Developing For The Web With Extreme Safety

github.com/DillonKearns
What Does Safety Look Like In Your Codebase?
Elm

- Language for client-side web apps
- Compiles to JavaScript
- Unlike TypeScript, ES2015, PureScript, etc., Elm lives in a sandbox
- Elm can call JavaScript, but not directly
- Based on Haskell
- Language choices favor simplicity
- Explicit over terse
- No runtime errors

github.com/DillonKearns/talks @dillontkearns
Safety Tips

1. Traceability
2. Domain Modeling
3. Consumer-Driven APIs
1) Traceability

- Uncertainty as data
- Managed effects

github.com/DillonKearns/talks @dillontkearns
1) Traceability

uncertaintyAsData -- How do you deal with uncertainty?
managedEffects

- No null, Maybe union instead

github.com/DillonKearns/talks @dillontkearns
Uncertainty as Data

The `Maybe` type

type Maybe value
    = Just value
    | Nothing

Examples

```
List.head [100, 200, 300] == Just 100

List.head [] == Nothing
```

github.com/DillonKearns/talks @dillontkearns
1) Traceability

uncertaintyAsData -- How do you deal with uncertainty?

managedEffects

- Can't raise exceptions
- No global exception handlers squelching errors
- No propagating hard-to-debug nulls from exception handlers
- Errors are traceable
- Exceptions are like `GOTOs`

github.com/DillonKearns/talks @dillontkearns
Uncertainty as Data

The Result type

type Result data error
  = Ok data
  | Err error

Examples

String.toInt "-42" == Ok -42
String.toInt "3.1" == Err "could not convert string '3.1' to an Int"

github.com/DillonKearns/talks @dillontkearns
Dealing With Uncertainty

```haskell
parseIntOrNegativeOne :: IntString -> Int
let
    toIntResult = String.toInt intString
in
    case toIntResult of
        Ok intValue -> intValue
        Err errorMessage -> -1

case can also be written as

Result.withDefault (-1) result
```

github.com/DillonKearns/talks @dillontkearns
1) **Traceability**

**uncertaintyAsData**

**managedEffects**

- Funnel all side-effects to one place
- Single place where non-determinism/side effects stem from
- Single place where the results of non-determinism/side-effects funnels into

github.com/DillonKearns/talks @dillontkearns
The Elm Architecture

Pure Functions

Side Effects & Mutation

update

view

Model
Msg

Model
Cmd

HTML

Elm

Virtual DOM
2) Domain Modeling

- Make Impossible States Impossible
- Contract-Driven Design
- More compiler guarantees => More focused tests & fewer bugs

github.com/DillonKearns/talks @dillontkearns
2) Domain Modeling

contracts -- decoders

impossibleStates -- union types

- Without contracts, errors creep deep into the system
  => harder to trace
- Untyped => Typed
- No exceptions, errors are just data
- Annoyance at first, can't live without it once you get used to it

github.com/DillonKearns/talks @dillontkearns
Decoding JSON

```json
json = { "first": "Beverly", "last": "Crusher" }
```

Discover contract violations at the gate

```go
decodeString (field "first" string) json
-- =>
Err "Expecting an object with a field named `name` but instead got: "...
```
Decoding JSON

json = { "first": "Beverly", "last": "Crusher" }

...Decode.decodeString (Decode.field "name" Decode.string) json
-- => Ok "Beverly"
Data Modeling Example

JavaScript
{
    hasError: true,
    errorMessage : 'Error message from server',
    doneLoading: true,
    data: null
}

github.com/DillonKearns/talks @dillontkearns
<table>
<thead>
<tr>
<th>data</th>
<th>doneLoading</th>
<th>hasError</th>
<th>errorMessage</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>Ø</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>✓</td>
<td>✓</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>Ø</td>
<td>✓</td>
<td>Ø</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>Ø</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>Ø</td>
<td>✓</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>✗</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>✓</td>
</tr>
</tbody>
</table>

github.com/DillonKearns/talks @dillontkearns
Elm Data Modeling

type RemoteData data -- we're defining a union type
  = NotAsked
  | Loading
  | Failure Http.Error
  | Success data

Constructing Union Types

Success { first = "James", last = "Kirk" }
Failure Http.Timeout
Loading

github.com/DillonKearns/talks @dillontkearns
3) Consumer-Driven APIs

What does UX for APIs have to do with safety?

- Guide towards correct use
- Minimal interface
- Trying to support everything is a hazard
- Start opinionated, minimal
- Introduce flexibility as needed

github.com/DillonKearns/talks @dillontkearns
Style Elements Package

- Fully specify your layouts inline
- No cascading, no indirection
- Plain code, no separate DSL for classes, ids, stylesheets
- If there is a layout problem, you know exactly where to look
- Helps enforce accessibility

github.com/DillonKearns/talks @dillontkearns
Misleading Error

```
~/src/myproject elm test --init

No tests found in the test/ (or tests/) directory.

NOTE: Make sure you’re running elm-test from your project's root directory, where its elm-package.json lives.

To generate some initial tests to get things going, run elm-test init
~/src/myproject
```
var args = minimist(process.argv.slice(2), {
  alias: {
    help: "h",
    fuzz: "f",
    seed: "s",
    compiler: "c",
    "add-dependencies": "a",
    report: "r",
    watch: "w"
  },
  boolean: ["warn", "version", "help", "watch"],
  string: ["add-dependencies", "compiler", "seed", "report", "fuzz"]
});

function runElmTest() {
  checkNodeVersion();

  if (args._[0] == "init") {
    var cmdArgs = Init.init();
    var cmd = [pathToElmPackage, "install", "--yes"].concat(cmdArgs).join(" ");

    child_process.execSync(cmd, { stdio: "inherit", cwd: Init.elmPackageDir });

    process.exit(0);
  }
}
- Declarative vs. Imperative
- Upfront contract enforcement (eager)
- Can't access data that won't be present
- Helps enforce good UX
GraphQL Enforce Contracts at the Gate

```graphql
{
  human(id: "1000") {
    name
    height
  }
}
```

github.com/DillonKearns/talks @dillontkearns
Minimize Constructs

Variables

```graphql
query HeroName($episode: Episode) {
  hero(episode: $episode) {
    name
  }
}
```

github.com/DillonKearns/talks @dillontkearns
Minimize Constructs

Aliases & Fragments

```json
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

github.com/DillonKearns/talks @dillontkearns
Graphqelm

- Generate Elm code based on your GraphQL schema
- If it compiles, it's valid
- Types are known at compile-time

[github.com/DillonKearns/talks](https://github.com/DillonKearns/talks) @dillontkearns
Takeaways

- Work backwards from constraints
- Guarantees are better than discipline
- Fewer features means more safety
(1) **Traceability**
- Uncertainty as data
- Managed effects

(2) **Domain Modeling**
- Make Impossible States Impossible
- Contract-Driven Design

(3) **Consumer-Driven APIs**
- Guide towards correct use
- Minimal interface
- Opinionated, minimal

[github.com/DillonKearns/talks @dillontkearns](https://github.com/DillonKearns/talks @dillontkearns)
Thank You!

Questions?

github.com/DillonKearns/talks @dillontkearns