

A beginner-friendly environment for exploring error messages in the Clojure programming language.

Tristan Kalvoda, Elena Machkasova, Jaydon Stanislawski, and
John Walbran

University of Minnesota, Morris

Midwest Instruction and Computing Symposium, April 2025

Outline

- 1 Overview of Clojure and Its Error Messages
 - Clojure language and Syntax
 - Clojure's Error Messages
- 2 Babel Project
 - Motivation and Goals
 - Setup
 - Exception Processing
- 3 Morse Viewers
- 4 Current State of the Project and Future Work
 - Screenshots
 - Future Work

Clojure Language and Syntax

What is Clojure? - Clojure language and Syntax

- Clojure is a part of the Lisp language family
- Syntax
 - prefix notation (operators before operands).
 - expressions are surrounded by parentheses.

Example: `(/ 9 3)` denotes 9 divided by 3

Clojure Language and Syntax

- Clojure is implemented in Java and runs on the Java Virtual Machine (JVM)
- Clojure code → Java code → JVM bytecode → executed on JVM

Clojure Language and Syntax

Clojure's REPL

- interactive environment for code evaluation
- Read → Evaluate → Print → Loop

```
Clojure 1.11.4
user=> (map pos? [-1 0 1])
(false false true)
user=>
```

Clojure's Error Messages

Clojure Exceptions

- an event or error that disrupts the normal flow of a program's execution
- Clojure syntax errors will also result in a Java exception

Error Messages

- generate when a exception occurs
- provide error type, cause, and location

Clojure's Error Messages

Anatomy of a Clojure Error Message

```
=> (/ 9 0)
```

```
Execution error (ArithmeticException) at user/eval1  
(REPL:1).
```

Divide by zero

- `ArithmeticException`: The type of error that occurred.
- `user/eval1 (REPL:1)`: The location where the error happened (in this case, REPL, line 1).
- `Divide by zero`: The description of the error's cause.

Motivation and Goals

Motivation

- Want to design a learning tool for beginners to Clojure and programming as a whole
- Clojure error messages contain awkward phrasing that may impede understanding

Example

Consider the error produced by the form below. What does it mean?

```
=> (count 1)
```

```
Execution error (UnsupportedOperationException) at  
user/eval1529 (REPL:1).
```

```
count not supported on this type: Long
```


Motivation and Goals

Goals

- Build an interactive, beginner-friendly tool for error processing in Clojure
- Simplify error messages to be more intuitive, removing jargon and clutter

Jaydon: More stuff here? Or delete entirely?

Setup

Overview of Babel

- Replaces native Clojure error messages
- Messages produced by Babel broadly fall into two types:

Spec errors

- Babel makes use of the Clojure *spec* library to catch errors on most core function calls
- This allows specifications on existing Clojure functions (see next slide)

Non-spec errors

- Regex is used to identify simpler, more common messages, e.g. division by zero
- We maintain a dictionary of these errors

Setup

About *spec*

- Tooling to define requirements on properties of a sequence or collection
- Function arguments come in a sequence
- Requirements are predicates, e.g., can be used to check argument types, count, values, etc.
- We bind specs to core Clojure functions, forcing an error if requirements aren't met
- Error reports from spec are far more detailed than native Clojure error messages

Usage

- Launching a REPL server in the Babel repository allows the tool to "hook" to it
- All messages displayed in the terminal are generated by Babel rather than Clojure

Incomplete. This slide will talk a bit more about how Babel operates in the REPL's terminal window; doesn't currently have IDE integration but could be a nice future goal. Babel can also be run with separate files, though a bit clunky

Exception Processing

Incomplete. This slide will describe how Babel manipulates the messages it receives to produce the message seen by the end user. The exception data is "classified" and passed to one of many helper functions which returns a new string. The modified message is printed to the terminal in place of the original message.

Sending Data to Morse

- The Clojure REPL does not provide the proper hooks to effectively manipulate error message data.

Sending Data to Morse

- The Clojure REPL does not provide the proper hooks to effectively manipulate error message data.
- To get around this, we need to initialize Babel within a sub-REPL of the parent REPL session.

Sending Data to Morse

- The Clojure REPL does not provide the proper hooks to effectively manipulate error message data.
- To get around this, we need to initialize Babel within a sub-REPL of the parent REPL session.
- Creating a sub-REPL allows us to introduce hooks that let us add preprocessing steps.

Sub-REPL hooks

Babel uses the following hooks as part of error processing:

- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.

Sub-REPL hooks

Babel uses the following hooks as part of error processing:

- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.
- `:eval` Behavior on form evaluation. Stores the command and sends it to REPL and Morse.

Sub-REPL hooks

Babel uses the following hooks as part of error processing:

- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.
- `:eval` Behavior on form evaluation. Stores the command and sends it to REPL and Morse.
- `:caught` Behavior on caught exception. Processes the error in Babel, and changes the Exception-String processing to instead create a vector of labelled pairs.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring form evaluation.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring from evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring from evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.
- The introduction of the error labeling and prototyping this was pivotal in enabling data formatting.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring from evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.
- The introduction of the error labeling and prototyping this was pivotal in enabling data formatting.
- We currently have a small number of error messages labeled for demonstration purposes.


```
user=> (even? 1 2)  
Execution error (ArityException) at user/eval2044 (REPL:1).  
Wrong number of args (2) passed to: clojure.core/even?
```

Figure: The output for the form `(even? 1 2)` in default Clojure.

```
Babel=> (even? 1 2)
(even? 1 2)
Wrong number of arguments in (even? 1 2): the function even? expects one argument but was given two arguments.
In Clojure interactive session on line 1.
```

Figure: The output for the form `(even? 1 2)` in Babel through the REPL.

The screenshot shows the Babel web interface. On the left is a sidebar with a 'Follow REPL' checkbox, a 'browse' button, and a table with two columns: 'expr' and 'val'. The table contains the following rows:

expr	val
error-map	(form (even? ...
(even? 1 2)	{via [[type cl...

The main area displays the error message: "Your code had an error in *Clojure interactive session* on line 1." Below this is a code editor showing the form `(even? 1 2)`. The error message states: "Wrong number of arguments in (even? 1 2): the function **even?** expects one argument but was given two arguments. In Clojure interactive session on line 1." Below the error message is a link to the documentation: https://clojuredocs.org/clojure.core/even_q. At the bottom of the interface are navigation buttons: '<', '<<', and '>>'.

Figure: The output for the form `(even? 1 2)` in Babel with Morse.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Refining the end user work flow between working code and erroring code.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Refining the end user work flow between working code and erroring code.
- Develop Morse viewers for other information, such as the stack trace, and full java error messages.

Future Work (cont.)

- We plan to run a usability study about our developments after we have greater feature coverage.

Future Work (cont.)

- We plan to run a usability study about our developments after we have greater feature coverage.
- We are going to use the results to guide further design.

Future Work (cont.)

- We plan to run a usability study about our developments after we have greater feature coverage.
- We are going to use the results to guide further design.
- We hope to explore IDE (VSCode) integration for possible work-flow refinements.

Acknowledgements

This work was supported in part by Morris Academic Partnership (MAP) and UMN Undergraduate Research Opportunity (UROP).

We thank Joe Lane for introducing us to Morse tools and for numerous helpful discussions.

Discussion

Questions?