End the Software Crisis!

Nicole Rauch
@nicolerauch

Michael Sperber
@sperbsen
• software project development
• Scala, Clojure, Erlang, Haskell, F#, OCaml
• training, coaching
• co-organize BOB conference

www.active-group.de
funktionale-programmierung.de
Nicole Rauch

• Independent Software Dev and Dev Coach
• EventStorming - Domain-Driven Design
• Software Craftsmanship
• React.js and Redux
• Functional Programming
Software Crisis

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.


Wikipedia: Software crisis
Crisis? What Crisis?
Software Crisis

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

Wikipedia: Software crisis
This Crisis!

• Pentium FDIV bug (1994)
• Denver airport (1995)
• German Toll Collect (2003)
• healthcare.gov (2013)
• Meltdown/Spectre (2018)
Functional Programming!

1. immutable data
2. less coupling
3. verification
4. catamorphisms
5. bifunctors
6. monads
7. monadic profunctors
8. Kleisli arrows
ELEPHANT IN THE
i.T. ROOM?

LACK OF
HUMAN INTERACTIONS,
COMMUNICATION &
UNDERSTANDING...

YOU "IDIOT!"

SIDE-EFFECT
YOURSELF!

FP
MONADS

OOP FOREVER
(SOLID)
Agile Revolution

Warum haben wir diesen Zustand immer noch?

Funktioniert „Agile Software Engineering“ wirklich?

29% 
Neue Features

53% 
Wartung, Komplexität

18% 
„Services Professionalisierung“

Fahd Al-Fatish: Software Engineering - the roots, Karlsruher Entwicklertage 2017
Weitere Folie vom Fahd
Imperative Programming

room1.exit(elephant)
hallway.enter(elephant)
hallway.exit(elephant)
room2.enter(elephant)
Reality and Snapshots
Problem

model

change

model'

view

hopefully corresponding

change

view'

model

change

view
MVC
React

- model
- new model
- view
- new view

User

generates

generates
class Square extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: null,
    };
  }
  render() {
    return (
      <button className="square" onClick={() => this.setState({value: 'X'})} >
        {this.state.value}
      </button>
    );
  }
}
setState antipattern

Dan Abramov on Twitter: "When people say setState() is an anti ...  
https://twitter.com/dan_abramov/status/725090047557558272?lang=en  
Apr 26, 2016 - There is no “anti-pattern” in having a <Dropdown> keep isOpen in local state. If it was an anti-pattern, React wouldn’t have this feature.
Alan Kay on OO

Though OOP came from many motivations, two were central. [...] to find a more flexible version of assignment, and then to try to eliminate it altogether.

Alan Kay, *History of Smalltalk*  
Communications of the ACM, 1996
# OO vs. State

[Diagram showing examples of classes with their variables and methods]

**Examples of classes**

<table>
<thead>
<tr>
<th><strong>Name (Identifier)</strong></th>
<th><strong>Student</strong></th>
<th><strong>Circle</strong></th>
<th><strong>SoccerPlayer</strong></th>
<th><strong>Car</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variables</strong></td>
<td>name</td>
<td>radius</td>
<td>name</td>
<td>plateNumber</td>
</tr>
<tr>
<td>(Static attributes)</td>
<td>gpa</td>
<td>color</td>
<td>number</td>
<td>xLocation</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>getName()</td>
<td>getRadius()</td>
<td>run()</td>
<td>yLocation</td>
</tr>
<tr>
<td>(Dynamic behaviors)</td>
<td>setGpa()</td>
<td>getArea()</td>
<td>jump()</td>
<td>speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kickBall()</td>
<td>move()</td>
</tr>
</tbody>
</table>

https://www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html
Valid use of inheritance

- Model inheritance
  - Subtype inheritance
  - View inheritance
  - Restriction inheritance
    - Functional variation inheritance
    - Type variation inheritance

- Variation inheritance
  - Extension inheritance
  - Reification inheritance
    - Structure inheritance
    - Implementation inheritance

- Software inheritance
  - Facility inheritance
    - Constant inheritance
  - Machine inheritance
FP for the Win!

