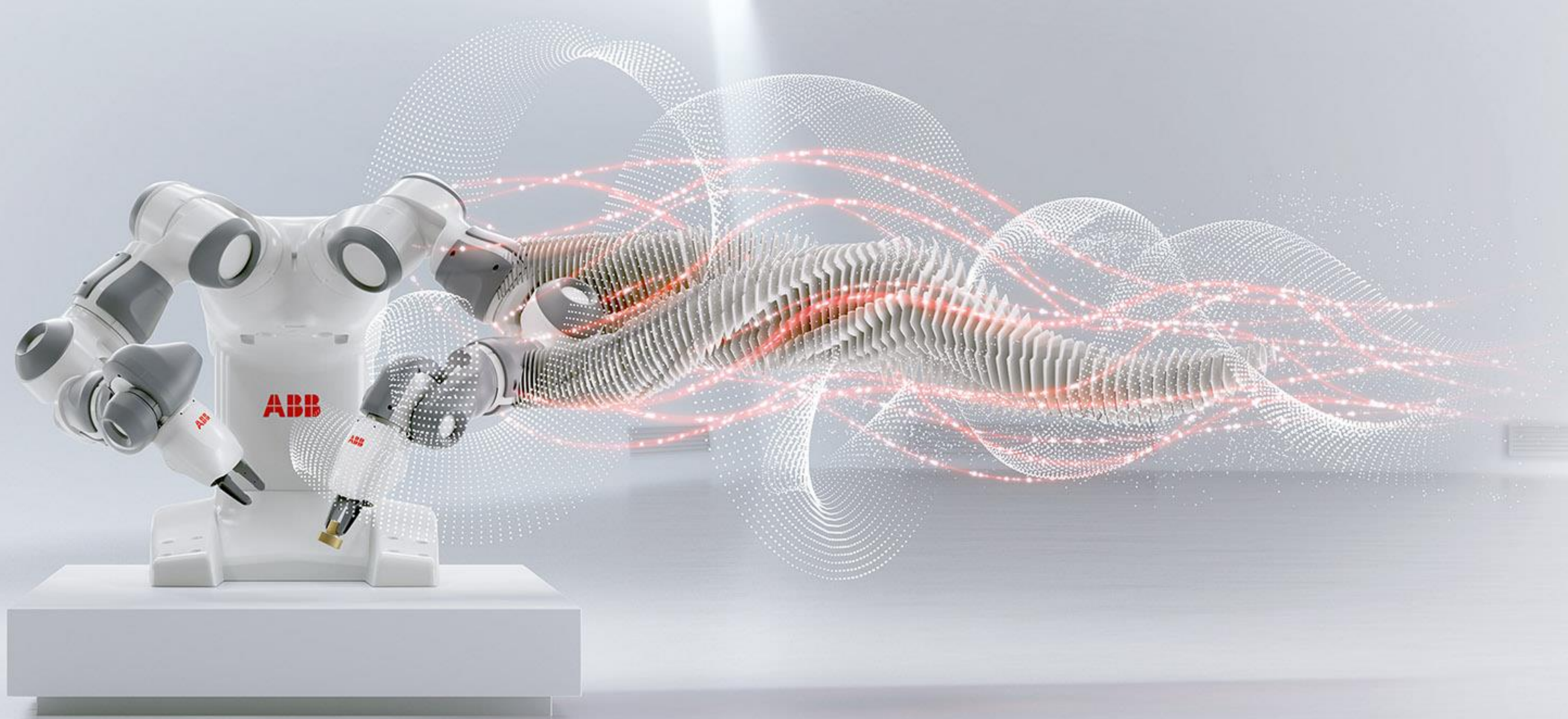


Functional programming in the ”real” world

Stefan Karlsson

Agenda

- What is the image of FP in industry, in my experience
- Why is FP relevant for the industry, in my experience
- Strategies and learnings from promoting FP in industry

Goal

- To assure you that FP is of high importance
- To inspire you to become a champion for FP

Audience

- What is your experience?
- 220x15x8 = 26400h (excluding my spare time coding)
- Interrupt with questions if anything is unclear!

Pre-university era

The Secret of Monkey Island
Created and Designed by Ron Gilbert

- Exposed to computing via C64/128 and NES (~1986)
  - Grown ups left us alone, they had no experience
- Amiga (~1992)
  - Entered my first program in Easy AMOS
- Apple
  - Basic
- PC – 486DX 33Mhz (~1995)
  - Pascal
  - C++

Education and Professional experience

- M.Sc. in Computer Science 2004
- M.Sc. In Computer Engineering 2005 (Civilingenjör)
- Software engineer at ABB 2006-2009
  - System 800xA, DCS
  - Lived and worked in Japan for 6 months
- Software engineer at Packsize 2009-2014
  - Machine manufacturer for packaging machines
- Senior Software engineer at ABB 2014-current
  - Services around System 800xA such as Mobile applications
- ARRAY Industrial PhD Student 2019-current

PACKSIZE
ON DEMAND PACKAGING
PACKSIZE
Track 3
Track 2

My road to functional programming

- Took a course in Haskell in the early 200X
- Then.. Nothing
- In 2011-2012 I felt a “peak” with C#
  - The same problems surfaced over and over..
- Started to look around for other languages
- Tried several languages (Python, Scala, F#, etc.)
- Found Clojure
- Clojure (FP it turns out) addressed many of the problems I had observed in industry
  - Distributed stateful objects with uncontrolled mutations
- Today, not using FP is painful..
- (Notice how many FP features have been added to C#/Java)

Many of the early programming language trade offs are not true any longer

- Immutability have a cost
- If all you have is 4096KB of memory, then update-in-place might be the only choice
- Many languages are based on premises true in the 1960-1970
  - Memory was small and expensive
  - Disk was small and expensive
  - CPUs were single threaded
- Today, developers are more expensive then hardware
  - If it takes me 2x longer to write code that express 10x more it is a win
  - A common case in Clojure is a 10x reduction in LOC compared to a Java solution

Simple Made Easy – Rich Hickey

- Highly influential talk for me on my road to FP
  - Mentions many of the pains I had experienced as a practitioner in industry.
- https://www.infoq.com/presentations/Simple-Made-Easy-QCon-London-2012/

A common industry attitude

“ Functional programing is nice for toy problems, but you can’t make real things with it”

