Lazytest
Better Living Through Protocols

Stuart Sierra

Clojure NYC
April 15, 2010

@stuartsierra #clojure

clojure.core: tests as metadata

```
(defn add
  ([x y] (+ x y))
  {:test (fn [] (assert (= 7 (add 3 4))))})


(test #'add)
:ok
```

clojure.test assertions

```
(is (= 4 (+ 2 2)))
true

(is (= 5 (+ 2 2)))
FAIL in ...
expected: (= 5 (+ 2 2))
  actual: (not (= 5 4))

(is (instance? Integer (/ 3 5)))
FAIL in ...
expected: (instance? Integer (/ 3 5))
  actual: clojure.lang.Ratio
```

clojure.test assertions

```
(macroexpand '(is (= 7 (+ 3 4)))

(try
 (let [values# (list 7 (+ 3 4))
       result# (apply = values#)]
   (if result#
     (report {:type :pass, ...})
     (report {:type :fail, ...})
 (catch java.lang.Throwable t#
   (report {:type :error, ...})))
```

clojure.test assertions

```
(is (thrown? ArithmeticException (/ 1 0)))
#<ArithmeticException java.lang.ArithmeticException: Divide
by zero>


(is (thrown? IllegalArgumentException (/ 1 0)))
ERROR in ..
expected: (thrown? IllegalArgumentException (/ 1 0))
  actual: java.lang.ArithmeticException: Divide by zero
 at clojure.lang.Numbers.divide (Numbers.java:138)
    user/eval (NO_SOURCE_FILE:1)
    clojure.lang.Compiler.eval (Compiler.java:4580)
    clojure.core/eval (core.clj:1728)
    swank.commands.basic/eval_region (basic.clj:36)
```

clojure.test tests in place

```
(with-test
    (defn add [x y] (+ x y))
  (is (= 7 (add 3 4)))
  (is (= 8 (add 2 2))))

(run-tests)
Testing user

FAIL in (add) ...
expected: (= 8 (add 2 2))
  actual: (not (= 8 4))

Ran 1 tests containing 2 assertions.
1 failures, 0 errors.
```

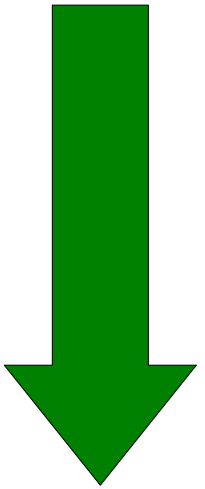
clojure.test tests in isolation

```
(deftest addition
  (is (= 4 (add 2 2)))
  (is (= 7 (add 3 4)))
  (is (= 9 (add 5 5))))
```

```
(addition)
FAIL in (addition) ...
expected: (= 9 (add 5 5))
  actual: (not (= 9 10))
```

assertions with shared structure

```
(deftest addition
  (are [sum x y] (= sum (add x y))
       4   2 2
       7   3 4
       9   5 5))
```



```
(deftest addition
  (is (= 4 (add 2 2)))
  (is (= 7 (add 3 4)))
  (is (= 9 (add 5 5))))
```

clojure.test doc strings

```
(deftest foo
  (testing "The foo function"
     (testing "when called with no args"
        (is (thrown? Exception (foo))
            "should throw an Exception")))))
```

```
"The foo function when called with no args should
throw an Exception."
```

Fixtures

```
(defn my-fixture [f]
  ... setup ...
  (f)
  ... teardown ...)

(use-fixtures :each my-per-test-fixture)

(use-fixtures :once my-general-fixture)
```

clojure.test in the wild

```
(deftest test-set-system-properties
  (testing "set and then unset a property using keywords"
    (let [propname :contrib.test-set-system-properties]
      (is (nil? (get-system-property propname)))
      (set-system-properties {propname :foo})
      (is (= "foo") (get-system-property propname))
      (set-system-properties {propname nil})
      (is (nil? (get-system-property propname))))))
```

clojure.test/are in the wild

```
(deftest test-last
  (are [x y] (= (last x) y))
    nil nil
    []  nil
    [1] 1
    ...

(deftest test-last
  (are [x y] (= x y)
    (last nil) nil
    (last [])  nil
    (last [1]) 1
    ...
```

