CLOJURE MEETUP - 17/05/2017

CLOJURE.SPEC IN PRACTICE

GOALS FOR TODAY

- Intro to Clojure Spec
- Practical examples
- What works, what does not

TOPICS FOR TODAY

- Support for generative testing
- Contracts vs Types
- Entities vs Associations

IN THEORY…

SPECS: THE GOOD PARTS

GOALS FOR CLOJURE.SPEC?

- Specify shape and invariants on data
- Add robustness, keep flexibility (change)
- Offer leverage on specification

SPECIFICATION ON DATA

```
{:player/blue   24
 :player/red    17
 :player/green  19}
```

SPECIFICATION ON DATA

```
(s/def :player/players
  #{:player/blue
    :player/red
    :player/green})
```

SPECIFICATION ON DATA

```
(s/def :player/scores
  (s/map-of
    :player/players
    pos-int?))
```

ROBUSTNESS

- Access to run-time values
- Access to good granularity
- Access to relation between instances

ROBUSTNESS

```
(s/def :game/board
  (s/every
    (s/every
      :player/players
      :count height)
      :count width))
```

FLEXIBILITY

- Avoid the coupling of type systems
- Express a commitment, a contract
- With evolution of software in mind

FLEXIBILITY

```
{:game/board       board
 :game/scores      scores
 :game/transitions transitions}
```

WHAT YOU HAVE, NOT CAN'T HAVE

```
(s/def :game/turn
  (s/keys :req
    [:game/board
     :game/scores]))
```

PRESENCE VS CONFORMANCE

```
(s/def :game/transitions map?)

{:game/board       board
 :game/scores      scores
 :game/transitions []} ;;BOOM!
```

LEVERAGE ON SPECIFICATION

- Validation, Instrumentation
- Parsing, Destructuring
- Generate samples / tests

GENERATING SAMPLES

```
(gen/valid? :game/turn turn)
```

```
(gen/conform :game/turn turn)
```

```
(gen/sample (s/gen :game/turn) 1)
```

SPECS IN PRACTICE…

SPECIFYING OUR GAME TURN

EXPRESSING COMMITMENT

- Define a contract with the client
  - Preconditions (never ask for more)
  - Postconditions (never provide less)
- Allows change, provides flexibility

EXPRESSING COMMITMENT

```
(s/fdef next-turn
  :args
  (s/cat :turn :game/turn
         :move :game/transition)
  :ret   :game/turn)
```

SYMMETRY FORCES FULL COMMITMENT

- Need transitions for next-turn
- Cannot provide the service without
- But no desire to commit to its presence

A DESIGN SIGN OF BAD DESIGN

- Implementation vs Informational
- Even if full commitment, bad generation
- Best spec is in term of succession => types

A BETTER DESIGN?

- Separate information from mechanism
- Generation with succession
- Provide spec for information only

SPECS IN PRACTICE…

ENTITIES VS ASSOCIATIONS

A DIFFERENT SPEC FOR SCORES

```
(s/def :player/red   int?)
(s/def :player/green int?)
(s/def :player/blue  int?)

(s/def :game/scores
  (s/keys :req
    [:player/red
     :player/green
     :player/blue]))
```

ENTITY VS ASSOCIATIONS

- Cannot use the key for other entity
- Use key sets for entities (membership)
- Use map-of for association

SPECS IN PRACTICE…

THE BINARY TREE CHALLENGE

THE BINARY TREE CHALLENGE

```
class BinaryTree<A>
{

    A value;
    BinaryTree<A> lhs;
    BinaryTree<A> rhs;

};
```

ASSOCIATIONS

```
(s/def :int-tree
  (s/cat
    :value int?
    :children
    (s/map-of #{:lhs :rhs}
              :int-tree))
```

USING MACROS FOR GENERICS

```
(def-btree-of
  :int-tree int?)

(def-btree-of
  :string-tree string?)
```

CLOJURE SPEC IN PRACTICE

CONCLUSION & LINKS

FINDING PROPERTIES

- Spec information, not implementation
- Contract system, not a type system
- Distinguish associations from entities

RESOURCES

- https://github.com/QuentinDuval/ClojureMeetup-2017-05-17
- https://clojure.org/about/spec
- https://clojure.org/guides/spec
- https://www.youtube.com/watch?v=oyLBGkS5ICk