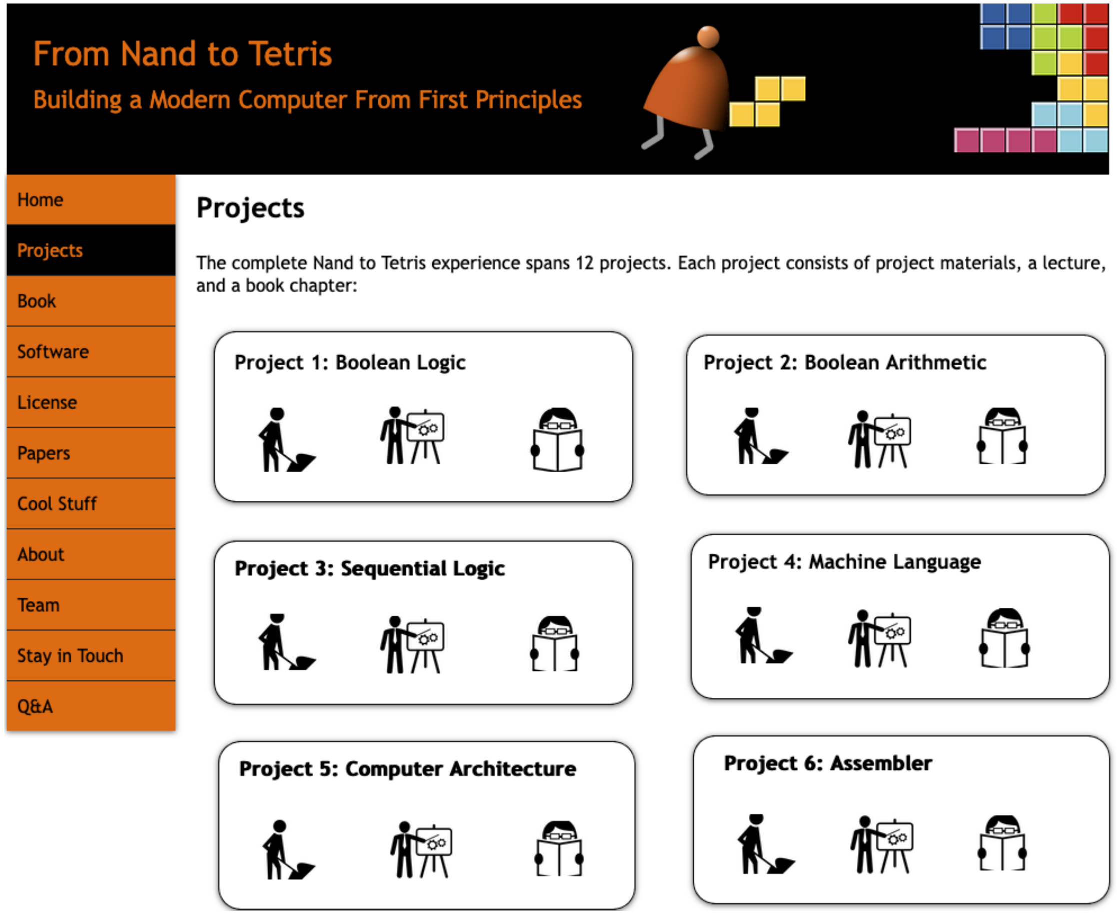
How functional programming was (is?) taught at university

- Mathy type of exercises
- No projects of building ”real” things
- So the impression that FP is for academic exercises persists
- Out of 5 years in CS education at the university I did 5p (7,5p) FP out of 200p (300p)
  - Hard for students to see the value with such low exposure
- “You build real things with real languages like C/C++/C#/Java”

Start building “real stuff”

- CRUD Web Application
- UI front end
- Compiler
- Etc.

I practiced Clojure by doing this



From Nand to Tetris

Building a Modern Computer From First Principles

Projects

The complete Nand to Tetris experience spans 12 projects. Each project consists of project materials, a lecture, and a book chapter:

Project 1: Boolean Logic

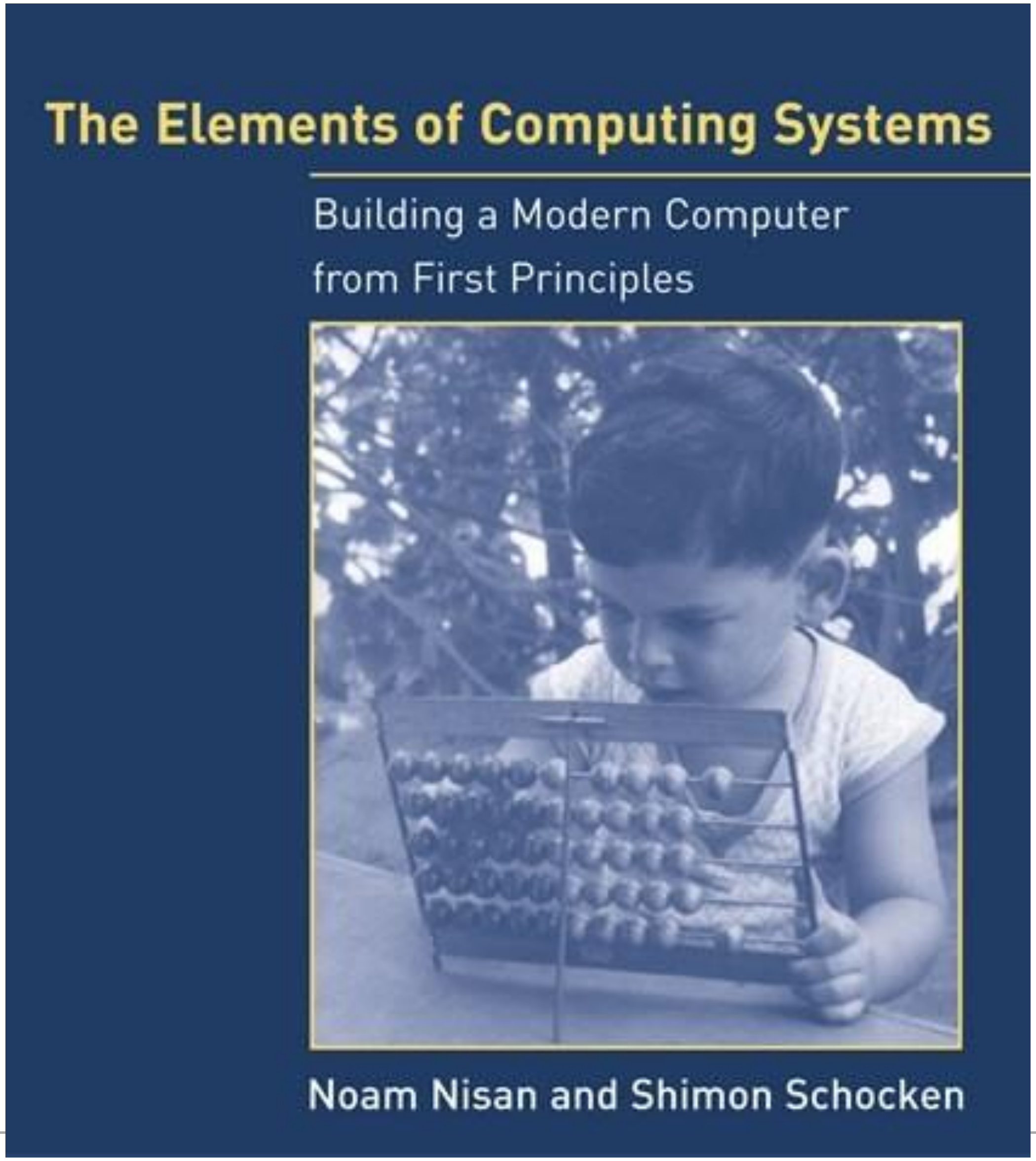
Project 2: Boolean Arithmetic

Project 3: Sequential Logic

Project 4: Machine Language

Project 5: Computer Architecture

Project 6: Assembler


The Elements of Computing Systems
Building a Modern Computer from First Principles
Noam Nisan and Shimon Schocken

