyer "0", :future-value -0.5}) false]
 [(#:credit{:owner "X", :value 2.25, :buyer "", :future-value -1.25}) false])
```

Study case - there must be a better way!

1. Developed business logic functions
2. Picked singular example to test ~~[hope to choose wisely]~~

```
(s/fdef valid?
  :args (s/cat :credit ::credit-spec)
  :ret boolean?)

(spec-test/check `credit/valid?)
({:spec #object[clojure.spec.alpha$fspec_impl$reify__2524 0x1a1a6380 "clojure.s
  :clojure.spec.test.check/ret
  {:result true, :pass? true, :num-tests 1000, :time-elapsed-ms 314, :seed 1570
```

Study case - Next one!

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database ~~*[hope not to be prod]*~~

Study Case - Mocking Database

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database ~~[hope not to be prod]~~

```
(mount/defstate datomic-test
  :start (database/start-datomic! cfg)
  :stop (database/stop-datomic! cfg))

(facts "Let's talk to databases now!"

  (mount/start-with-states {#'database/server #'datomic-test})

  (let [credit (first (gen/sample (s/gen ::credit/credit-spec)))]
    (fact "Saving the credit into datomic TEST databse"
      (credit/save! credit)) => truthy)
  (mount/stop))
```

Study Case - Mocking Database

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database ~~[hope not to be prod]~~

```
user>
user>
user> (go)
19-10-09 04:31:47 arch INFO [study-test.database:9] - creating the PROD database
19-10-09 04:31:47 arch INFO [study-test.database:13] - schemas being created in PROD database
;; => :ready{:sym study-test.credit/valid?}
19-10-09 04:31:59 arch INFO [study-test.database:9] - creating the TEST database
19-10-09 04:31:59 arch INFO [study-test.database:13] - schemas being created in TEST database
19-10-09 04:31:59 arch INFO [study-test.database:20] - {:env "TEST", :uri "datomic:mem://credit_TEST"}
19-10-09 04:31:59 arch INFO [study-test.database:21] - deleting TEST database
user>
```

Study Case - Last one!

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database
4. Left database in bad state ~~*[hope no one uses this afterwards]*~~

Study Case - Setup and TearDown?

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database
4. Left database in bad state ~~[hope no one uses this afterwards]~~

```clojure
(facts "Ok, improving on the above facts.."
  (with-state-changes [(before :facts (mount/start-with-states {#'database/serv
                       (after :facts (mount/stop))]
    (let [credit (first (gen/sample (s/gen ::credit/credit-spec)))]
      (fact "Saving the credit into datomic TEST databse"
        (credit/save! credit)) => truthy)))
```

Study Case - Setup and TearDown?

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database
4. Left database in bad state ~~*[hope no one uses this afterwards]*~~

```
(facts "Ok, improving on the above facts.."
  (with-state-changes [(before :contents (mount/start-with-states {#'database/s
                        (after :contents (mount/stop))]
    (let [credit (first (gen/sample (s/gen ::credit/credit-spec)))]
      (fact "Saving the credit into datomic TEST databse"
        (credit/save! credit)) => truthy)))
```

Study Case - Setup and TearDown?

1. Developed business logic functions
2. Picked singular example to test
3. Connected to a real database
4. Left database in bad state

BOOM
NAILED IT!
imgflip.com

Next steps

There is a Github repository with all the code: Studing Tests

1. Please, play around
2. Submit PR to practice writing tests on the other functions
3. Call for help on the project:
    1. Mock HTTP Request
    2. Mock RabbitMQ Pub/Sub
    3. Implementing Custom Generators on Specs
    4. Creating fixtures with Midje
    5. Examples of Stubbing with Midje
4. Please, we need a Python project like this one!!

Thanks