Building a Clojure App from the Bottom Up

Twin Cities Code Camp #23
April 13, 2019

Kurt Christensen
@projectileboy

Who am I?

Kurt Christensen

- Programmer
  - What I've been paid to write
    - Pascal, C, x86 Assembly, C++, SQL, Java, Javascript, VB.NET, Bash, Clojure, C#, Ruby, Groovy
  - What I *might* actually know
    - Whatever I've touched in the last 24 hours..?
- Product/Process/Technical Coach
  - Small startups, single teams
  - Big companies, multiple teams across many time zones

Who are you?

Who here has ever tried using Clojure?

- For fun?
- At work? Currently?

Who here has used a functional programming language?

Motivation

- Inspired by Stuart Halloway's talk, "Running with Scissors: Live Coding with Data"
  - https://www.youtube.com/watch?v=Qx0-pViyIDU
- Share what's good about Clojure
- Share what's good about the Clojure workflow
  - See if any of it applies to *your* world

Outline

- Taste of Clojure
- The Clojure ecosystem
- Thinking in Clojure
- The REPL
- REPL workflow
- Building a simple application at the REPL
- What about tests? Or the debugger?
- Resources
- Q & A

Why Lisp?

"... I think I can give a kind of argument that might be convincing. The source code of the Viaweb editor was probably about 20-25% macros. Macros are harder to write than ordinary Lisp functions, and it's considered to be bad style to use them when they're not necessary. So every macro in that code is there because it has to be. What that means is that at least 20-25% of the code in this program is doing things that you can't easily do in any other language. However skeptical the Blub programmer might be about my claims for the mysterious powers of Lisp, this ought to make him curious."

Paul Graham, "Beating the Averages"

http://www.paulgraham.com/avg.html

Why Clojure?

- A Lisp you can actually *use*
  - JVM, CLR, Javascript (Clojurescript)
- Easy interop with underlying platform
- Immutable by default
- Consistent ways of working with various data
- Functional, but with very nice concurrency primitives for managing state when necessary

...and so on: clojure.spec, multimethods,
focus on backward compatibility, etc etc etc

What does Clojure look like?

```
> (+ 1 2)
3
> (defn add [x y] (+ x y))
#'user/add
> (defn add [& args] (apply + args))
#'user/add
> (add 1 2 3 4 5)
15
```

Immutable Data

```
> (def m {:a 1 :b 2 :c 3})
#'user/m
> (assoc m :d 4)
{:a 1, :b 2, :c 3, :d 4}
> m
{:a 1, :b 2, :c 3}
```

Data Collections

```
> (:a {:a 100 :b 200 :c 300}) ; Map
100
> ([100 200 300] 0) ; Vector
100
> (#{100 200 300} 100) ; Set
100
```

In Clojure, we are usually working with very simple data types: primitives, Strings, Maps, Vectors, Sets (there are also lists, but lists are usually function calls)

Shapes of Data

- File stream
- JSON
- SQL query result sets
- Maps

...etc...

In Clojure, almost everything you care about is a sequence, and you work with all sequences in roughly the same way

Macros

```
> (defmacro flip [y x f] (list f x y))
#'user/flip
> (flip 1 2 +)
3
> (defmacro flip [& args]
    `(apply ~(last args) '~(butlast args)))
#'user/flip
> (flip 1 2 3 4 +)
10
> (macroexpand '(flip 1 2 3 4 +))
(clojure.core/apply + (quote (1 2 3 4)))
```

...and much much more!

But this isn't meant to be an exhaustive introduction to Clojure!

Instead, we want to see what it *feels* like to build Clojure code at a REPL - how it is a natural extension of how one thinks about writing code in Clojure. This is a style of working that can apply to F#, Python, Ruby, Groovy, etc - the latter languages aren't often written in a functional style, but they could!