Some high-profile industry users

NETFLIX

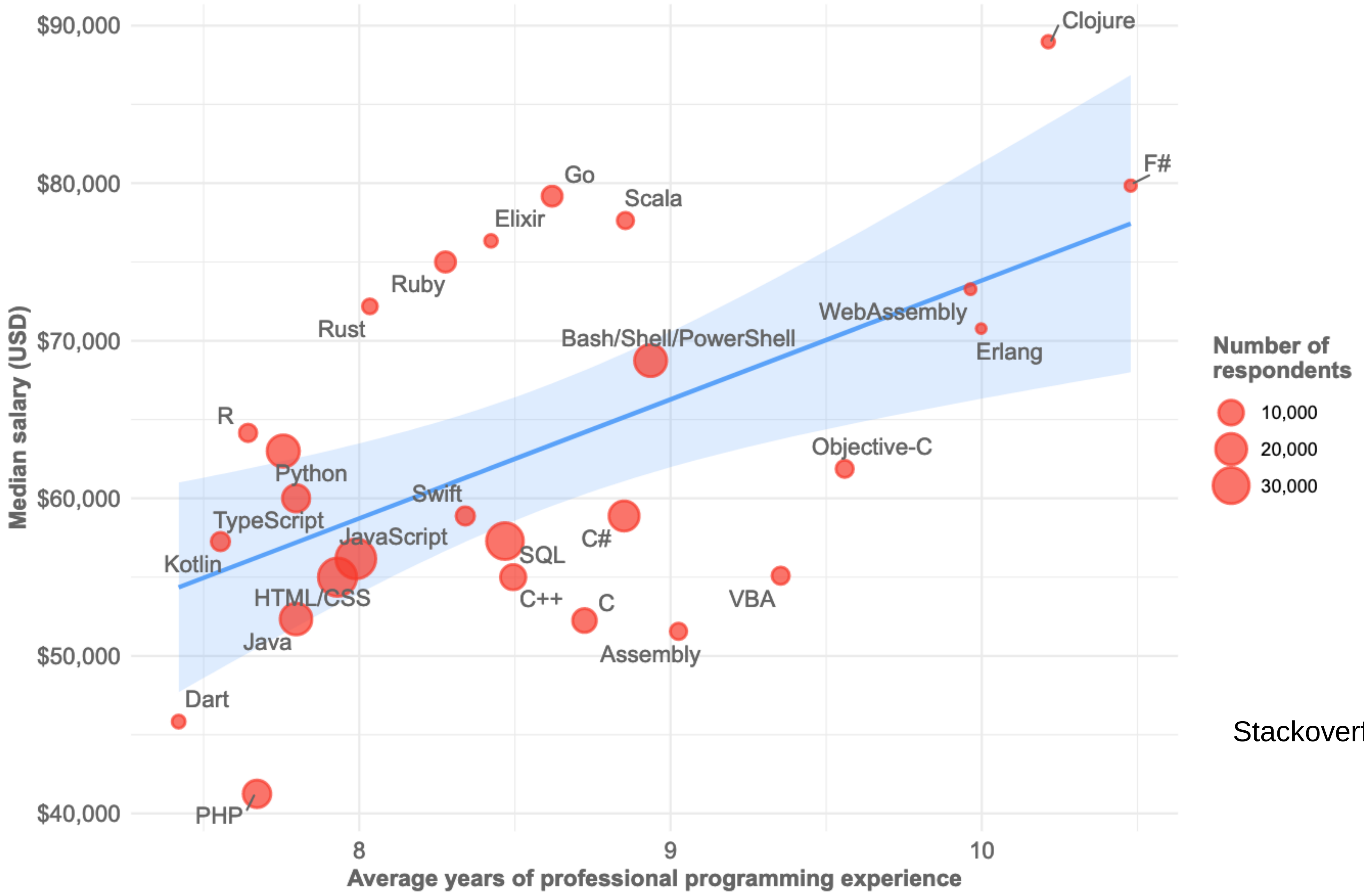
cisco™

Walmart

Klarna.

