What Do You Mean?

@KevlinHenney

WTF Do You Mean?

@KevlinHenney

The difficulty of literature is not to write, but to write what you mean; not to affect your reader, but to affect him precisely as you wish.

Robert Louis Stevenson
“Truth of Intercourse”

Any program is a model of a model within a theory of a model of an abstraction of some portion of the world or of some universe of discourse.

Meir M Lehman

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

> The purpose of abstraction is *not* to be vague, but to create a new semantic level in which one can be absolutely precise.

Edsger W Dijkstra
“The Humble Programmer”

It’s just semantics.

It’s just meaning.

software

system of
meaning

code

tests

scripts

codified knowledge

knowledge acquisition

learning

communication

social negotiation

model of participation

software architecture

design

synthesis

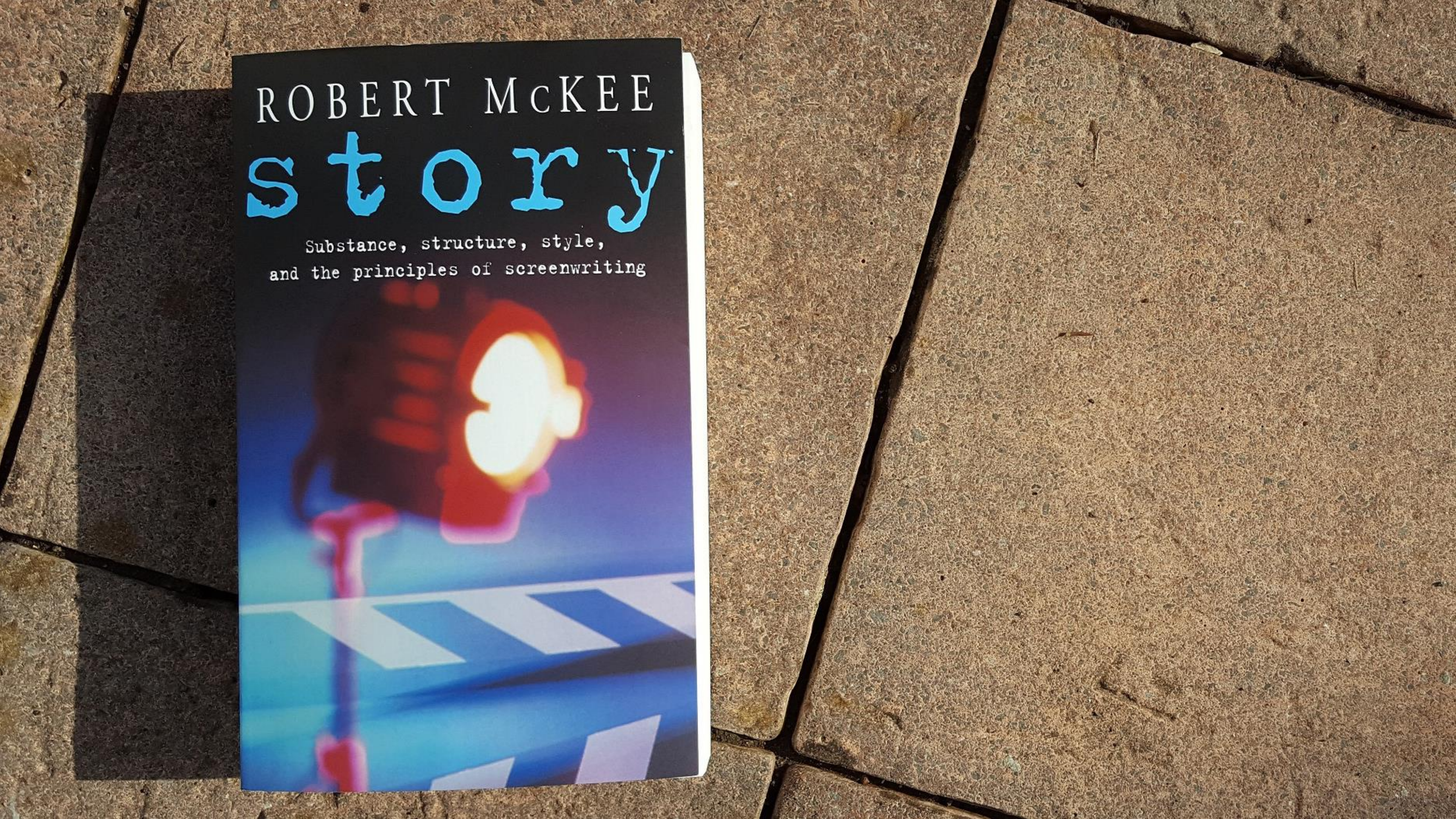
analysis

systole

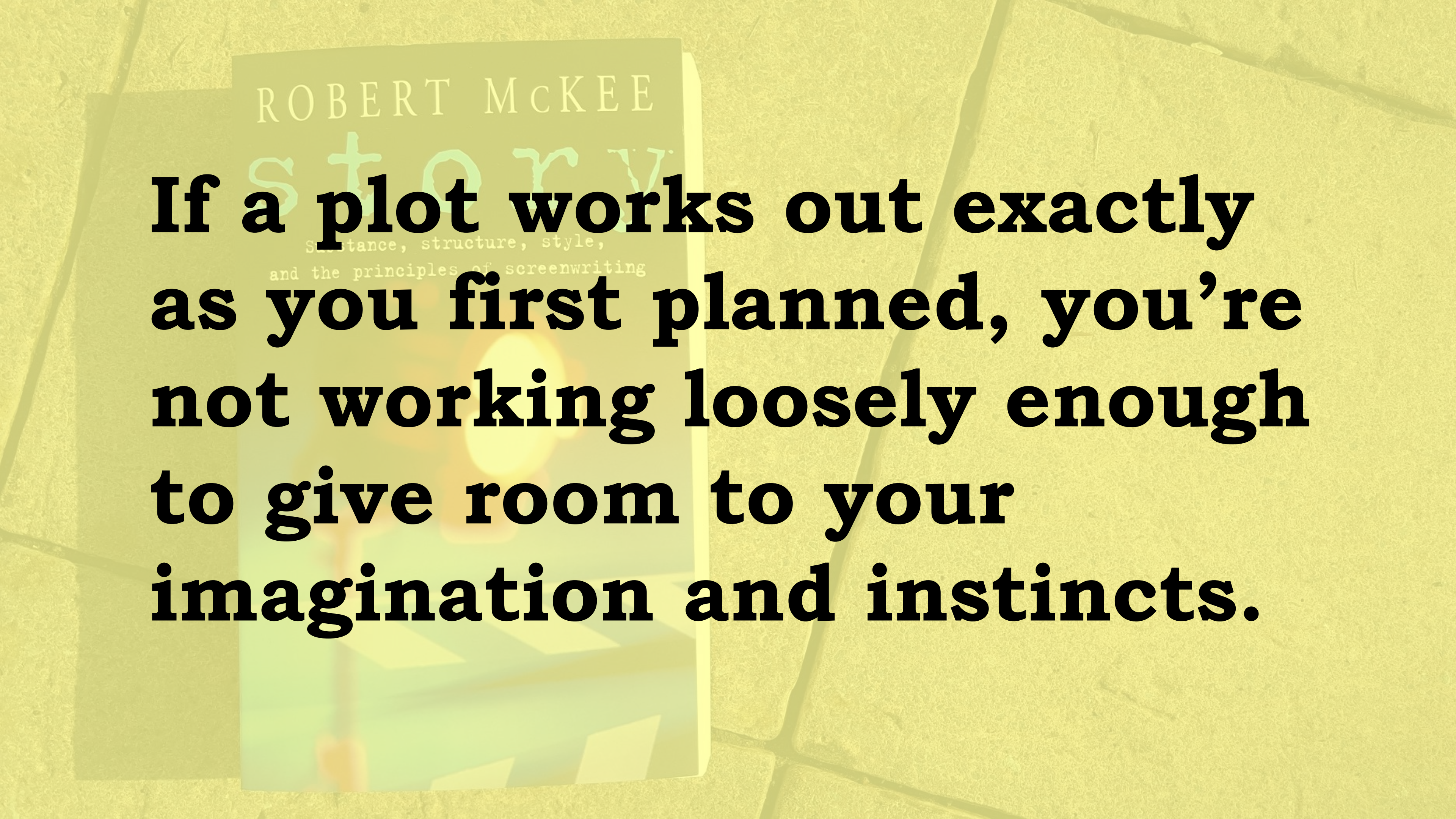
diastole

The only kind of writing is rewriting.

Ernest Hemingway



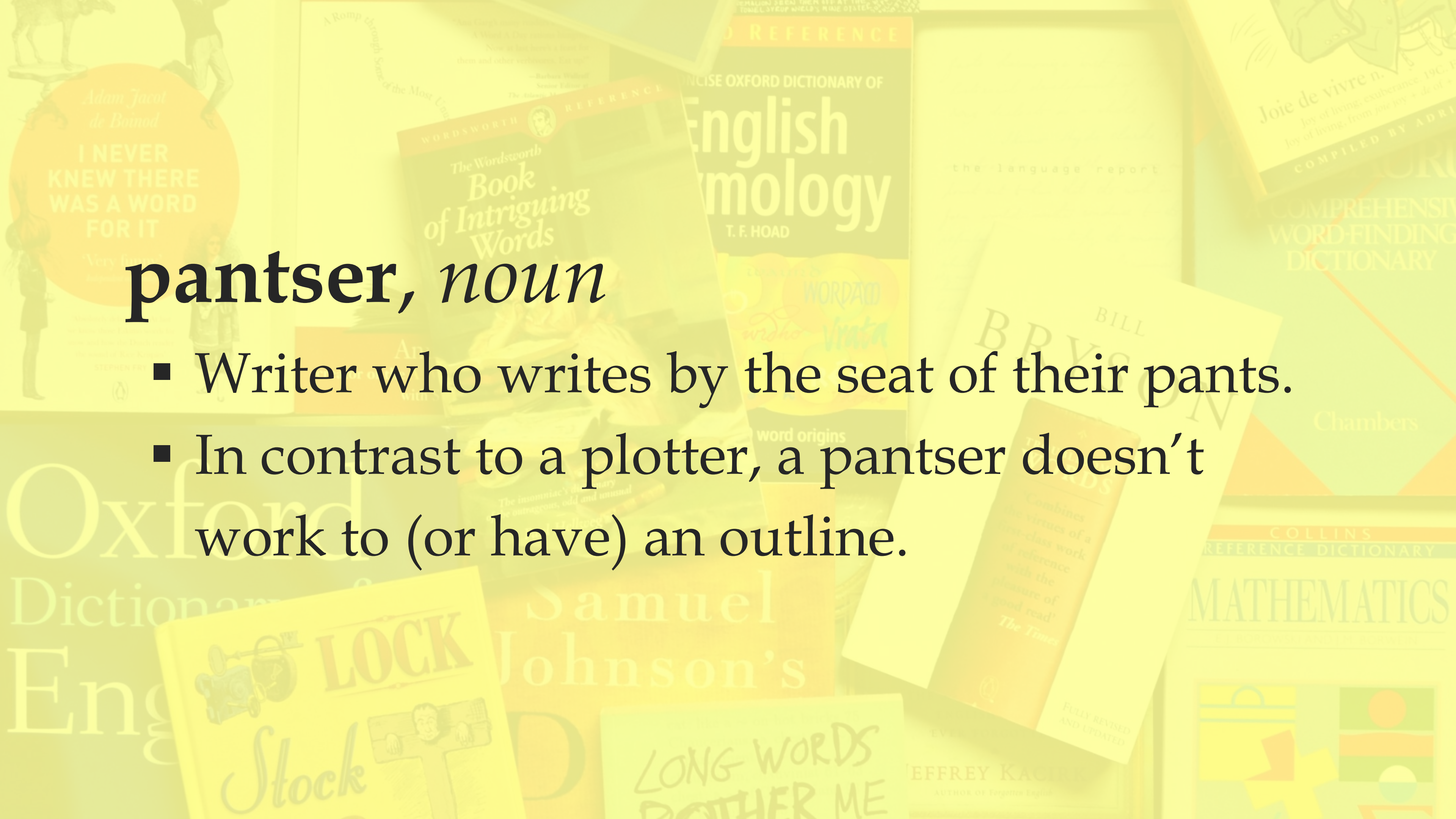
ROBERT McKEE
story
Substance, structure, style,
and the principles of screenwriting