- FIXME: like the first one
- higher productivity
- less complexity
- better modularity

Source: F# for fun and profit, Cycles and modularity in the wild
# Yale Study

<table>
<thead>
<tr>
<th>Language</th>
<th>Lines of code</th>
<th>Lines of documentation</th>
<th>Development time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Haskell</td>
<td>85</td>
<td>465</td>
<td>10</td>
</tr>
<tr>
<td>(2) Ada</td>
<td>767</td>
<td>714</td>
<td>23</td>
</tr>
<tr>
<td>(3) Ada9X</td>
<td>800</td>
<td>200</td>
<td>28</td>
</tr>
<tr>
<td>(4) C++</td>
<td>1105</td>
<td>130</td>
<td>–</td>
</tr>
<tr>
<td>(5) Awk/Nawk</td>
<td>250</td>
<td>150</td>
<td>–</td>
</tr>
<tr>
<td>(6) Rapide</td>
<td>157</td>
<td>0</td>
<td>54</td>
</tr>
<tr>
<td>(7) Griffin</td>
<td>251</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>(8) Proteus</td>
<td>293</td>
<td>79</td>
<td>26</td>
</tr>
<tr>
<td>(9) Relational Lisp</td>
<td>274</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>(10) Haskell</td>
<td>156</td>
<td>112</td>
<td>8</td>
</tr>
</tbody>
</table>

Hudak, Jones: Haskell vs. Ada vs. C++ vs. Awk vs. ...  
An Experiment in Software Prototyping Productivity, Yale University, 1993
The Real\textsuperscript{(TM)} Questions

- What adds business value?
- What do the customers want?
- What do the domain experts talk about?
- How can we transform this into code?
Domain-Driven Design

Establish shared mental model across
  Stakeholders
  Domain Experts
  Developers
  Code
DDD with OO

- Classes can be used as Entities
- They can also serve as Value Objects
- Aggregates can be expressed through references
- Boundaries are part of most languages
- "The Code is the Design"
Huh?

- Student
  - Name
  - Address
  - Phone Number
  - Email Address
  - Student Number
  - Average Mark
  - Is Eligible To Enroll
  - Get Seminars Taken

- Enrollment
  - Marks Received
    - Get Average To Date
    - Get Final Mark

- Seminar
  - Name
  - Seminar Number
  - Fees
  - Add Student
  - Drop Student

- Professor
  - Name
  - Address
  - Phone Number
  - Email Address
  - Salary

- on waiting list
- 0..* (ordered, FIFO)
- 1..* in
- 1

- ?Some seminars may not have an instructor?
public interface ShowerProduct {
    double soapProportion();
}
public class Soap implements ShowerProduct {
    private final String color;
    private final double ph;

    public Soap(String color, double ph) {
        this.color = color;
        this.ph = ph;
    }

    public double soapProportion() {
        return 1.0;
    }
}
enum HairType { OILY, DRY, NORMAL, DANDRUFF };

public class Shampoo implements ShowerProduct {
    private final String color;
    private final HairType hairType;

    public Shampoo(String color, HairType hairType) {
        this.color = color;
        this.hairType = hairType;
    }

    public double soapProportion() {
        return 0.0;
    }
}

public class Mixture implements ShowerProduct {
    private final ShowerProduct product1;
    private final double proportion1;
    private final ShowerProduct product2;

    public Mixture(ShowerProduct product1, double proportion1,
                   ShowerProduct product2) {
        this.product1 = product1;
        this.proportion1 = proportion1;
        this.product2 = product2;
    }

    public double soapProportion() {
        return (this.product1.soapProportion() * this.proportion1) +
               (this.product2.soapProportion() * (1.0 - this.proportion1));
    }
}
Haskell

data Color = Color String

data PH = PH Double

data HairType = Oily | Dry | Normal | Dandruff

data ShowerProduct =
  Soap Color PH
  | Shampoo Color HairType
  | Mixture Double ShowerProduct ShowerProduct
Haskell

\[
\text{soapProportion} :: \text{ShowerProduct} \to \text{Double}
\]

\[
\begin{align*}
\text{soapProportion} (\text{Soap } _) & = 1.0 \\
\text{soapProportion} (\text{Shampoo } _) & = 0.0 \\
\text{soapProportion} (\text{Mixture } p1 \ sp1 \ sp2) & = \\
& p1 \times (\text{soapProportion} \ sp1) + \\
& (1 - p1) \times (\text{soapProportion} \ sp2)
\end{align*}
\]
data Color = Color String deriving Eq

data PH = PH Double deriving Eq

data HairType = Oily | Dry | Normal | Dandruff
  deriving Eq

data ShowerProduct =
  Soap Color PH
  |
  Shampoo Color HairType
  |
  Mixture Double ShowerProduct Double ShowerProduct
  deriving Eq
data Id = Id Int deriving Eq

data Entity state = Entity Id state

instance Eq (Entity a) where
  (Entity id1 _) == (Entity id2 _) = id1 == id2

data Amount = Amount Double

type Stash = Entity (ShowerProduct, Amount)
Side Effects ...

- FIXME
- predictable projects
- easier testing
- easier transition to new tech / multicore / ...
- new domain insights