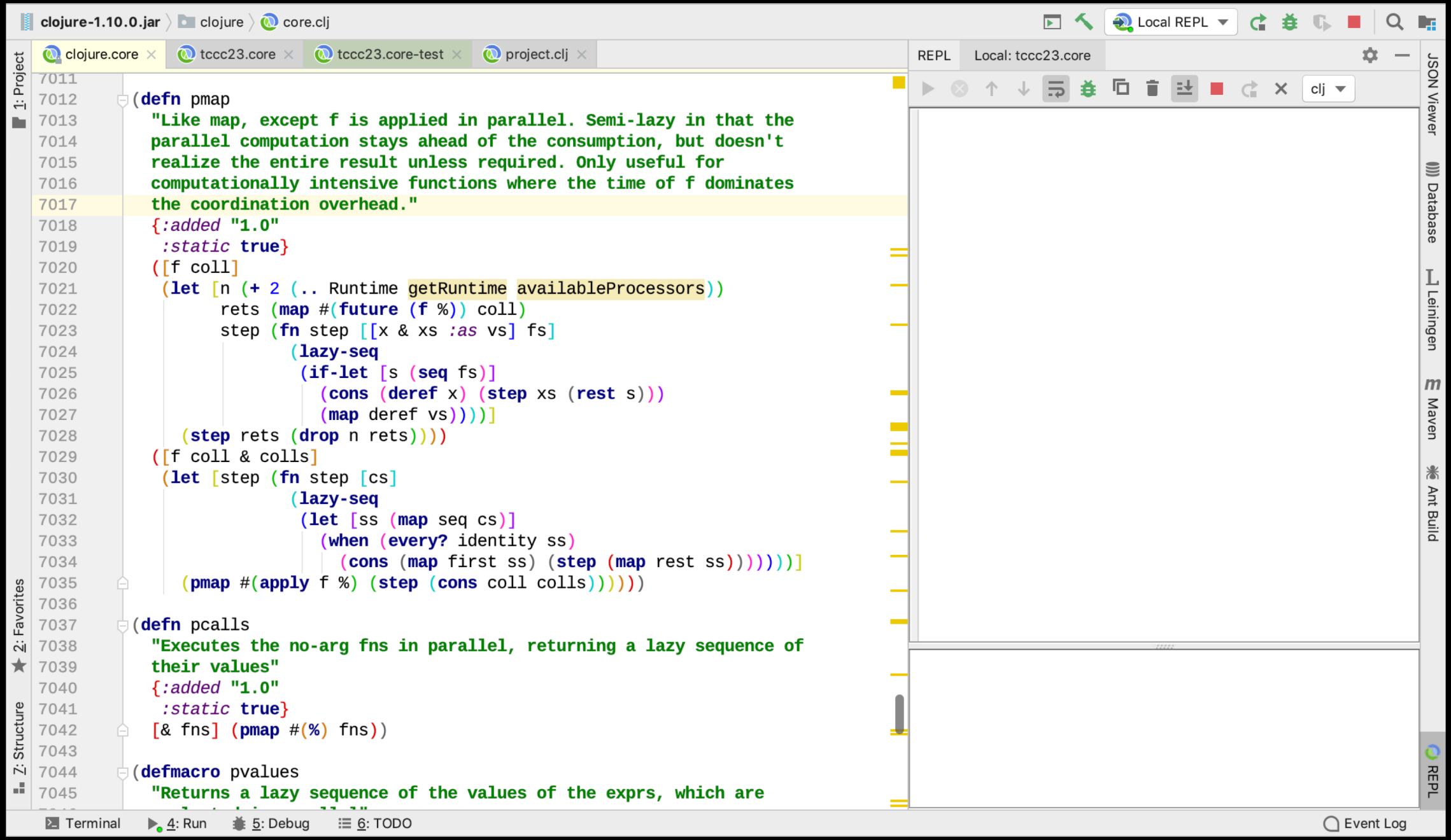
How does one *think* in Clojure?

In Clojure, one tends to think in terms of transforming streams of data - functional programming is like signal processing

- map
- filter
- reduce

Assemble your pieces in a let statement

(also, prefer destructuring to direct access!)