ROBERT McKEE
story
Substance, structure, style,
and the principles of screenwriting

If a plot works out exactly as you first planned, you’re not working loosely enough to give room to your imagination and instincts.

I NEVER KNEW THERE WAS A WORD FOR IT
Adam Jacot de Boinod
The Wordsworth Book of Intriguing Words
Paul Hellweg
Concise Oxford Dictionary of English Etymology
T. F. HOAD
the language report
Bill Bryson
Troublesome Words
A Comprehensive Word-Finding Dictionary
Chambers
Oxford Dictionary of English
Lock Stock
Samuel Johnson's
Collins Reference Dictionary Mathematics
E. J. Borowski and J. M. Borwein
Long Words Bother Me
Jeffrey Kacirk

/ WordFriday



pantser, *noun*

- Writer who writes by the seat of their pants.
- In contrast to a plotter, a pantser doesn’t work to (or have) an outline.

pants

thongs

language

programming

natural

algorithm

algorithm, *noun*

- a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer

Concise Oxford English Dictionary

procedure

The main difference is that the **procedure** can halt or need not halt. But the **algorithm** always halts and gives you the output.

https://www.quora.com/What-is-the-difference-between-an-algorithm-and-a-procedure algorithm

algorism

algorisme

algorismus

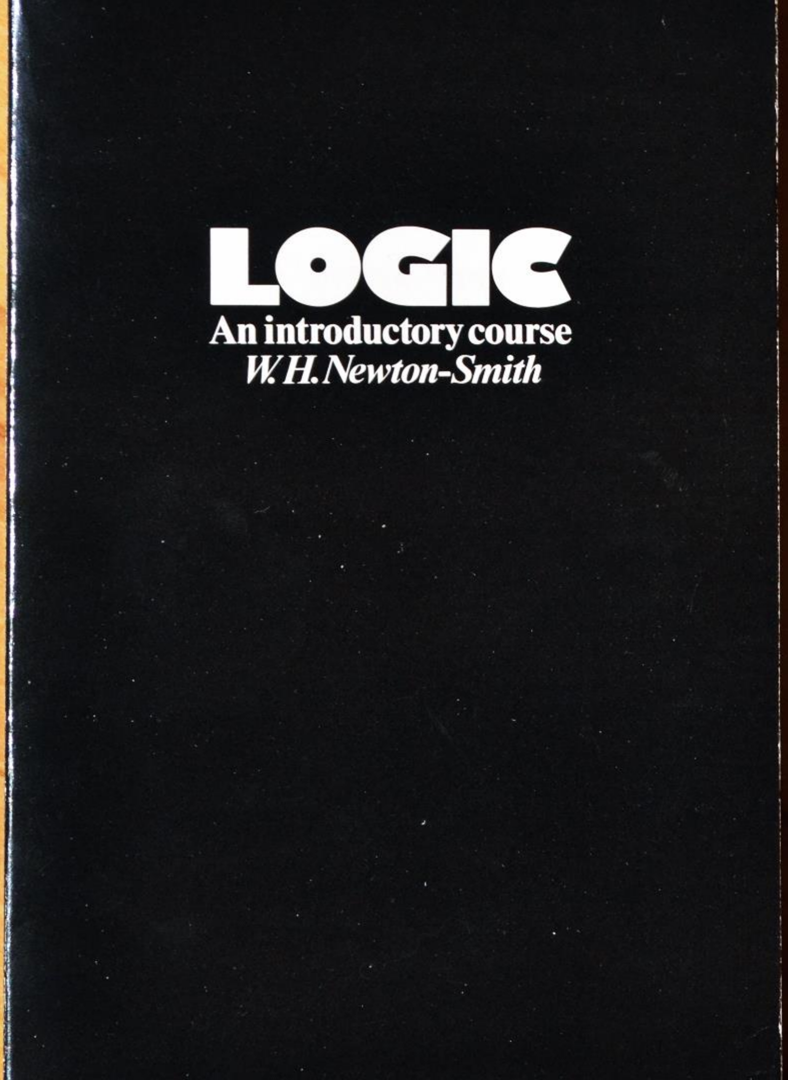
الــخوارزمــي

خوارزمی

algorithm

<algorithm>

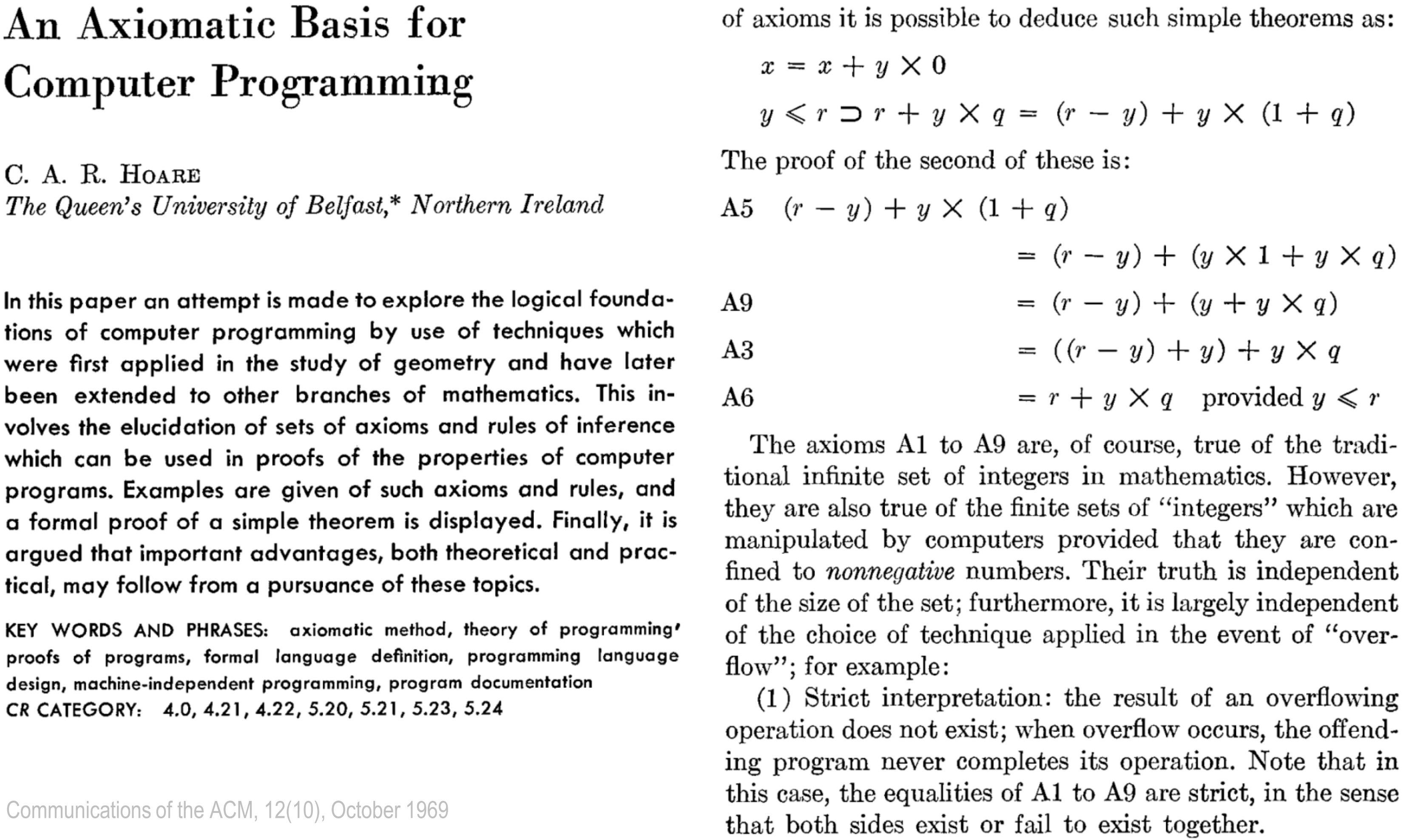
```
std::sort
```



LOGIC

An introductory course

W. H. Newton-Smith



An Axiomatic Basis for Computer Programming

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

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

Communications of the ACM, 12(10), October 1969

of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

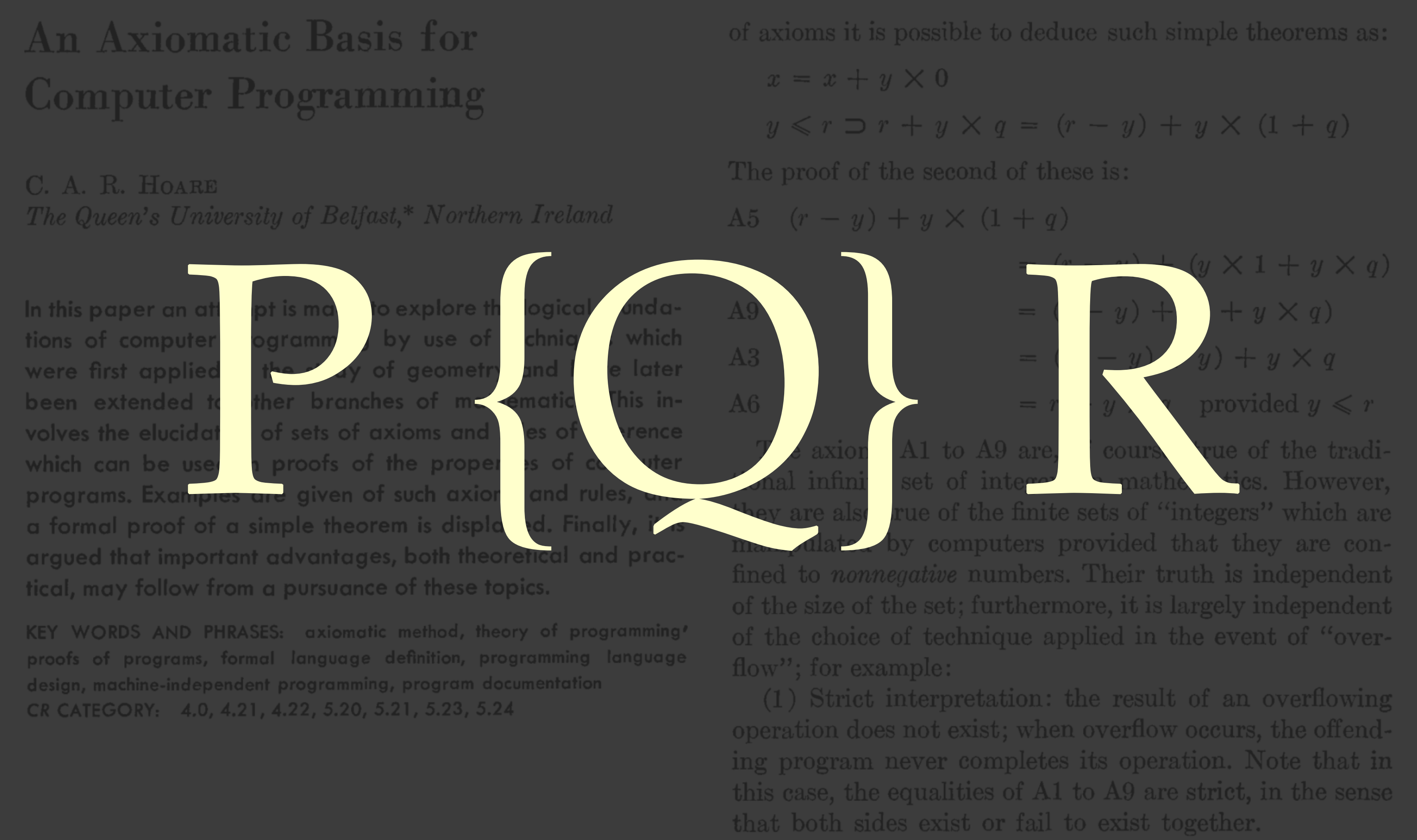
The proof of the second of these is:

