Developing Beginner-Friendly User Interactions for the Clojure Programming Language

Henry Fellows, Aaron Lemmon, Max Magnuson, Emma Sax, Paul Schliep, and Elena Machkasova
Midwest Instruction and Computing Symposium
April 11, 2015
Table of contents

1. Introduction to the Project
2. Overview of Clojure
3. Error Messages
4. Technical Setup
Clojure in an introductory course

- Developed in 2007 by Rich Hickey
- Member of the Lisp family
- Felleisen et al found Lisp languages to be useful in introductory courses
- Current UMM course uses a Lisp language
Motivations for the project

- ClojurEd
  - ongoing project at UMM
  - introduce Clojure in an introductory course
- Our work focuses on error messages in Clojure
  - error messages are a useful learning tool
  - focus on usability
Overview of Clojure

- Dynamically typed
- Data types immutable by default
- Functional
- Runs on the Java Virtual Machine (JVM)
- Read-eval-print-loop (REPL)
  - interactive environment
  - useful for development and debugging
Prefix notation

- Clojure uses prefix notation
  - parentheses
  - parameters
    - `<function-name> <argument 1> <argument 2>`
  - `+` is a built-in function, not an operator
    - `(+ 5 5)`
    - `-> 10`
def can be used to define variables
(def my-string "Hello World")
my-string
-> "Hello World"

defn can be used to define functions
(defn increment-number [number] (+ number 1))
(increment-number 2)
-> 3
Collections

- All collections can contain any number of values of any data type
  - lists
    - (1 2 "foo" :a 9 "bar")
  - hashmaps (key-value pairs)
    - {:a 1, :b 2, :c 3}
Anonymous functions

Anonymous functions are a way to implement a function on the fly, and use it only once

- `map` takes a function and a collection as arguments, and applies the function to the entire collection

```
(map (fn [number] (+ number 1)) '(0 1 2 3))
-> (1 2 3 4)
```
Lazy sequences

- Only a needed portion of a sequence is evaluated
  - `take` takes the first $n$ elements of a collection
  - `range` returns an infinite sequence of non-negative integers beginning at 0
    - `(take 10 (range))`
    - $\rightarrow$ (0 1 2 3 4 5 6 7 8 9)
  - Fibonacci sequence
    - (1 1 2 3 5 8 13 ... )
Error messages should be:

- helpful for debugging
- easy to understand
- not intimidating

Clojure error messages are confusing
**Clojure error messages: examples**

- **EOF while reading, starting at line 3,**
  compiling:(compilation_errors/eof.clj:4:1)

- **IllegalArgumentException** Parameter declaration
  * should be a vector
  clojure.core/assert-valid-fdecl (core.clj:6842)

- **ClassCastException** java.lang.String cannot be cast to clojure.lang.IPersistentCollection
  clojure.core/conj (core.clj:83)
Exception in thread "main" java.lang.RuntimeException: EOF while reading, starting at line 3, compiling:(compilation_errors/eof.clj:4:1)
at clojure.lang.Compiler.load(Compiler.java:7137)
at clojure.lang.RT.loadResourceScript(RT.java:370)
at clojure.lang.RT.loadResourceScript(RT.java:361)
at clojure.lang.RT.load(RT.java:440)
at clojure.lang.RT.load(RT.java:411)
at clojure.core$load$fn__5066.invoke(core.clj:5641)
at clojure.core$load.doInvoke(core.clj:5640)
at clojure.lang.RestFn.invoke(RestFn.java:408)
at clojure.core$load_one.invoke(core.clj:5446)
at clojure.core$load_lib$fn__5015.invoke(core.clj:5486)
at clojure.core$load_lib.doInvoke(core.clj:5485)
at clojure.lang.RestFn.applyTo(RestFn.java:143)
Error message transformation: approaches

- Function substitution approach
  - add type-checking preconditions
  - type mismatch produces custom message

- Try/catch approach
  - wrap user’s code in try/catch block
  - capture error message
  - check against collection of regular expressions
  - replace with improved message
Error message transformation: discussion

- Function substitution approach
  - able to recover argument values and use them for a more meaningful message
    (+ 2 "apple")
    In function +, the second argument "apple" must be a number but is a string.
  - unable to handle compilation exceptions
  - cannot handle user-made functions
  - impractical to redefine all existing functions

- Try/catch approach
  - able to handle both runtime and compilation exceptions
  - usually cannot recover argument values
User scenarios: example slide 1

- Erroneous code fragment:
  
  ```clojure
  print(Hello World)
  ```

- Clojure error message:
  
  ```java
  CompilerException java.lang.RuntimeException: Unable to resolve symbol: Hello in this context
  ```

- Our modified error message:
  
  `Compilation error: name Hello is undefined`

- Student edit:
  
  ```clojure
  print("Hello World")
  ```
User scenarios: example slide 2

- Student edit:
  
  ```clojure
  print("Hello World")
  ```

- Clojure error message:
  
  ```java
  ClassCastException java.lang.String cannot be cast to clojure.lang.IFn
  ```

- Our modified error message:
  
  Attempted to use a string, but a function was expected.

- Corrected code:
  
  ```clojure
  (print "Hello World")
  ```
Hints: how they are helpful

- Errors can have several underlying causes
- Multiple hints per error
- Provide more human interpretation of the issue
- Offer suggestions on how to resolve issues
Erroneous code fragment:
print(Hello World)

Improved error message:
Compilation error: name Hello is undefined

Hint:

It looks like Clojure is expecting that Hello is something named in your program. If you wanted Hello and any following words to be plain text, try surrounding them with double quotes. If Hello is referring to something named in your program, make sure it is spelled correctly.
Hints: future work

- Hints still a work in progress
- Provide links to Clojure documentation
- Develop more user scenarios
- Usability study with students
Setup

- Current IDE options
  - LightTable
    - more established
  - Nightcode
    - newer, promising IDE
  - ongoing open source projects
  - IDE independent approach

- Project manager
  - Leiningen
  - popular in Clojure community
Error handling

- Initial approach of try/catch
- Maintain consistency between runtime, compile time, REPL
- Issues with laziness
- Try/catch student code does not catch all
Developing Beginner-Friendly User Interactions for the Clojure Programming Language

Current workflow diagram

Key
- Code Flow
- Errors
- Defined By

Dependencies
- Console
- Leiningen
- Student Code (REPL, Code)
- nREPL
- Compiler

Developing Beginner-Friendly User Interactions for the Clojure Programming Language
Issues

- Installation of Leiningen is nontrivial
- Command line acts as a barrier
- We do not want students managing dependencies
- No integration with new error handling system

Developing Beginner-Friendly User Interactions for the Clojure Programming Language
Developing Beginner-Friendly User Interactions for the Clojure Programming Language
Future work

- Error handling
  - currently have regular expressions
  - need user scenarios
  - usability studies

- Technical implementation
  - implement items in the proposed diagram
  - work on user interface through the IDE
Acknowledgments

Our research was sponsored by:

- HHMI
- UMN UROP
- UMM MAP

Thank you!
Any questions?