It Depends...

@KevlinHenney

novice

advanced beginner

novice

competent
advanced beginner
novice

proficient
competent
advanced beginner
novice

expert
proficient
competent
advanced beginner
novice

expert
proficient
competent
advanced beginner
novice

The Dreyfus Model of Skill Acquisition

expert intuitive

proficient

competent

advanced beginner

novice analytical

expert holistic

proficient

competent

advanced beginner

novice decomposed

expert situational

proficient

competent

advanced beginner

novice non-situational

expert context-sensitive

proficient

competent

advanced beginner

novice context-free

expert "it depends..."

proficient

competent

advanced beginner

novice "always/never..."

shuhari

shu·ha·ri

守破離

守破離

imitate

守破離

innovate

守破離

invent

What do I think? This code sucks.

Teedy Deigh
The Way of the Consultant

What do I think?

Well... it's not all bad! Nothing that some aggressive, merciless and inconsiderate refactoring couldn't solve.

Teedy Deigh

The Way of the Consultant

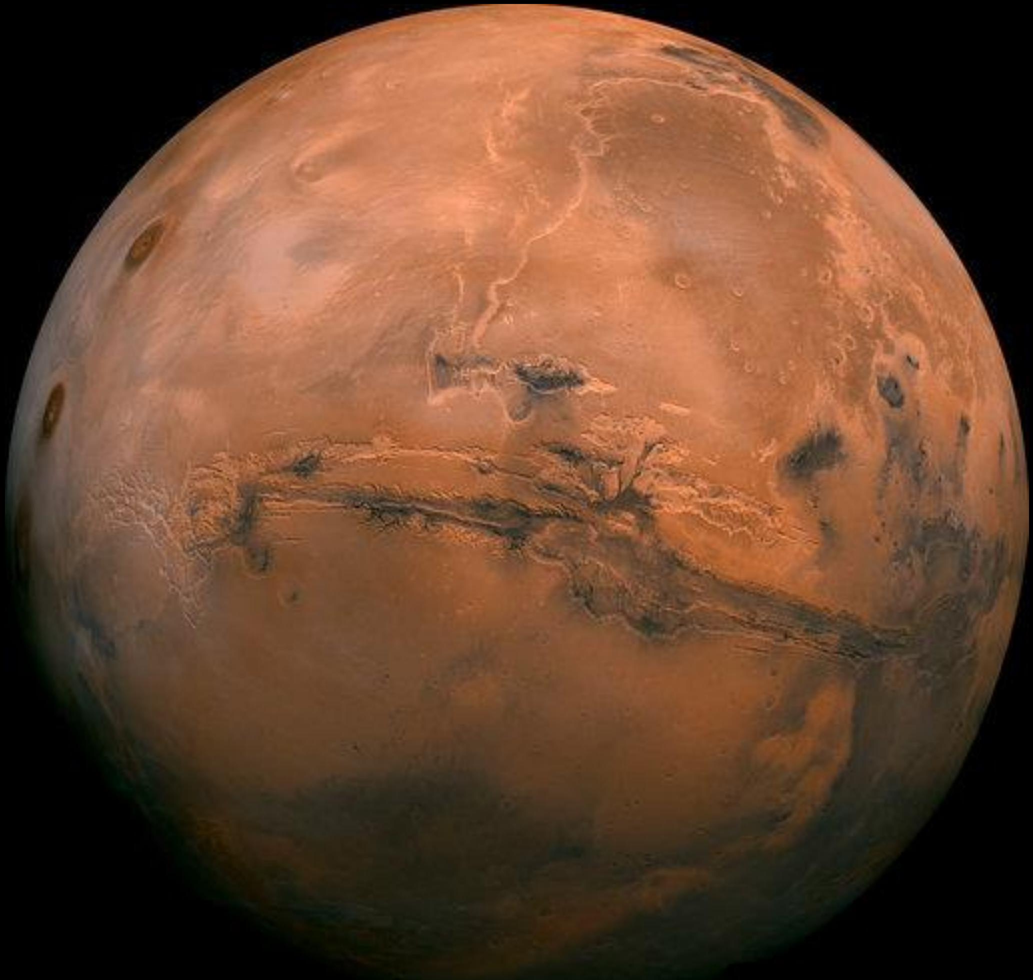
What do I think?

Although there are aspects of the system's design that are sound, the solution as a whole may be better aligned with the needs of the business by leveraging the synergies of complementary solution paths. The resulting amelioration of quality will be further enhanced by the displacement of vestigial solution components extant from the status quo.

Teedy Deigh
The Way of the Consultant

What do I think?
It depends.

Teedy Deigh
The Way of the Consultant




DRY


DRY


Don’t
Repeat
Yourself


Kevlin Henney 编
李军 译 吕骏 审校
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn
O'REILLY®
オライリー・ジャパン
97
97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY®
Edited by Kevlin Henney

Beware the Share

Udi Dahan

As I worked through my first feature, I took extra care to put in place everything I had learned — commenting, logging, pulling out shared code into libraries where possible, the works.

The code review that I had felt so ready for came as a rude awakening — reuse was frowned upon!

Udi Dahan

How could this be? Throughout college, reuse was held up as the epitome of quality software engineering.

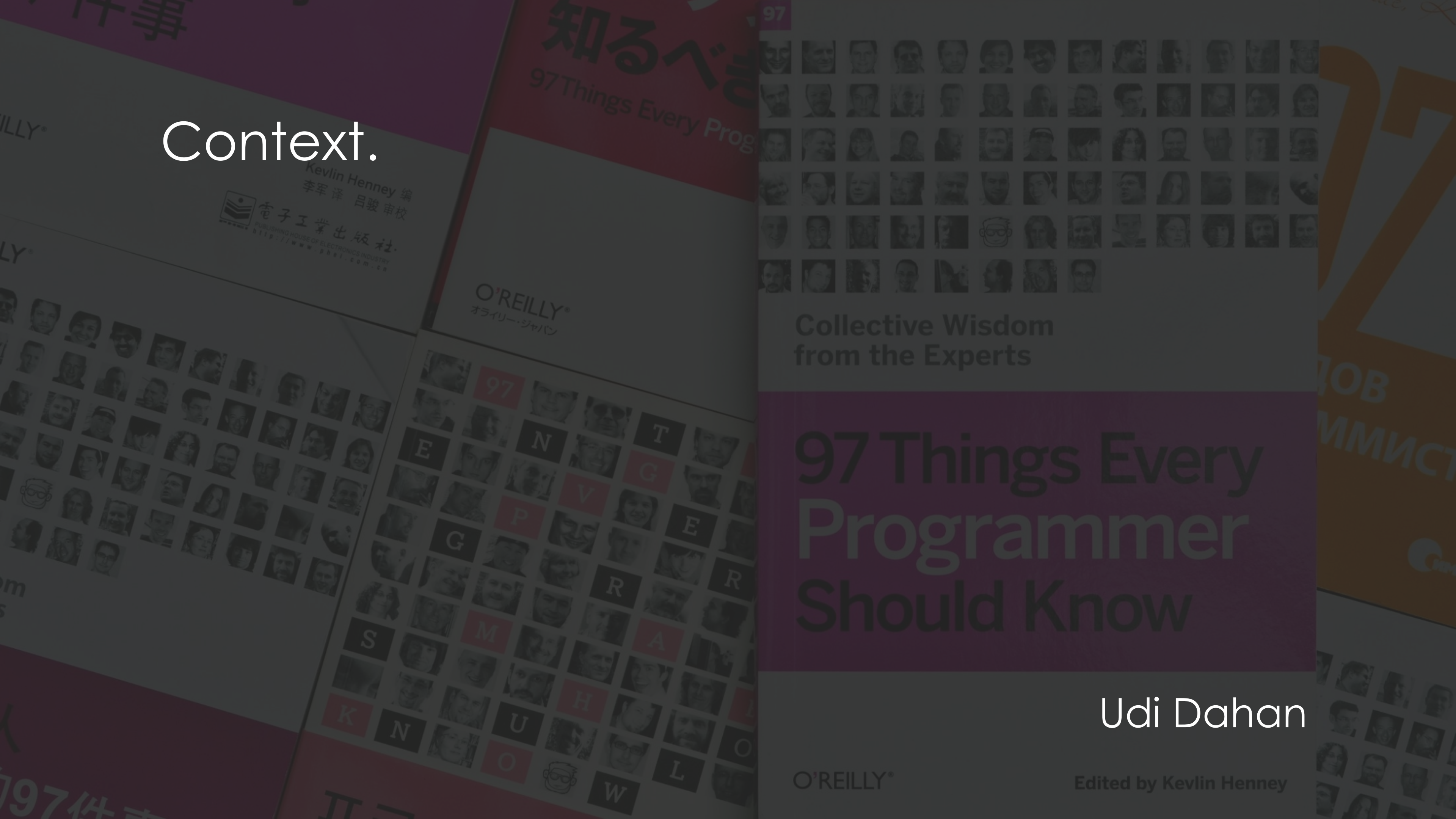
All the articles I had read, the textbooks, the seasoned software professionals who taught me — was it all wrong?

Udi Dahan

It turns out that I was missing something critical.

Udi Dahan

Context.



Udi Dahan

The fact that two wildly different parts of the system performed some logic in the same way meant less than I thought.

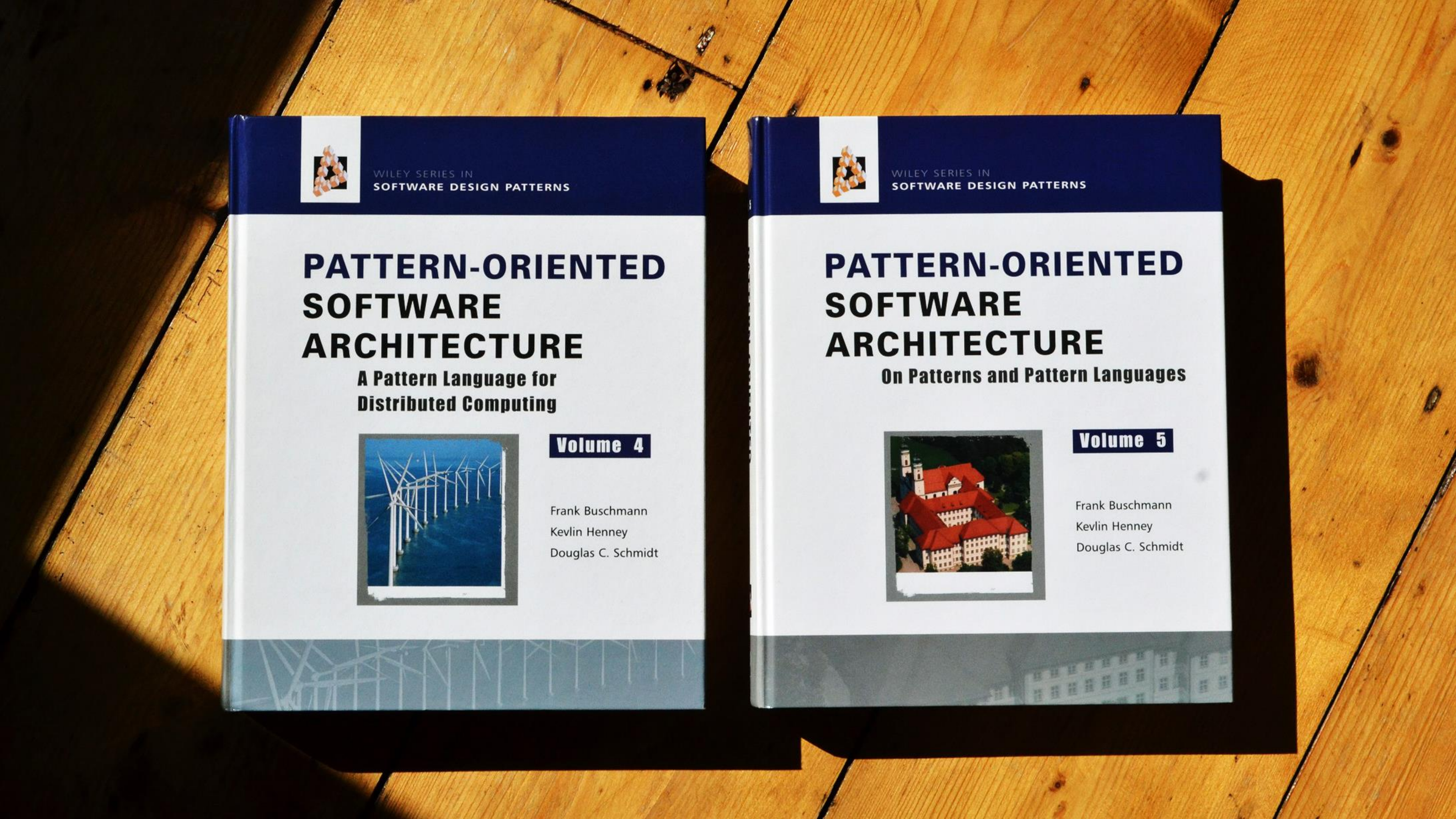
Up until I had pulled out those libraries of shared code, these parts were not dependent on each other. Each could evolve independently.