$$\begin{array}{ll}
\text{A5} & (r - y) + y \times (1 + q) \\
 & \qquad = (r - y) + (y \times 1 + y \times q) \\
\text{A9} & \qquad = (r - y) + (y + y \times q) \\
\text{A3} & \qquad = ((r - y) + y) + y \times q \\
\text{A6} & \qquad = r + y \times q \quad \text{provided } y \leqslant r
\end{array}$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

P{Q} R



An Axiomatic Basis for Computer Programming

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

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

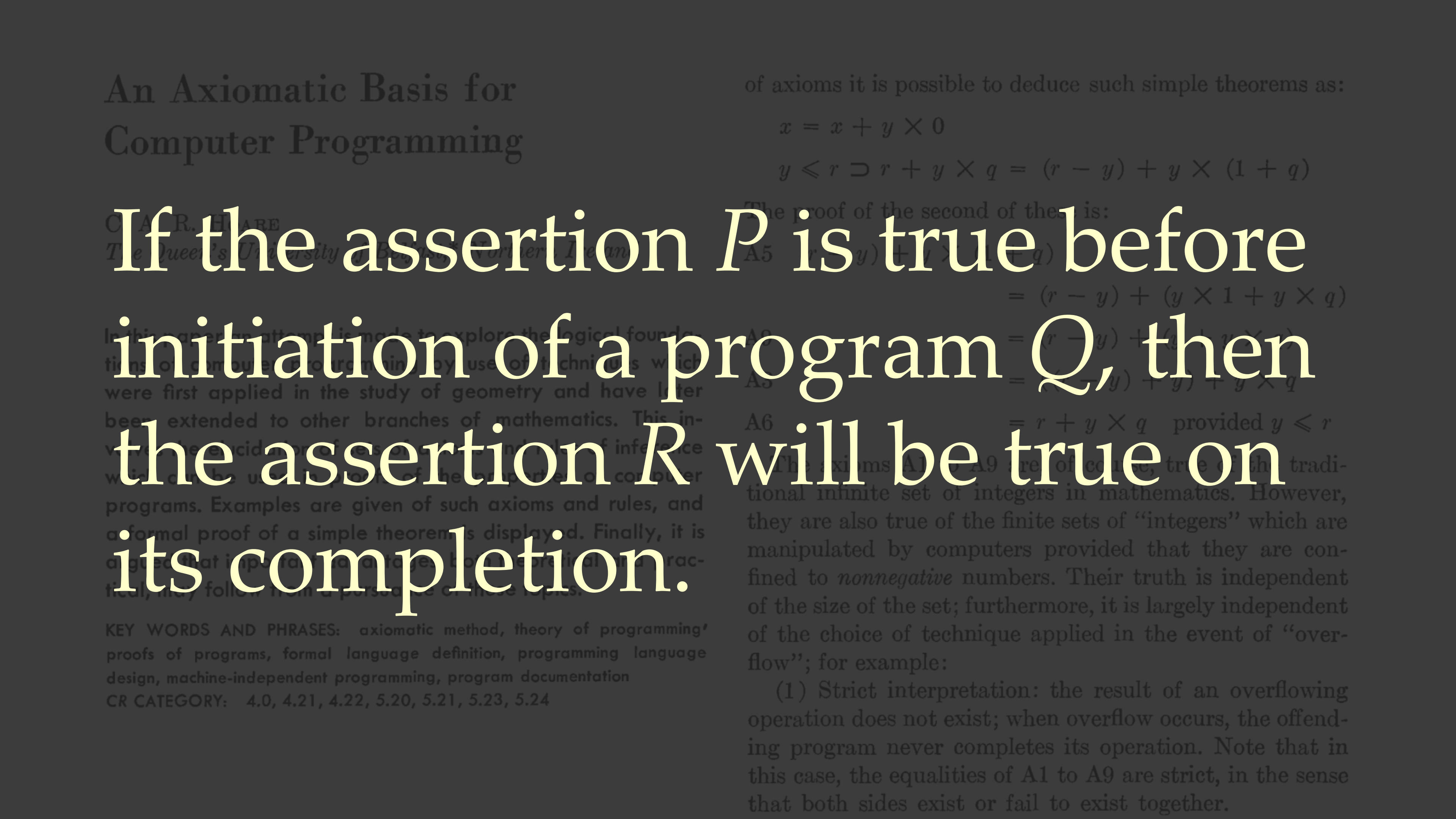
The proof of the second of these is:

A5	$(r - y) + y \times (1 + q)$
	$= (r - y) + (y \times 1 + y \times q)$
A9	$= (r - y) + (y + y \times q)$
A3	$= ((r - y) + y) + y \times q$
A6	$= r + y \times q$ provided $y \leqslant r$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

If the assertion *P* is true before initiation of a program *Q*, then the assertion *R* will be true on its completion.