clojure-1.10.0.jar
clojure
core.clj
Local REPL
clojure.core
tccc23.core
tccc23.core-test
project.clj
REPL
Local: tccc23.core
clj
1: Project
2: Favorites
7: Structure
JSON Viewer
Database
Leiningen
Maven
Ant Build
REPL
Terminal
4: Run
5: Debug
6: TODO
Event Log
7011
7012 (defn pmap
7013 "Like map, except f is applied in parallel. Semi-lazy in that the
7014 parallel computation stays ahead of the consumption, but doesn't
7015 realize the entire result unless required. Only useful for
7016 computationally intensive functions where the time of f dominates
7017 the coordination overhead."
7018 {:added "1.0"
7019 :static true}
7020 ([f coll]
7021 (let [n (+ 2 (.. Runtime getRuntime availableProcessors))
7022 rets (map #(future (f %)) coll)
7023 step (fn step [[x & xs :as vs] fs]
7024 (lazy-seq
7025 (if-let [s (seq fs)]
7026 (cons (deref x) (step xs (rest s)))
7027 (map deref vs))))]
7028 (step rets (drop n rets))))
7029 ([f coll & colls]
7030 (let [step (fn step [cs]
7031 (lazy-seq
7032 (let [ss (map seq cs)]
7033 (when (every? identity ss)
7034 (cons (map first ss) (step (map rest ss)))))))]
7035 (pmap #(apply f %) (step (cons coll colls))))))
7036
7037 (defn pcalls
7038 "Executes the no-arg fns in parallel, returning a lazy sequence of