ebay™

Salary and Experience by Language

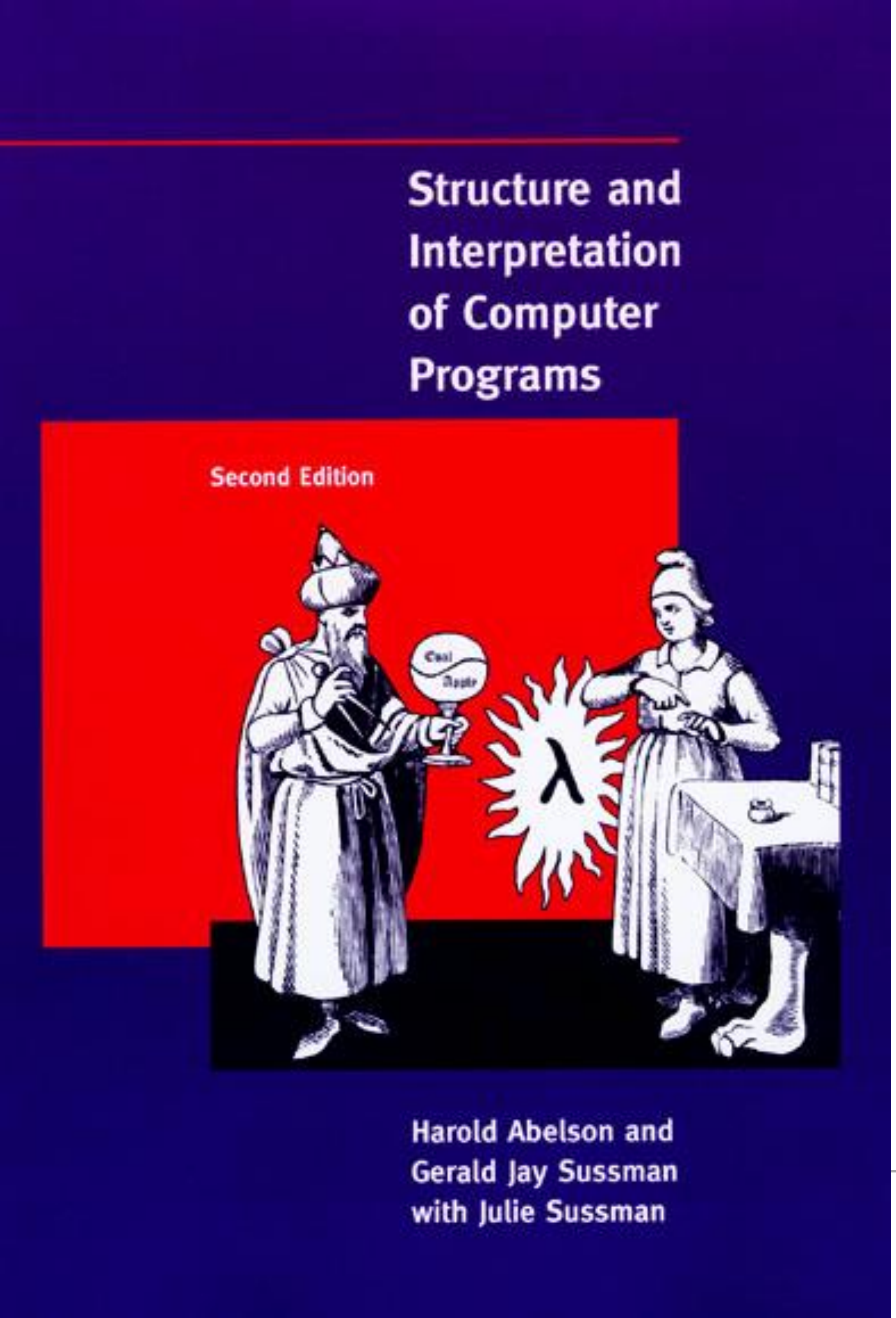


Stackoverflow survey 2019

My theory of why experienced programmers are drawn to FP

Mutable State

- It will make any non-trivial application harder to reason about
  - Generations of programmers’ effort wasted..
- The basis of logic depends on things not changing
  - Debugging concurrent systems with mutable state is extremely hard
- In my experience, it do not scale!
  - Immutability and data driven pure functions do


Structure and Interpretation of Computer Programs
Second Edition
λ
Harold Abelson and Gerald Jay Sussman with Julie Sussman

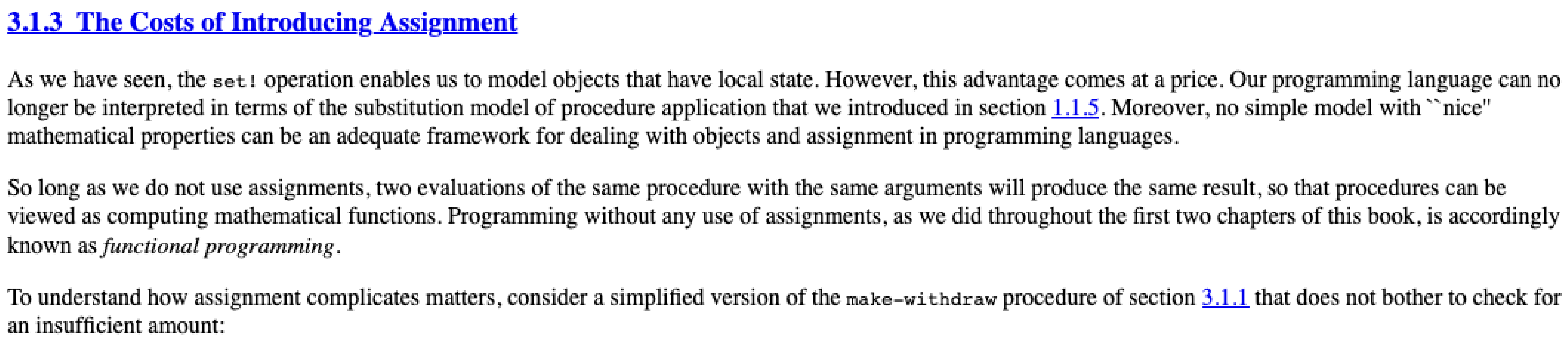
SICP - ”The wizard book”

Uses Scheme

”If programming was a religion, this would be the holy book” - Stefan Karlsson

https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book.html

WHY DIDN’T ANYONE TELL ME!



3.1.3 The Costs of Introducing Assignment

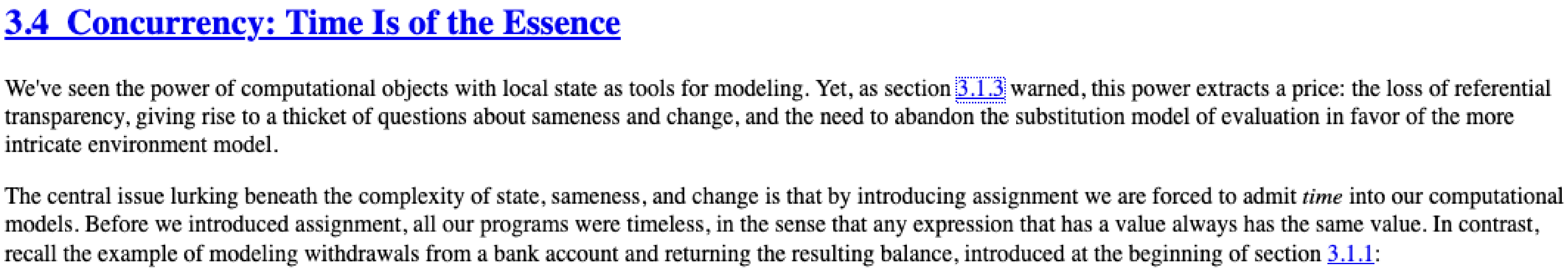
As we have seen, the `set!` operation enables us to model objects that have local state. However, this advantage comes at a price. Our programming language can no longer be interpreted in terms of the substitution model of procedure application that we introduced in section 1.1.5. Moreover, no simple model with ``nice'' mathematical properties can be an adequate framework for dealing with objects and assignment in programming languages.

So long as we do not use assignments, two evaluations of the same procedure with the same arguments will produce the same result, so that procedures can be viewed as computing mathematical functions. Programming without any use of assignments, as we did throughout the first two chapters of this book, is accordingly known as *functional programming*.

To understand how assignment complicates matters, consider a simplified version of the `make-withdraw` procedure of section 3.1.1 that does not bother to check for an insufficient amount:

Basis for logic!

