LIsp Framework for Testing

An introduction
and more

Outline

- Testing and LIFT
- What’s good about LIFT
- Problems with LIFT
- What I’d really like to see

Why Test?

- Machines are so fast and storage capacities are so huge that we face orders of magnitude more room for confusion, the propagation and diffusion of which are easily inadvertently mechanized.

  -- Edsger W. Dijkstra memo 1243

source: http://www.cs.utexas.edu/users/EWD/

Guidelines for Design

- Being a better programmer ... is about avoiding those complexities that increase the number of reasoning steps needed to keep the design under strict intellectual control.

-- Edsger W. Dijkstra memo 1209

Why Test?

- We test to make sure we did what we wanted to do,
- We test to make sure we have not undone what we did not want to undo.

Testing in XP

- Extreme programming advocates test-a-little / code-a-little programming
  - In part as a way of growing a design
  - Enables rapid prototyping and fearless refactoring
- A test system provides confidence and increases the speed of development--changes are less error prone and experimentation becomes easier.

Testing in Lisp…

- Common Lisp has no standard testing tools.
- Most testing is done by writing ad hoc code or by using the conditional evaluation reader macro #+
- This style is fine for interactive development but does not support regression testing or long term stability

What is LIFT?

- LIFT is a set of macros that make building regression tests as easy to build as interactive ones.
- LIFT is a framework that makes it easy to structure tests into a hierarchy and to give each test its own working environment.
- LIFT is in the family of Kent Beck inspired SUnit testing tools

LIFT has friends

(alphabetically)

- CLUnit
  - Adrian
- FiveAM
  - Baringer
- SchemeUnit
  - Welsh et. al.
- XPTest
  - Brozefsky
- …
- And non SUnit based
  - Franz Allegro's
  - Richard Water's RT
  - …

- Others probably exist
- All are worthy
- Combining them would be worthwhile…

Simple Example

```
(defparameter *filename* "test-file")
(defparameter *test-data* '(1 2 3 4))

;; setup
(with-open-file
 (s *filename* :direction :output)
   (write *test-data* :stream s))
==> (1 2 3 4)

;; the test
(with-open-file (s *filename*)
  (equal (read s) *test-data*))
==> T

;; cleanup
(delete-file *filename*)
```

```
(deftest test-file-system ()
  ((filename "test-file")
   (data '(1 2 3 4)))
  (:setup (with-open-file
              (s filename
                 :direction :output)
              (write data :stream s)))
  (:teardown (delete-file filename)))

(addtest (test-file-system)
  read-what-was-written
  (with-open-file (s filename)
    (ensure (equal (read s) data))))
==> (prints) Test passed!
```

Success!

LIFT in a nutshell

- deftest
  - Creates a test class (which will include many tests). The class provides a place for common variables and for test setup and teardown.
- addtest
  - Adds a test case to a test class
- run-tests
  - Runs the test cases in a test class (and its subclasses)

Supporting Players

- undeftest
  - Remove a test case from a test class
- Ensure, ensure-equal, ensure-warning and ensure-error
  - Tests an assertion and logs failures and errors
- Plus...
  - Some variables to control default behavior

One More Example

```
(deftest test-binary-search-tree ()
  ((b (make-container 'binary-search-tree)))
  (:setup (empty! b))
  (:tests
   ((insert-item b 2)                    ;; test #1
    (ensure (not (empty-p b))))
   (ensure (empty-p b))))                ;; test #2

(addtest
 (insert-item b 2)
 (insert-item b 3)
 (delete-item b 2)
 (ensure-equal (size b) 5000)))
```

Whoopts!

```
Test Suite: TEST-BINARY-SEARCH-TREE -- 1 Failure, 0 Errors ***
;------------------------------------
Failure: TEST-BINARY-SEARCH-TREE.TEST-4
Condition: Ensure-equal: 1 is not EQUAL to 5000 in ((SIZE B) 5000)
Code: ((INSERT-ITEM B 2) (INSERT-ITEM B 3) (DELETE-ITEM B 2)
       (ENSURE-EQUAL (SIZE B) 5000))
```

Outline

- Testing and LIFT
- **What’s good about LIFT**
- Problems with LIFT
- What I’d really like to see

The Good

- It fits with Lisp
  - Interactive
  - Clean
  - Simple
- It does what it is supposed to do!