An Axiomatic Basis for Computer Programming

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

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

The proof of the second of these is:

$$
\begin{array}{ll}
\text{A5} & (r - y) + y \times (1 + q) \\
 & = (r - y) + (y \times 1 + y \times q) \\
\text{A9} & = (r - y) + (y + y \times q) \\
\text{A3} & = ((r - y) + y) + y \times q \\
\text{A6} & = r + y \times q \quad \text{provided } y \leqslant r
\end{array}
$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

$$\{P\}\ Q\ \{R\}$$

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // post: is_sorted(begin, end)
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

An Axiomatic Basis for Computer Programming

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

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

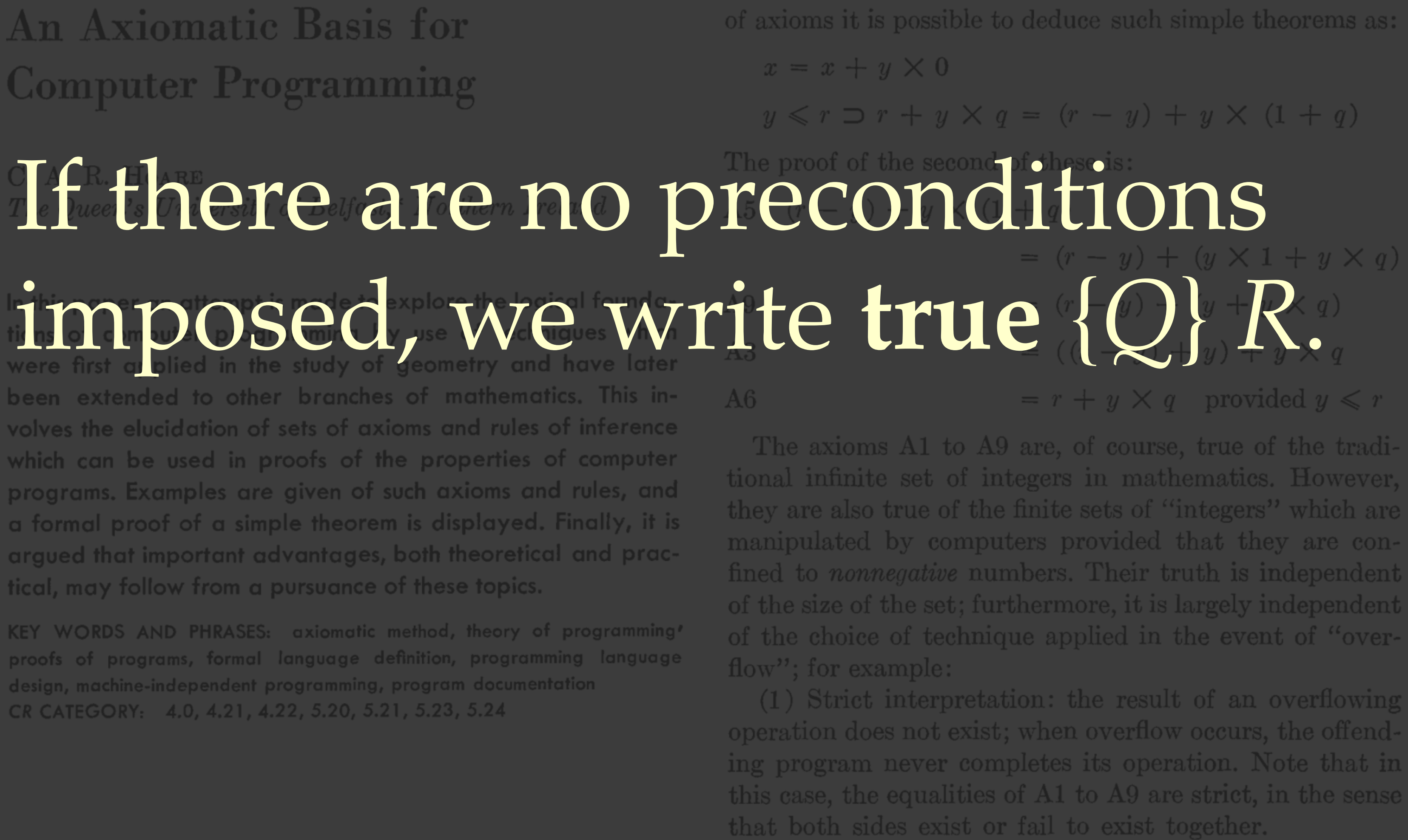
The proof of the second of these is:

$$\begin{array}{ll}
\text{A5} & (r - y) + y \times (1 + q) \\
 & \qquad = (r - y) + (y \times 1 + y \times q) \\
\text{A9} & \qquad = (r - y) + (y + y \times q) \\
\text{A3} & \qquad = ((r - y) + y) + y \times q \\
\text{A6} & \qquad = r + y \times q \quad \text{provided } y \leqslant r
\end{array}$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

If there are no preconditions imposed, we write **true** $\{Q\}\,R$.



An Axiomatic Basis for Computer Programming

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

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

The proof of the second of these is:

A5	$(r - y) + y \times (1 + q)$
	$= (r - y) + (y \times 1 + y \times q)$
A9	$= (r - y) + (y + y \times q)$
A3	$= ((r - y) + y) + y \times q$
A6	$= r + y \times q$ provided $y \leqslant r$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: true
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: begin and end are valid iterators
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: begin and end are valid iterators
    //   from the same range
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: begin and end are valid iterators
    //     from the same range and begin does not
    //     follow end
    // post: is_sorted(begin, end) and
    //     the values from the resulting range are
    //     a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: end is reachable from begin
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end);
    // pre: end is reachable from begin
    // post: is_sorted(begin, end) and
    //   the values from the resulting range are
    //   a permutation of the original values
```

```
template<typename Iterator>
  void sort(Iterator begin, Iterator end)

    [[ post: is_sorted(begin, end) ]];
```

`std::sort`

```
std::qsort
```

```
std::vector<int> values {3, 1, 4, 1, 5, 9};
const std::vector<int> sorted {1, 1, 3, 4, 5, 9};



std::sort(values.begin(), values.end());
assert(values == sorted);
```

algorithm?

$$O(n \log n)$$

$$O(n^2)$$

