Things We Never Told Anyone About Functional Programming

Michael Sperber

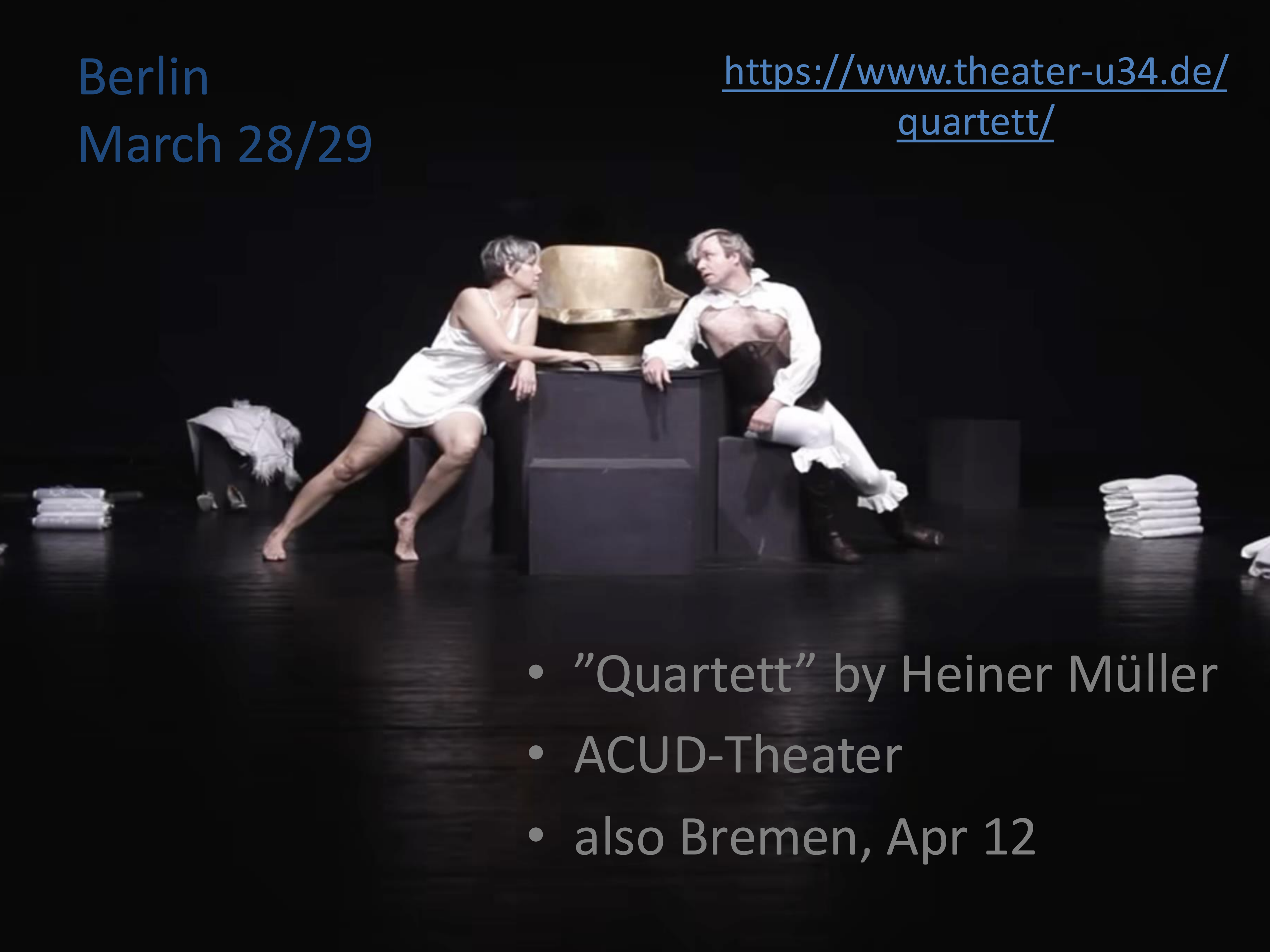
@sperbsen@discuss.systems

@ctive group

- software development with FP
- software architecture
- consulting
- software reviews
- training

www.active-group.de

funktionale-programmierung.de



Berlin
March 28/29

https://www.theater-u34.de/quartett/

- ”Quartett” by Heiner Müller
- ACUD-Theater
- also Bremen, Apr 12

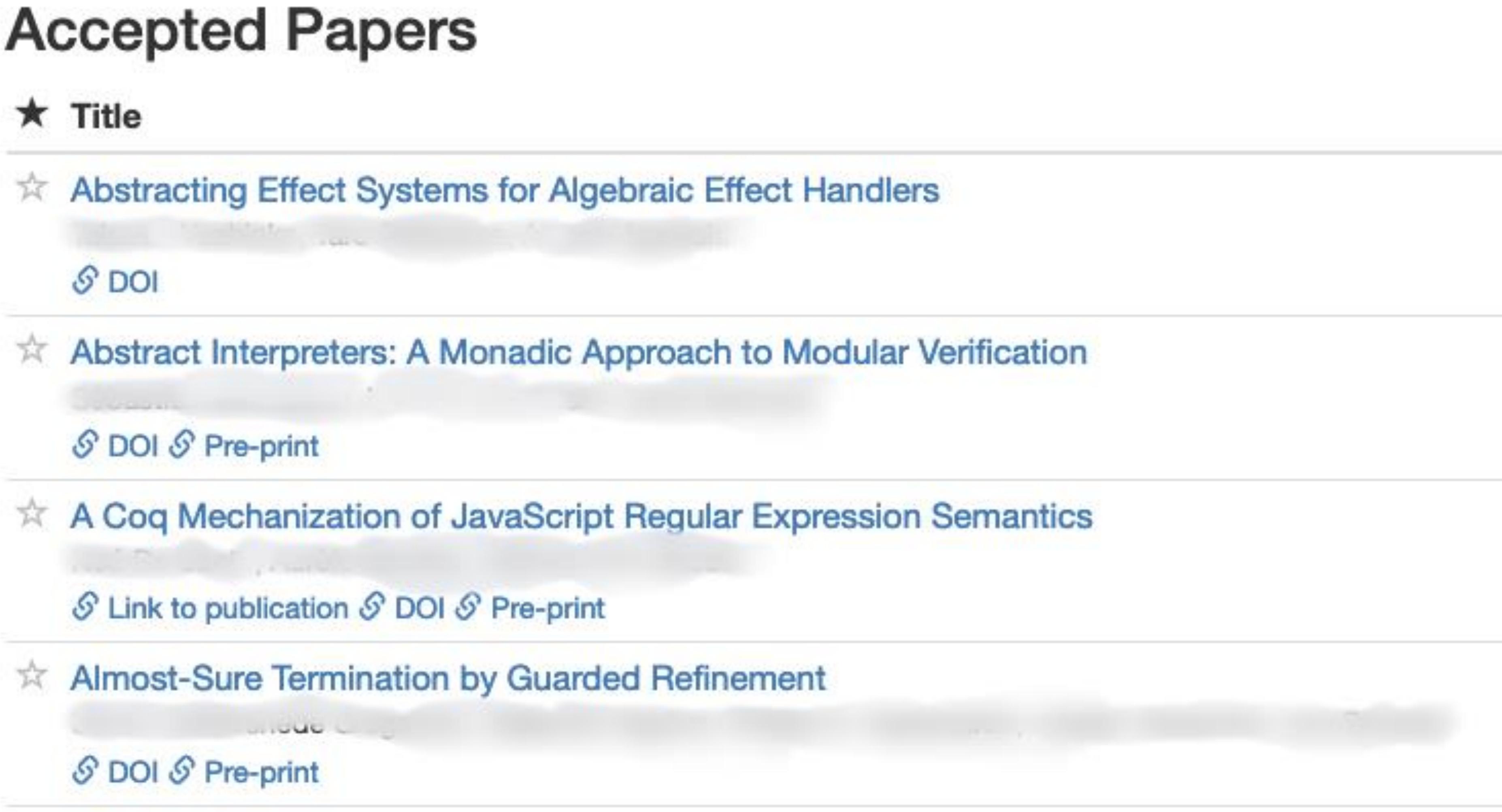
FP

What is it good for?

Functional **Programming**

Build Software Systems to Fulfill **Requirements**

ICFP Papers and Events



Accepted Papers

★ **Title**

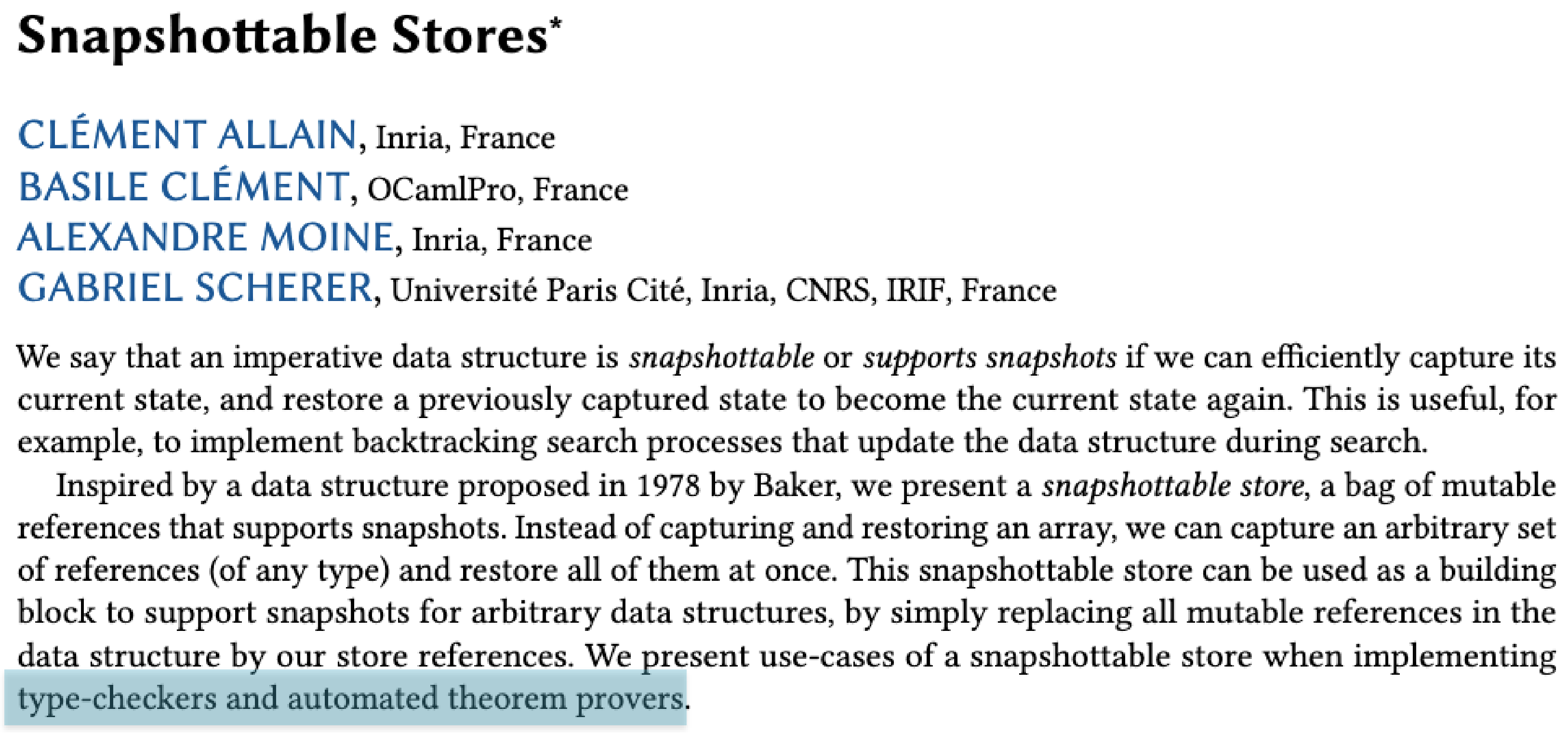
☆ Abstracting Effect Systems for Algebraic Effect Handlers
[redacted]
DOI

