NST: A unit testing system for Common Lisp
- or -
Honning the tester’s vocabulary

John Maraist
SIFT, LLC
Minneapolis, Minnesota, USA

International Lisp Conference 2010
What’s in a test system?

At some level, all (Lisp) test systems let you define four artifacts:

- **Fixtures** establishing consistent test environments.
- **Criteria** that describe correct behavior.
- **Tests** applying a criterion to specific exemplars.
- **Groups** of related tests.
Fixtures
Faithful context reproduction

• Provide a convenient, consistent environment for test evaluation.
  • Local name bindings.
  • Setup- and cleanup routines.

• For example:
  • Database or other resource configuration.
  • Interchangeable sets of bindings.
Criteria

What is “right”?

• Non-nil vs. nil.
• Normal completion vs. thrown error.
  • Easily extended — `assert-equal`, `assert-error`, etc.
  • Sequence of assertions.
• More detailed properties of evaluation outcomes, result values.
Tests

What do we check?

- Could be combined with criteria — single form evaluated, null-tested.
- More generally: association of criterion and form.
  - Via assertion/sequence of assertions.
  - Comparison to “answer key”.
Groups
A family name

- Convenient reference for a responsibly-large, hopefully growth-prone, set of tests.
- Invoke tests, request results by single name.
- We’ll always have package...
NST design philosophy

In NST:

- All four testing artifacts
  - Criteria
  - Tests
  - Groups
  - Fixtures are separately defined and named.
- Incorporate frequent operations are supported as features of the test system — not requiring additional programming of the underlying test framework representation.
- Oriented towards “permanent” test suites — artifacts in code files:
  - Artifacts in files, loaded e.g. as a test system from ASDF.
  - Use REPL for digging into results, re-running individual tests.
What’s different

From a high-level user view, NST is similar to LIFT or FiveAM.
- Different macro names, keywords, etc.
- But simple tests end up looking about the same.

The main difference is in NST’s treatment of criteria. Each NST criterion:
- Can be abstracted over criterion arguments.
- Can encapsulate several different individual checks.
- Can aggregate multiple reports of failure or error.
Why complex criteria?

The benefit of these complex criteria is to better scale up to larger, more complicated test objects.

- Arguments allow minor variations of correctness criteria within a general rubric.
- Reduce verbosity of each test when invoking named criterion.
- Reduce number of tests since separate checks not needed...
  - Without sacrificing level of detail.
  - Without one discovered failure hiding other issues.
Examples — basics

Some simple tests:

(def-test-group some-number-tests ()
   (def-test it-is-even
       (:predicate evenp)
       40)
   (def-test hey-not-even
       (:predicate evenp)
       41)
   (def-test is-an-integer
       (:predicate integerp)
       40)
   (def-test hey-not-integer
       (:predicate integerp)
       40.5))
Examples — basics

Results from these tests:

Group some-number-tests: 2 of 4 passed
  - Check hey-not-even failed
    - Predicate evenp fails for (41)
  - Check hey-not-integer failed
    - Predicate integerp fails for (40.5)
Examples — combining criteria

Two things to check:

(def-test-group more-number-tests ()
  (def-test even-int-40
    (:all (:predicate evenp) (:predicate integerp)) 40)
  (def-test even-int-40half
    (:all (:predicate evenp) (:predicate integerp)) 40.5)
  (def-test even-int-41
    (:all (:predicate evenp) (:predicate integerp)) 41))
Examples — combining criteria

Results from these tests:

Group more-number-tests: 1 of 3 passed
  - Check even-int-41 failed
    - Predicate evenp fails for (41)
    - Check even-int-40half raised an error:
      Errors:
      - the value of excl::x is 40.5, which is not of type integer.
      Failures:
      - Predicate integerp fails for (40.5)
TOTAL: 1 of 3 passed
(2 failed, 1 error, 0 warnings)
Examples — naming criteria

We can name new criteria:

```lisp
(def-criterion-alias (:even-int)
   `(:all (:predicate evenp) (:predicate integerp)))
```

```lisp
(def-test-group still-more-number-tests ()
  (def-test even-int-40
    :even-int
    40)
  (def-test even-int-40half
    :even-int
    40.5)
  (def-test even-int-41
    :even-int
    41))
```
Examples — checking list elements

```
(def-test-group num-list-tests ()
  (def-test num-list
    (:each :even-int)
    '(40 40.5 41)))
```
Examples — checking list elements

Results from these tests:

Group num-list-tests: 0 of 1 passed
  - Check num-list raised an error:
    Errors:
    - the value of excl::x is 40.5, which is not of type integer.
    Failures:
    - Predicate integerp fails for (40.5)
TOTAL: 0 of 1 passed
  (1 failed, 1 error, 0 warnings)
A separate language for criteria

Really?

- Not without disadvantages: harder to debug/investigate a criterion interactively.
- Canard: “just program Lisp.”
  - NST’s :all vs. CL’s and.
  - NST’s :each vs. CL’s every.
  - If we want the expressiveness that arises from easily detecting multiple issues, the short-circuiting tendencies of CL’s functions are not what we want.
- Move from dispatch on these symbols to directly callable functions?
In the paper

- A more complete review of NST’s features and syntax.
- An overview of the implementation.
- Detailed comparison of about a dozen CL test systems.
Conclusion

NST adds a useful new test abstraction to Lisp’s testing toolkit.

- Good experience with larger test suites.
- Integration with ASDF.
- Support for QuickCheck-style sampled invariant testing.
- Some experimental support for:
  - More OO-style test methods.
  - JUnit XML output.
- Development continuing.

How to get NST:

- ASDF-install.
- CLiki: cliki.net/NST.