```
std::vector<int> values {3, 1, 4, 1, 5, 9};
const std::vector<int> sorted {1, 1, 3, 4, 5, 9};







permutation_sort(values.begin(), values.end());
assert(values == sorted);
```

```
std::vector<int> values {3, 1, 4, 1, 5, 9};
const std::vector<int> sorted {1, 1, 3, 4, 5, 9};
template<typename Iterator>
void permutation_sort(Iterator begin, Iterator end)
{
    while (std::next_permutation(begin, end))
        ;
}
permutation_sort(values.begin(), values.end());
assert(values == sorted);
```

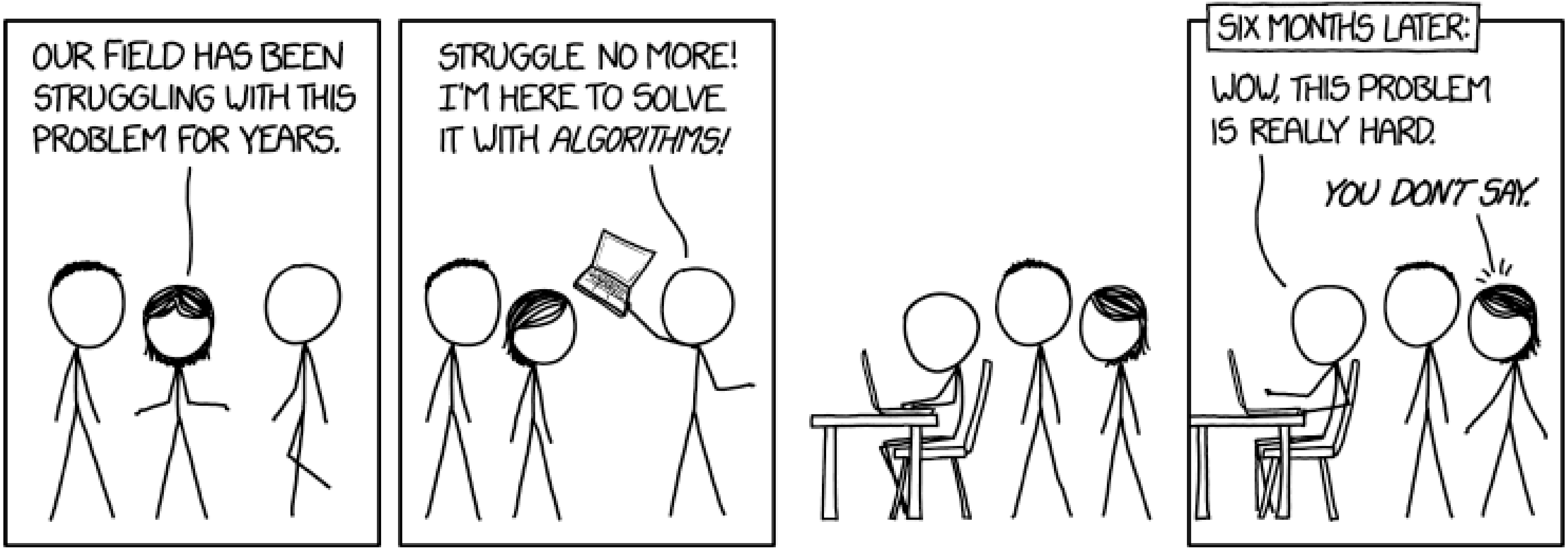
$$O(n!)$$