7039 their values"
7040 {:added "1.0"
7041 :static true}
7042 [& fns] (pmap #(%) fns))
7043
7044 (defmacro pvalues
7045 "Returns a lazy sequence of the values of the exprs, which are

The Clojure Ecosystem

- Clojure (currently at 1.10)
- Leiningen (aka lein)
- Editor / IDE
  - Text editor + command-line REPL!
  - Repl.it - quick and easy way to start any language
  - Emacs / CIDER
  - IntelliJ / Cursive
  - VSCode / Clojure or Calva extensions

The Clojure Ecosystem

- Various libraries for different ways of working with relational databases
  - Also, Datomic
- Variety of web application / API frameworks
  - We'll use Pedestal as our example (http://pedestal.io/guides/hello-world)
- Great APIs for AWS
- Many scientific and machine learning libraries

...and so on, and so on...

The REPL

- REPL = Read -> Eval -> Print Loop
- Not merely a shell! It *is* the running process!
- You can attach a REPL to an already-running process

Navigating the REPL

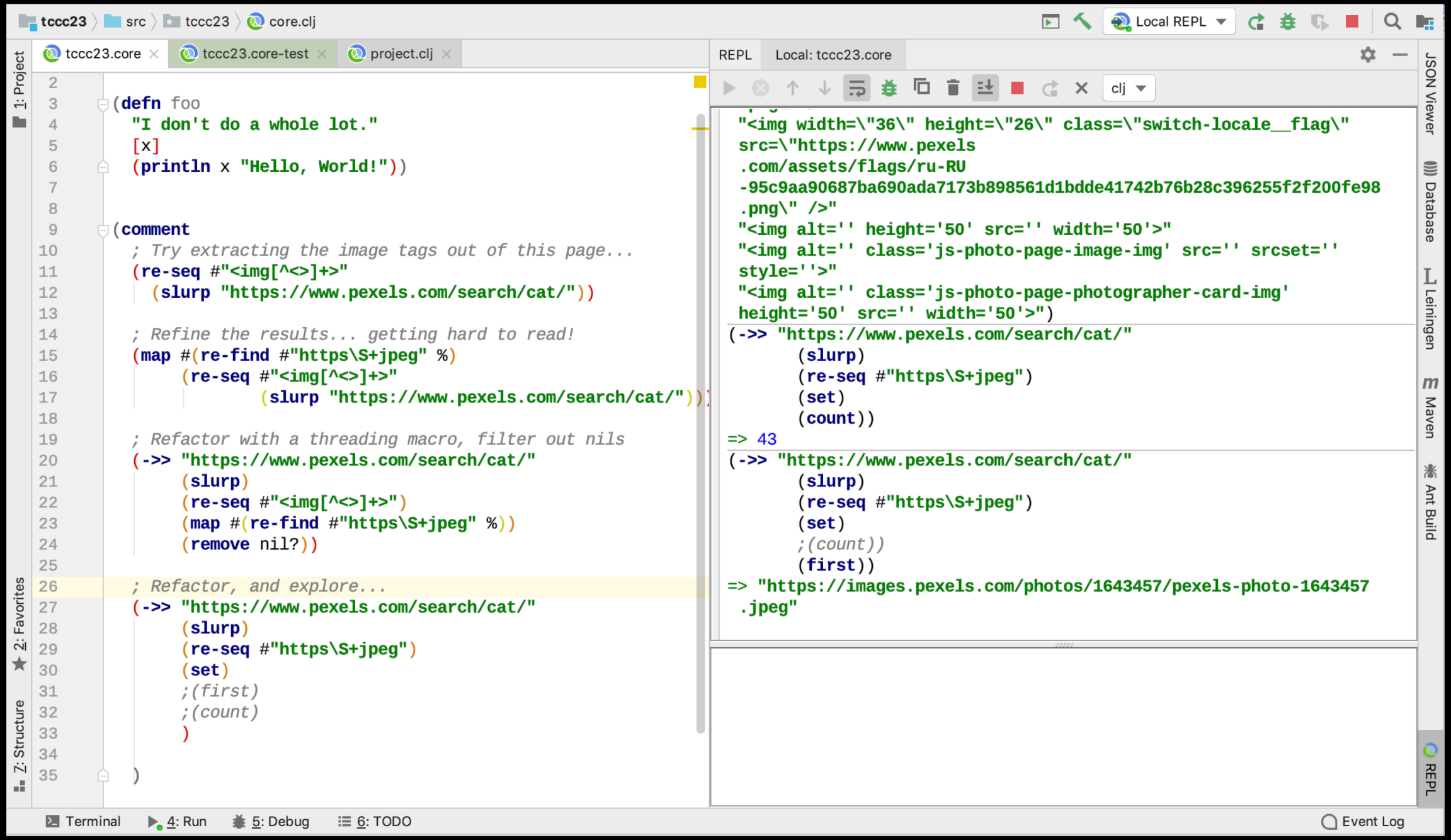
- Choices:
  - emacs / cider
  - Cursive (IntelliJ)
  - https://repl.it
  - VS Code
  - More…
- I'll be using Cursive, using a REPL launched by Leiningen
- Cursive has handy keyboard shortcuts!
- Be aware of namespaces!