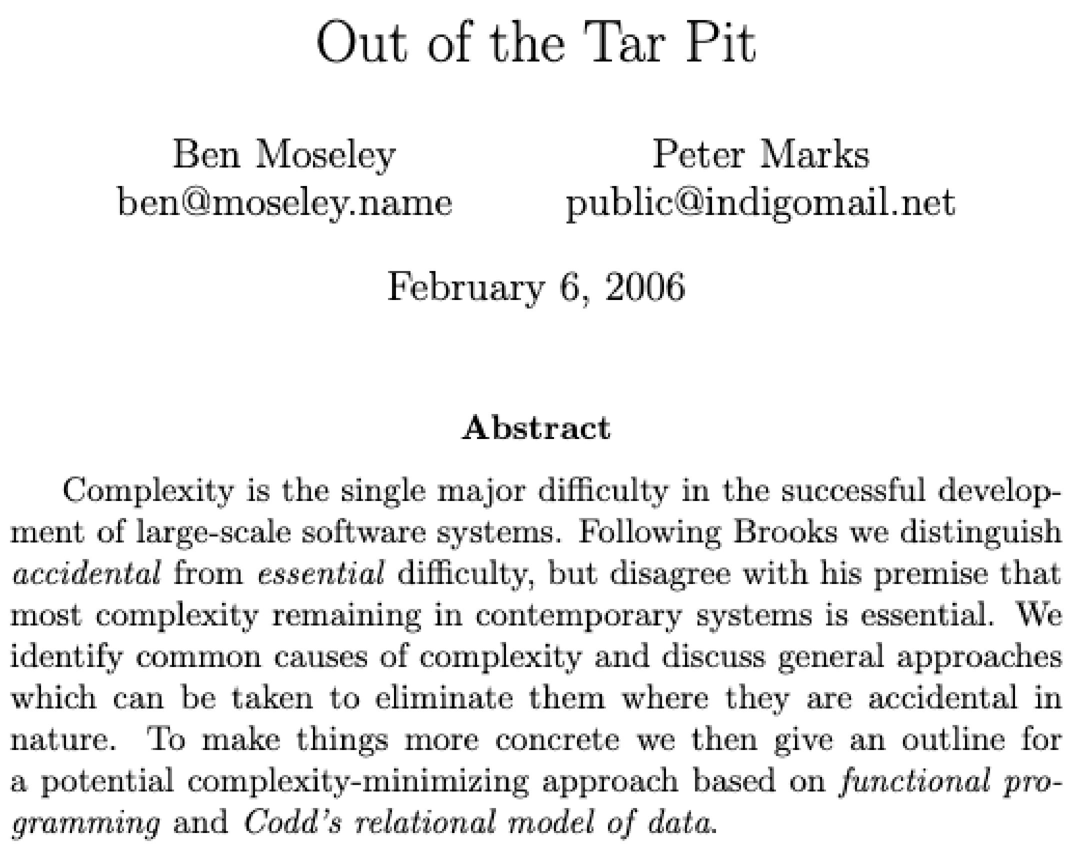
Again..



3.4 Concurrency: Time Is of the Essence

We've seen the power of computational objects with local state as tools for modeling. Yet, as section 3.1.3 warned, this power extracts a price: the loss of referential transparency, giving rise to a thicket of questions about sameness and change, and the need to abandon the substitution model of evaluation in favor of the more intricate environment model.

The central issue lurking beneath the complexity of state, sameness, and change is that by introducing assignment we are forced to admit *time* into our computational models. Before we introduced assignment, all our programs were timeless, in the sense that any expression that has a value always has the same value. In contrast, recall the example of modeling withdrawals from a bank account and returning the resulting balance, introduced at the beginning of section 3.1.1:



Out of the Tar Pit

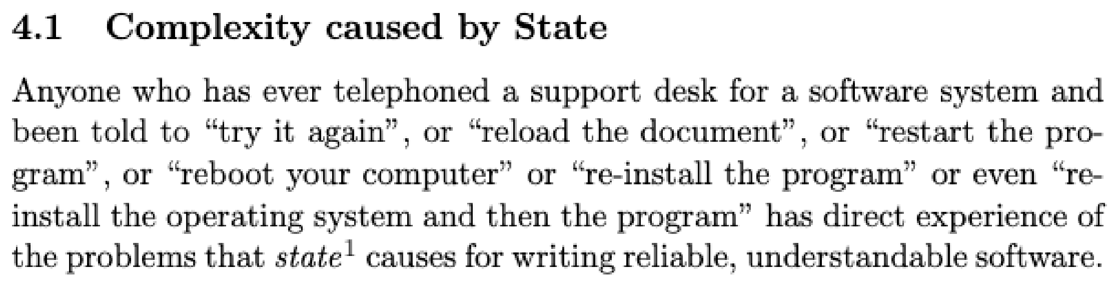
Ben Moseley
ben@moseley.name

Peter Marks
public@indigomail.net

February 6, 2006

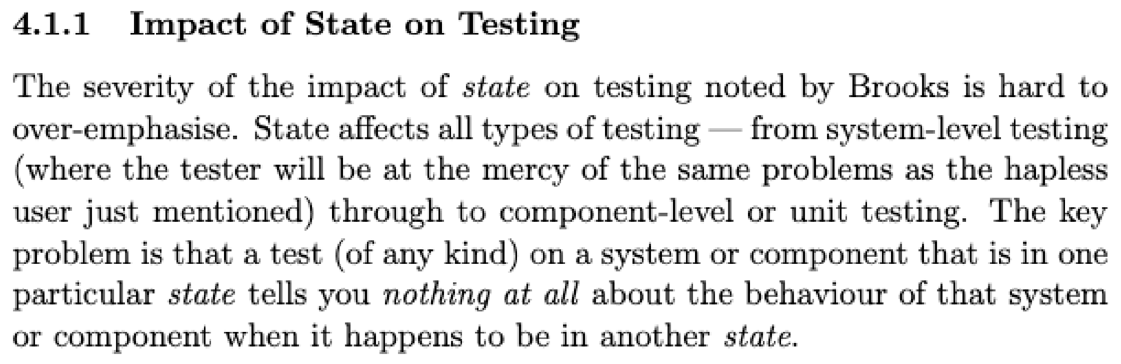
Abstract

Complexity is the single major difficulty in the successful development of large-scale software systems. Following Brooks we distinguish *accidental* from *essential* difficulty, but disagree with his premise that most complexity remaining in contemporary systems is essential. We identify common causes of complexity and discuss general approaches which can be taken to eliminate them where they are accidental in nature. To make things more concrete we then give an outline for a potential complexity-minimizing approach based on *functional programming* and *Codd's relational model of data.*



4.1 Complexity caused by State

Anyone who has ever telephoned a support desk for a software system and been told to "try it again", or "reload the document", or "restart the program", or "reboot your computer" or "re-install the program" or even "re-install the operating system and then the program" has direct experience of the problems that *state*[1] causes for writing reliable, understandable software.