☆ Abstract Interpreters: A Monadic Approach to Modular Verification
[redacted]
DOI Pre-print

☆ A Coq Mechanization of JavaScript Regular Expression Semantics
[redacted]
Link to publication DOI Pre-print

☆ Almost-Sure Termination by Guarded Refinement
[redacted]
DOI Pre-print

ICFP 2024 Distinguished Paper



Snapshottable Stores*

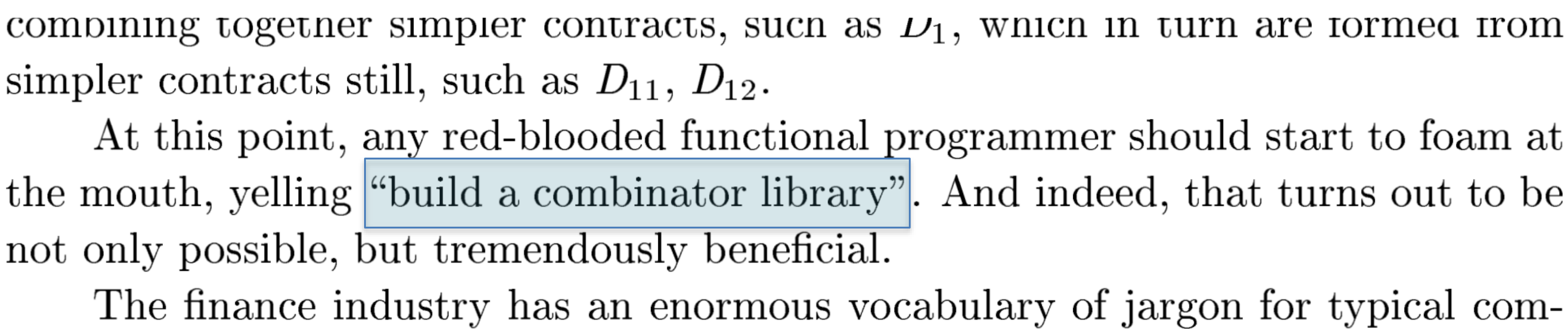
CLÉMENT ALLAIN, Inria, France
BASILE CLÉMENT, OCamlPro, France
ALEXANDRE MOINE, Inria, France
GABRIEL SCHERER, Université Paris Cité, Inria, CNRS, IRIF, France

We say that an imperative data structure is *snapshottable* or *supports snapshots* if we can efficiently capture its current state, and restore a previously captured state to become the current state again. This is useful, for example, to implement backtracking search processes that update the data structure during search.

Inspired by a data structure proposed in 1978 by Baker, we present a *snapshottable store*, a bag of mutable references that supports snapshots. Instead of capturing and restoring an array, we can capture an arbitrary set of references (of any type) and restore all of them at once. This snapshottable store can be used as a building block to support snapshots for arbitrary data structures, by simply replacing all mutable references in the data structure by our store references. We present use-cases of a snapshottable store when implementing type-checkers and automated theorem provers.

"use-cases ... type-checkers and automated theorem provers"

S. L. Peyton Jones and J.-M. Eber
How to write a financial contract. 2001



combining together simpler contracts, such as D_1, which in turn are formed from simpler contracts still, such as D_{11}, D_{12}.

At this point, any red-blooded functional programmer should start to foam at the mouth, yelling "build a combinator library". And indeed, that turns out to be not only possible, but tremendously beneficial.

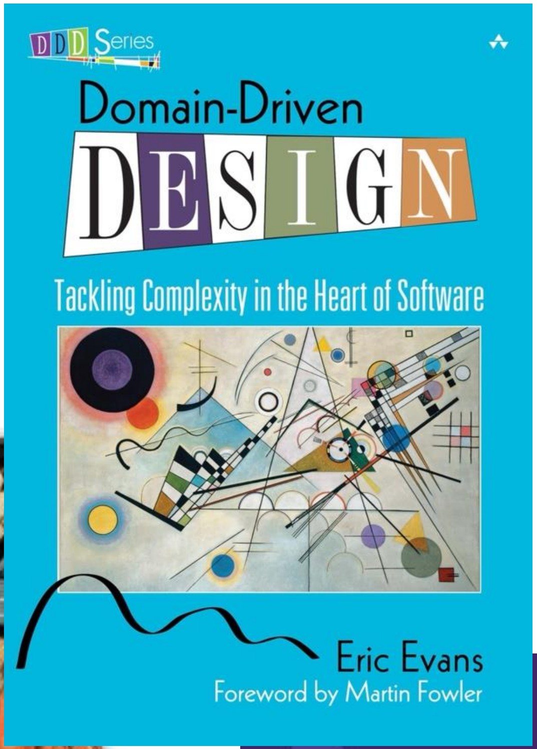
The finance industry has an enormous vocabulary of jargon for typical com-

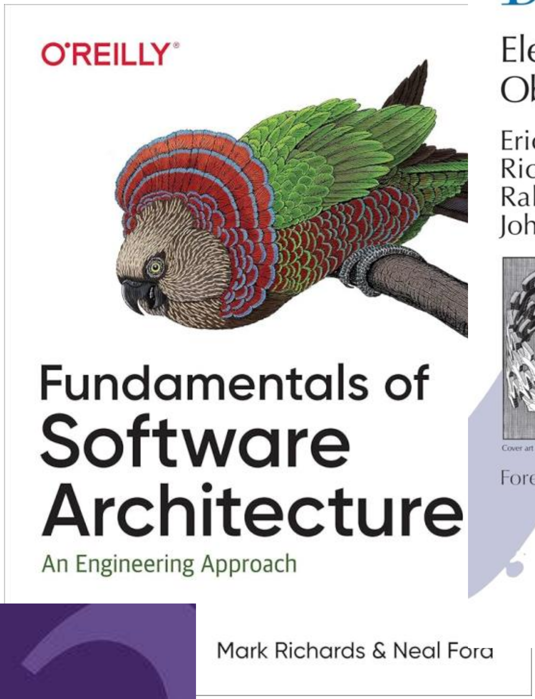
Build a combinator library!

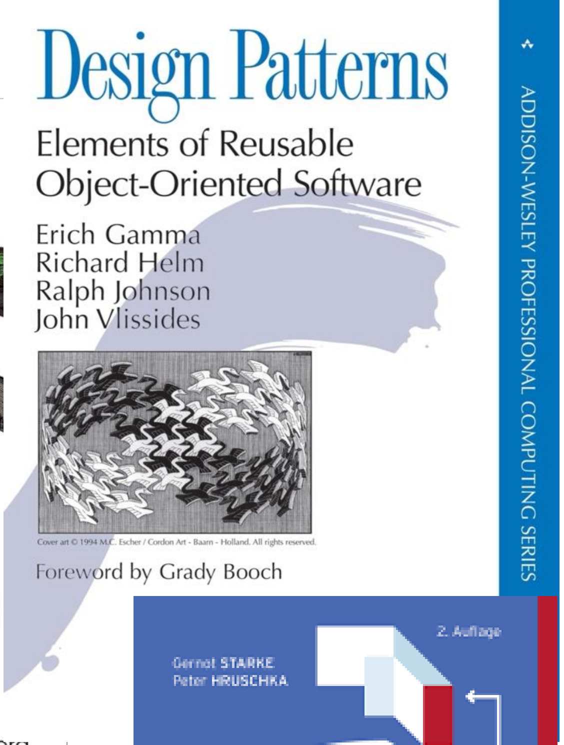
https://www.flickr.com/photos/tambako/4237063719

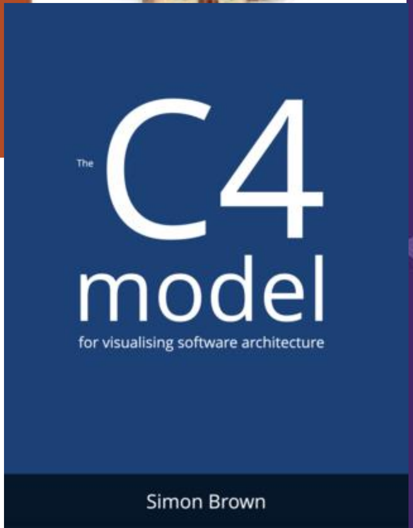
¬FP

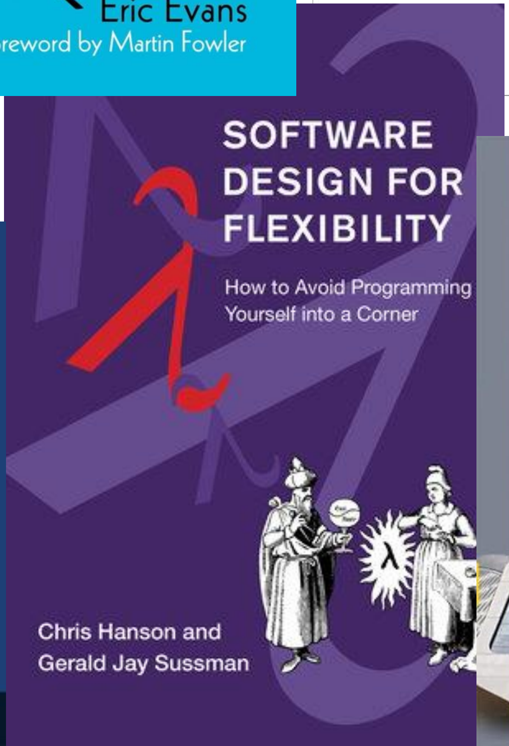
Collaborative Software Design
How to facilitate domain modeling decisions
Evelyn van Kelle
Gien Verschatse
Kenny Baas-Schwegler
MANNING

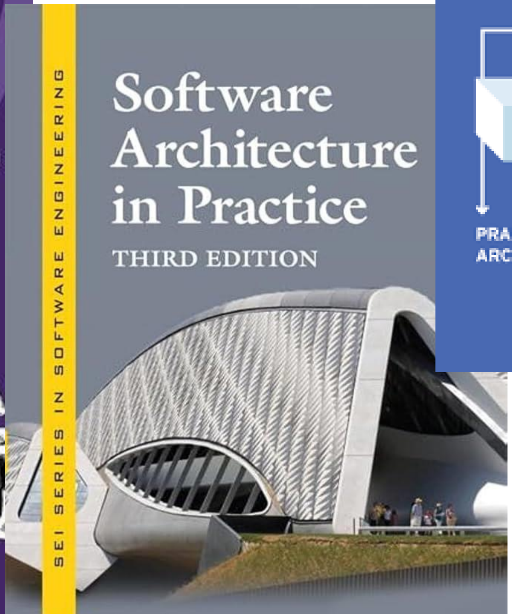

DDD Series
Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


O'REILLY
Fundamentals of Software Architecture
An Engineering Approach
Mark Richards & Neal Ford


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


The C4 model
for visualising software architecture
Simon Brown

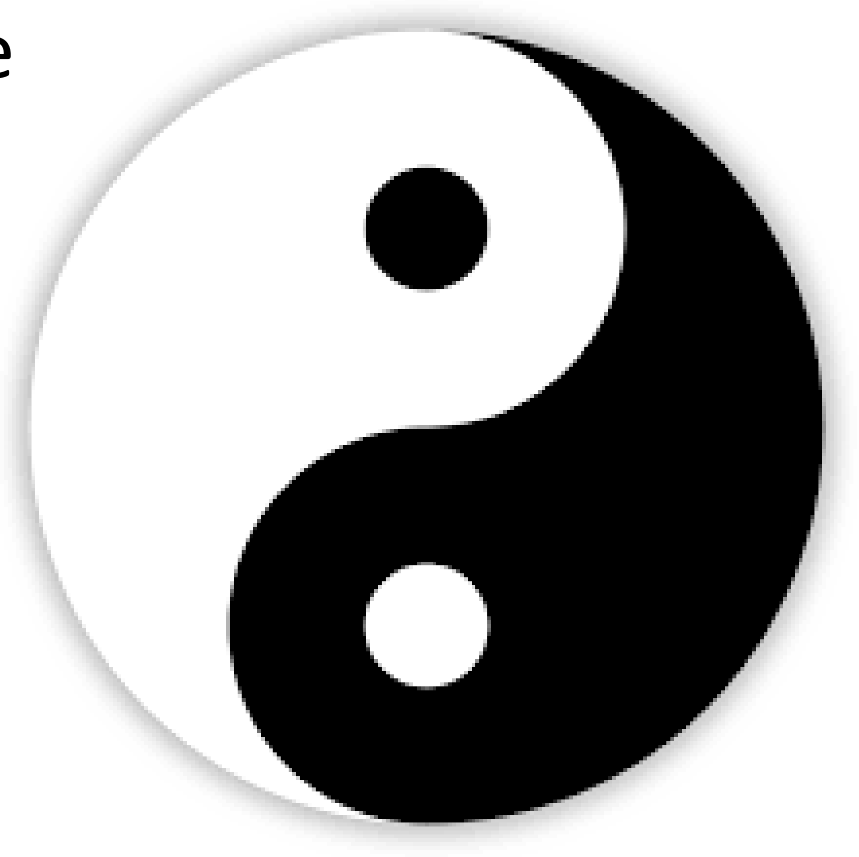

SOFTWARE DESIGN FOR FLEXIBILITY
How to Avoid Programming Yourself into a Corner
Chris Hanson and Gerald Jay Sussman