Transform data, a step at a time

Little functions compose into big functions

Poke, explore, break things...

(live demo)

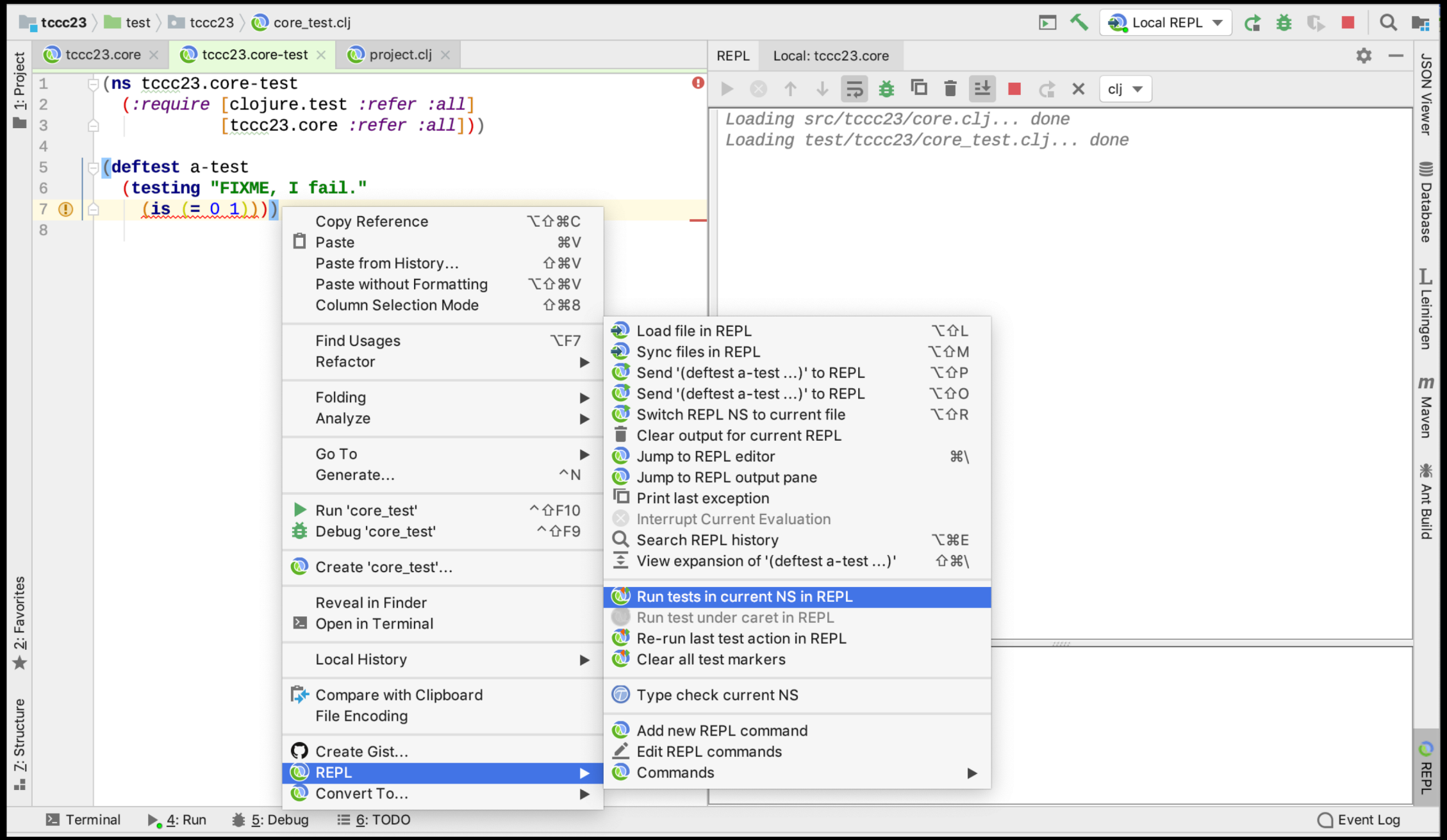

tccc23 src tccc23 core.clj
Local REPL
tccc23.core
tccc23.core-test
project.clj
(defn foo
"I don't do a whole lot."
[x]
(println x "Hello, World!"))
(comment
; Try extracting the image tags out of this page...
(re-seq #"<img[^<>]+>"
(slurp "https://www.pexels.com/search/cat/"))
; Refine the results... getting hard to read!
(map #(re-find #"https\S+jpeg" %)
(re-seq #"<img[^<>]+>"
(slurp "https://www.pexels.com/search/cat/")))
; Refactor with a threading macro, filter out nils
(->> "https://www.pexels.com/search/cat/"
(slurp)
(re-seq #"<img[^<>]+>")
(map #(re-find #"https\S+jpeg" %))
(remove nil?))
; Refactor, and explore...
(->> "https://www.pexels.com/search/cat/"
(slurp)
(re-seq #"https\S+jpeg")
(set)
;(first)
;(count)
)
)
REPL
Local: tccc23.core
clj
"<img width=\"36\" height=\"26\" class=\"switch-locale__flag\"
src=\"https://www.pexels
.com/assets/flags/ru-RU
-95c9aa90687ba690ada7173b898561d1bdde41742b76b28c396255f2f200fe98
.png\" />"
"<img alt='' height='50' src='' width='50'>"
"<img alt='' class='js-photo-page-image-img' src='' srcset=''
style=''>"
"<img alt='' class='js-photo-page-photographer-card-img'
height='50' src='' width='50'>")
(->> "https://www.pexels.com/search/cat/"
(slurp)
(re-seq #"https\S+jpeg")
(set)
(count))
=> 43
(->> "https://www.pexels.com/search/cat/"
(slurp)
(re-seq #"https\S+jpeg")
(set)
;(count))
(first))
=> "https://images.pexels.com/photos/1643457/pexels-photo-1643457
.jpeg"
1: Project
2: Favorites
7: Structure
JSON Viewer
Database
Leiningen
Maven
Ant Build
REPL
Terminal
4: Run
5: Debug
6: TODO
Event Log