4.1.1 Impact of State on Testing

The severity of the impact of *state* on testing noted by Brooks is hard to over-emphasise. State affects all types of testing — from system-level testing (where the tester will be at the mercy of the same problems as the hapless user just mentioned) through to component-level or unit testing. The key problem is that a test (of any kind) on a system or component that is in one particular *state* tells you *nothing at all* about the behaviour of that system or component when it happens to be in another *state*.

Rich Hickey fan club

https://github.com/tallesl/Rich-Hickey-fanclub

A collection about Rich Hickey's works on the internet.

State again..

Global variables taught as bad

- But Fields in OO are ok for some reason..
- What's the difference?

Guard your state with warning signs

- Make it explicit!
- Transactions for all state not just DB!
- FP solves this with immutability by default

But we need some state to make interesting programs

There are immutable databases!

C R U X

Datomic

User interface – great gateway to FP

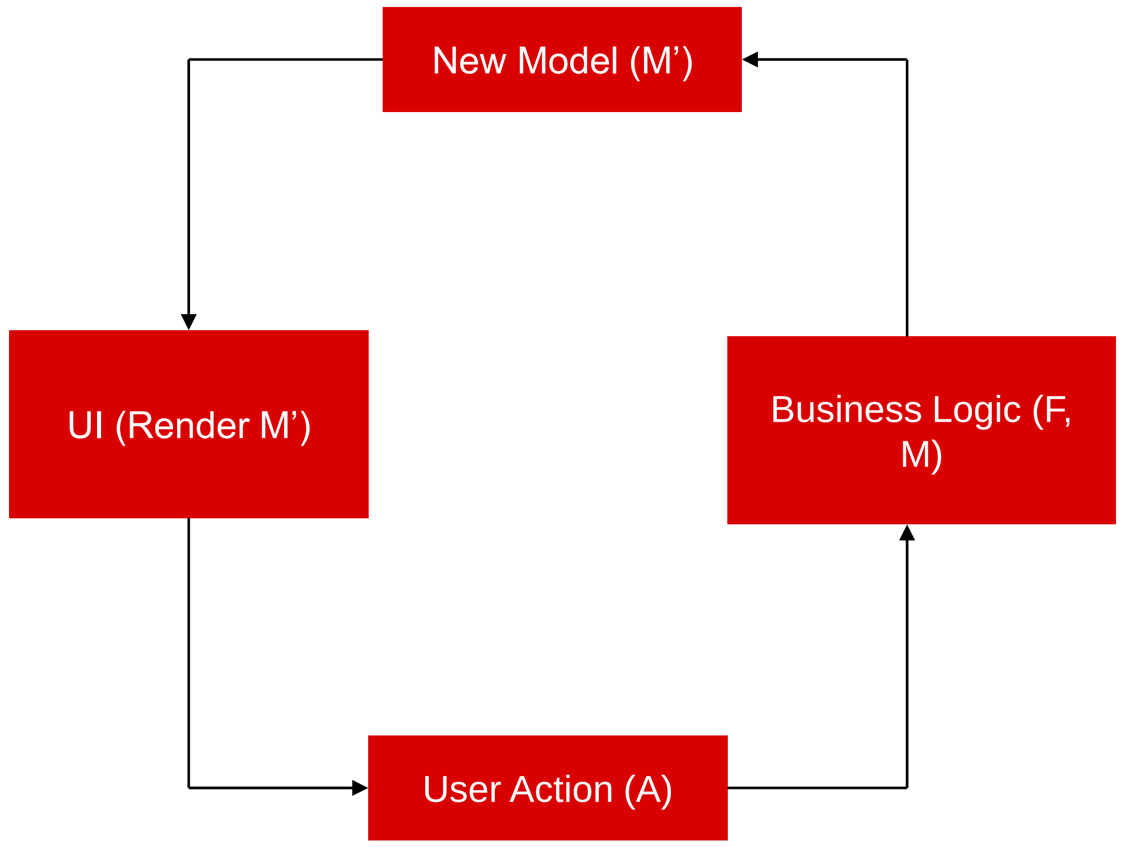
React

Vue.js

The power of immutability, a ClojureScript **<u>wrapper</u>** around React was faster than React!

Functional UI

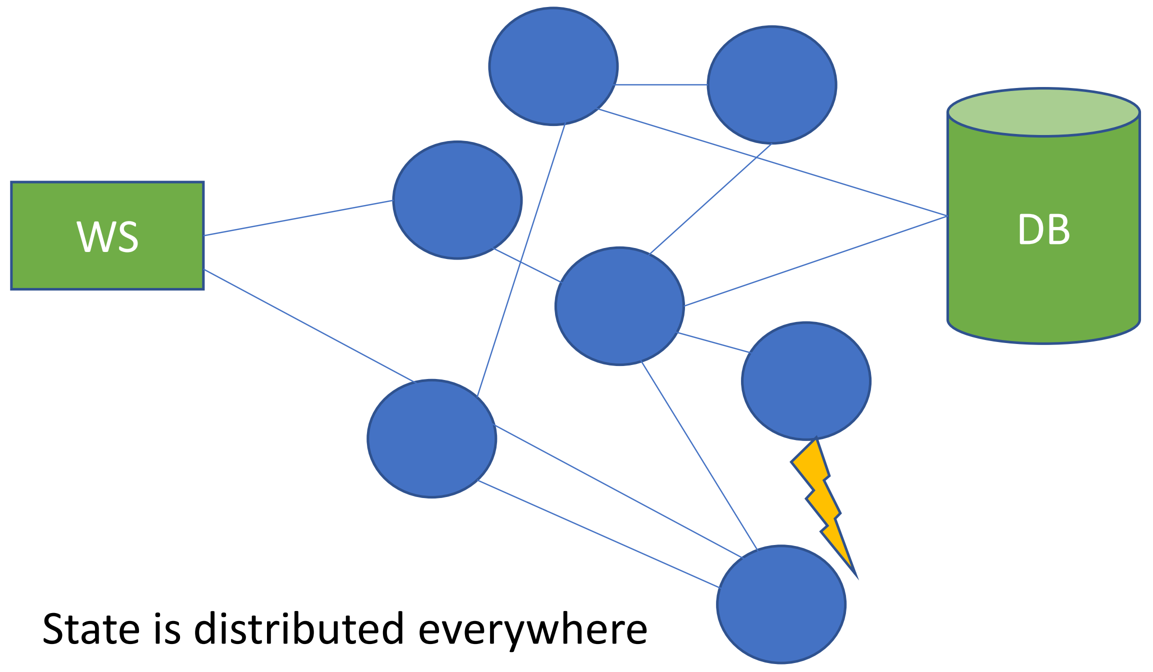
M’ = F(M, A)