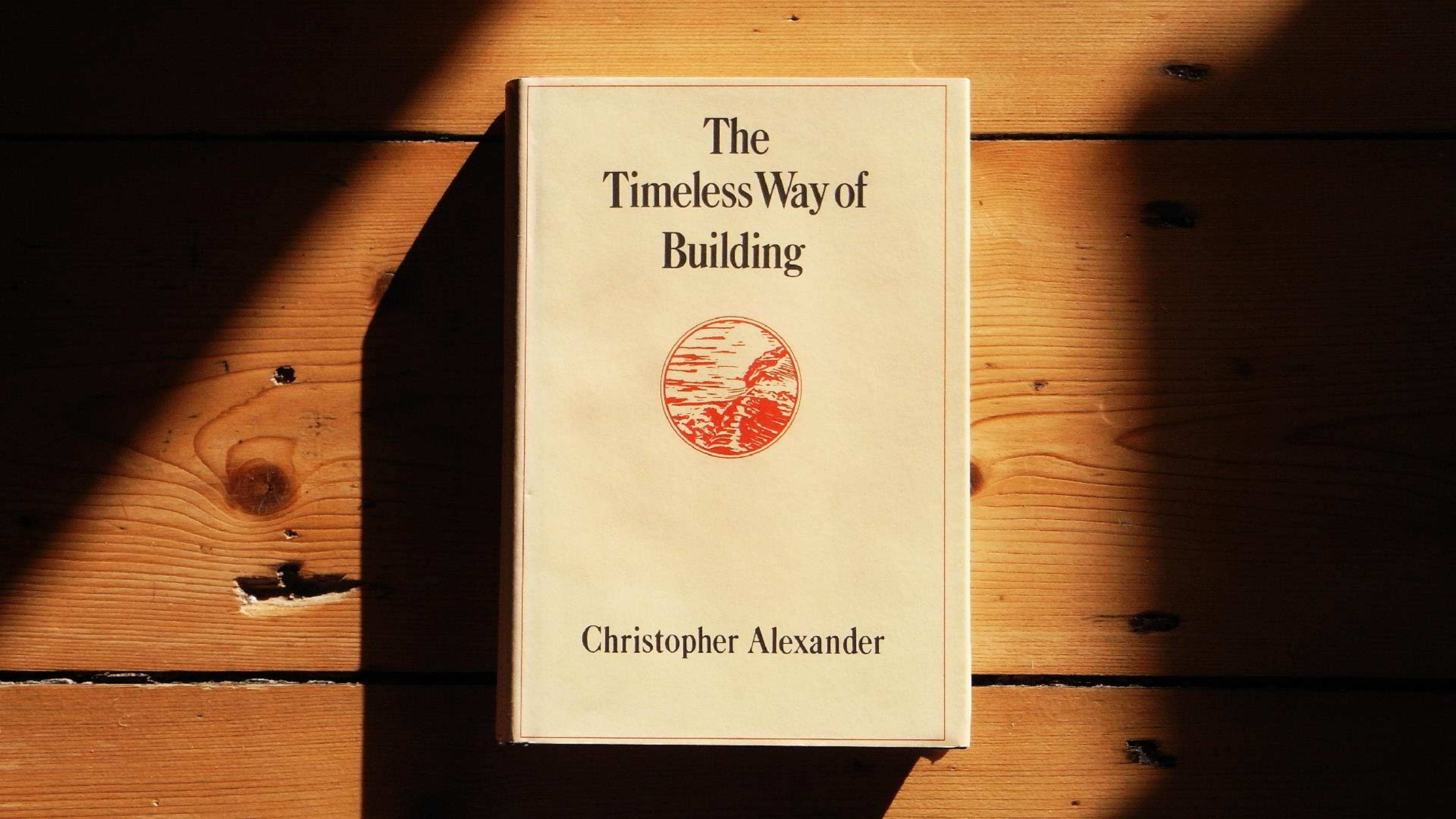
Udi Dahan

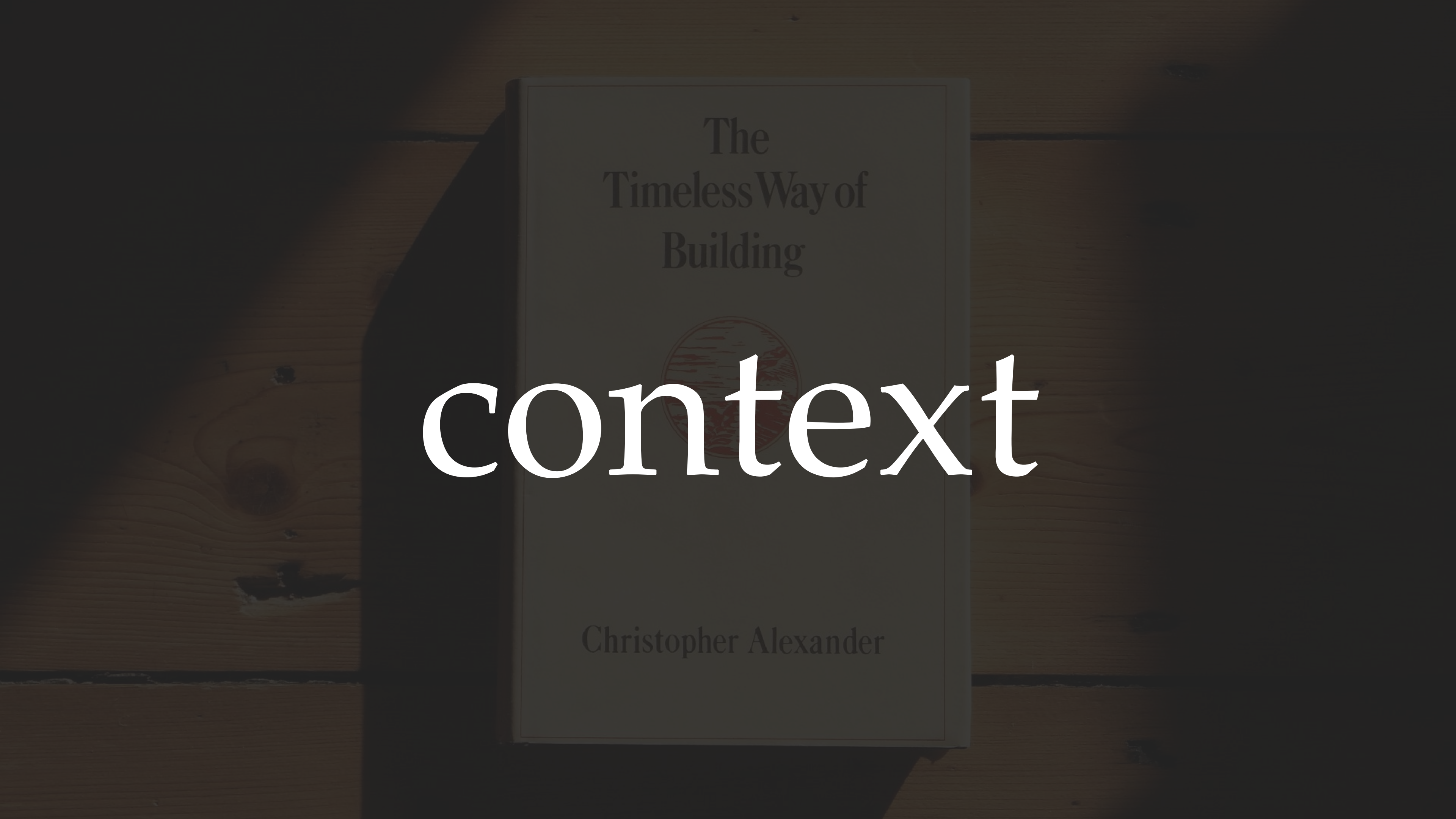
directives

principles

patterns


WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

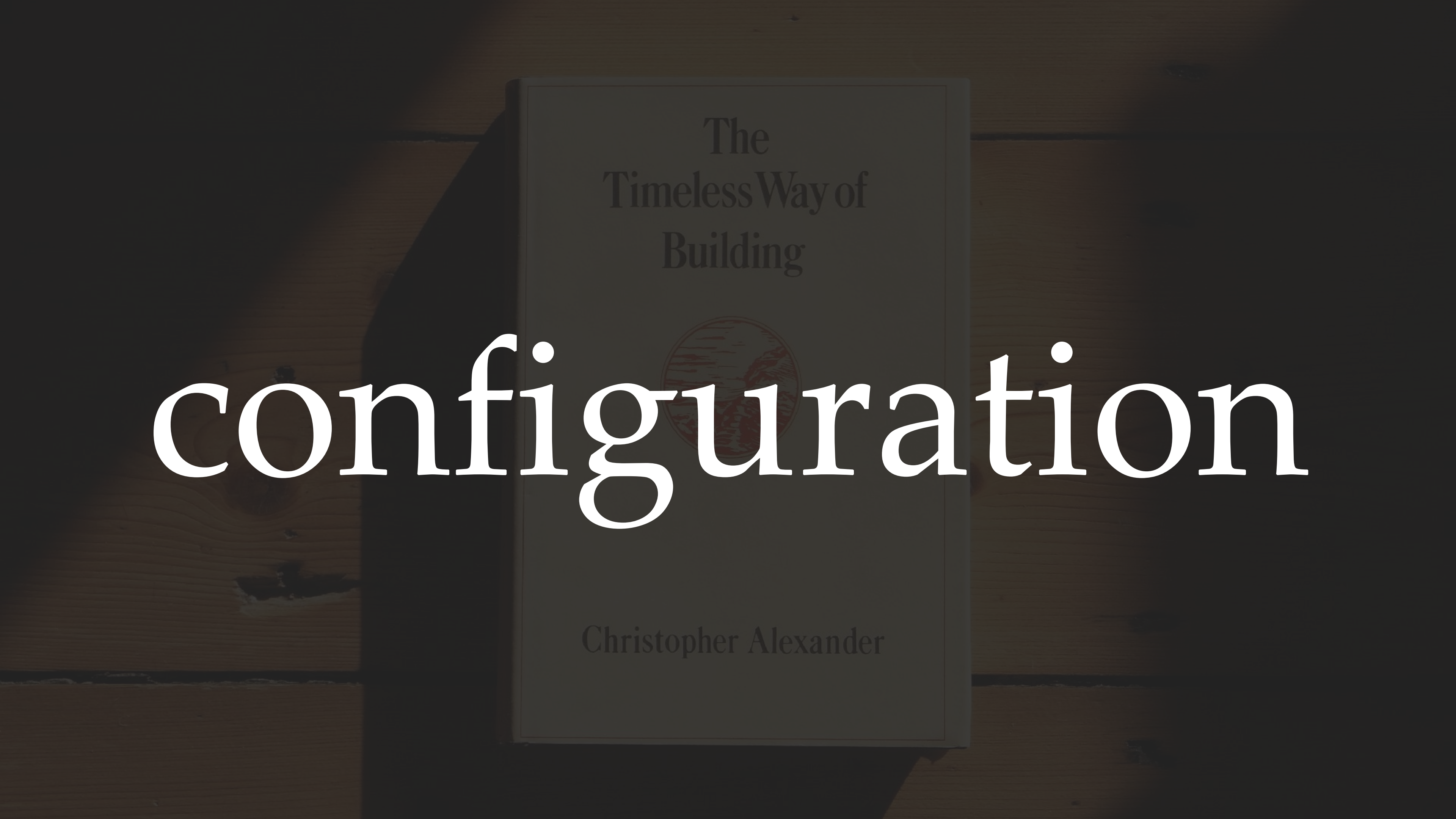

The
Timeless Way of
Building
Christopher Alexander


The
Timeless Way of
Building
context
Christopher Alexander

conflicting forces

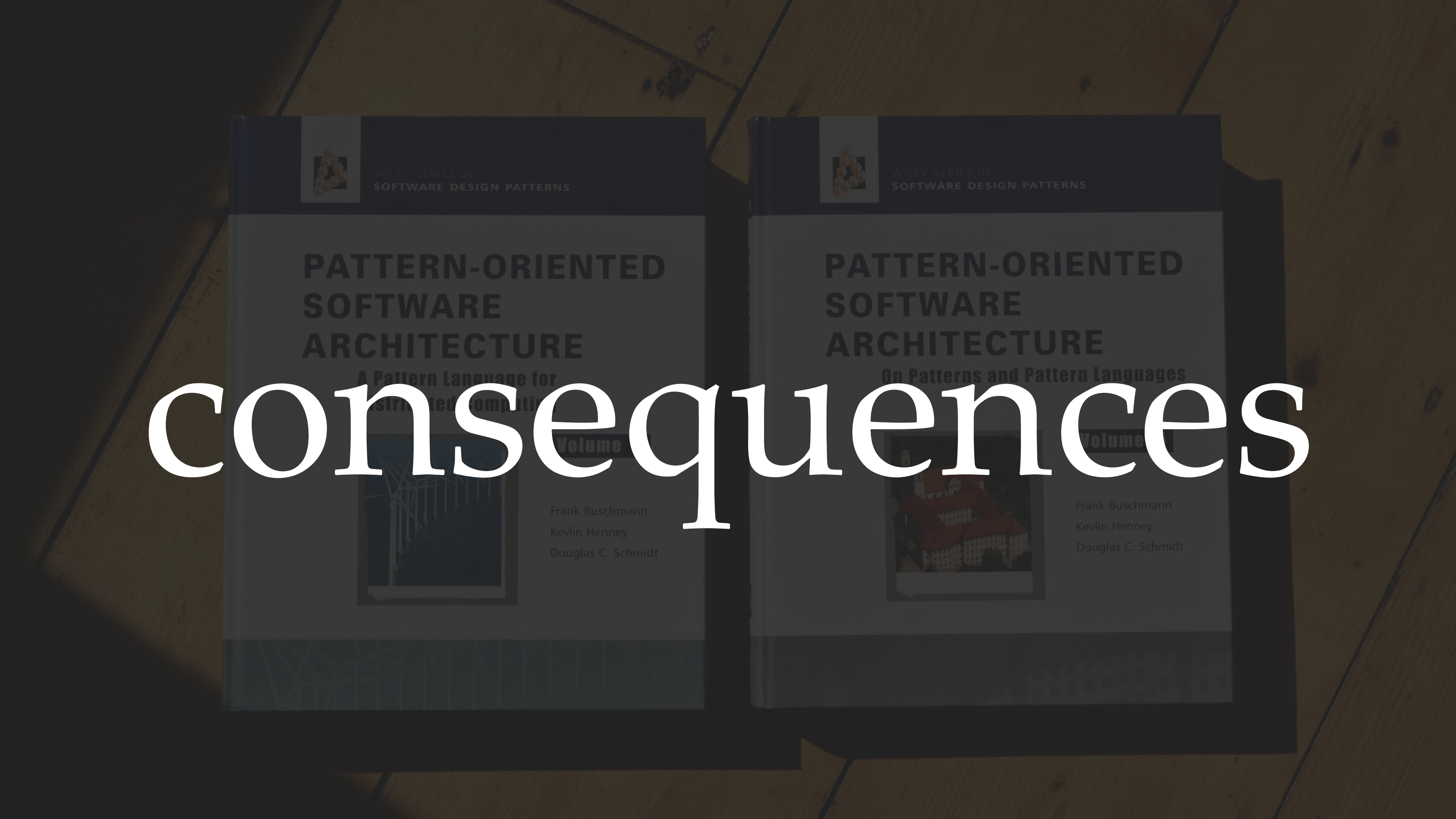
problem


WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt


The
Timeless Way of
Building
Christopher Alexander
configuration

solution


WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
A Pattern Language for
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt


consequences
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
A Pattern Language for
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
On Patterns and Pattern Languages
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

patterns

design

code

There’s no such thing as bad weather, only unsuitable clothing.

mutex

bottleneck

scoped_lock

synchronized

StringBuffer

String

DOI:10.1145/2844112

We need it, we can afford it, and the time is now.

BY PAT HELLAND

Immutability Changes Everything

latches has become harder ... latch latency loses lots of ... opportunities. Keeping ... copies of lots of data is now afford... and one payoff is reduced coordina... challenges.

Storage is increasing as the cost ... terabyte of disk keeps dropping ... means a lot of data can be kept ... long time. Distribution is ... ing as more and more data and ... are spread across a great dista... Data within a data center seems "fa... away." Data within a many-core ... may seem "far away." Ambiguity ... increasing when trying to coordina... with systems that are far away—... stuff has happened since you ... heard the news. Can you take ... with incomplete knowledge? Can ... wait for enough knowledge?

Turtles all the way down." As ... ous technological ...

There's no such thing as thread-unsafe code, only unsuitable threading.

StringBuffer

StringBuilder

static


includes
the C
Rationale
The
C
Standard
Incorporating Technical
Corrigendum No. 1

using the equivalent of the following algorithm.

```
char *asctime(const struct tm *timeptr)
{
     static const char wday_name[7][3] = {
          "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
     };
     static const char mon_name[12][3] = {
          "Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
     };
     static char result[26];
     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
          wday_name[timeptr->tm_wday],
          mon_name[timeptr->tm_mon],
          timeptr->tm_mday, timeptr->tm_hour,
          timeptr->tm_min, timeptr->tm_sec,
          1900 + timeptr->tm_year);
     return result;
}
```

Returns

3 The **asctime** function returns a pointer to the string.

7.23.3.2 The ctime function

7.23.3.3 The gm

Synopsis

1 #includ
struct

Description

2 The gmtime fun
down time, expres

Returns

3 The gmtime fun
specified time can

7.23.3.4 The lo

Synopsis

1 #incluc