Clojure 1.1 Pre/Postconditions

```
(defn my-function [x y]
  {:pre  [(integer? x) (pos? y)]
   :post [(integer? %) (> % 1000)]}
  ... the function body ...)
```

lazytest

- lazy test execution
- parallel test execution
- separate setup/teardown from assertions
- separate running tests from reporting results
- no dynamic binding

circumspec (Stuart Halloway)

- continuous testing
- regression testing
- ANSI-colored output
- BDD-style

old lazytest context/test/suite

```
(defcontext context-one [] 1)

(deftest my-test [x context-one]
   (pos? x) (= x 1))

(defsuite long-suite []
   my-test
   another-test
   a-third-test)
```

new lazytest: spec/is

```
(spec simple-addition
   (is (= 2 (+ 1 1))
       (= 4 (+ 2 2))))


(spec confused-addition
   (is "Two plus two is four"
       (= 4 (+ 2 2))
       "Two plus two is five?!"
       (= 5 (+ 2 2))))
```

nested specs

```
(spec minus "The minus function"
  (spec "when called with one argument"
    (spec "negates that argument"
      (is (= -1 (- 1))
          (= -2 (- 2)))))
  (spec "when called with two arguments"
    (spec "subtracts"
      (is (= 0 (- 5 5))
          "2 from 3 to get 1"
          (= 1 (- 3 2))))))
```

named nested specs

```
(spec minus "The minus function"
  (spec one-arg "when called with one argument"
    (spec negates "negates that argument"
      (is (= -1 (- 1))
          (= -2 (- 2)))))
  (spec two-arg "when called with two arguments"
    (spec subtracts "subtracts"
      (is (= 0 (- 5 5))
          "2 from 3 to get 1"
          (= 1 (- 3 2))))))


;; Call (minus) to run all the specs.
;; Call (subtracts) to run just that spec.
```

running specs

```
user=> (spec-report (the-specs))

Running specs at Thu Apr 15 13:05:37 EDT 2010
.....................................
Ran 45 assertions.
0 failures, 0 errors, 0 pending
```

failure reporting

```
(spec bad-spec "Bad arithmetic"
  (is "thinks 2 and 2 make 5"
      (= 5 (+ 2 2))))

user=> (spec-report (bad-spec))

Running bad-spec at Thu Apr 15 13:09:45 EDT 2010

FAIL
Doc:  Bad arithmetic thinks 2 and 2 make 5
Form: (= 5 (+ 2 2))
File: foo.clj
Line: 7

Ran 1 assertions.
1 failures, 0 errors, 0 pending
```

attaching spec metadata

```
;; file src/test/com/example/foo_spec.clj
(ns com.example.foo-spec
  (:use com.stuartsierra.lazytest))

(describe *ns* "The Foo library"
  (spec "should work" ...))
```

```
;; file src/main/com/example/foo.clj
(ns com.example.foo
    {:spec com.example.foo-spec})
```

```
user=> (run-spec 'com.example.foo)
```

continuous testing

```
user=> (def watcher (watch-spec "src"))


Running specs at Thu Apr 15 13:29:16 EDT 2010
.....................................
Ran 45 assertions.
0 failures, 0 errors, 0 pending
```

continuous testing

```
;; add a file com/stuartsierra/foo_spec.clj
(ns com.stuartsierra.foo-spec
  (:use [com.stuartsierra.lazytest]))

(describe *ns* "The Foo library")
```

```
Running specs at Thu Apr 15 13:33:15 EDT 2010
PENDING
NS:   com.stuartsierra.foo-spec
Doc:  The Foo library
Form: (com.stuartsierra.lazytest/spec)
File: com/stuartsierra/foo_spec.clj

Ran 0 assertions.
0 failures, 0 errors, 1 pending
```

continuous testing

```
;; edit the file, save
(ns com.stuartsierra.foo-spec
  (:use [com.stuartsierra.lazytest]))

(describe *ns* "The Foo library"
  (is (= 1 1)))
```

```
Running specs at Thu Apr 15 13:33:15 EDT 2010
.
Ran 1 assertions.
0 failures, 0 errors, 0 pending
```