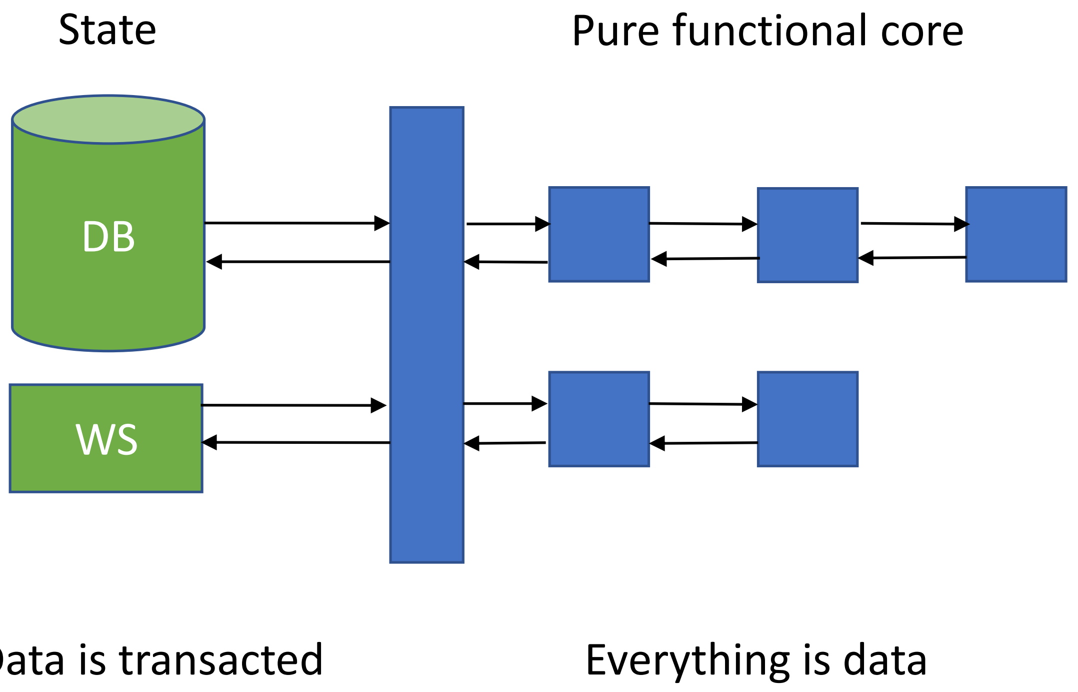
The UI renders the new model which is produced by applying the business logic function with the current model state and user action as input

Functional core, imperative shell

- Keep state at the edge
- Use the same FP principles on all levels of the system
- Simplifies every aspect of development



State is distributed everywhere



Data is transacted

Everything is data

The FP mutability trap

- F# and Scala mutable objects “trap”
- Tutorials from people with an OO mindset will just change the syntax not the principles
  - You get “Java in Scala”

Real reuse is the function not the object

”I think the lack of reusability comes in object-oriented languages, not functional languages. Because the problem with object-oriented languages is they’ve got all this implicit environment that they carry around with them. **You wanted a banana but what you got was a gorilla holding the banana and the entire jungle**.

If you have referentially transparent code, if you have pure functions — all the data comes in its input arguments and everything goes out and leave no state behind — it’s incredibly reusable.”

- Joe Armstrong

Joe Armstrong

- “Making reliable distributed systems in the presence of software errors”
  - Great read
- Inventor of Erlang
  - Functional and dynamically typed language aimed for reliability

A political battle not a technical one

- I have built successful services with Clojure
  - Had to throw it away due to change in management..
- Built web services with Clojure
  - Had to throw them away due to project management decision..
- Not once have there been any technical argument against FP

Industry do not always realize what it needs

“If I had asked people what they wanted, they would have said faster horses.”

- Henry Ford

- Industry usually hire for tools/languages
  - ”We want C#/Java programmers, because that is what we use”
- Instead, principles would be of more value
  - “We want developers who can build high quality systems”

Apply FP lessons

- Apply the lessons from FP even if you are forced to use other languages
  - Pure functions/methods can be applied in any language
  - Immutability can be applied in any language
  - But it requires much more discipline when the language do not help you

Another way in

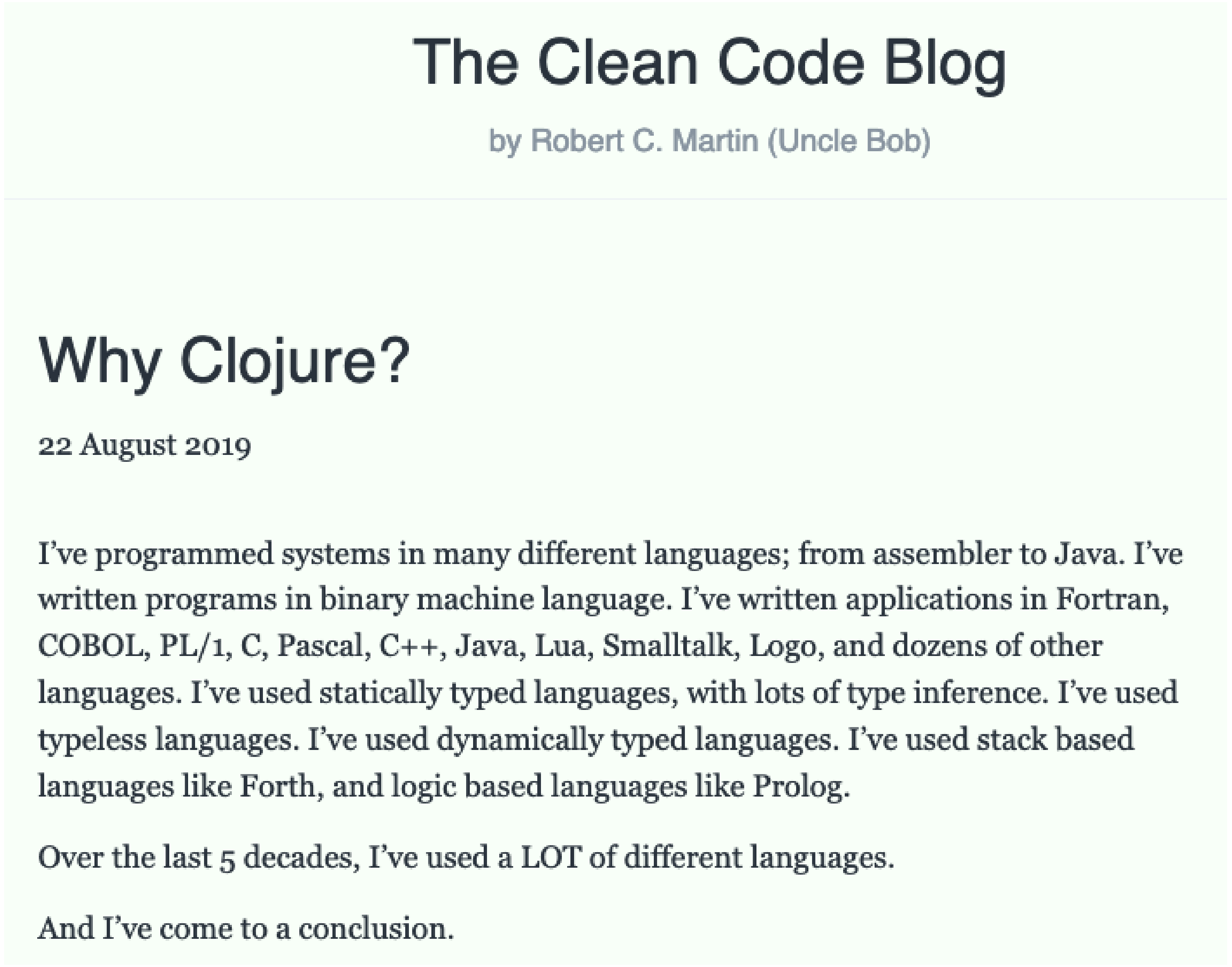
- Tooling is a way to show the benefits of FP
  - Do not effect production
- Tests
  - Property-based testing is strong in FP languages
  - Check out QuickCheck (FSCheck in F#)
  - John Hughes et al. (google scholar)
  - My paper : “QuickREST: Property-based Test Generation of OpenAPI-Described RESTful APIs”
    - Property-based testing in a functional language on real systems
    - https://arxiv.org/abs/1912.09686