struct

Description

2 The localtim
broken-down tim

Returns

3 The localtim

gets

```
char * gets(char * s);
```

```
puts
gets
```

```c
void example(void)
{
    char s[32];
    puts("What is your full name?");
    gets(s);
    ...
}
```

```
void example(void)
{
    char s[32];
    puts("What is your full name?");
    gets(s);
    ...
}
```

S-Programs

P-Programs

E-Programs

Meir M Lehman

“Programs, Life Cycles, and Laws of Software Evolution”

S-Programs

P-Programs

E-Programs

Programs whose function is formally defined by and derivable from a specification.

Meir M Lehman

“Programs, Life Cycles, and Laws of Software Evolution”

programming pearls

By Jon Bentley

WRITING CORRECT PROGRAMS

In the late 1960s people were talking about the promise of programs that verify the correctness of other programs. Unfortunately, it is now the middle of the 1980s, and, with precious few exceptions, there is still little more than talk about automated verification systems. Despite unrealized expectations, however, the research on program verification has given us something far more valuable than a black box that gobbles programs and flashes "good" or "bad"—we now have a fundamental understanding of computer programming.

The purpose of this column is to show how that fundamental understanding can help programmers write correct programs. But before we get to the subject itself, we must keep it in perspective. Coding skill is just one small part of writing correct programs. The majority of the task is the subject of the three previous columns: problem definition, algorithm design, and data structure selection. If you perform those tasks well, then writing correct code is usually easy.

The Challenge of Binary Search

I've given this problem as an in-class assignment in courses at Bell Labs and IBM. The professional programmers had one hour (sometimes more) to convert the above description into a program in the language of their choice; a high-level pseudocode was fine. At the end of the specified time, almost all the programmers reported that they had correct code for the task. We would then take 30 minutes to examine their code, which the programmers did with test cases. In many different classes and with over a hundred programmers, the results varied little: 90 percent of the programmers found bugs in their code (and I wasn't always convinced of the correctness of the code in which no bugs were found).

I found this amazing: only about 10 percent of professional programmers were able to get this small program right. But they aren't the only ones to find this task difficult. In the history in Section 6.2.1 of his *Sorting and Searching*, Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962.

Communications of the ACM, 26(12), December 1983

```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = low + ((high - low) / 2);
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

Probably faster, and arguably as clear

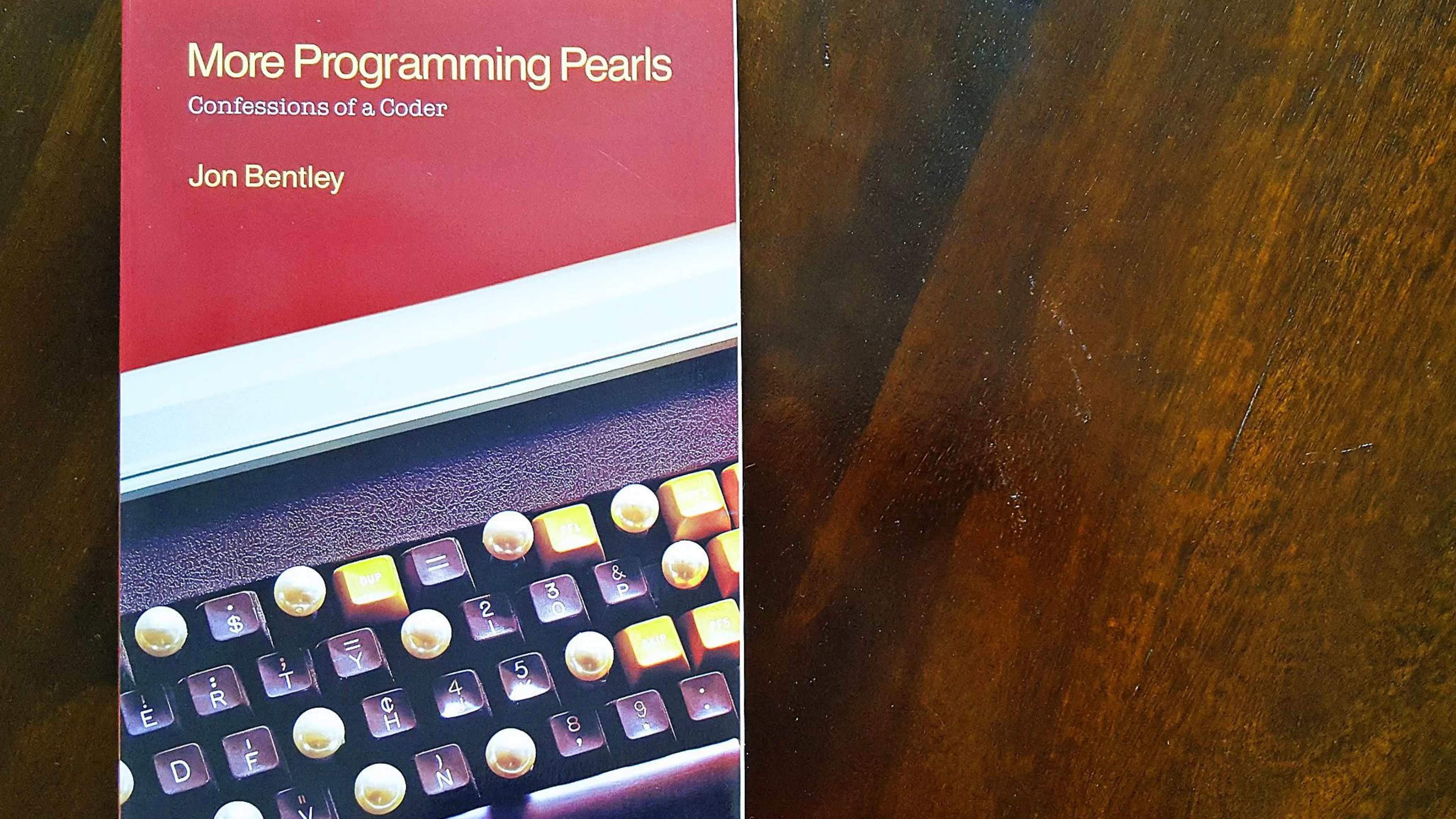
```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