SEI SERIES IN SOFTWARE ENGINEERING
Software Architecture in Practice
THIRD EDITION

2. Auflage
Gernot STARKE
Peter HRUSCHKA
arc42 in Aktion
PRAKTISCHE TIPPS ZUR ARCHITEKTURDOKUMENTATION
HANSER
INNOQ

Software Systems (for People)

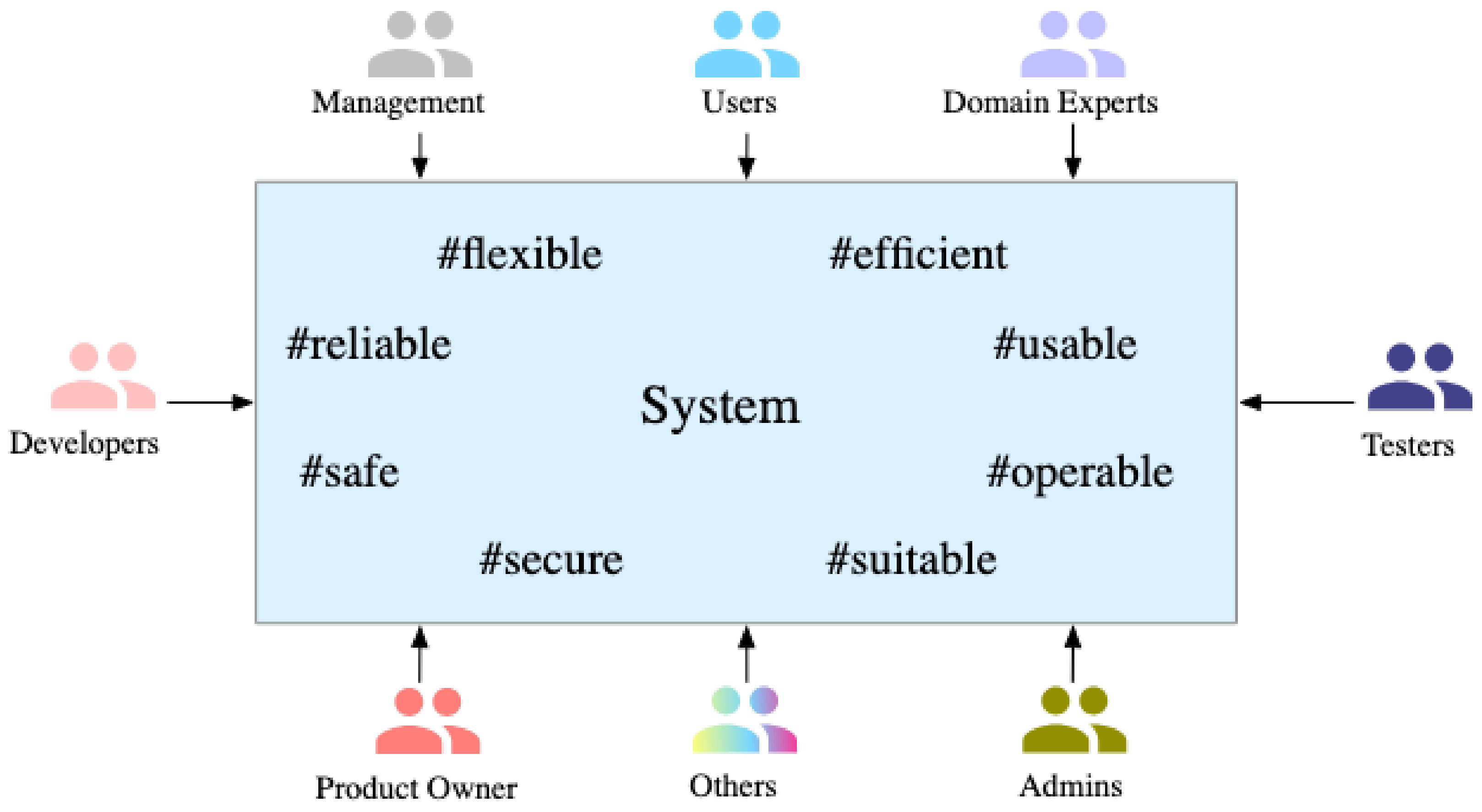
- fulfill (human) requirements
- make lives better
- programmed by people



What It Takes

- connecting requirements and software
- programming in the small
- programming in the *large* and in the *long*
- human activities around this

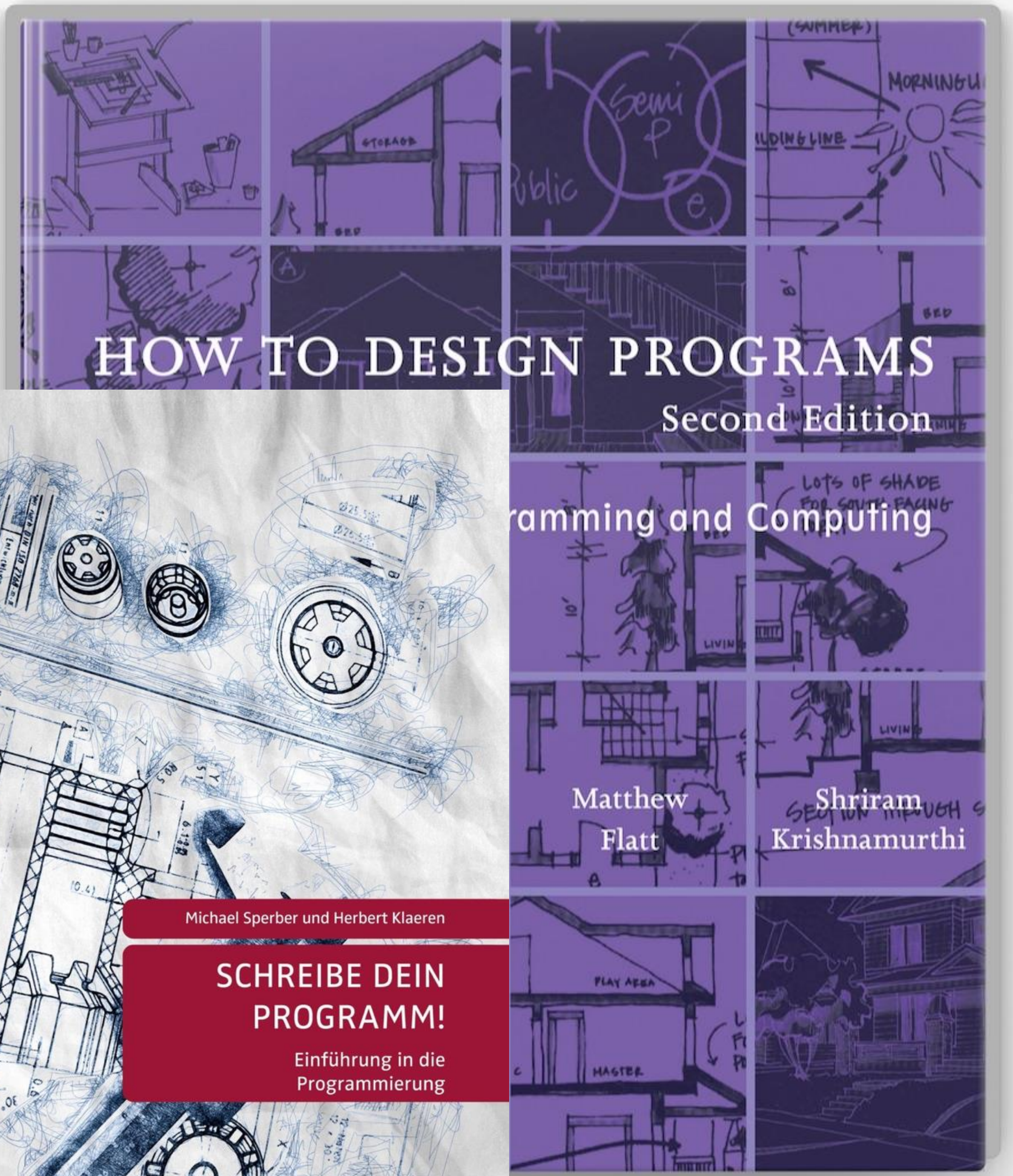
Software System Qualities



https://quality.arc42.org/articles/arc42-quality-model

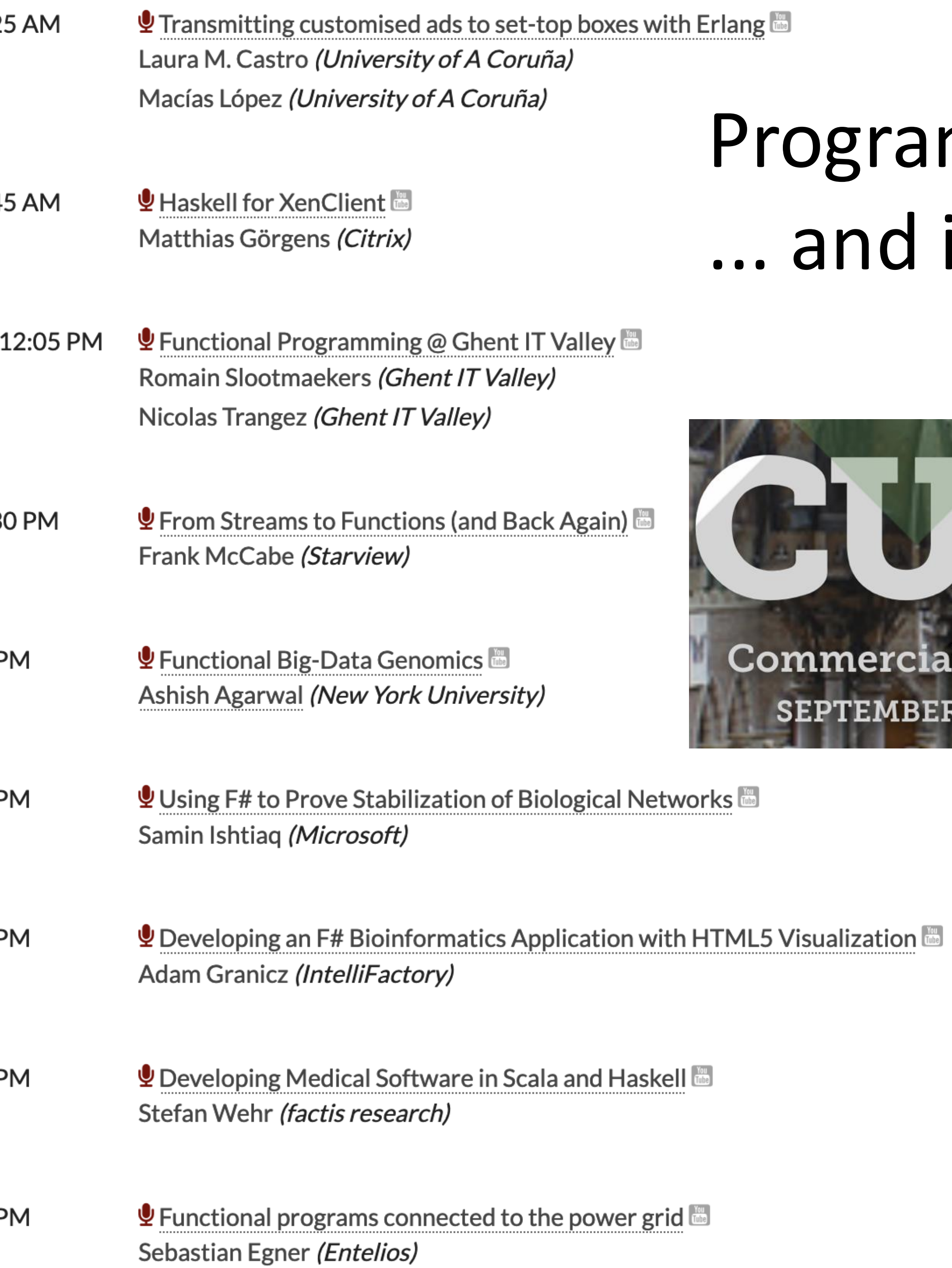
(Functional) Requirements and Programming in the Small

- data
- functions


HOW TO DESIGN PROGRAMS
Second Edition
ramming and Computing
Matthew Flatt
Shriram Krishnamurthi

Michael Sperber und Herbert Klaeren
SCHREIBE DEIN PROGRAMM!
Einführung in die Programmierung
TÜBINGEN UNIVERSITY PRESS

Programming in the *large* … and in the *long*?