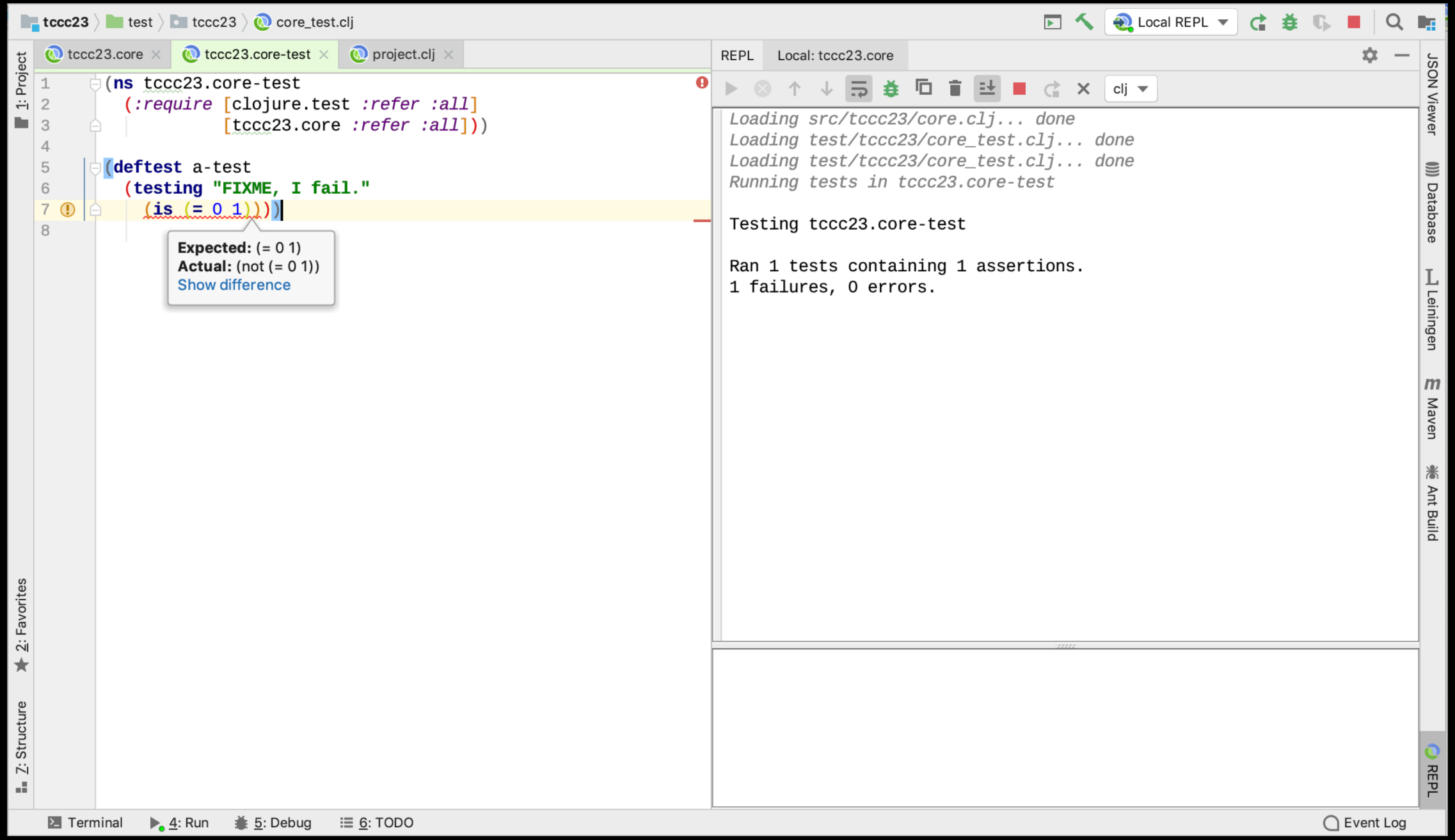
Rich comment blocks

- Sometimes just a capture (or subset) of your REPL session!
  - But usually curated...
- In practice, often serves as more helpful documentation than unit tests

But where are my unit tests?

- They exist...


tccc23 test tccc23 core_test.clj
Local REPL
tccc23.core
tccc23.core-test
project.clj
(ns tccc23.core-test
(:require [clojure.test :refer :all]
[tccc23.core :refer :all]))
(deftest a-test
(testing "FIXME, I fail."
(is (= 0 1)))
REPL
Local: tccc23.core
clj
Loading src/tccc23/core.clj... done
Loading test/tccc23/core_test.clj... done
Copy Reference ⌥⇧⌘C
Paste ⌘V
Paste from History... ⇧⌘V
Paste without Formatting ⌥⇧⌘V
Column Selection Mode ⇧⌘8
Find Usages ⌥F7
Refactor
Folding
Analyze
Go To
Generate... ^N
Run 'core_test' ^⇧F10
Debug 'core_test' ^⇧F9
Create 'core_test'...
Reveal in Finder
Open in Terminal
Local History
Compare with Clipboard
File Encoding
Create Gist...
REPL
Convert To...
Load file in REPL ⌥⇧L
Sync files in REPL ⌥⇧M
Send '(deftest a-test ...)' to REPL ⌥⇧P
Send '(deftest a-test ...)' to REPL ⌥⇧O
Switch REPL NS to current file ⌥⇧R
Clear output for current REPL
Jump to REPL editor ⌘\
Jump to REPL output pane
Print last exception
Interrupt Current Evaluation
Search REPL history ⌥⌘E
View expansion of '(deftest a-test ...)' ⇧⌘\
Run tests in current NS in REPL
Run test under caret in REPL
Re-run last test action in REPL
Clear all test markers
Type check current NS
Add new REPL command
Edit REPL commands
Commands
1: Project
2: Favorites
7: Structure
JSON Viewer
Database
Leiningen
Maven
Ant Build
REPL
Terminal
4: Run
5: Debug
6: TODO
Event Log