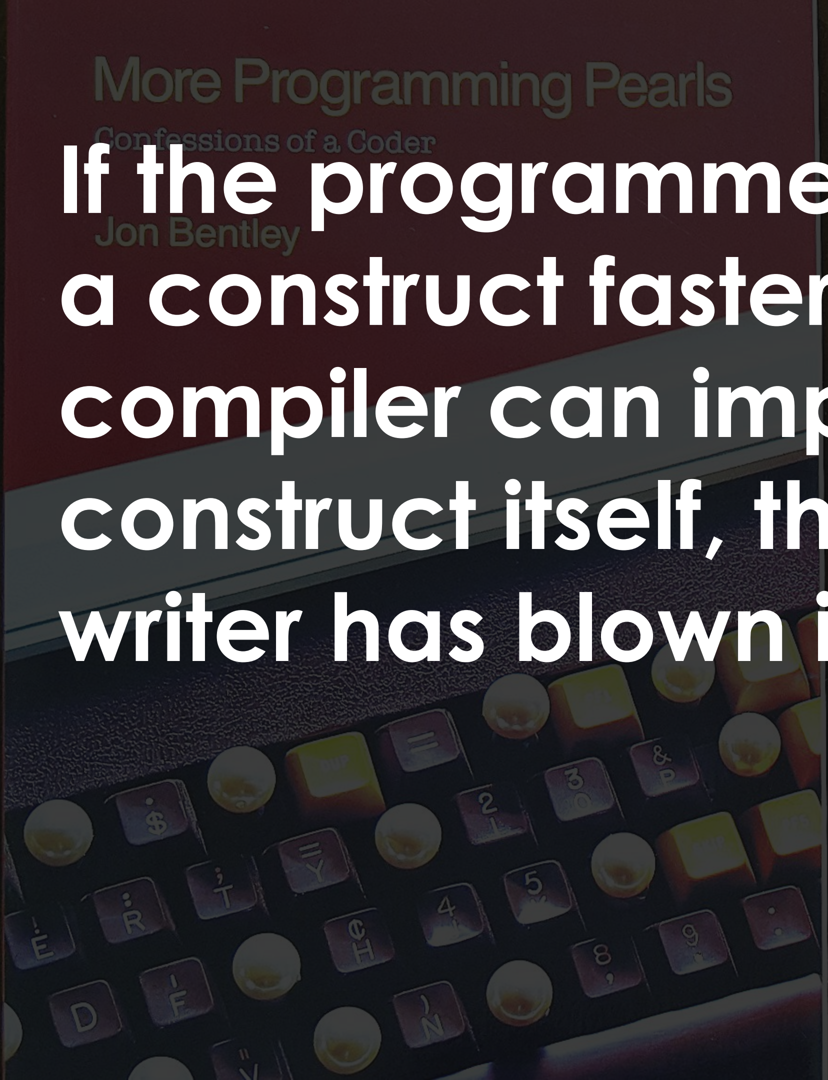
ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html


/Theory/In/Practice
Beautiful
Code
Leading Programmers Explain How They Think
O'REILLY
Edited by Andy Oram & Greg Wilson

Probably faster but may be obscure to most Java developers (including me)

Alberto Savoia


More Programming Pearls
Confessions of a Coder
Jon Bentley



If the programmer can simulate a construct faster than the compiler can implement the construct itself, then the compiler writer has blown it badly.

Guy L Steele, Jr

Simple Testing Can Prevent Most Critical Failures

An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues,
Xu Zhao, Yongle Zhang, Pranay U Jain & Michael Stumm

usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf

A majority of the production failures (77%) can be reproduced by a unit test.

usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf

The general lesson that I take away from this bug is humility: It is hard to write even the smallest piece of code correctly, and our whole world runs on big, complex pieces of code.

Joshua Bloch

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

S-Programs

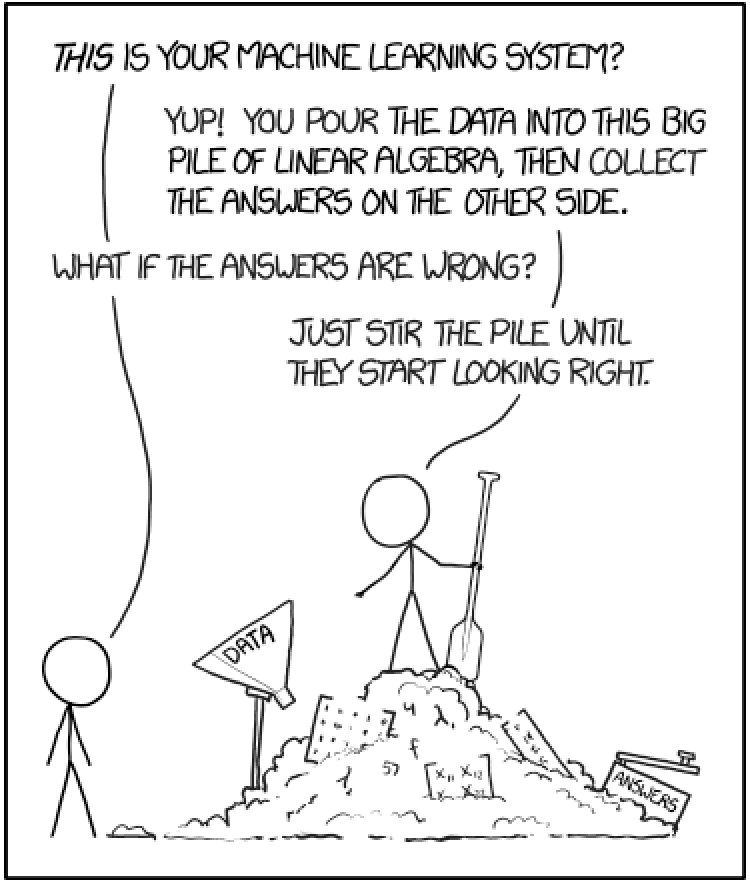
P-Programs

E-Programs

The acceptability of a solution is determined by the environment in which it is embedded.

Meir M Lehman

"Programs, Life Cycles, and Laws of Software Evolution"



xkcd.com/1838/

AI is characterized by output that isn't strictly dependent on the input or on the algorithm: the output of an AI system depends critically on a training process, in which the program learns how to perform its task. Training differentiates AI from traditional software applications and data analysis.

Mike Loukides

oreilly.com/radar/planning-for-ai/

Explanations must be wrong. They cannot have perfect fidelity with respect to the original model. If the explanation was completely faithful to what the original model computes, the explanation would equal the original model, and one would not need the original model in the first place, only the explanation.

Cynthia Rudin

“Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”

“Machine learning” is a fancy way of saying “finding patterns in data”.

Laurie Penny

theguardian.com/commentisfree/2017/apr/20/robots-racist-sexist-people-machines-ai-language

Of course, as Lydia Nicholas […] explains, all this data "has to have been collected in the past, and since society changes, you can end up with patterns that reflect the past. If those patterns are used to make decisions that affect people's lives you end up with unacceptable discrimination."

Laurie Penny

theguardian.com/commentisfree/2017/apr/20/robots-racist-sexist-people-machines-ai-language

S-Programs

P-Programs

E-Programs

Programs that mechanize a human or societal activity. The program has become a part of the world it models, it is embedded in it.

Meir M Lehman

"Programs, Life Cycles, and Laws of Software Evolution"

The Making of a Fly: The Genetics of Animal Design (Paperback)

by Peter A. Lawrence

‹ Return to product information

Always pay through Amazon.com's Shopping Cart or 1-Click.
Learn more about Safe Online Shopping and our safe buying guarantee.

Price at a Glance

List Price: ~~$70.00~~

Used: from **$35.54**

New: from **$1,730,045.91**

Have one to sell? Sell yours here

All | **New** (2 from $1,730,045.91) | **Used** (15 from $35.54)

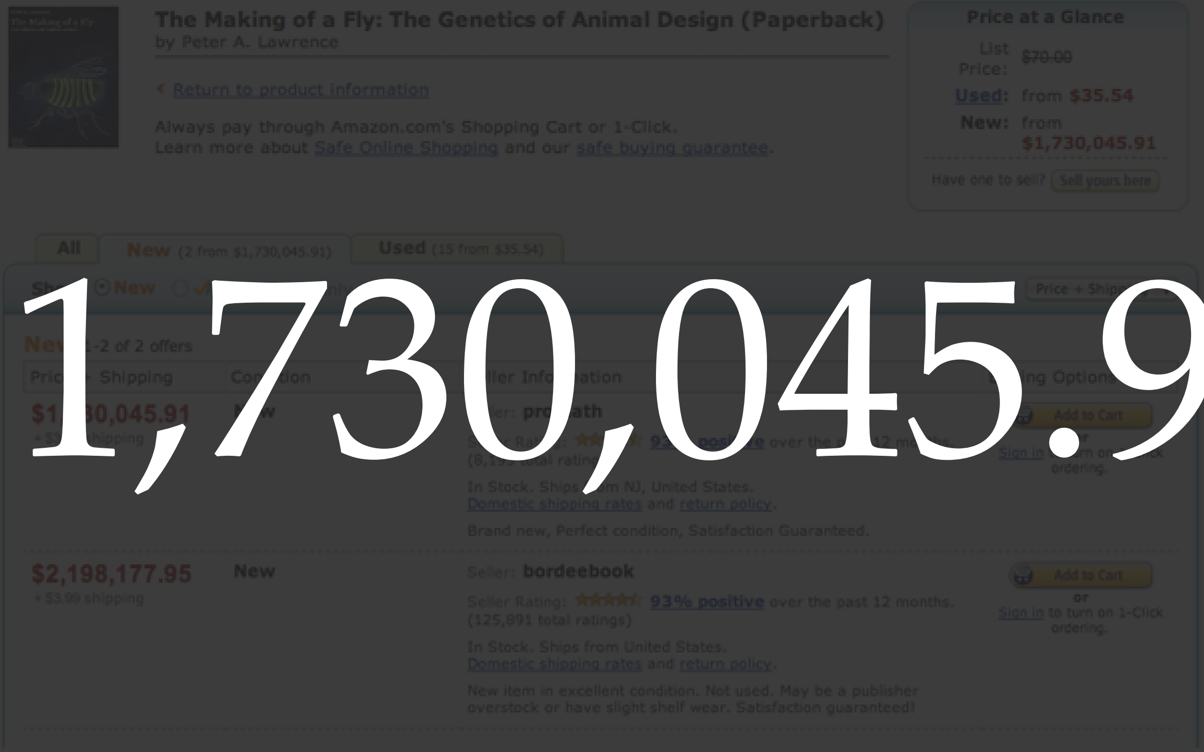
Show New · Prime offers only (0) — **Sorted by** Price + Shipping