25 AM Transmitting customised ads to set-top boxes with Erlang
Laura M. Castro *(University of A Coruña)*
Macías López *(University of A Coruña)*

45 AM Haskell for XenClient
Matthias Görgens *(Citrix)*

12:05 PM Functional Programming @ Ghent IT Valley
Romain Slootmaekers *(Ghent IT Valley)*
Nicolas Trangez *(Ghent IT Valley)*

30 PM From Streams to Functions (and Back Again)
Frank McCabe *(Starview)*

PM Functional Big-Data Genomics
Ashish Agarwal *(New York University)*

PM Using F# to Prove Stabilization of Biological Networks
Samin Ishtiaq *(Microsoft)*

PM Developing an F# Bioinformatics Application with HTML5 Visualization
Adam Granicz *(IntelliFactory)*

PM Developing Medical Software in Scala and Haskell
Stefan Wehr *(factis research)*

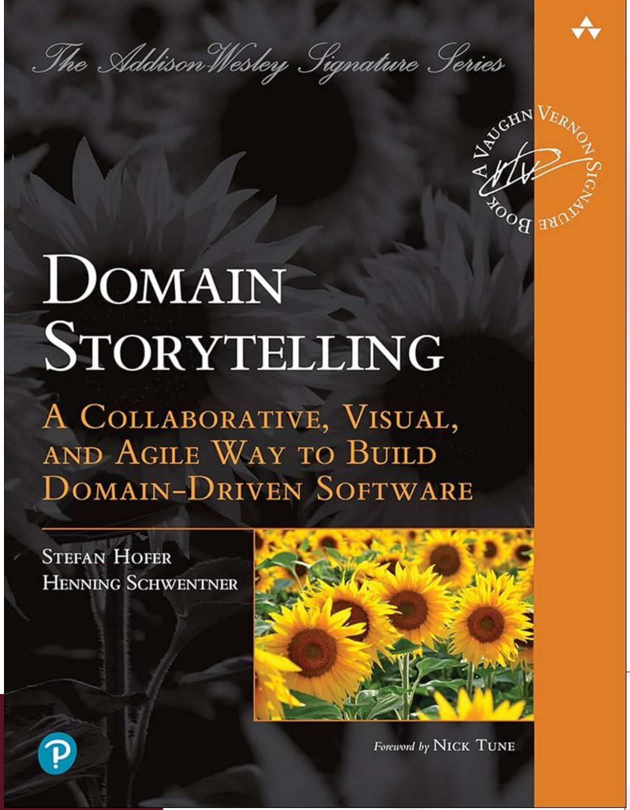
PM Functional programs connected to the power grid
Sebastian Egner *(Entelios)*

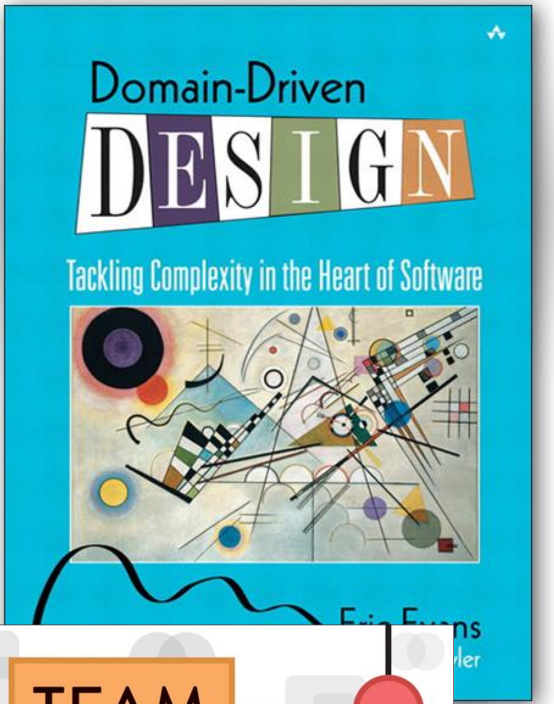
CUFP 2017
Commercial Users of Functional Programming
SEPTEMBER 7TH-9TH — OXFORD, UNITED KINGDOM

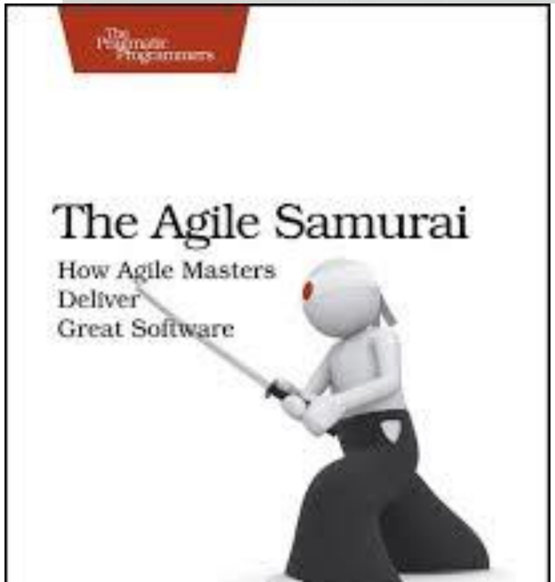
Human Activities

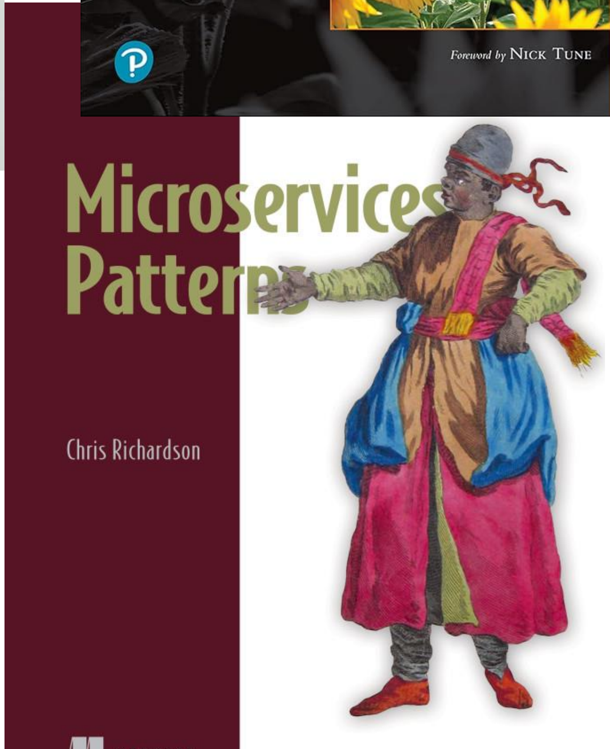
Elsewhere ...

EVENT STORMING
Alberto Brandolini
INVITE THE RIGHT PEOPLE
FACILITATOR
PROVIDE AN UNLIMITED MODELLING SURFACE
AN ACT OF DELIBERATE COLLECTIVE LEARNING
SILO BOUNDARY CROSSED
ALTERNATIVE MODEL EMERGED
DEEPER INSIGHT ACHIEVED
VALIDATE IT!
WRITE A PROTOTYPE
ENTERPRISE MODEL SKETCHED
PLACE DOMAIN EVENTS ALONG A TIMELINE
DEVELOPER
DOMAIN EXPERT
WARNING! IT MAY BECOME ADDICTIVE
VISIBLE REPRESENTATION OF OUR CURRENT UNDERSTANDING
ASK MORE QUESTIONS


The Addison-Wesley Signature Series
A Vaughn Vernon Signature Book
Domain Storytelling
A Collaborative, Visual, and Agile Way to Build Domain-Driven Software
Stefan Hofer
Henning Schwentner
Foreword by Nick Tune




The Pragmatic Programmers
The Agile Samurai
How Agile Masters Deliver Great Software



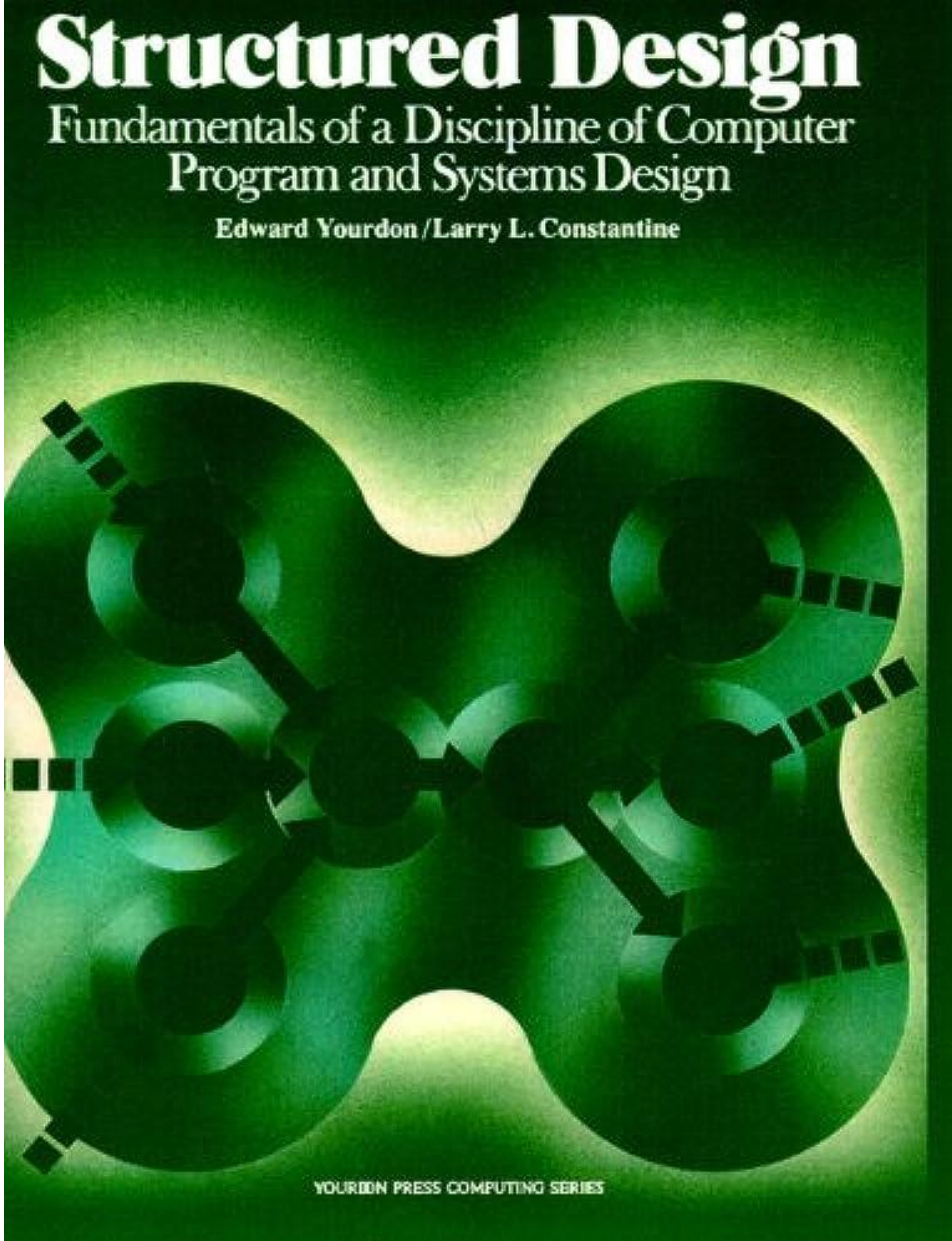
{OOP}

What It Takes

- connecting requirements and software ✓ ✗
- programming in the small ✓ ✓
- programming in the *large* and in the *long* ✓ ✗
- human activities around this ✗ ✗