Outline

- Testing and LIFT
- What's good about LIFT
- **Problems with LIFT**
- What I'd really like to see

What’s Wrong

- Testing Macros
- Test Organization
  - Tool integration
  - GUI, etc.

Macro Problems

- Start with a macro and a test

```
(defmacro my-macro (a b c)
  `(+ ,a (* ,b ,c)))

(deftest test-my-macro () ())
(addtest test-1
  (ensure-equal (my-macro 1 2 3) 7))
==> (prints) Test Passed!
```

- Change the macro

```
(defmacro my-macro (a b c)
  `(list ,a ,b ,c))

(run-test)
==> (prints) Test Passed!
```

Uh Oh!

Macro Problems - 2

- Work-around: test the expansion

```
(addtest (test-bind)
  expand-2-in-1
  (ensure (equal (macroexpand
                  '(bind ((a 1) b)
                     (declare (fixnum a) (dynamic-extent b))))
                 '(let ((a 1))
                    (declare (type fixnum a))
                    (let (b)
                      (declare (dynamic-extent b)))))))
```

  - But depends on how the macro works, not on what the macro is supposed to *do*
- Can write macros to call functions and then test these functions
- Could use reflection to automatically re-evaluate tests with changed macros…

Organization

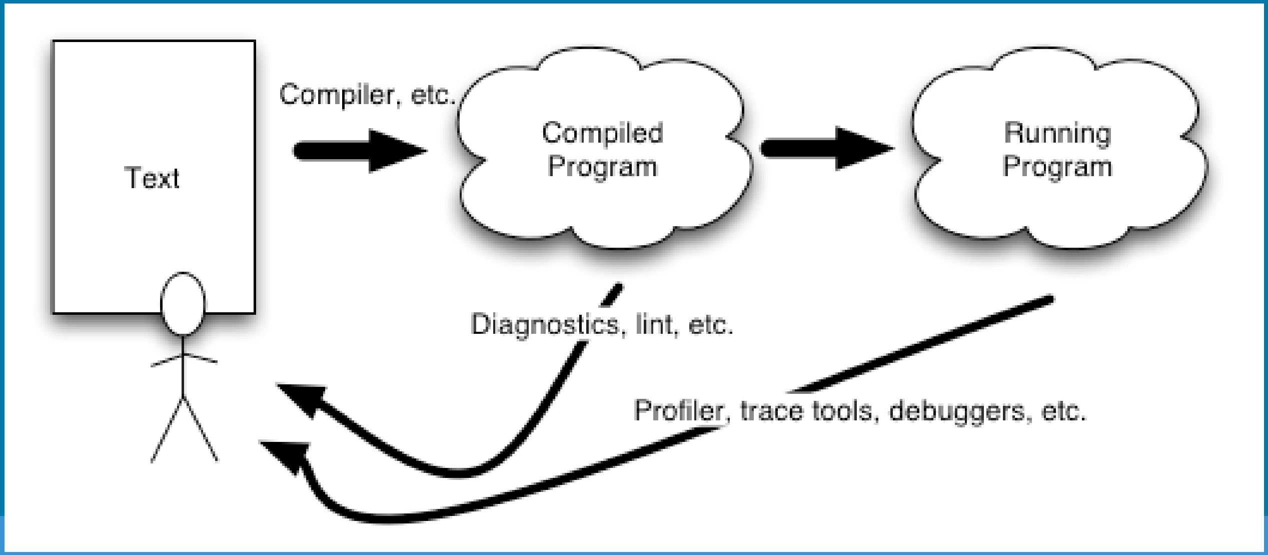
- So many tests, where to put ‘em
  - Same file? Different files?
- So many tests, how to set up a hierarchy
- So much old code, how to write tests for it all
- Need tools to manage it all…
  - CVS integration, GUI, etc.