tccc23 / test / tccc23 / core_test.clj
Local REPL
tccc23.core
tccc23.core-test
project.clj
1 (ns tccc23.core-test
2 (:require [clojure.test :refer :all]
3 [tccc23.core :refer :all]))
4
5 (deftest a-test
6 (testing "FIXME, I fail."
7 (is (= 0 1))))
8
Expected: (= 0 1)
Actual: (not (= 0 1))
Show difference
REPL
Local: tccc23.core
clj
Loading src/tccc23/core.clj... done
Loading test/tccc23/core_test.clj... done
Loading test/tccc23/core_test.clj... done
Running tests in tccc23.core-test
Testing tccc23.core-test
Ran 1 tests containing 1 assertions.
1 failures, 0 errors.
1: Project
2: Favorites
7: Structure
JSON Viewer
Database
Leiningen
Maven
Ant Build
REPL
Terminal
4: Run
5: Debug
6: TODO
Event Log

But where are my unit tests?

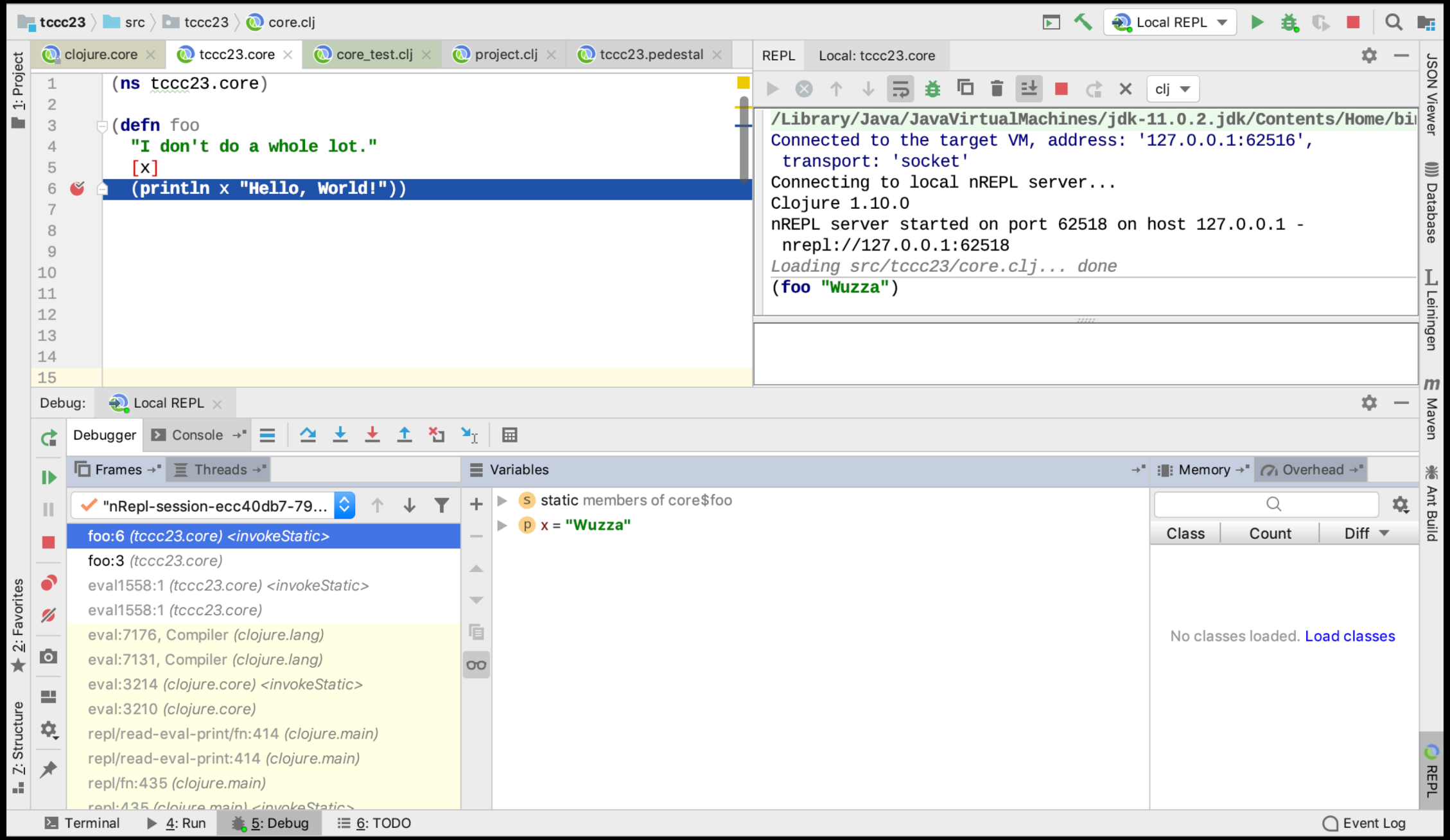
- They exist, but... what do you typically use unit tests for?
  - To see that something behaves the way you expect
  - To help you understand how you want something to behave
- Unit tests provide a fast feedback loop - unless you have a REPL, which is a faster feedback loop
- Functional code with immutable data eliminates the need for many unit tests