Programming in the Large and in the Long

$$€ = \Delta$$


Structured Design

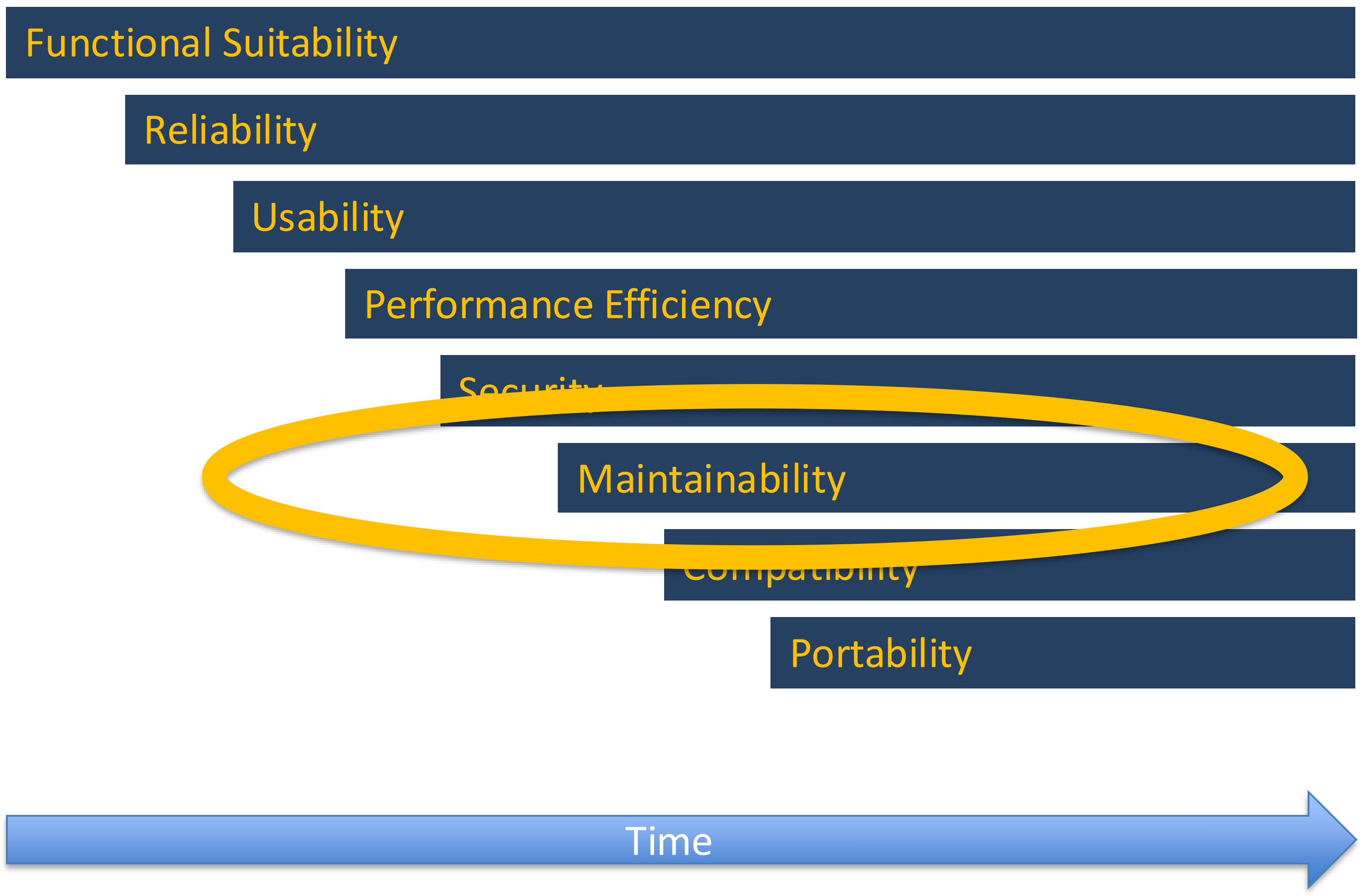
Fundamentals of a Discipline of Computer Program and Systems Design

Edward Yourdon / Larry L. Constantine

YOURDON PRESS COMPUTING SERIES

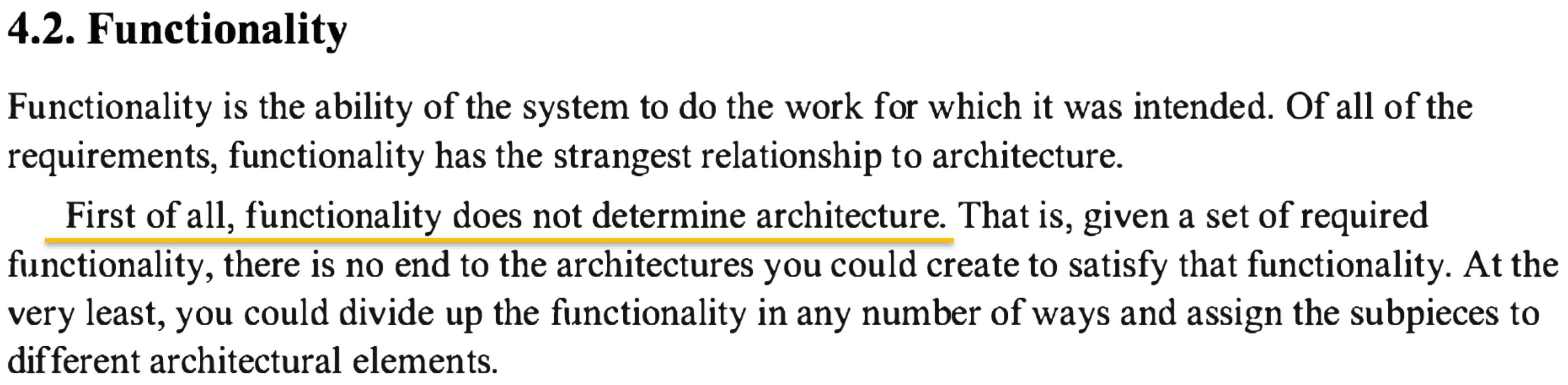
1979

Evolutionary Software Quality



Source: Markus Harrer, OOP 2023

Functionality and Architecture



4.2. Functionality

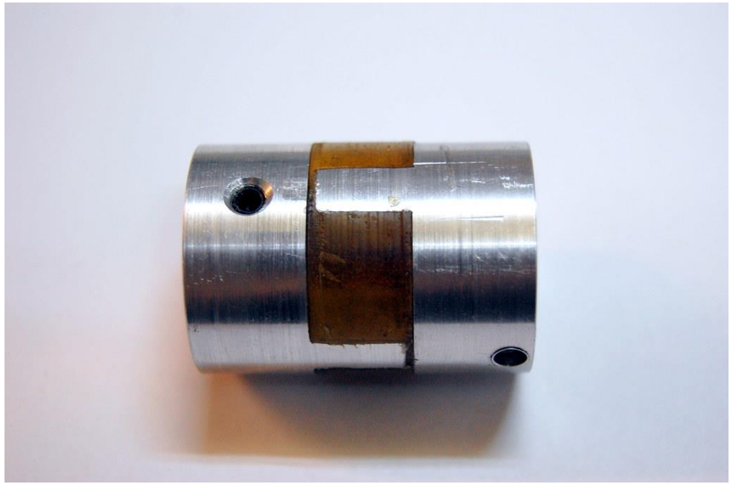
Functionality is the ability of the system to do the work for which it was intended. Of all of the requirements, functionality has the strangest relationship to architecture.

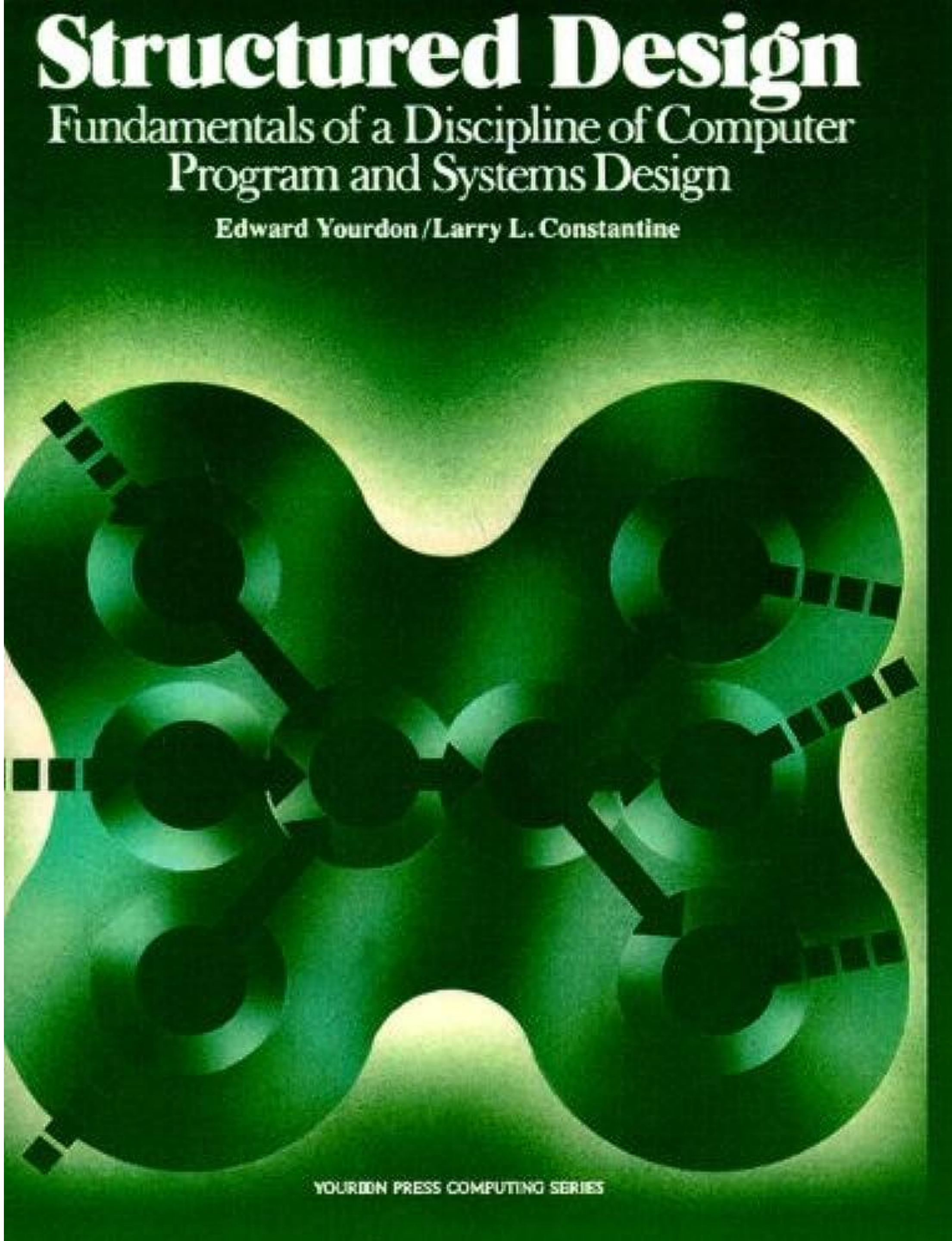
First of all, functionality does not determine architecture. That is, given a set of required functionality, there is no end to the architectures you could create to satisfy that functionality. At the very least, you could divide up the functionality in any number of ways and assign the subpieces to different architectural elements.

