Designing a Comparative Usability Study of Error Messages

Henry Fellows, Thomas Hagen, Sean Stockholm,
and Elena Machkasova

Midwest Instruction and Computing Symposium

April 23, 2016

Table of contents

Our project

- ClojurEd
  - Ongoing project at UMM
  - Goal: use Clojure in an introductory course
  - Educators have found Lisp languages (e.g. Racket) to be useful in introductory courses
  - Functional abstraction allows clearer abstraction: separating the data from the means of interacting with that data.
  - Current UMM course uses a Lisp language

Current state of the project

- Our work focuses on error messages in Clojure
  - Error messages are a useful learning tool
  - Primary means of communication
  - Focus on usability
  - Summer 2015: developed an alternative system of error messages
  - Current goal: evaluate their effectiveness
  - Would like to compare to usual Clojure and to Racket

Lisp

- Clojure and Racket are Lisps
- Lisp is a functional dynamically typed language
- Data immutable by default
- Read-eval-print-loop (REPL)
  - interactive environment
  - useful for development and debugging

Lisp prefix notation

- Lisp uses prefix notation
  - parentheses
  - parameters
    `(<function-name> <argument 1> <argument 2>)`
  - + is a built-in function, not an operator

    ```
    (+ 5 5)
    -> 10
    ```

  - Elena: Add something about anonymous functions

Overview of Racket

Elena: Possibly move to earlier

Overview of Clojure

`Elena: Some of this can be moved to differences between Racket and Clojure, some eliminated`

- Developed in 2007 by Rich Hickey
- Member of the Lisp family
- Runs on the Java Virtual Machine (JVM)
- Used in industry, especially for parallelism and data science
  - Horrific error messages
  - Beginner tools are under development

Study goals

The goal of the usability study is to compare usability of error messages:

- Our error messages to the original Clojure.
- Our error messages to Racket.

We adapted an approach developed by Marceau, Fisler, and Krishnamurthi for evaluation of Racket error messages: does an edit after seeing an error is closer to the solution or farther?

Usability study: students are asked to correct several code fragments in Clojure and in Racket.

Experimental Setup

- Participant given a review of Racket
- Participant given a series of program fragments in Racket to correct
- Participant presented with a brief overview of Clojure
- Participant given a series of program fragments in Clojure to correct (with either default Clojure messages or our messages)
- The screen recorded at a regular intervals
- The participants asked a few questions at the end

Study Participants

- Volunteer students
- Taken Racket introductory course
- No/little experience with Clojure
- Recruitment from department mailing list
- Students compensated for time thanks to gift from Cognitect, Inc.

Data Evaluation

- Continuous screen capture allows extraction of numeric data
  - Time to solve
  - Iterations to solve
  - Problems solved
- Interview allows us to gauge perception of error messages

Question Selection

- Testing the usability of similar programming languages
- Are you testing syntax, semantics, or error messages?
- Select elements of the languages that have similar syntax and semantics
- Sometimes you have to sacrifice idiomatic code for testable code

Meaningful & Accessible questions

The code examples should:

- be simple enough to understand the intent with 2-3 test examples.
- have mistakes that a beginner would make (e.g. switched function arguments, a mistyped identifier...)
- be simple to fix (challenging: may be caused by multiple issues, and a beginner; beginners may make more complex changes than needed)
- use the same simple set of features in Racket and in Clojure (use `equal?` in Racket rather than `check-expect` since Clojure = is roughly the same).

Example 1

Racket version:

```
(define (select-even elements)
  (foldl (lambda (x y) (if (even? y) (cons x y) y))
                       '() elements))
```

Clojure version:

```
(defn select-even [elements]
  (reduce (fn [x y] (if (even? x) (cons y x) y))
                    '() elements))
```

Test case:

```
(= '(2 4 6 8) (select-even '(1 2 3 4 5 6 7 8 9)))
```

Example 1: error messages

Racket error:

```
even?: expects integer, given '()
```

Original Clojure:

```
java.lang.IllegalArgumentException:
Argument must be an integer:
clojure.lang.PersistentList$EmptyList@1
                core.clj:1355 clojure.core/even?
```

Our Clojure:

```
Error: In function even?, the first argument () must be
an integer number but is a list.
```

Example 2

Racket version:

```
(define (my-length elements)
   (cond
   [(empty? elements) 0]
   [else (+ 1 (my-length (first elements)))]))
```

Clojure version:

```
(defn my-length [elements]
  (if (empty? elements) 0 (+ 1
                     (my-length (first elements)))))
```

Test cases:

```
(= (my-length '(5 4 3 2 1)) 5)
(= (my-length '()) 0)
(= (my-length '(1 3 5 7 9 11)) 6)
```

Example 2: error messages

Racket error:

```
first: expects a non-empty list; given: 1
```

Original Clojure:

```
IllegalArgumentException Don't know how to create ISeq
from: java.lang.Long  clojure.lang.RT.seqFrom
(RT.java:528)
```

Our Clojure:

```
Error: In function first, the first argument 5 must be
a sequence but is a number.
```

Conclusion

There isn't much literature about systematic evaluation of error messages.
Designing a comparative usability study is challenging.
We look forward to the results of the study.

Acknowledgments

Our research was sponsored by:

- HHMI
- UMN UROP
- Coginitect, Inc providing funding for participants' compensation

Thank you!
Any questions?