New 1-2 of 2 offers

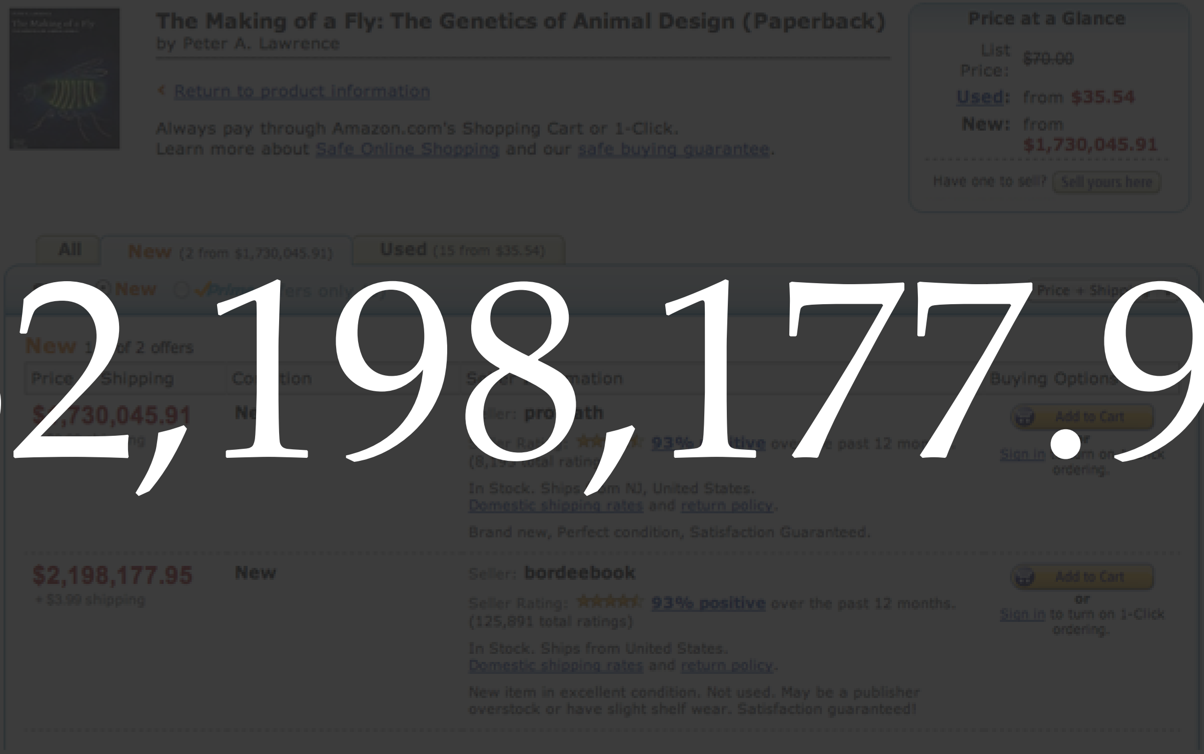
Price + Shipping	Condition	Seller Information	Buying Options
$1,730,045.91 + $3.99 shipping	**New**	Seller: **profnath** Seller Rating: 93% positive over the past 12 months. (8,193 total ratings) In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.	Add to Cart or Sign in to turn on 1-Click ordering.
$2,198,177.95 + $3.99 shipping	**New**	Seller: **bordeebook** Seller Rating: 93% positive over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!	Add to Cart or Sign in to turn on 1-Click ordering.

michaeleisen.org/blog/?p=358

$1,730,045.91

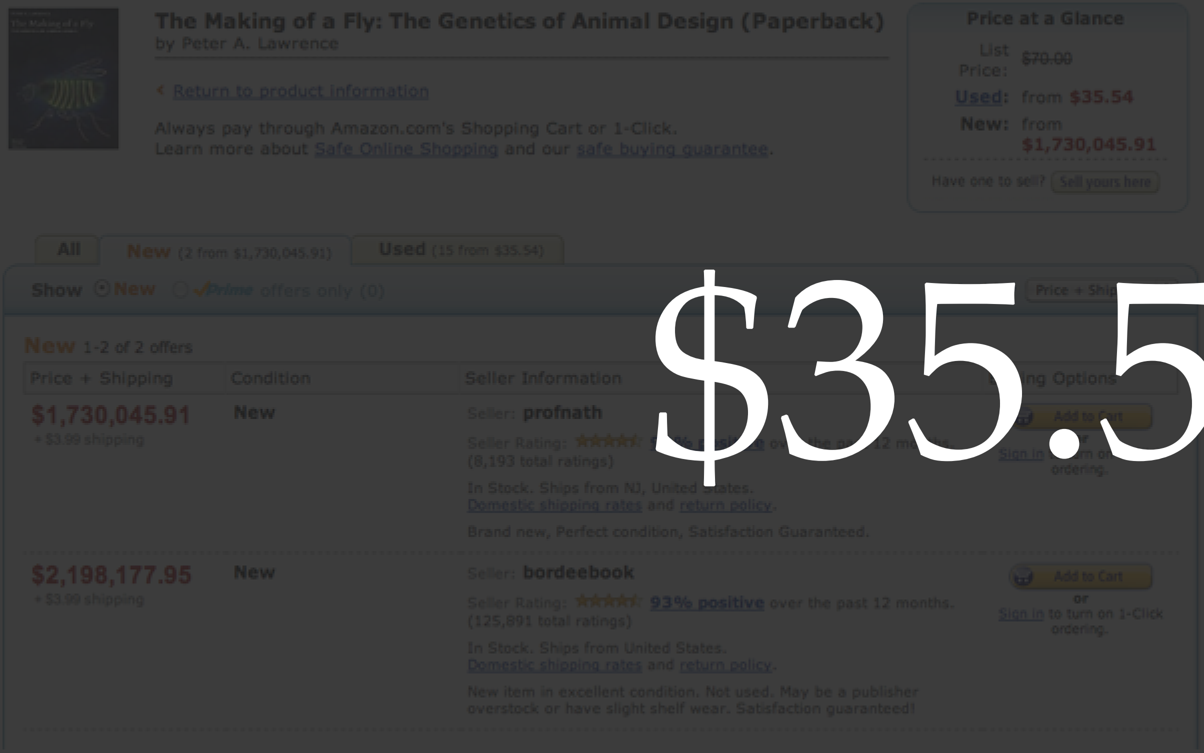


michaeleisen.org/blog/?p=358



$2,198,177.95

michaeleisen.org/blog/?p=358



$35.54

michaeleisen.org/blog/?p=358

	profnath	bordeebook	profnath over previous bordeebook	bordeebook over profnath
8-Apr	$1,730,045.91	$2,198,177.95		1.27059
9-Apr	$2,194,443.04	$2,788,233.00	0.99830	1.27059
10-Apr	$2,783,493.00	$3,536,675.57	0.99830	1.27059
11-Apr	$3,530,663.65	$4,486,021.69	0.99830	1.27059
12-Apr	$4,478,395.76	$5,690,199.43	0.99830	1.27059
13-Apr	$5,680,526.66	$7,217,612.38	0.99830	1.27059

michaeleisen.org/blog/?p=358

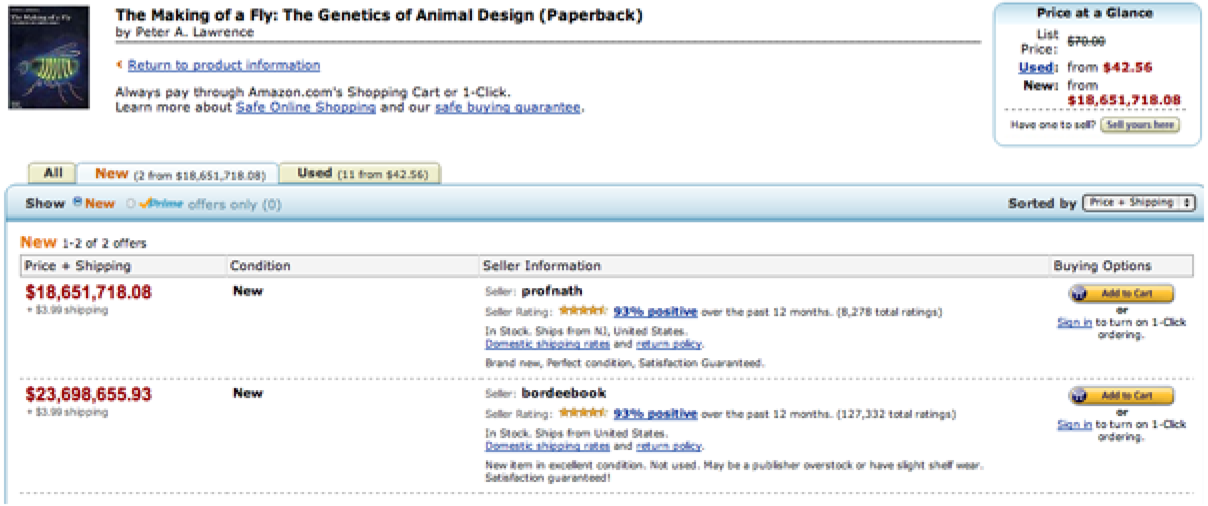
0.99830

	profnath	bordeebook	profnath over previous bordeebook	bordeebook over profnath
8-Apr	[illegible]	[illegible]	[illegible]	[illegible]
9-Apr	[illegible]	[illegible]	[illegible]	[illegible]
10-Apr	[illegible]	[illegible]	[illegible]	[illegible]
11-Apr	$3,530,663.65	$4,486,021.69	0.99830	1.27059
12-Apr	$4,478,395.76	$5,690,199.43	0.99830	1.27059
13-Apr	$5,680,526.66	$7,217,612.38	0.99830	1.27059

1.27509

	profnath	bordeebook	profnath over previous bordeebook	bordeebook over profnath
8-Apr	$1,730,045.9[illegible]	$2,198,[illegible]7.95	[illegible]	[illegible]
9-Apr	$2,194,443.0[illegible]	$2,78[illegible],233.00	0.998[illegible]0	1.2705[illegible]
10-Apr	$2,783,493.[illegible]	[illegible]	[illegible]9830	1.2[illegible]059
11-Apr	$3,530,663.65	$4,486,021.69	0.99830	1.27059
12-Apr	$4,478,395.76	$5,690,199.43	0.99830	1.27059
13-Apr	$5,680,526.66	$7,217,612.38	0.99830	1.27059

michaeleisen.org/blog/?p=358



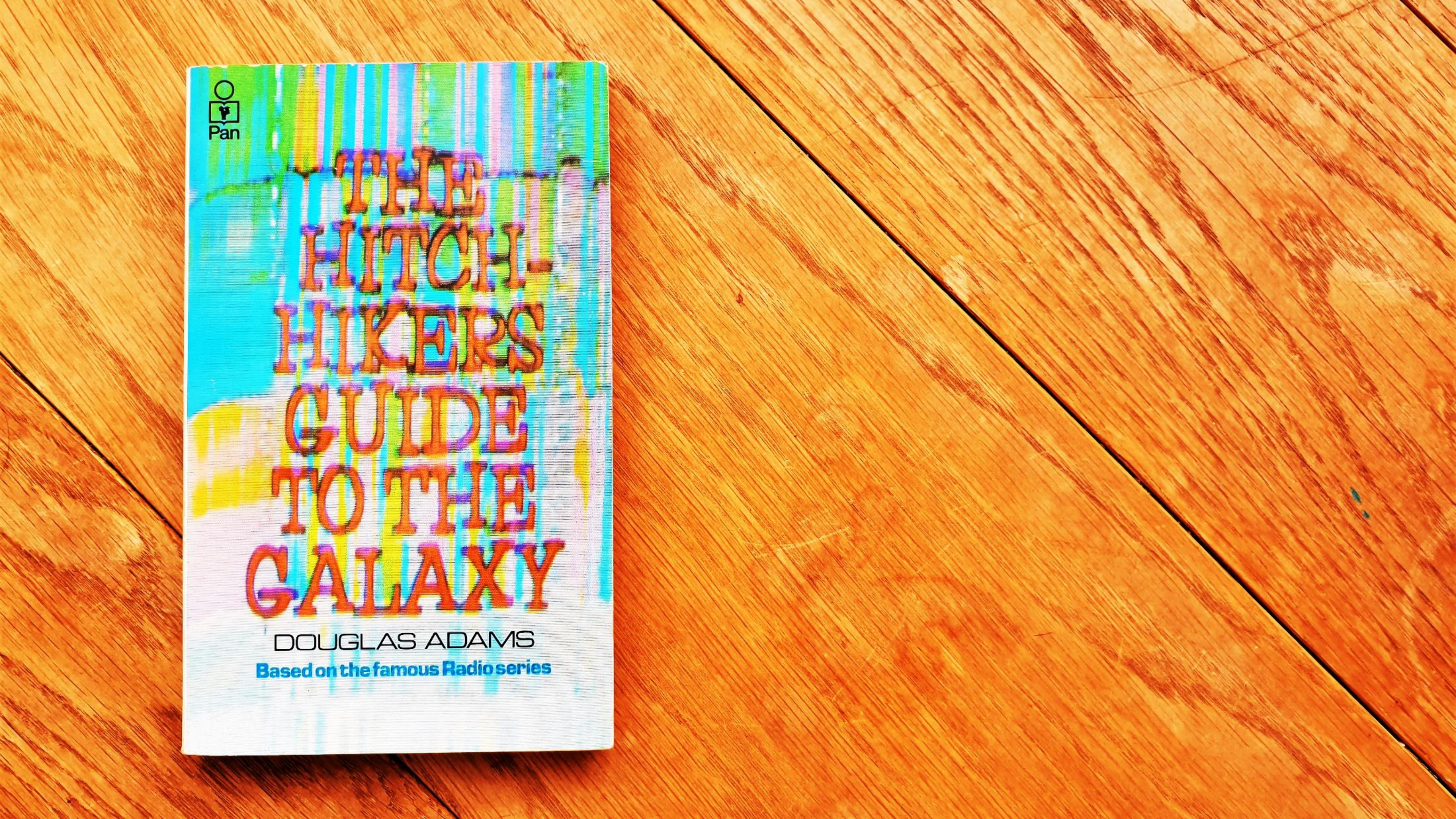
michaeleisen.org/blog/?p=358

$23,698,655.93

michaeleisen.org/blog/?p=358


101 Things I Learned
in Architecture School
Matthew Frederick


Always design a thing by considering it in its next larger context.
Eliel Saarinen
101 Things
Matthew Frederick
School