Bass, Kazman, Clements: *Software Architecture in Practice*

Programming in the Large and in the Long

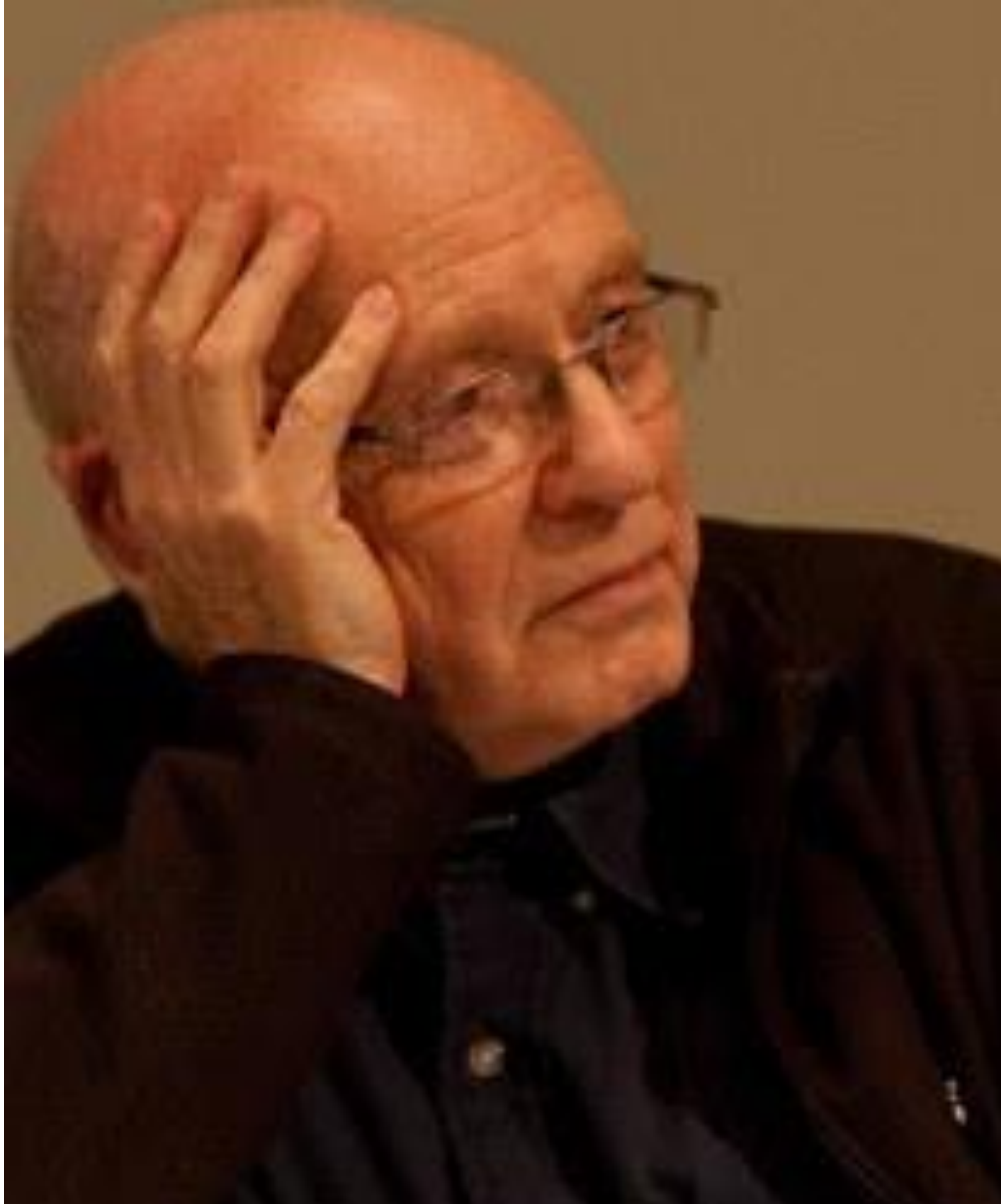
$€ = \Delta$



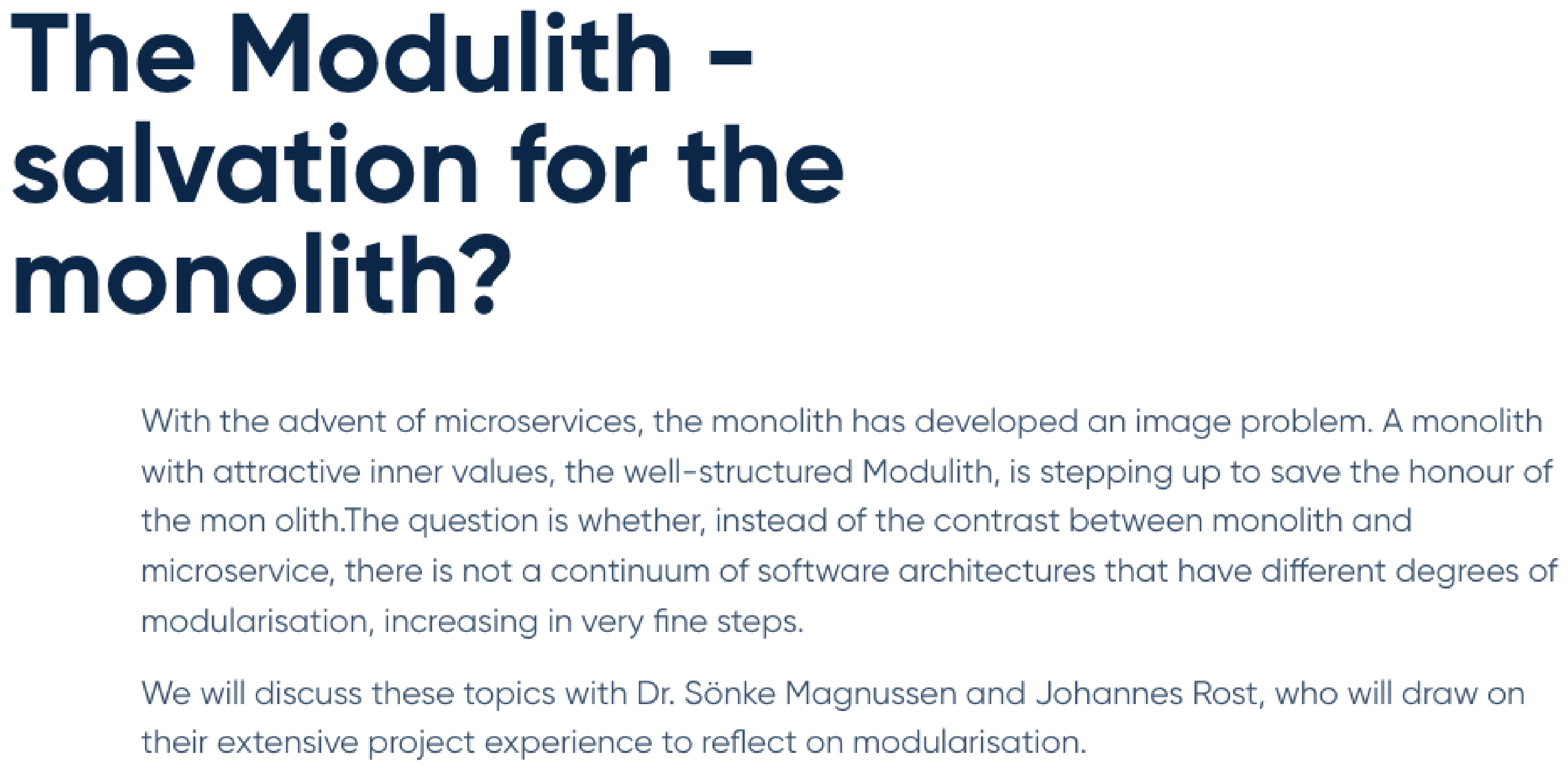

Structured Design
Fundamentals of a Discipline of Computer Program and Systems Design
Edward Yourdon / Larry L. Constantine
YOURDON PRESS COMPUTING SERIES

David Parnas (1972)

"We propose [...] that one begins with a list of difficult **design decisions** or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.."



The Monolith is Unmodular



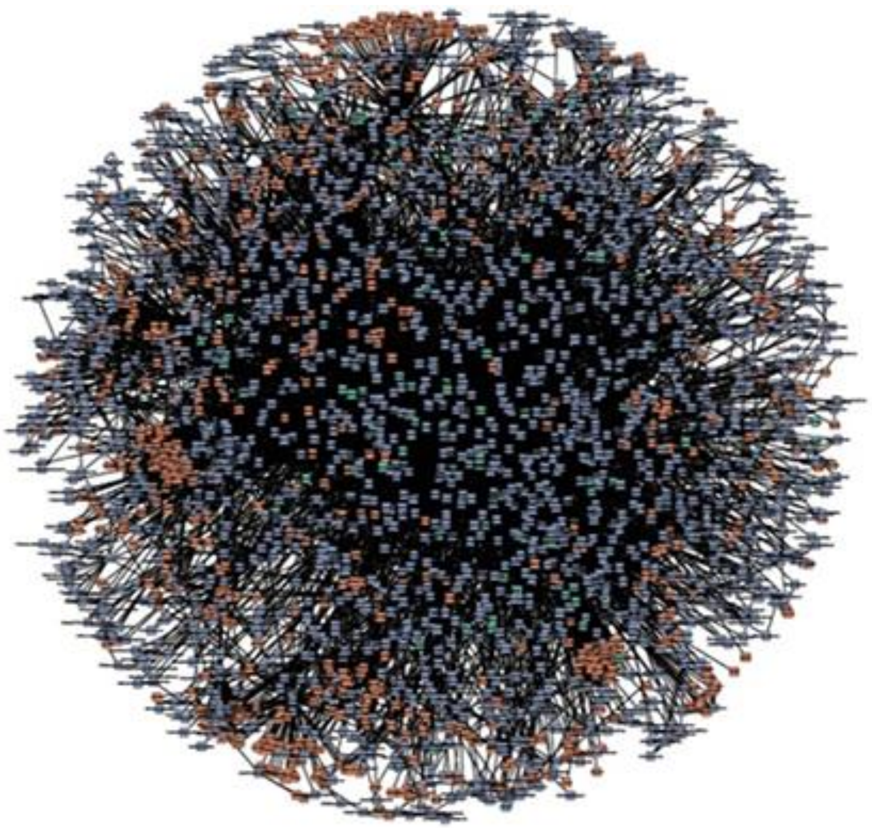
The Modulith – salvation for the monolith?

With the advent of microservices, the monolith has developed an image problem. A monolith with attractive inner values, the well-structured Modulith, is stepping up to save the honour of the mon olith.The question is whether, instead of the contrast between monolith and microservice, there is not a continuum of software architectures that have different degrees of modularisation, increasing in very fine steps.