contexts

```
(defcontext the-context "docstring?" []
   ... body of "before" function ...
   ... returns some state ...
   :after [x]
   ... body of "after" function ...
   ... the state is in 'x' ...
   ... return value is ignored ...)


(spec ...
  (given [z the-context]
    (spec ...
      (is (= z ...
```

composed contexts

```
(defcontext two-refs []
  [(ref 1) (ref 1)])

(defcontext buncha-threads [rs two-refs]
  (let [[ra rb] rs]
    (doall
     (for [i (range 50), f [inc dec]]
       (doto (Thread. #(dosync (alter ra f)
                               (alter rb f)))
         (.start)))))
  :after [threads]
  (doseq [t threads]
    (.stop t)))
```

composed contexts

```
(describe *ns*
 (spec ref-stress-test
   "Two refs, updated and read in transactions"
   (given [rs two-refs
           tt buncha-threads]
     (is "should always have consistent values."
         (let [[ra rb] rs]
           (every? true?
              (for [i (range 100000)]
                (let [[a b] (dosync [@ra @rb])]
                  (= a b)))))))))
```

lazytest protocols

```
(defprotocol TestInvokable
  (invoke-test [t active]))

(defprotocol TestResult
  (success? [r])
  (pending? [r])
  (error? [r])
  (container? [r]))
```

lazytest datatypes

```
(defrecord TestResultContainer [source children]
  TestResult
    (success? [this] (every? success? children))
    (pending? [this] (if (seq children)
                       false true))
    (error? [this] false)
    (container? [this] true))

(defrecord TestFailed [source states]
  TestResult
    (success? [this] false)
    (pending? [this] false)
    (error? [this] false)
    (container? [this] false))
```

lazytest datatypes

```
(defrecord SimpleAssertion [pred]
  clojure.lang.IFn
    (invoke [this] (invoke-test this {}))
  TestInvokable
    (invoke-test [this active]
      (try
        (if (pred)
          (TestPassed. this nil)
          (TestFailed. this nil))
        (catch Throwable t
          (TestThrown. this nil t)))))
```

lazytest datatypes

```
(defrecord ContextualAssertion [contexts pred]
  clojure.lang.IFn
    (invoke [this] (invoke-test this {}))
  TestInvokable
    (invoke-test [this active]
      (let [merged (reduce open-context ...)
            states (map merged contexts)]
        (try
         (if (apply pred states)
           (TestPassed. this states)
           (TestFailed. this states))
         (catch Throwable t
           (TestThrown. this states t))
         (finally
          (close-local-contexts ...))))))
```

"given" macro

```
(given [x the-context]
  ...)

;; expands to:

(let [#^{::given true} x the-context]
  ...)
```

"is" macro

```
(is (= x 1))

;; expands to:

(let [givens (filter #(::given (meta %))
                     (keys &env))]
  ;; if there are givens
  (ContextualAssertion. (fn [x] (= x 1))
  ;; otherwise, there are no givens
  (SimpleAssertion. ...)
```

"spec" macro

```
(spec foo "the foo spec" ...)

;; expands to:

((fn [] (SimpleContainer. ...))

;; when a name is given:

(intern *ns* the-name the-spec)
```

"are"

```
(are [x y z] (= (+ x y) z)
   2  2  4
   3  2  5
   8 -1  7)

;; Equivalent to:

(is (= (+ 2 2) 4)
    (= (+ 3 2) 5)
    (= (+ 8 -1) 7))
```

"spec-do"

```
(spec-do foo "the foo spec" []
  ;; ... arbitrary code ...
  (assert (= foo bar))
  ;; ... more code ...
  (assert (still = foo bar))
  ;; ... cleanup ...
  )
```

lazytest todo

- parallel testing
- repeat assertions with different contexts
- regression tests (for-all)
- test-runner GUI
- growl / libnotify
- syntax for once/each contexts
- release!

Me!

- Web: http://stuartsierra.com/
- Email: mail@stuartsierra.com
- Twitter: @stuartsierra
- Github: http://github.com/stuartsierra
- IRC: stuartsierra in #clojure