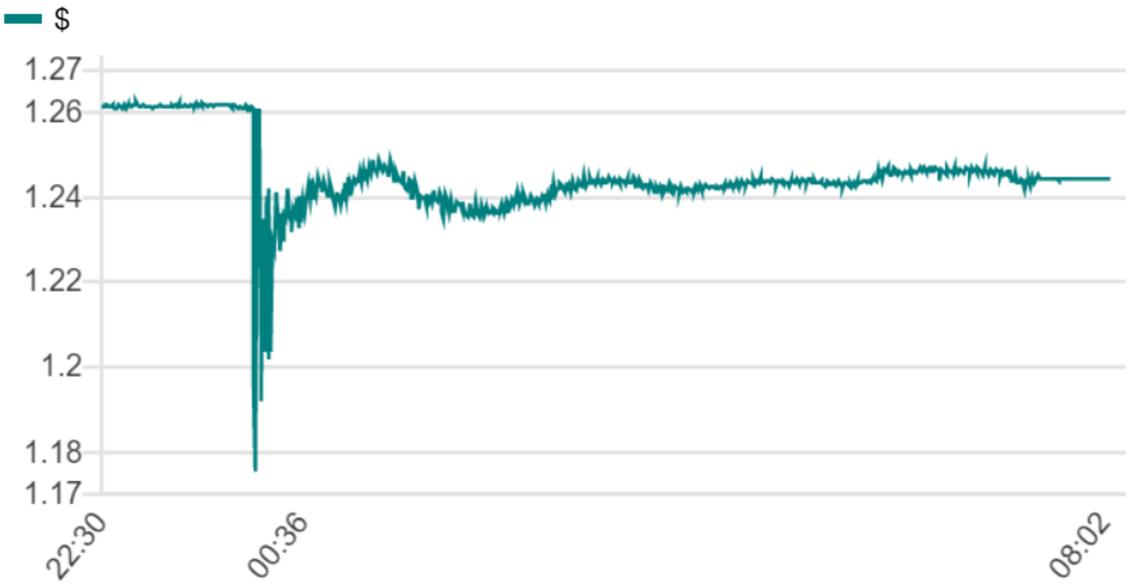

Pan
THE HITCH-HIKERS GUIDE TO THE GALAXY
DOUGLAS ADAMS
Based on the famous Radio series

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.

Sterling flash crash

£/$, 6-7 October



Source: Bloomberg

BBC

The pound has dived on Asian markets with automated trading being blamed for the volatility.

bbc.co.uk/news/business-37582150

Digital devices tune out small errors while creating opportunities for large errors.

Earl Wiener

S-Programs

P-Programs

E-Programs

Meir M Lehman

“Programs, Life Cycles, and Laws of Software Evolution”

S	Closed
P	Closed
E	Open

S	Defined
P	Undefined
E	Undefined

S	Definable
P	Definable
E	Undefinable

To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge.

Grace Hopper

0. lack of ignorance
1. lack of knowledge
2. lack of awareness
3. lack of process
4. meta-ignorance

Phillip G Armour

Five Orders of Ignorance

known knowns
known unknowns
unknown unknowns
unknowable unknowns

known knowns
known unknowns
unknown unknowns
unknowable unknowns

known knowns
known unknowns
unknown unknowns
unknowable unknowns

known knowns
known unknowns
unknown unknowns
unknowable unknowns

known knowns
known unknowns
unknown unknowns
unknowable unknowns

I know that I know nothing.

Socrates *

* Possibly

ULSS

Ultra-Large-Scale Systems

Unknowable

Decentralised

Evolving

Heterogeneous

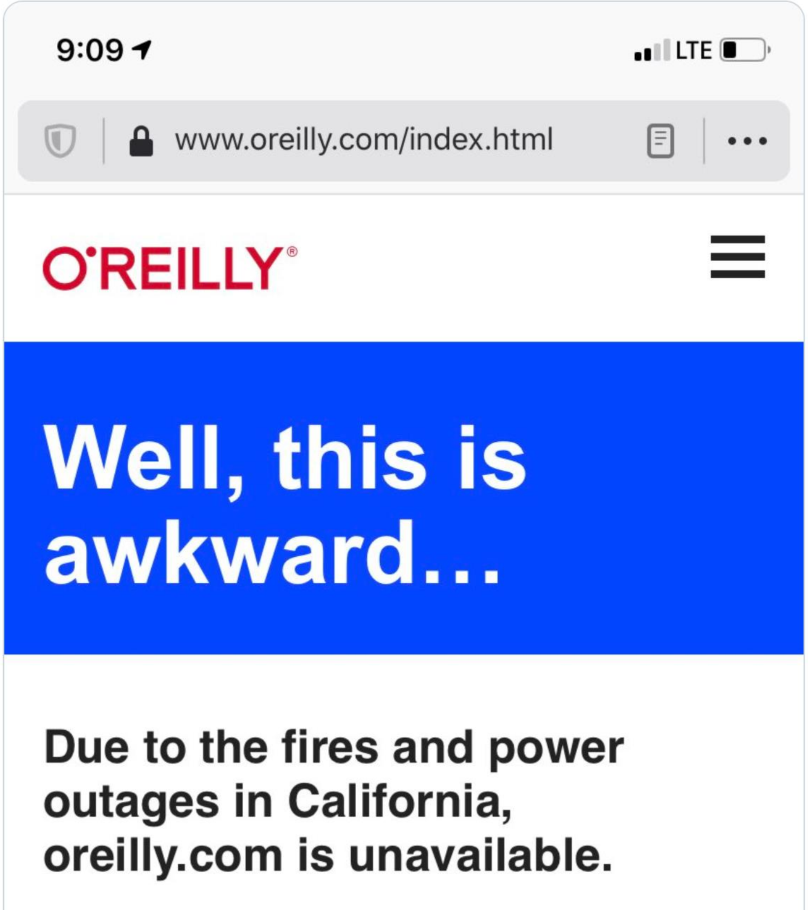
Failing

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

Charlie Morris
@cdmo

Fire in California, can't read your ebook in Pennsylvania



twitter.com/cdmo/status/1188806227079389184

It is a feature of a distributed system that it may not be in a consistent state, but it is a bug for a client to contradict itself.

twitter.com/KevlinHenney/status/1351956942877552646

Brewer’s theorem

CAP theorem

C

A

P

Consistency
Availability
Partition tolerance

Consistency
Availability
Partition tolerance

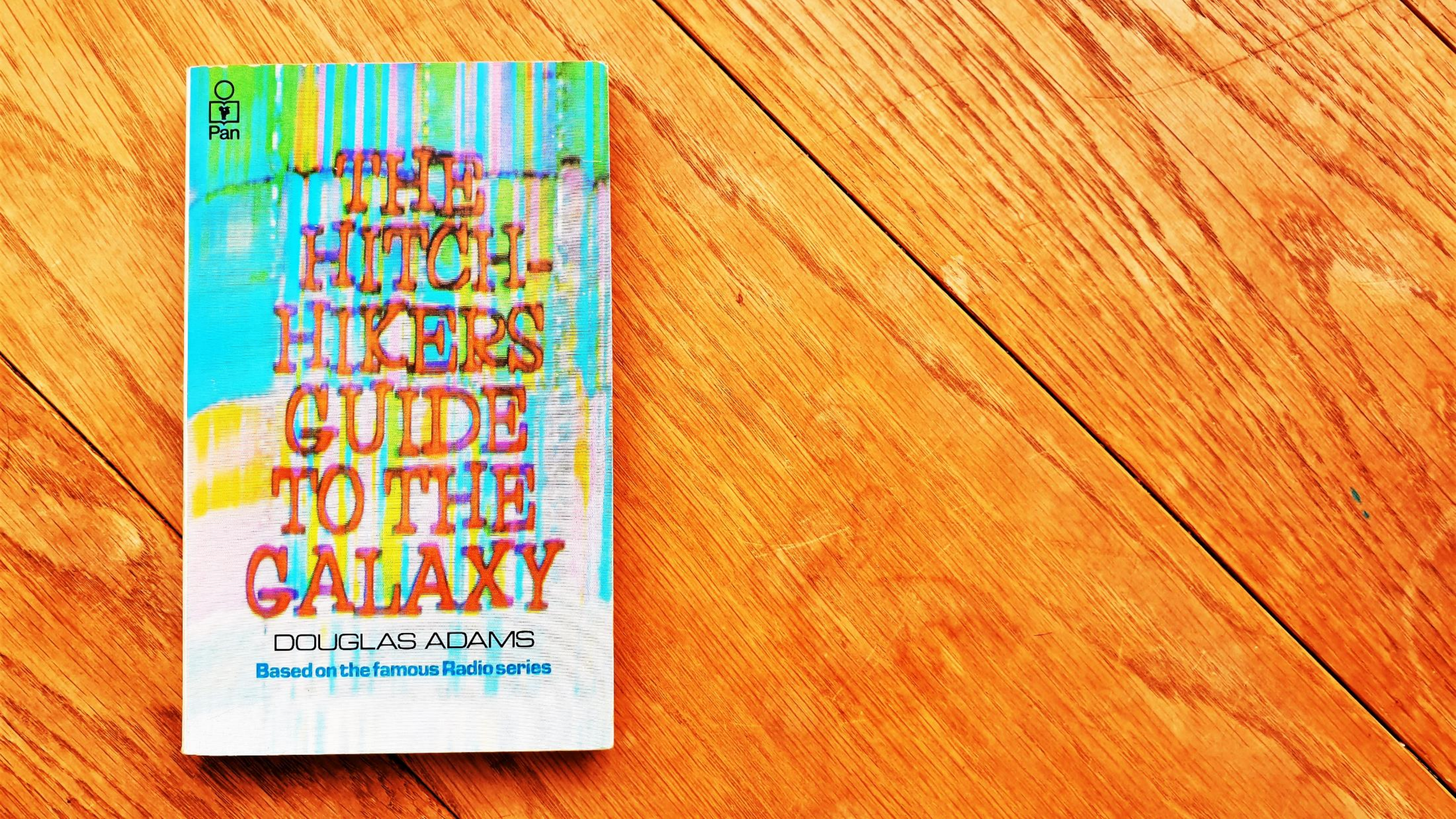
Consistency
Availability
Partition tolerance

Consistency

Availability

Partition tolerance

$$\Delta x \, \Delta p \geq \frac{\hbar}{2}$$


Pan
THE HITCH-HIKERS GUIDE TO THE GALAXY
DOUGLAS ADAMS
Based on the famous Radio series

We demand rigidly defined areas of doubt and uncertainty!