Outline

- Testing and LIFT
- What’s good about LIFT
- Problems with LIFT
- **What I’d really like to see**

Where we are

- The code is “ours” (and only ours)
- We get feedback from the compiler, from tools and from execution
- Rinse, lather, repeat



What’s missing? Design Checks

- Even with organic growth, design suffers
  - No way to “codify” the design
  - Implementation forces compromise
- Too much left to the programmer
  - All that room for Dijkstra’s “mechanically propagated confusion”

What we need

- Specify code and meta-code
  - Contracts, constraints, capabilities
  - Types, yes but also:
    - This class can never be instantiated directly (generally at run-time),
    - This class should always precede that class in the precedence list (generally static)
    - These classes never get used (runtime information)

And more!

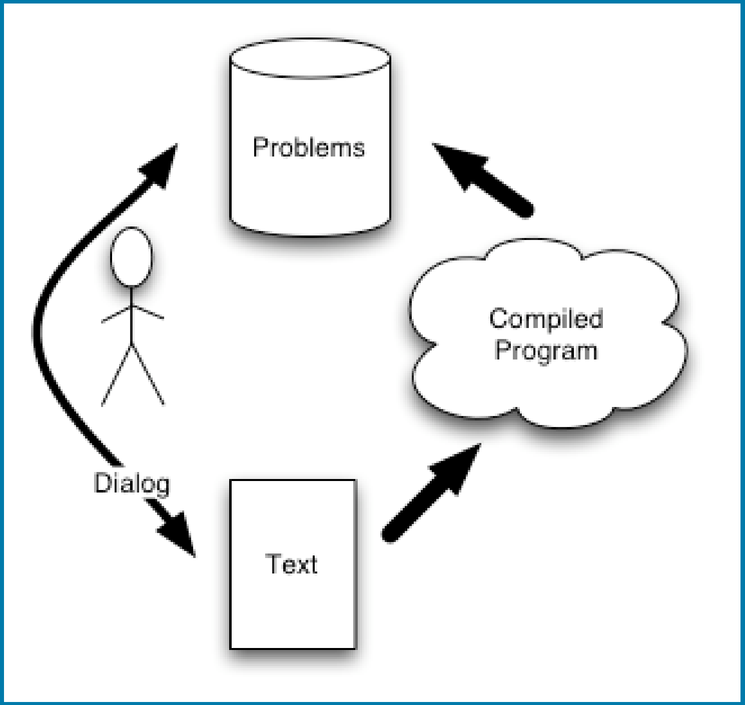
- Specification of protocols
  - Issues in the Design and Specification of Class Libraries. Kiczales and Lamping, 1992
- Examples:
  - Every method of this GF must call that GF
  - This method may not be overridden (only extended via :after methods)
  - If this method is overridden, then so must that method

What’s missing: sketching

- Human’s are not linear
  - Many of us like to sketch
- But compilers want things in *order*
- Lisp does this already
  - with-compilation-unit
- But it could do much more

What we need

- Track problems
  - Understand what actions will fix them
    - cf. Constraint satisfaction
  - Interact with person
  - Facilitate coding as sketching



YA Programmers Asst.

- Let the programming environment share the code and interact with the programmer
  - This special variable is declared in “file-1” but used in “file-2” which is loaded before “file-1”. Do you want to fix it?
  - The variable “foo” isn’t used by the function, do you want to declare it ignorable?
- These are mechanical corrections that a computer can notice and fix

