I can haz no Futures?

Michał Płachta
@miciek

ocado TECHNOLOGY

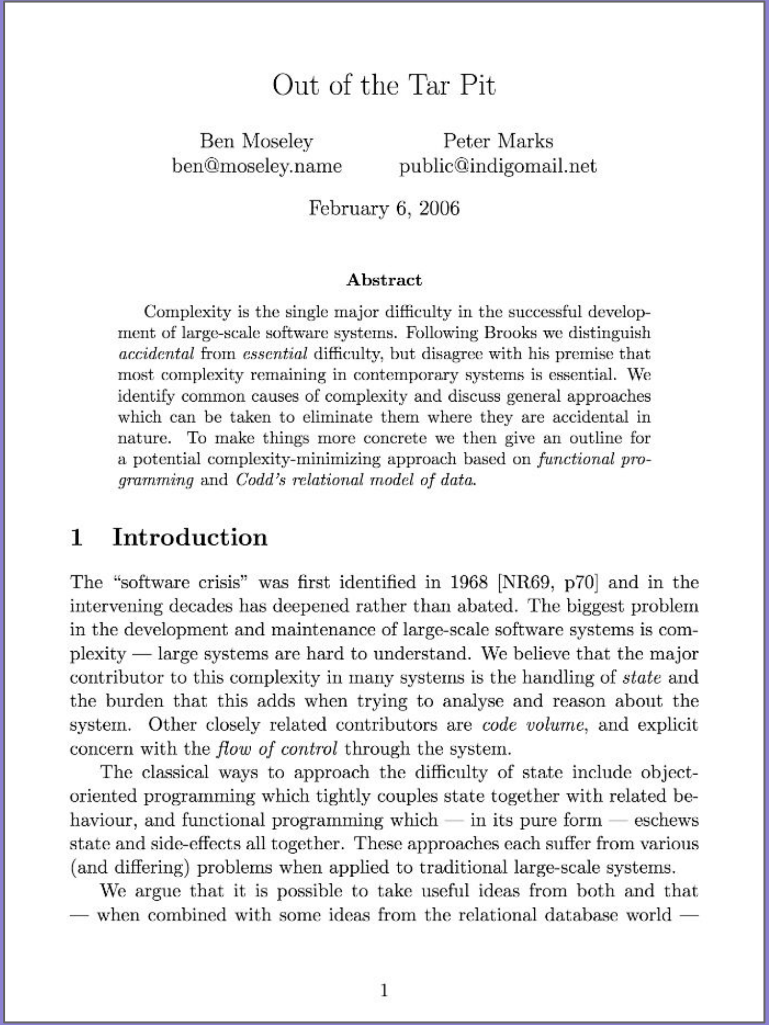
The Journey Ahead of Us

WHY?

REFACTORING
ADD FUNCTION

REFACTORING
ACTOR SYSTEM

Complexity is the root cause of the vast majority of problems with software today.



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.*

1 Introduction

The "software crisis" was first identified in 1968 [NR69, p70] and in the intervening decades has deepened rather than abated. The biggest problem in the development and maintenance of large-scale software systems is complexity — large systems are hard to understand. We believe that the major contributor to this complexity in many systems is the handling of *state* and the burden that this adds when trying to analyse and reason about the system. Other closely related contributors are *code volume*, and explicit concern with the *flow of control* through the system.

The classical ways to approach the difficulty of state include object-oriented programming which tightly couples state together with related behaviour, and functional programming which — in its pure form — eschews state and side-effects all together. These approaches each suffer from various (and differing) problems when applied to traditional large-scale systems.

We argue that it is possible to take useful ideas from both and that — when combined with some ideas from the relational database world —

1

Complexity

Essential

- The "heart" of your system
- The WHAT

Accidental

- The least important part of your system
- The HOW and WHEN

QUIZ

Here’s a return type of a function in a CRM system...

```
Future[Option[Customer]]
```