On Formally Undecidable Propositions Of Principia Mathematica And Related Systems

KURT GÖDEL

Translated by
B. MELTZER

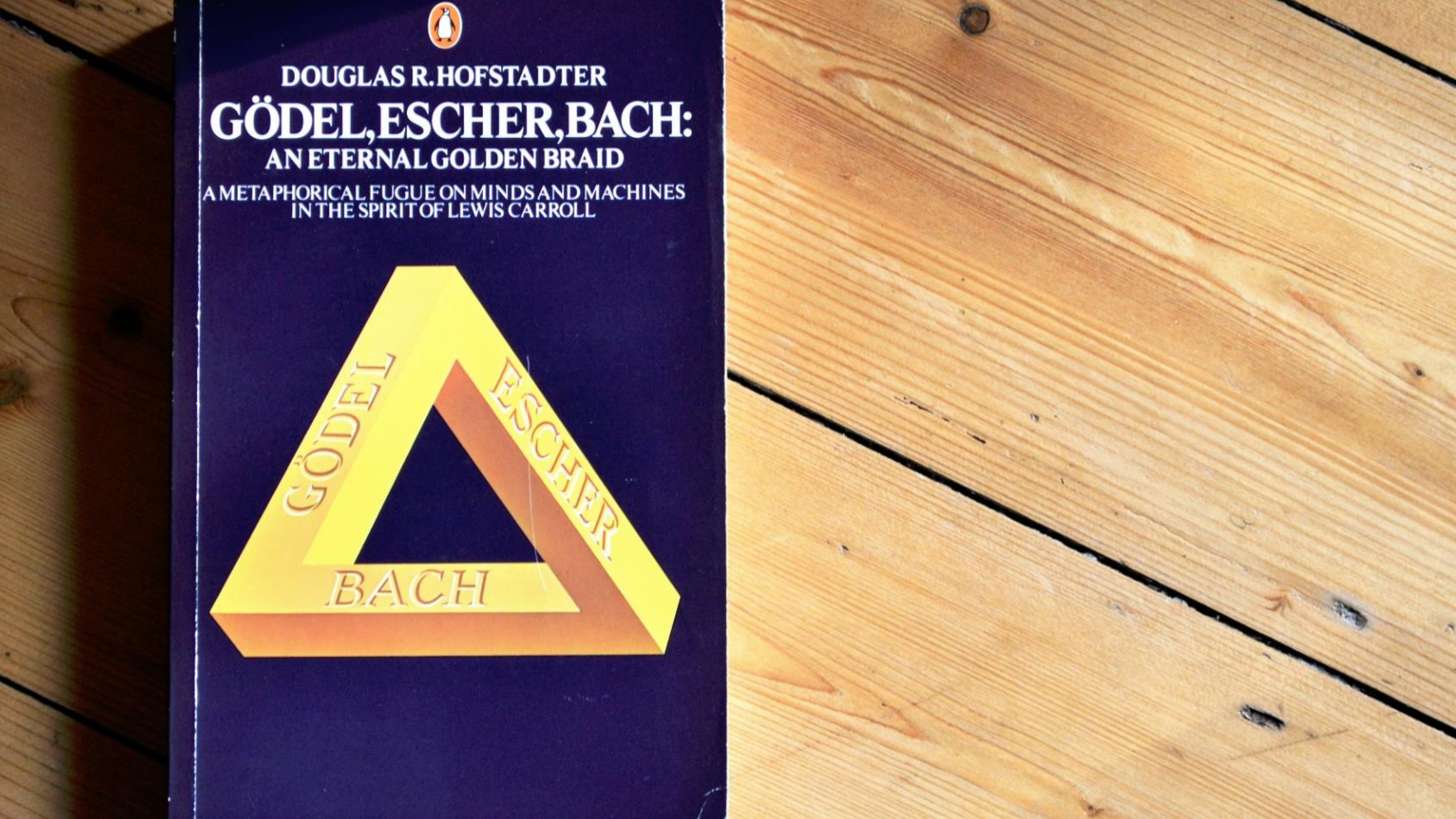
Introduction by
R B. BRAITHWAITE

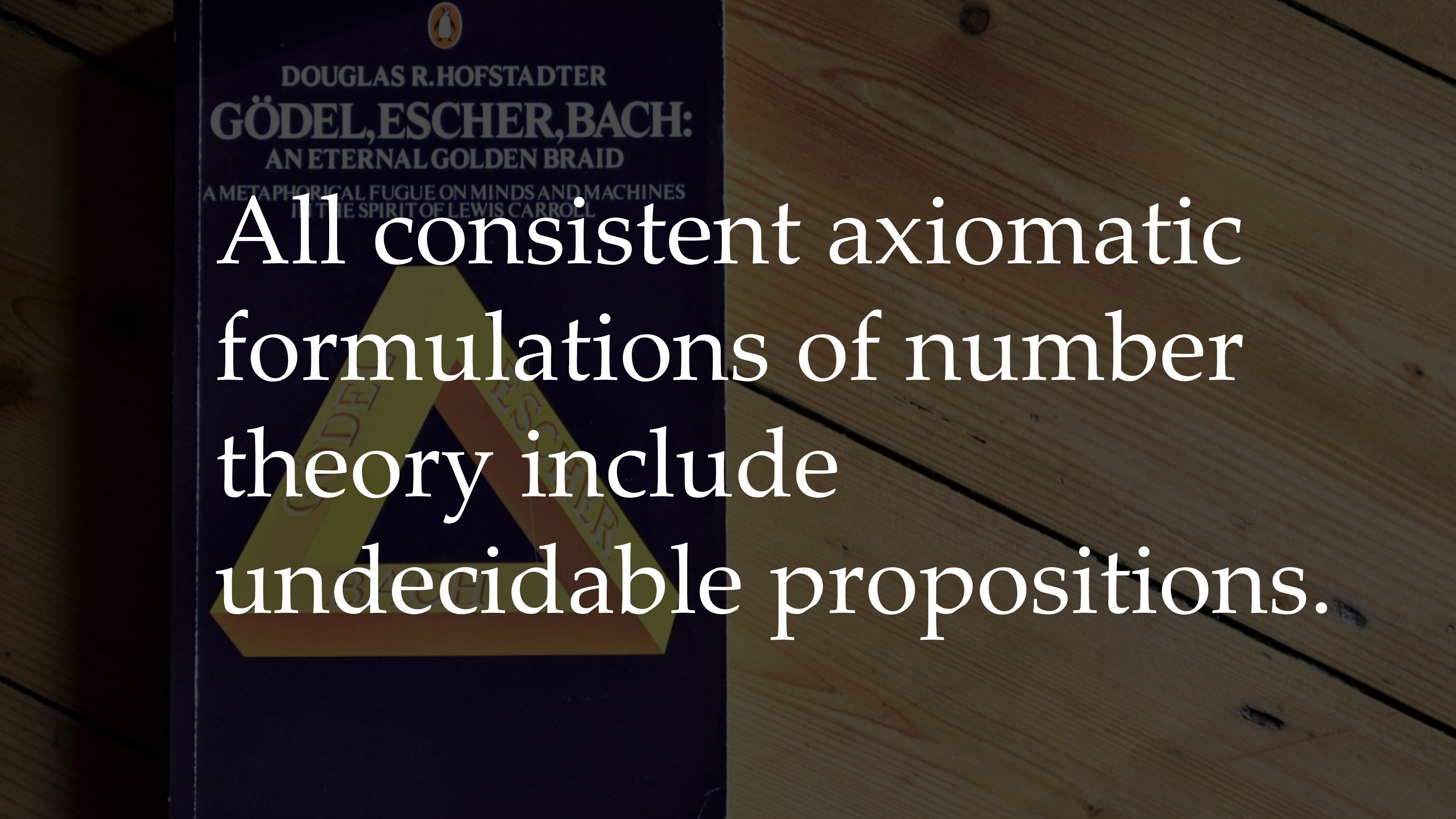
In 1911 Russell & Whitehead published Principia Mathematica, with the goal of providing a solid foundation for all of mathematics.

In 1931 Gödel's Incompleteness Theorem shattered the dream, showing that for any consistent axiomatic system there will always be theorems that cannot be proven within the system.

Adrian Colyer

blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/


DOUGLAS R. HOFSTADTER
GÖDEL, ESCHER, BACH:
AN ETERNAL GOLDEN BRAID
A METAPHORICAL FUGUE ON MINDS AND MACHINES
IN THE SPIRIT OF LEWIS CARROLL
GÖDEL
ESCHER
BACH



All consistent axiomatic formulations of number theory include undecidable propositions.

undecidable propositions

How long is a piece of string?

```c
size_t strlen(const char * s)
{
    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```c
size_t strlen(const char * s)
{
    assert(s != NULL);

    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```
size_t strlen(const char * s)
{
    assert(s != NULL);
    assert(∃n (s[n] == '\0'));
    assert(∀i∈0..n (s+i is a valid pointer));

    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```c
void well_defined(void)
{
    char s[] = "Be excellent to each other";
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
```

```
void insufficient_space(void)
{
    char s[5] = "Bogus";
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
```

```
void undefined_pointer_value(void)
{
    char * s;
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
```

One premise of many models of fairness in machine learning is that you can measure (‘prove’) fairness of a machine learning model from within the system – i.e. from properties of the model itself and perhaps the data it is trained on.

To show that a machine learning model is fair, you need information from outside of the system.

Adrian Colyer

blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/

AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

By Alonzo Church.

---

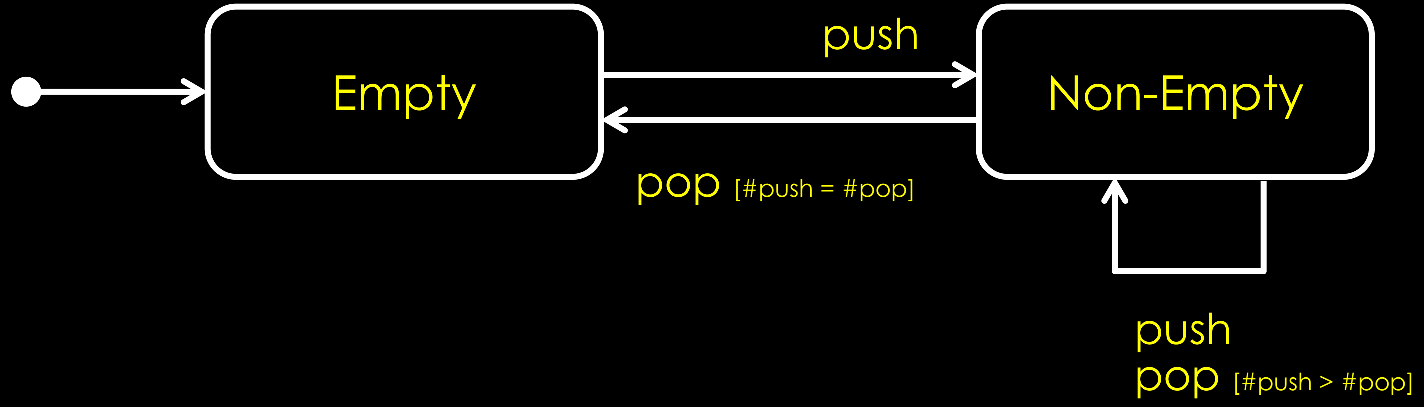
1. Introduction. There is a class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function f of n positive integers, such that $f(x_1, x_2, \cdots, x_n) = 2$ [2] is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving $x_1, x_2, \cdots, x_n$ as free variables.