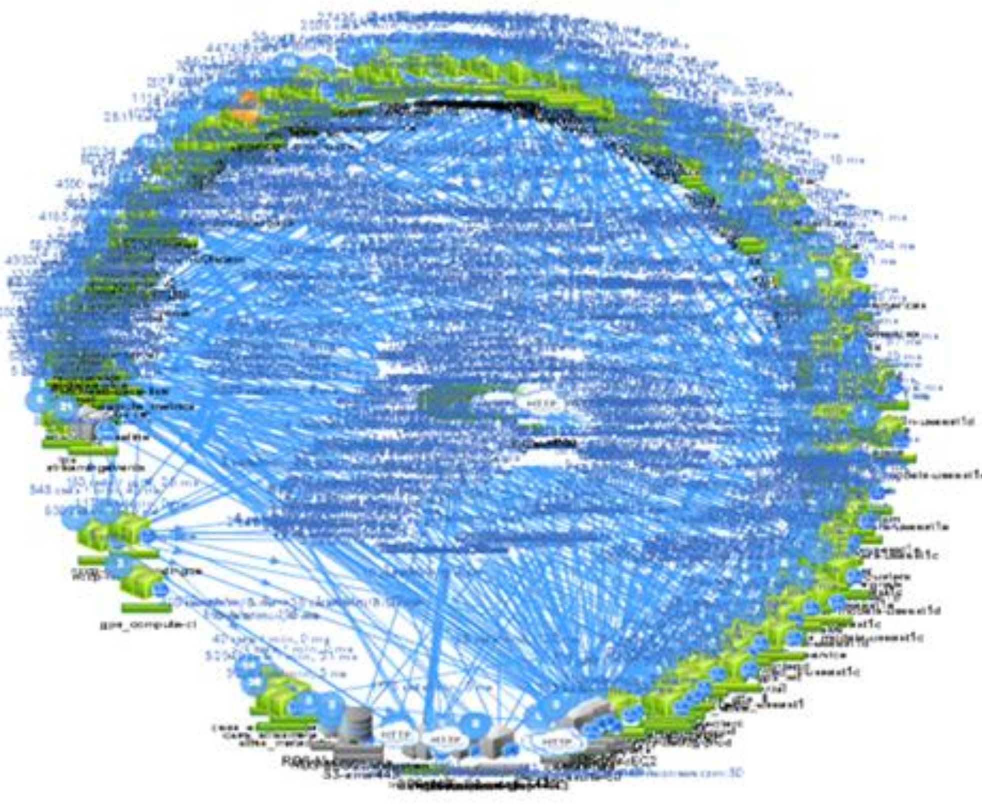
We will discuss these topics with Dr. Sönke Magnussen and Johannes Rost, who will draw on their extensive project experience to reflect on modularisation.

https://www.wps.de/en/news/the-modulith-salvation-for-the-monolith

Microservice Architectures



amazon.com



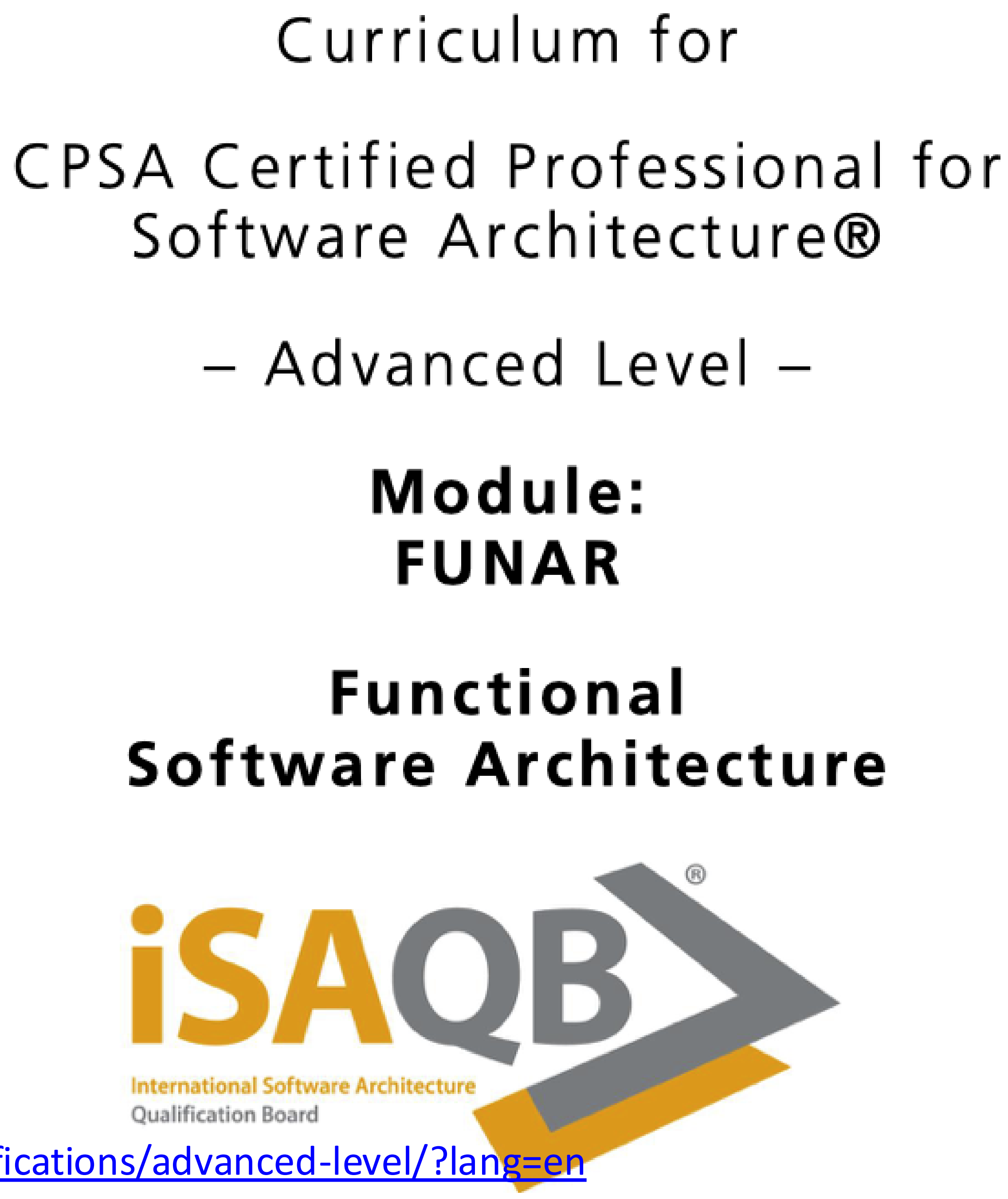
NETFLIX

Decoupled by Default

- immutability
- abstraction boundaries
- expressive interface languages
- expressive effects

λ

Putting it all together



Curriculum for

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Module: FUNAR

Functional Software Architecture

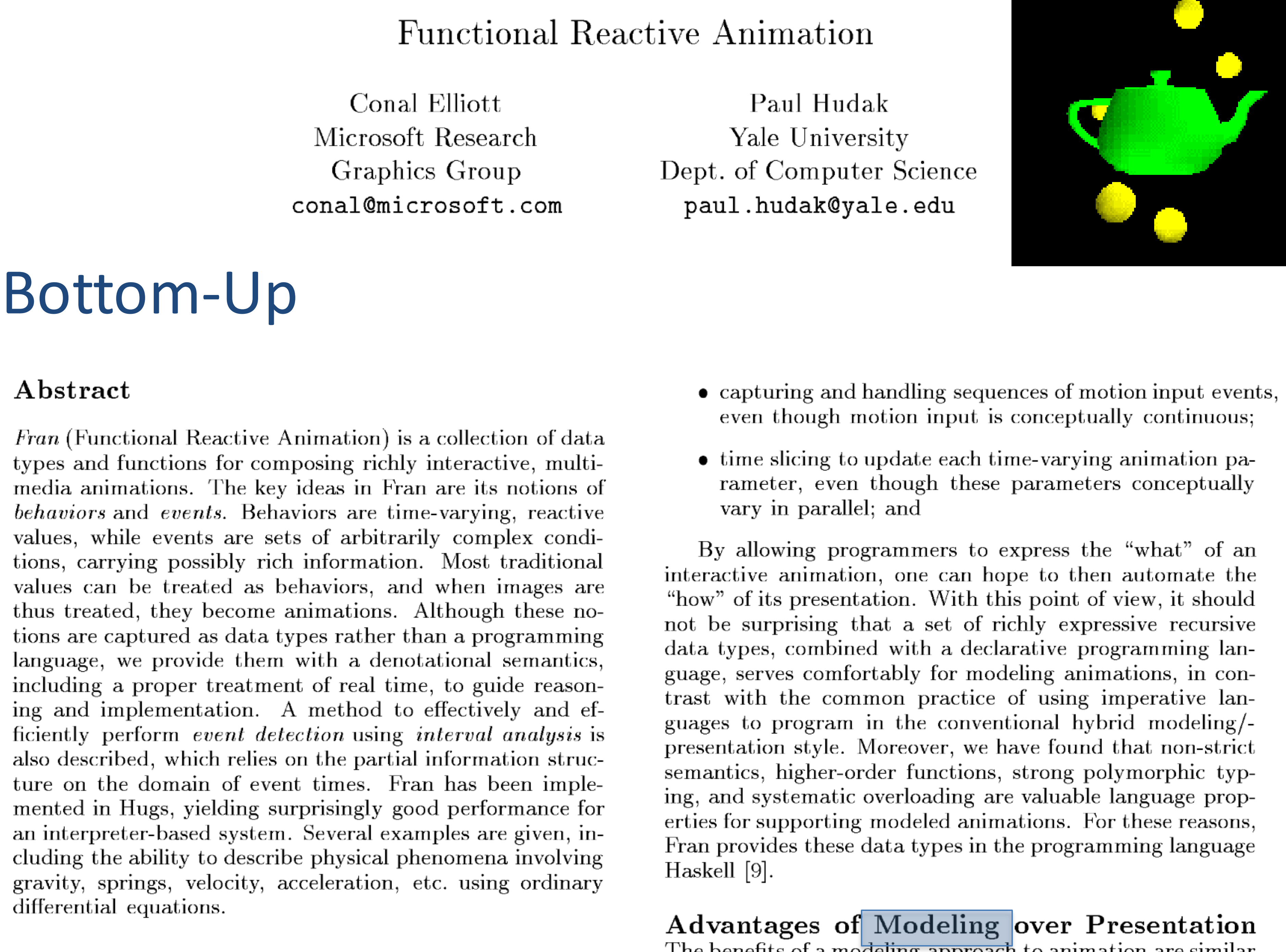
iSAQB®

International Software Architecture Qualification Board

https://www.isaqb.org/certifications/advanced-level/?lang=en

A Blueprint for Functional Software Design

- bottom-up
- start with functional qualities
- everything is data
- data modeling with sums and products
- abstraction
- combinator models
- algebraic structures
- mathematical properties



Functional Reactive Animation

Conal Elliott
Microsoft Research
Graphics Group
conal@microsoft.com

Paul Hudak
Yale University
Dept. of Computer Science
paul.hudak@yale.edu

Bottom-Up

Abstract

Fran (Functional Reactive Animation) is a collection of data types and functions for composing richly interactive, multimedia animations. The key ideas in Fran are its notions of *behaviors* and *events*. Behaviors are time-varying, reactive values, while events are sets of arbitrarily complex conditions, carrying possibly rich information. Most traditional values can be treated as behaviors, and when images are thus treated, they become animations. Although these notions are captured as data types rather than a programming language, we provide them with a denotational semantics, including a proper treatment of real time, to guide reasoning and implementation. A method to effectively and efficiently perform *event detection* using *interval analysis* is also described, which relies on the partial information structure on the domain of event times. Fran has been implemented in Hugs, yielding surprisingly good performance for an interpreter-based system. Several examples are given, including the ability to describe physical phenomena involving gravity, springs, velocity, acceleration, etc. using ordinary differential equations.