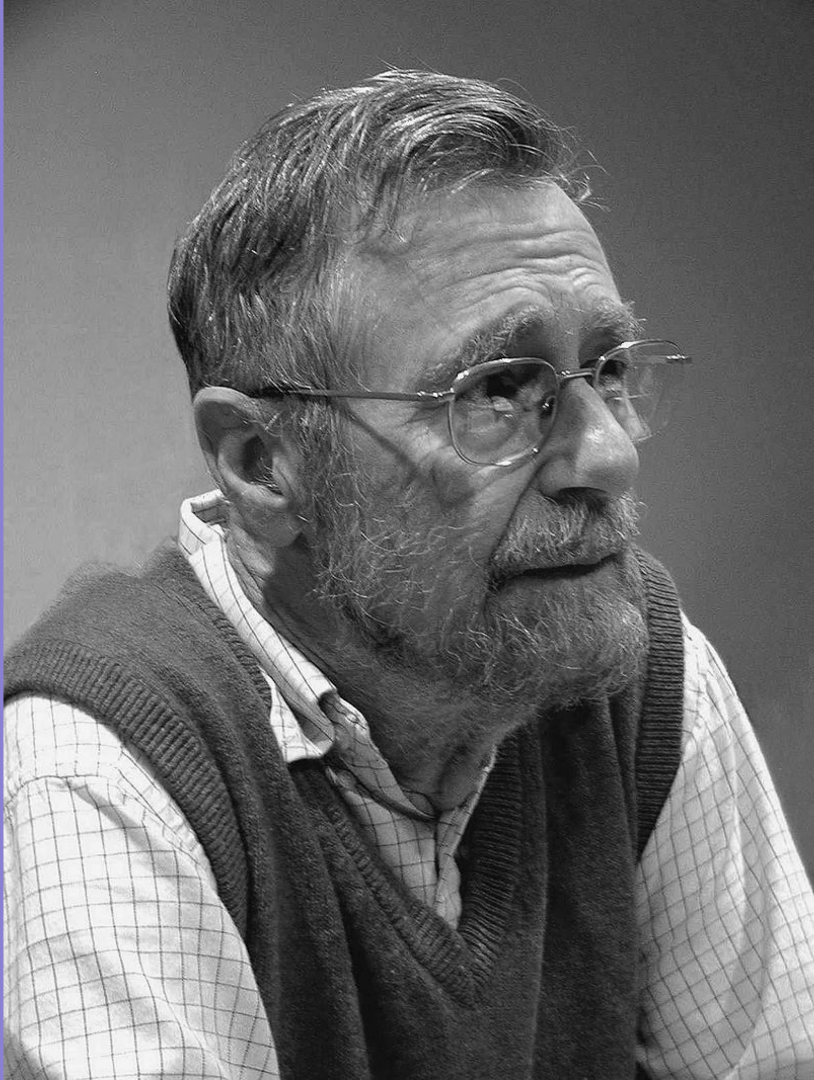
Accidental Complexity

Essential Complexity

“

We have to keep it **crisp**, **disentangled**, and **simple** if we refused to be crushed by the **complexities** of our own making...

Portrait by Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4204157



Managing Complexity

Live coding examples

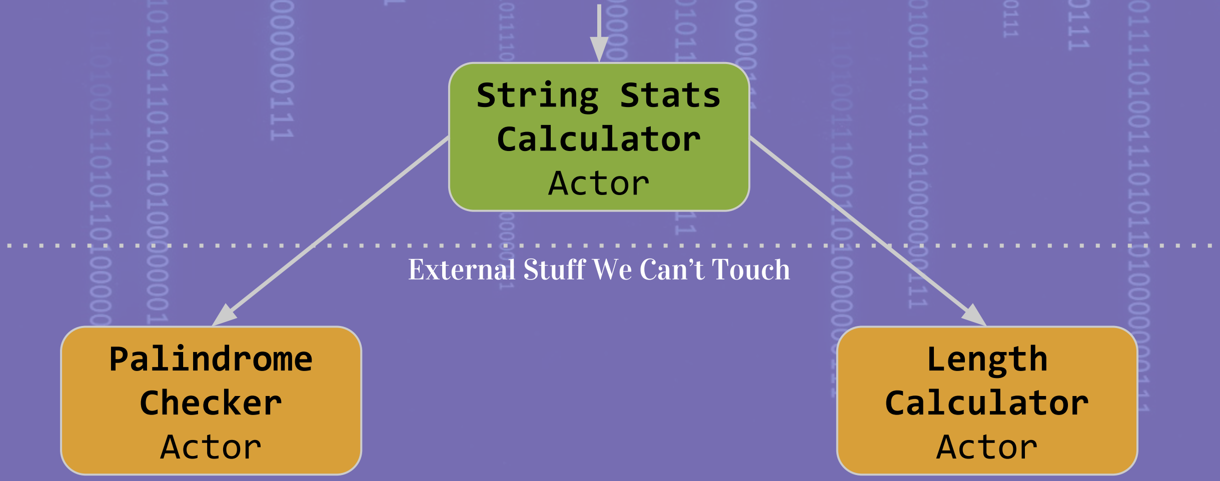
LIVE CODING #1

Let’s start with a very simple case...

```
def add(Int, Future[Int]): Future[Int]
```

LIVE CODING #2

```
def stringStats(String): Future[StringStats]
```



The Journey

Entangled COMPLEXITIES

Injecting **FUTUREs**

LOGIC vs EXECUTION

Thanks!

ANY QUESTIONS?

@miciek

www.michalplachta.com/contact