But where are my unit tests?

- clojure.spec goes above and beyond what is usually done with unit testing
  - clojure.spec.alpha/exercise
- Integration tests are still valuable!

But where’s my debugger?

- It exists...


tccc23 src tccc23 core.clj
Local REPL
clojure.core
tccc23.core
core_test.clj
project.clj
tccc23.pedestal
(ns tccc23.core)
(defn foo
"I don't do a whole lot."
[x]
(println x "Hello, World!"))
REPL
Local: tccc23.core
clj
/Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bi
Connected to the target VM, address: '127.0.0.1:62516',
transport: 'socket'
Connecting to local nREPL server...
Clojure 1.10.0
nREPL server started on port 62518 on host 127.0.0.1 -
nrepl://127.0.0.1:62518
Loading src/tccc23/core.clj... done
(foo "Wuzza")
Debug:
Local REPL
Debugger
Console
Frames
Threads
Variables
Memory
Overhead
"nRepl-session-ecc40db7-79...
foo:6 (tccc23.core) <invokeStatic>
foo:3 (tccc23.core)
eval1558:1 (tccc23.core) <invokeStatic>
eval1558:1 (tccc23.core)
eval:7176, Compiler (clojure.lang)
eval:7131, Compiler (clojure.lang)
eval:3214 (clojure.core) <invokeStatic>
eval:3210 (clojure.core)
repl/read-eval-print/fn:414 (clojure.main)
repl/read-eval-print:414 (clojure.main)
repl/fn:435 (clojure.main)
static members of core$foo
x = "Wuzza"
Class
Count
Diff
No classes loaded. Load classes
1: Project
2: Favorites
7: Structure
JSON Viewer
Database
Leiningen
Maven
Ant Build
REPL
Terminal
4: Run
5: Debug
6: TODO
Event Log

But where’s my debugger?

- It exists... But what do you typically use a debugger for?
  - To pause execution to see the state of an object or objects - because you don’t have a REPL
- But what if you didn’t have any state?
- What if you had state, but you could simply view the state whenever you wanted to, from the REPL?

Resources for Learning Clojure

- https://clojure.org
- Clojure for the Brave and True
  - https://www.braveclojure.com/clojure-for-the-brave-and-true
- http://4clojure.com
- http://clojurekoans.com
- The Joy of Clojure
  - https://www.manning.com/books/the-joy-of-clojure-second-edition
- Applied Clojure
  - https://pragprog.com/book/vmclojeco/clojure-applied

Resources for Using Clojure

- Community-driven documentation (different, both useful in different ways):
  - https://clojuredocs.org
  - http://clojure-doc.org
- https://leiningen.org
- Leiningen sample project file:
  - https://github.com/technomancy/leiningen/blob/master/sample.project.clj
- https://www.youtube.com/user/ClojureTV
  - In particular, any talks by Rich Hickey

Resources for Practitioners

- Clojurian Slack channels
  - http://clojurians.net
- Google Groups mailing list
  - https://groups.google.com/forum/#!forum/clojure
- https://www.clojure-toolbox.com/
- Clojure/conj 2019
  - Date and location TBD

Q & A

Primary font was Courier Prime Code; quote font was Candara.
No graphic designers were harmed in the making of this slide deck.