An example of such a problem is the problem to find a means of determining of any given positive integer n whether or not there exist positive integers x, y, z, such that $x^n + y^n = z^n$. For this may be interpreted, required to find an effectively calculable function f, such that $f(n)$ is equal to 2 if and only if there exist positive integers x, y, z, such that $x^n + y^n = z^n$. Clearly λ

```
push := make(chan string)
pop := make(chan string)

go Stack(push, pop)

push<- ”ACCU”
push<- ”2021”

Println(<-pop)
Println(<-pop)
```

```
2021
ACCU
```

```
push := make(chan string)
pop := make(chan string)

go Stack(push, pop)

Println(<-pop)
```


Empty
push
Non-Empty
pop [#push = #pop]
push
pop [#push > #pop]

Programming Techniques

S. L. Graham, R. L. Rivest Editors

Communicating Sequential Processes

C.A.R. Hoare
The Queen's University
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

CR Categories: 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPHARD [19]), actors (Hewitt [1]).

The traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program. Where the desire for greater speed has led to the introduction of parallelism, every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 6600) or by the software (as in an I/O control package, or a multiprogrammed operating system). However, developments of processor technology suggest that a multiprocessor machine, constructed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in Algol 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and

This form of failure is known as a deadlock.

Programming Techniques — S. L. Graham, R. L. Rivest, Editors

Communicating Sequential Processes

C.A.R. Hoare
The Queen's University
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of [illegible] sequential processes [illegible] program structuring method. [illegible] with a development of [illegible] guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

CR Categories: 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPHARD [19]), actors (Hewitt [1]).

The traditional stored program digital computer has [illegible] of a [illegible] greater [illegible] every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 66[illegible]) [illegible] by the software [illegible] [illegible] structed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in Algol 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and

```go
func Stack(push <-chan string, pop chan<- string) {
	var items [] string
	for {
		if depth := len(items); depth == 0 {
			items = append(items, <-push)
		} else {
			select {
			case newTop := <-push:
				items = append(items, newTop)
			case pop<- items[depth - 1]:
				items = items[:depth - 1]
			}
		}
	}
}
```

To iterate is human,
to recurse divine.

L Peter Deutsch

```go
func Stack(push <-chan string, pop chan<- string) {
    for {
        nonEmptyStack(push, pop, <-push)
    }
}
func nonEmptyStack(push <-chan string, pop chan<- string, top string)
{
    for {
        select {
        case newTop := <-push:
            nonEmptyStack(push, pop, newTop)
        case pop<- top:
            return
        }
    }
}
```

Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W Dijkstra

Notes on Structured Programming

```go
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
case _ = <-pop:
    test.Errorf(”empty stack can never be popped”)
case <-time.After(???):
}
```

```
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
case _ = <-pop:
    test.Errorf("empty stack can never be popped")
case <-time.After(time.Eternity):
}
```

```
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
case _ = <-pop:
    test.Errorf("empty stack can never be popped")
case <-time.After(time.Second):
}
```

Halting Problem

Halting Problem

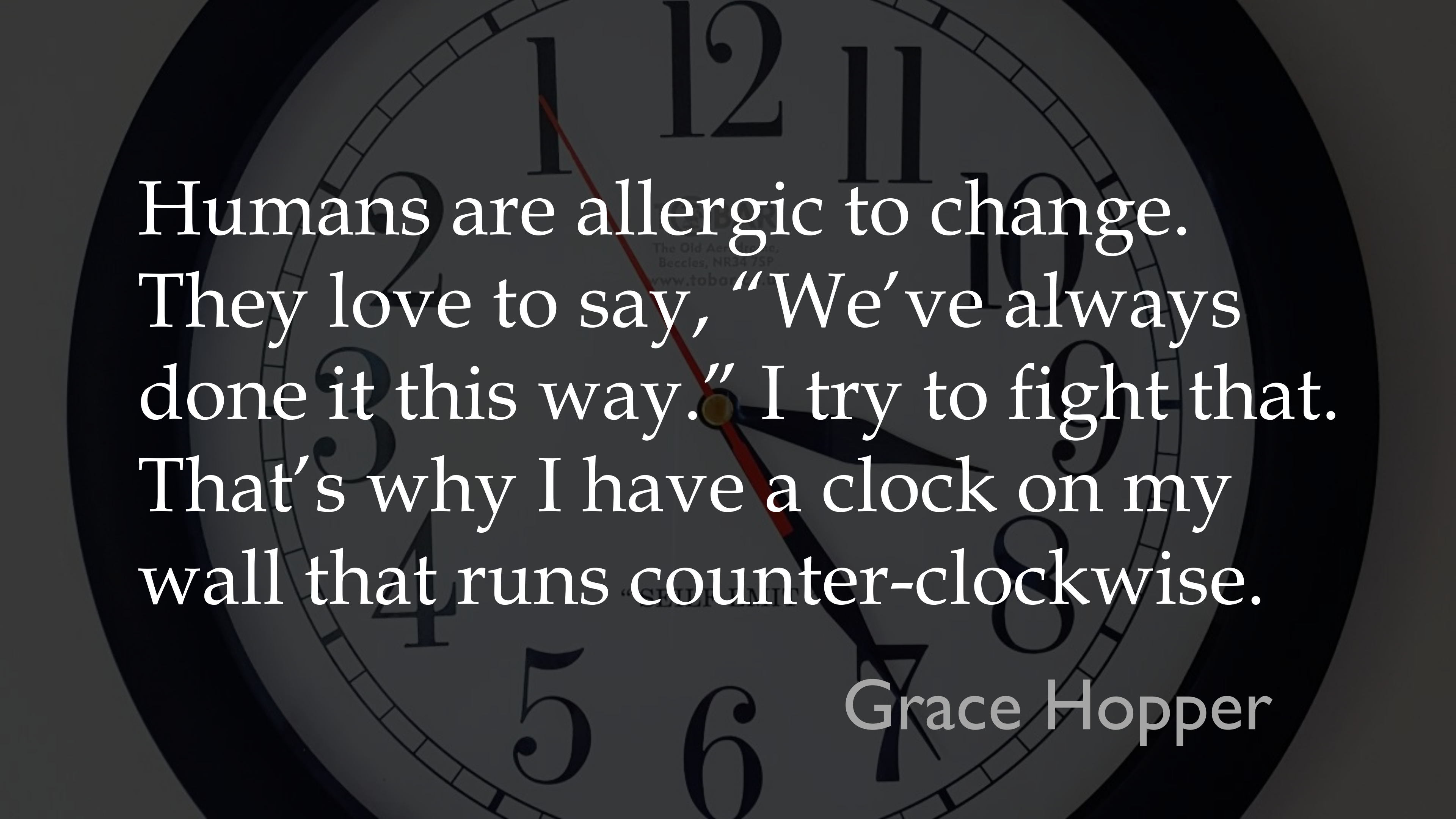
Prediction is very difficult, especially about the future.

Niels Bohr?

prioritise by business value


POLICE
PUBLIC CALL
BOX
FREE
PUBLIC
SPACE

prioritise by estimated business value


Humans are allergic to change.
They love to say, "We've always
done it this way." I try to fight that.
That's why I have a clock on my
wall that runs counter-clockwise.
Grace Hopper


TOBAR
The Old Aerodrome,
Beccles, NR34 7SP
www.tobar.co.uk
“ SEILF EMIT ”

Kevlin Henney
@KevlinHenney

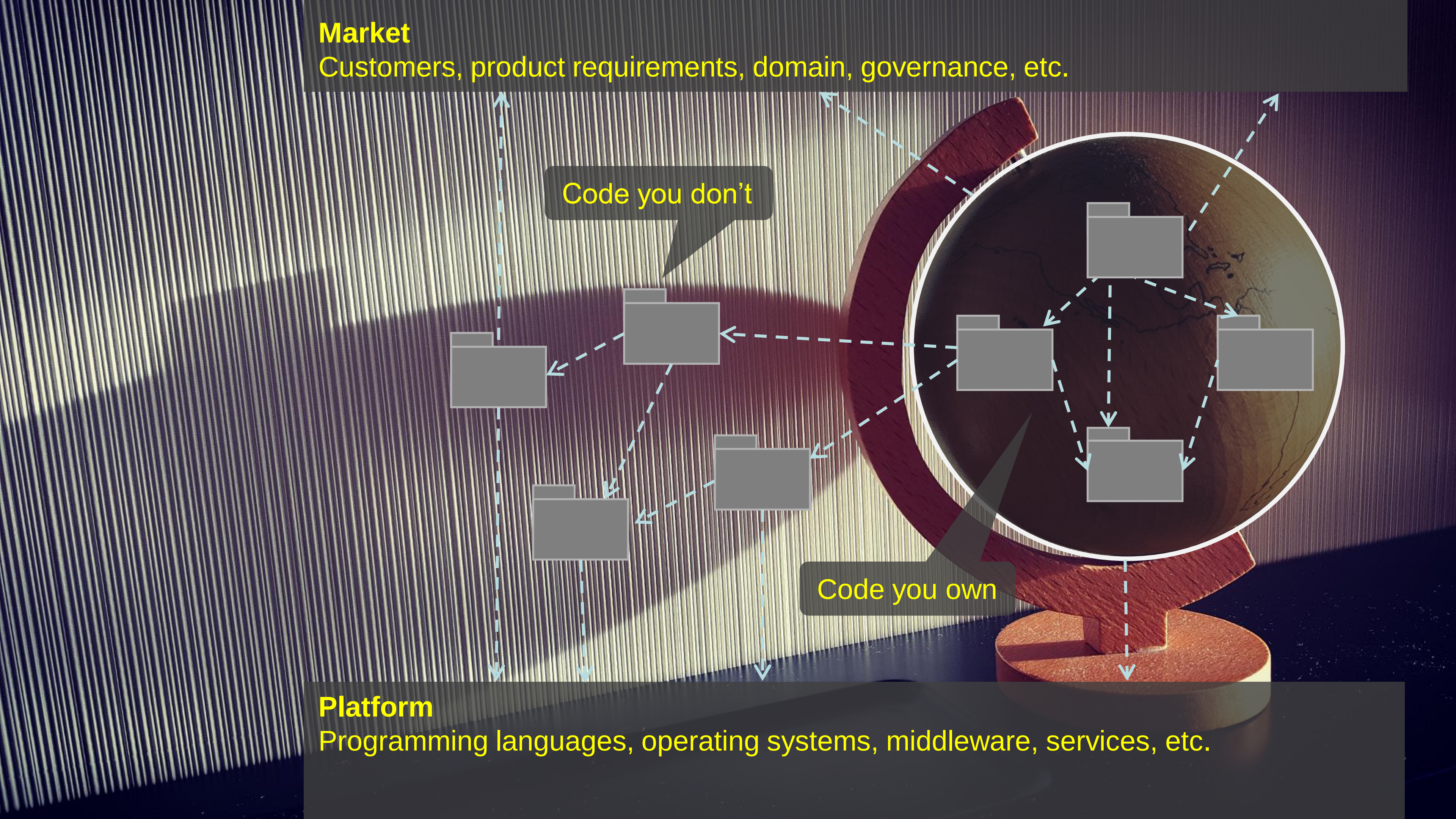
Epistemologically speaking, assumptions are the barefoot-trodden Lego bricks in the dark of knowledge. You don't know they're there until you know that they're there. And even if you know there are some there, you don't know exactly where and you'll still end up stepping on some.

26 2:29 PM - Apr 22, 2020

twitter.com/KevlinHenney/status/1252952622128128000

The connections between modules are the assumptions which the modules make about each other.

David Parnas


Market
Customers, product requirements, domain, governance, etc.
Code you don’t
Code you own
Platform
Programming languages, operating systems, middleware, services, etc.

It’s often not the direct dependencies of your project that bite you, but the dependencies of your dependencies, all the way on down to transitive closure.

Adrian Colyer

blog.acolyer.org/2020/09/21/watchman/

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

Careful, careful ... Don't fumble this like the JS world (Credit: Claus Rebler)

23 Mar 2016 at 01:24, Chris Williams

1322

Updated Programmers were left staring at broken builds and failed installations on Tuesday after someone toppled the Jenga tower of JavaScript.

A couple of hours ago, Azer Koçulu unpublished more than 250 of his modules from NPM, which is a popular package manager used by JavaScript projects to install dependencies.

www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/

When we try to pick out anything by itself, we find it hitched to everything else in the universe.

John Muir

It Depends...