Leverage the runtime already adopted

- F# runs on the CLR
- Clojure runs on the JVM/CLR/js
- Libraries can be reused
- The organization do not need to support another runtime

Learn it good and find allies

- You need to be able to answer ANY FP question
  - “What about this? What about that?”
- Try and find a coworker that is a learner
- I have been asked a 1000 times about the memory cost of immutability..



The Clean Code Blog

by Robert C. Martin (Uncle Bob)

Why Clojure?

22 August 2019

I've programmed systems in many different languages; from assembler to Java. I've written programs in binary machine language. I've written applications in Fortran, COBOL, PL/1, C, Pascal, C++, Java, Lua, Smalltalk, Logo, and dozens of other languages. I've used statically typed languages, with lots of type inference. I've used typeless languages. I've used dynamically typed languages. I've used stack based languages like Forth, and logic based languages like Prolog.

Over the last 5 decades, I've used a LOT of different languages.

And I've come to a conclusion.

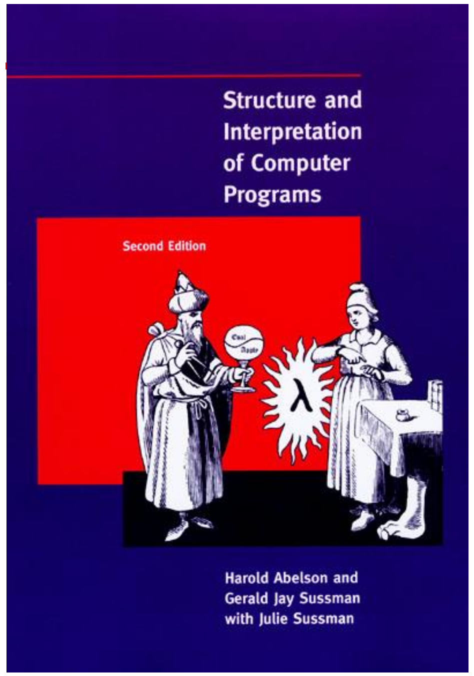
Be a champion of FP

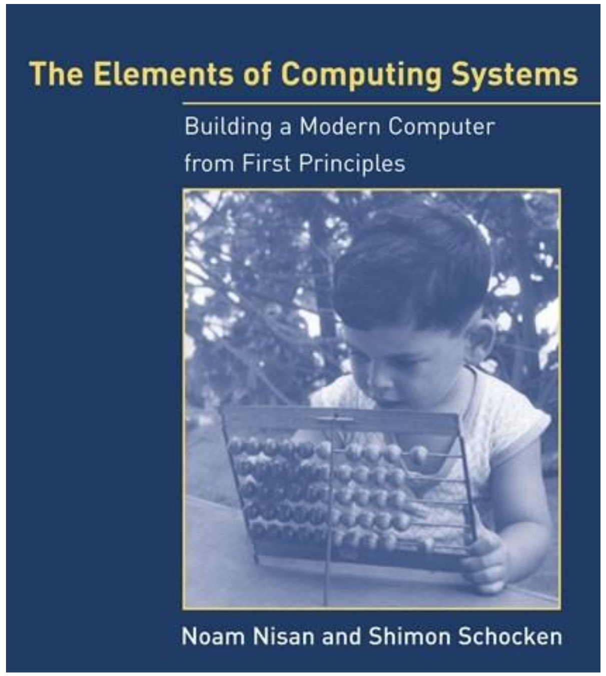
"First they ignore you, then they laugh at you, then they fight you, then you win"

- Gandhi

Clojure

- Clojure, invented by Rich Hickey in 2007
- Since you know F# it might be interesting to compare with Clojure
  - Dynamically typed, immutable, data driven LISP
- I choose Clojure based on rationality and experience not "Pop-culture"
- https://clojure.org/
- https://blog.cleancoder.com/uncle-bob/2019/08/22/WhyClojure.html

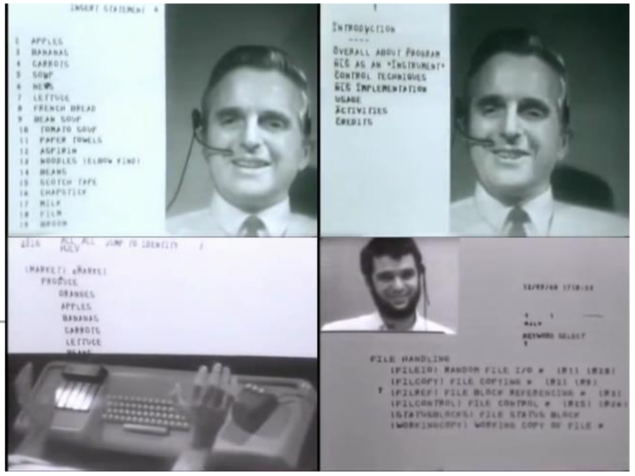

Structure and Interpretation of Computer Programs
Second Edition
Harold Abelson and Gerald Jay Sussman with Julie Sussman


The Elements of Computing Systems
Building a Modern Computer from First Principles
Noam Nisan and Shimon Schocken

A collection about Rich Hickey's works on the internet.

Functional Core – Imperative Edge – Never stop learning
The rest is details

@zteefo
stefan.l.karlsson@mdh.se
https://quality-developer.com/
https://github.com/zclj



ABB