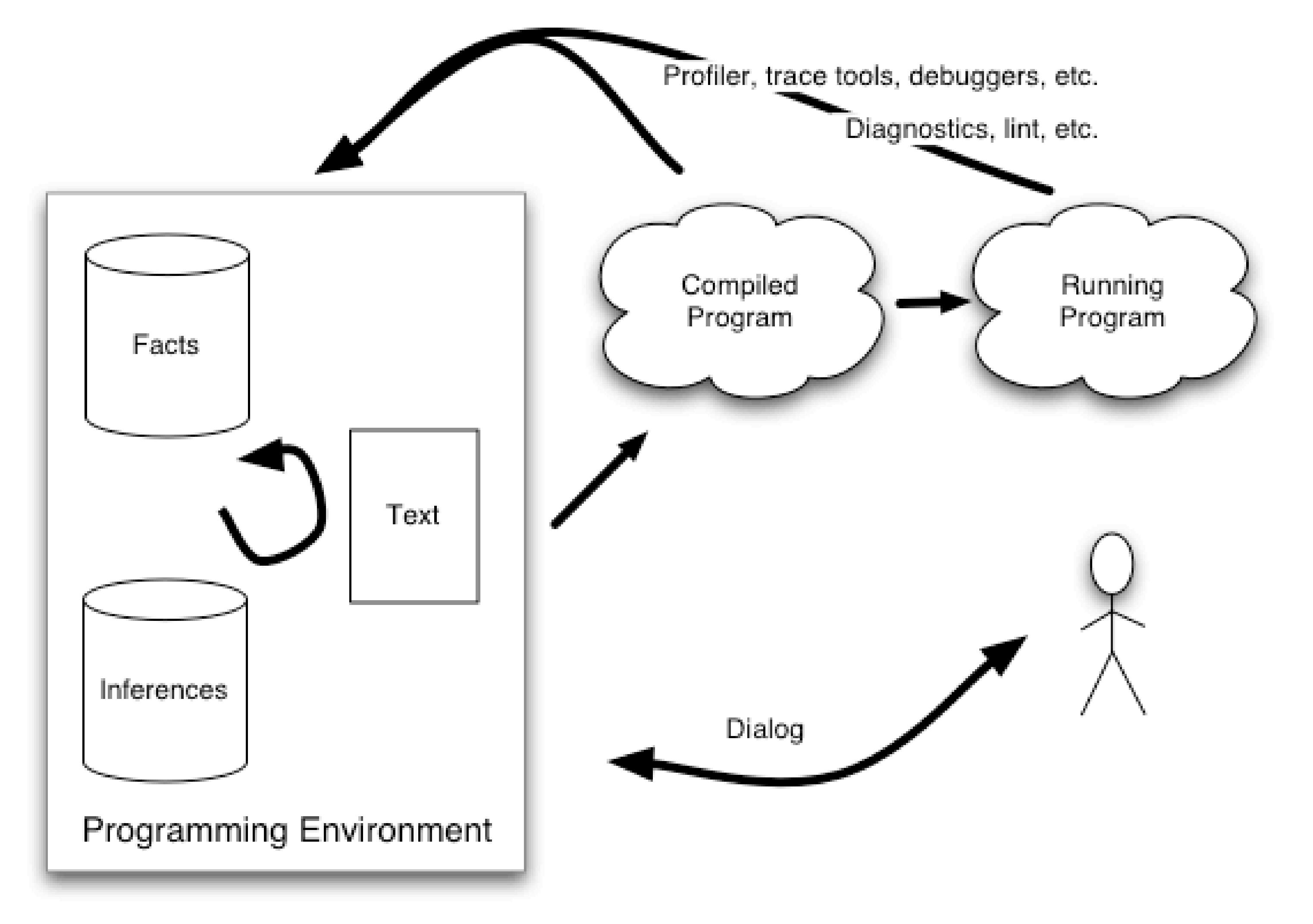
Declarative Compilation

- If compiler optimizations are encoded declaratively as rules and transformations
- Then the system could infer that knowing more about types could provide "meaningful" optimizations in speed or time
- I.e., the programming environment could say
  - "It looks as if array is a simple-vector, declaring it so should double the speed of function foo."

Related work

- Declarative Meta-programming
  - http://prog.vub.ac.be/research/DMP/
  - “The aim is to try to capture and formally express the interaction between the higher phases of software development (design, analysis, etc...) and the actual implementation level.”
- Flavors could capture parts of the design
  - :required-methods, :required-flavors, etc…
- Refactoring code browsers
  - Eclipse and IntelliJ for Java
  - Refactoring Browser for SmallTalk
- Programming by transformation, etc…

Big Fuzzy Bubble Picture



Caveat

- LIFT exists, the rest of this is hand waving.
- I’d love to hear comments, critiques and stories of what’s been done and what can be done better.
- E-mail me at gwking@metabang.com