- capturing and handling sequences of motion input events, even though motion input is conceptually continuous;
- time slicing to update each time-varying animation parameter, even though these parameters conceptually vary in parallel; and

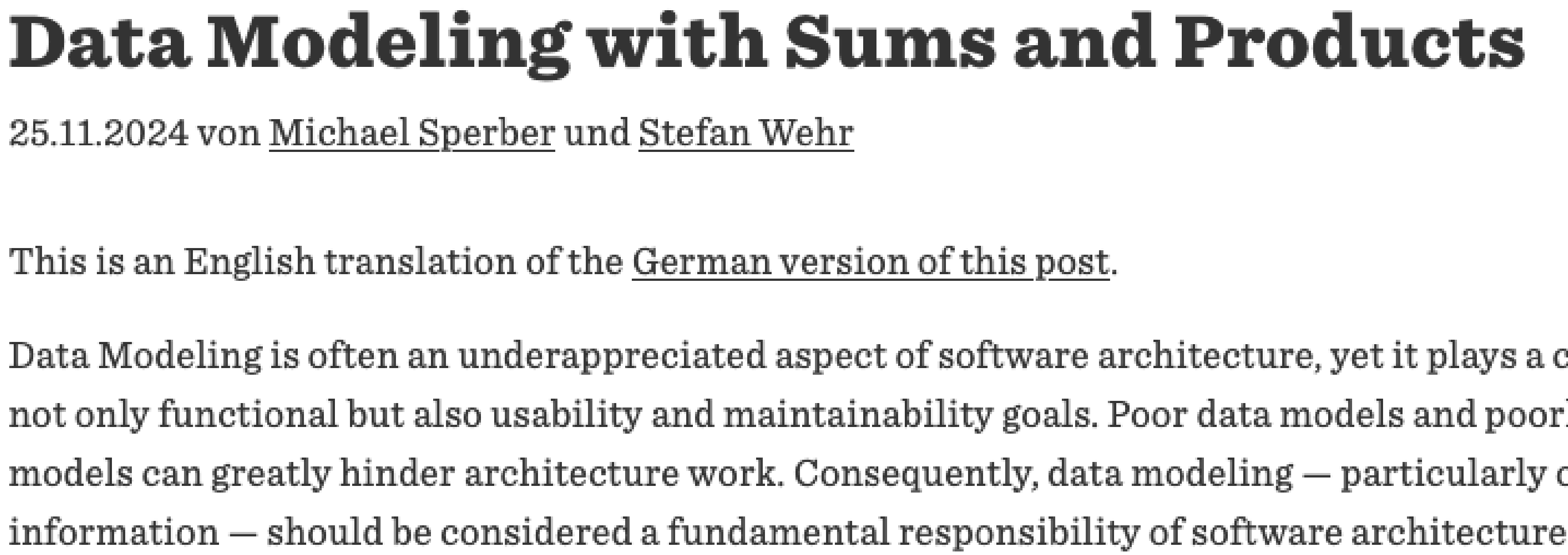
By allowing programmers to express the "what" of an interactive animation, one can hope to then automate the "how" of its presentation. With this point of view, it should not be surprising that a set of richly expressive recursive data types, combined with a declarative programming language, serves comfortably for modeling animations, in contrast with the common practice of using imperative languages to program in the conventional hybrid modeling/-presentation style. Moreover, we have found that non-strict semantics, higher-order functions, strong polymorphic typing, and systematic overloading are valuable language properties for supporting modeled animations. For these reasons, Fran provides these data types in the programming language Haskell [9].

Advantages of Modeling over Presentation

The benefits of a modeling approach to animation are similar

Data Modeling

- functions
- *data*
- products
- *sums* FTW



Data Modeling with Sums and Products

25.11.2024 von Michael Sperber und Stefan Wehr

This is an English translation of the German version of this post.

Data Modeling is often an underappreciated aspect of software architecture, yet it plays a c
not only functional but also usability and maintainability goals. Poor data models and poor
models can greatly hinder architecture work. Consequently, data modeling — particularly c
information — should be considered a fundamental responsibility of software architecture

https://funktionale-programmierung.de/2024/11/25/sums-products-english.html

What We Model When We Model - OO

```
interface Animal {
  void runOver();
  void feed(Amount amount);
}
```

Quelle: Wikipedia
CC Attribution 3.0 Unported

What We Model When We Model - FP

```haskell
runOverAnimal :: Animal -> Animal

feedAnimal :: Weight -> Animal -> Animal
```

Quelle: Wikipedia
CC Attribution 3.0 Unported

A World of Objects

(© Trustees of the British Museum)

Reality and Snapshots

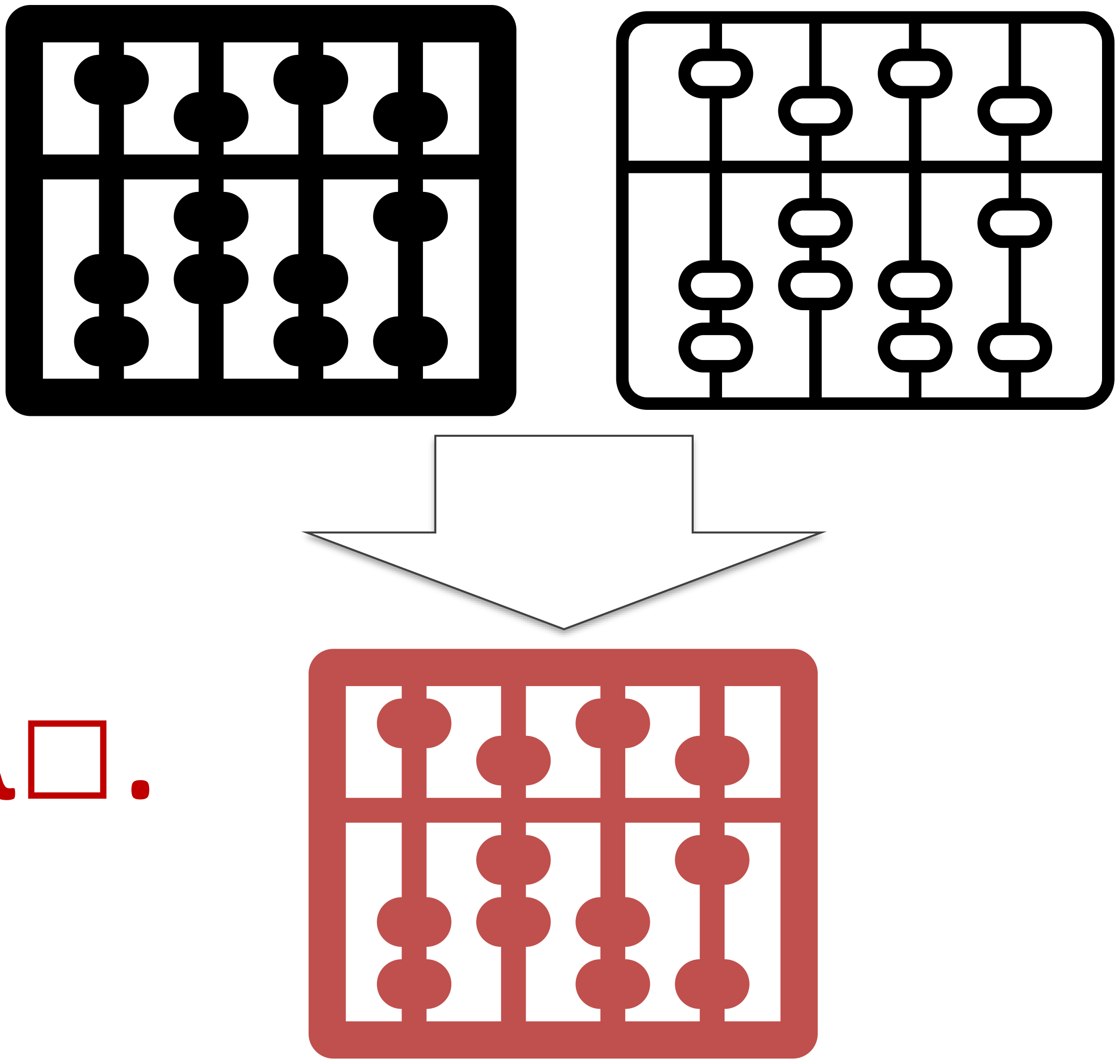
Architecture Evolution

$$\left.\begin{array}{l}\bullet\ \text{model domain with data}\\ \bullet\ \text{implement functions}\end{array}\right\}\text{functionality}$$

- change to achieve
  quality-specific requirements

low coupling => $\Delta \sim$ €

Abstraction



λ□.

Combinator Libraries

Combinator Libraries

- meet functional requirements
- facilitate communication
- decouple representation from behaior
- slow down change
- fulfill future requirements without or with small changes

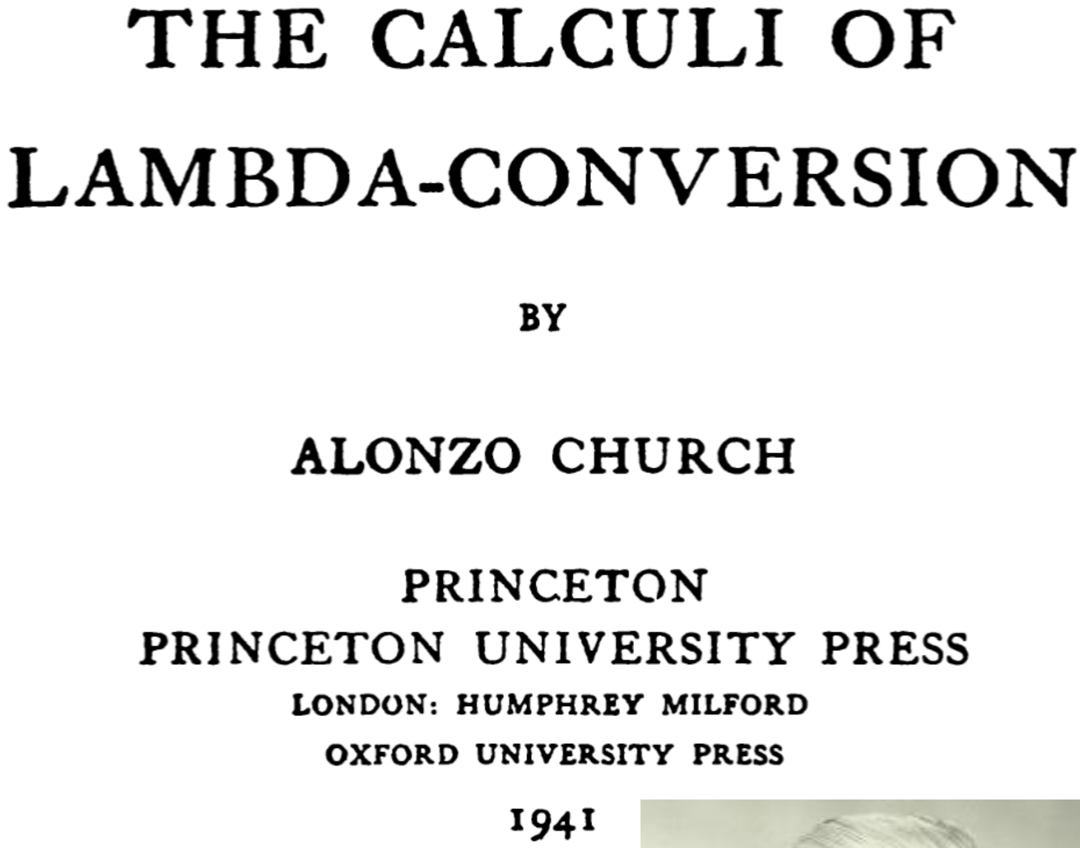
Algebraic Structures

$$\begin{tikzcd}
(M \otimes M) \otimes M \arrow[r, "\alpha"] \arrow[d, "\mu\otimes 1"'] & M \otimes (M \otimes M) \arrow[r, "1\otimes\mu"] & M \otimes M \arrow[d, "\mu"] \\
M \otimes M \arrow[rr, "\mu"'] & & M
\end{tikzcd}$$

$$\begin{tikzcd}
T(X) \arrow[r, "\eta_{T(X)}"] \arrow[d, "T(\eta_X)"'] \arrow[dr, equal] & T(T(X)) \arrow[d, "\mu_X"] \\
T(T(X)) \arrow[r, "\mu_X"'] & T(X)
\end{tikzcd}$$

Abstract Nonsense

Why *This*?



THE CALCULI OF
LAMBDA-CONVERSION

BY

ALONZO CHURCH

PRINCETON
PRINCETON UNIVERSITY PRESS
LONDON: HUMPHREY MILFORD
OXFORD UNIVERSITY PRESS

1941

λ

… profit?

Mathematics = Brain Patterns for Thinking

FP/Mathematics and Humans

- Model Human Understanding
  (rather than *the thing*)
- Immutability FTW
- Maths = Patterns for Human Understanding

FP Todos

- top-down FP design
  (and meet-in-the-middle)
- architecture documentation
  patterns & visualization
- stakeholder communication
- books
- cross-pollination with the non-FP types