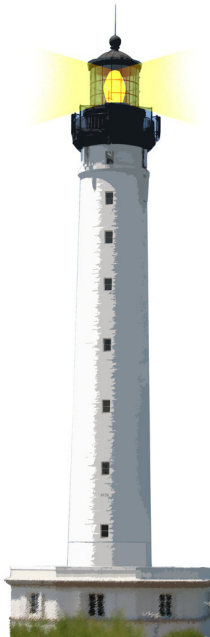
About Validation

A pretext to talk about delegation of actions

S. Ducasse



Goal

- How can we navigate an instance tree?
- With optional states that can be skipped?

The case: Validation

- We want to validate UI forms
- Nested components may want to validate or not their contents

Screenshot here

A tree of instances

Screenshot here

Validate first design

- Any presenter can validate its contents.
- Per dafault does not nothing.

```
SpPresenter >> validate
  ^ self
```

```
SpOptionPresenter >> validate

  | report |
  report := SpValidationReport new.
  self layout children do: [ :presenter | presenter validate ]
```

Analysis

- We need to have a report to know if the validation failed or not.
- Should validate return a report?
- If validate returns a report then we have to return an ok report for anybody.
- We do not want to force a report for all the tree instances.

Second design: let the object decide

Pass around a basket and let any sub instance decides if it wants to participate

```
SpPresenter >> validateInto: aReport
  ^ self
```

```
SpOptionPresenter >> validate

  | report |
  report := SpValidationReport new.
  self layout children do: [ :presenter | presenter validateInto: report ].
  ^ report
```

Local and global together

```
SpTextInputFieldWithValidation >> validateInto: aValidationReport
  self validate.
  aValidationReport addAll: validationErrors
```

Each validating subcomponent

- can bring its information to the report
- gets the responsibility to fill up the report

Conclusion

- Let the object decides if it wants to join a process but passing a container

A course by

S. Ducasse, L. Fabresse, G. Polito, and Pablo Tesone