```
std::vector<int> values {3, 1, 4, 1, 5, 9};
const std::vector<int> sorted {1, 1, 3, 4, 5, 9};
template<typename Iterator>
void bogosort(Iterator begin, Iterator end)
{
    while (!std::is_sorted(begin, end))
        std::random_shuffle(begin, end);
}
bogosort(values.begin(), values.end());
assert(values == sorted);
```

OMG!

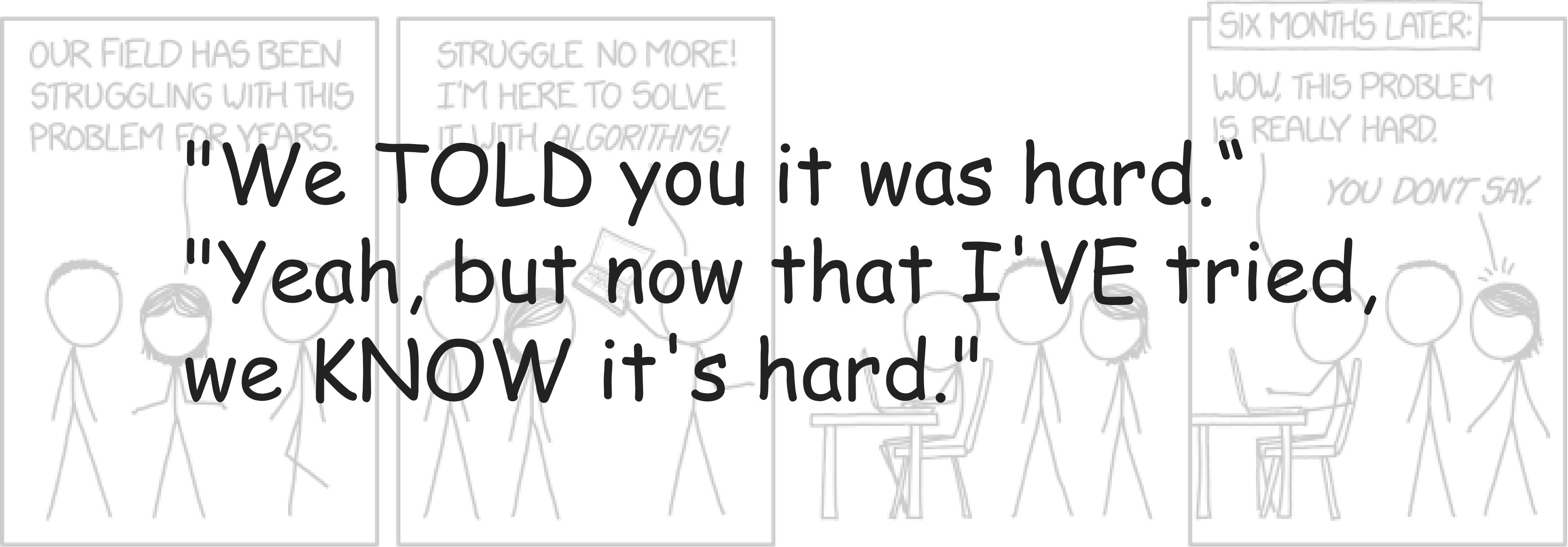
```
$ cat > sleepsort
while [ -n "$1" ]
do
  (sleep $1; echo $1) &
  shift
done
wait
$ chmod +x sleepsort
$ ./sleepsort 3 1 4 1 5 9
1
1
3
4
5
9
```

$$\mathrm{O}(n)$$


OUR FIELD HAS BEEN STRUGGLING WITH THIS PROBLEM FOR YEARS.
STRUGGLE NO MORE! I'M HERE TO SOLVE IT WITH *ALGORITHMS!*
SIX MONTHS LATER:
WOW, THIS PROBLEM IS REALLY HARD.
YOU DON'T SAY.

https://xkcd.com/1831/

"We TOLD you it was hard."
"Yeah, but now that I'VE tried, we KNOW it's hard."

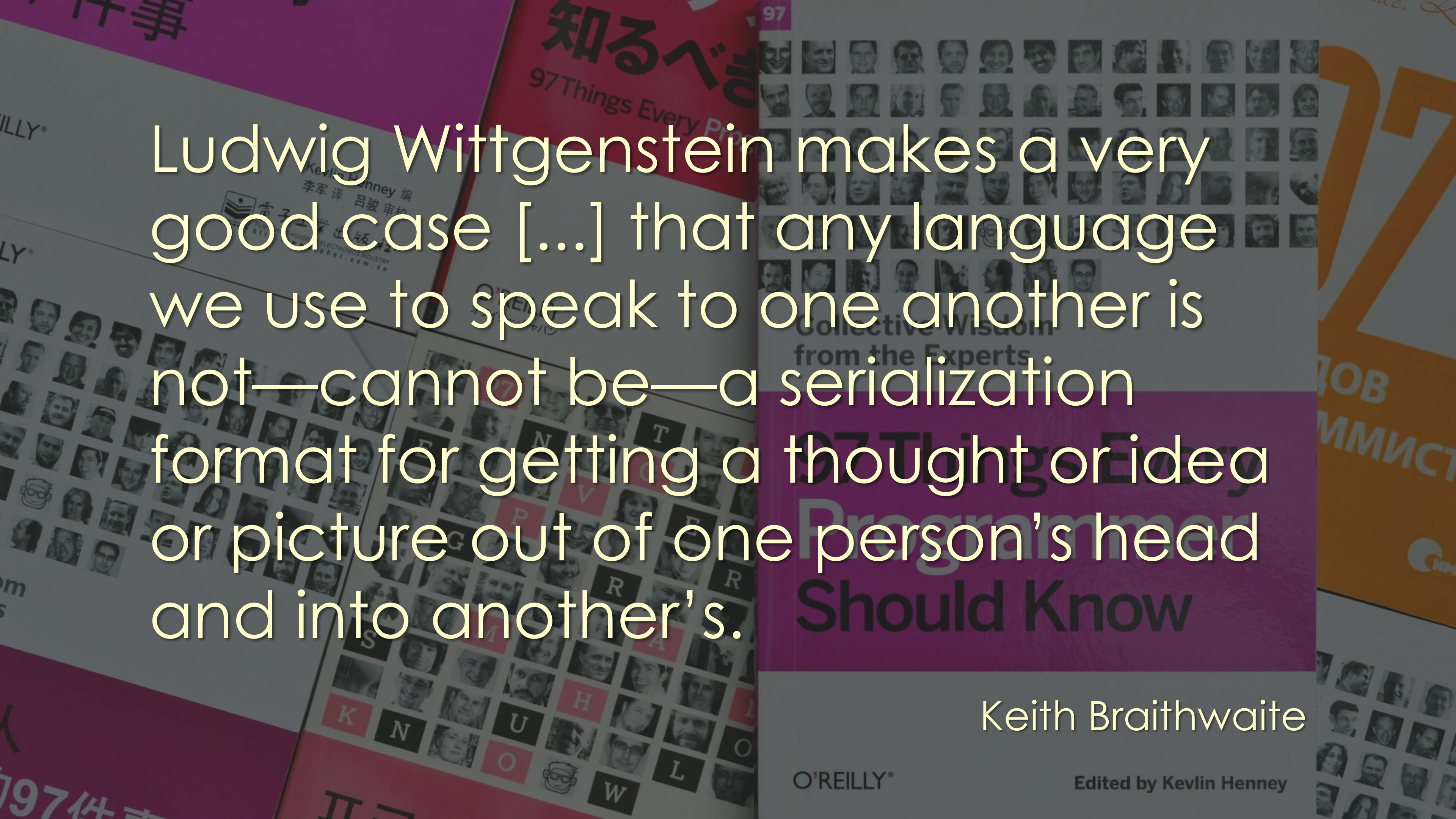

OUR FIELD HAS BEEN STRUGGLING WITH THIS PROBLEM FOR YEARS.
STRUGGLE NO MORE! I'M HERE TO SOLVE IT WITH *ALGORITHMS!*
SIX MONTHS LATER:
WOW, THIS PROBLEM IS REALLY HARD.
YOU DON'T SAY.

https://xkcd.com/1831/

97
Collective Wisdom from the Experts
97 Things Every Programmer Should Know
O'REILLY®
Edited by Kevlin Henney
知るべき
97 Things Every Prog
O'REILLY®
オライリー・ジャパン
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
97
E N T G V P E G R R S M A H K N U O L W O
ДОВ
ММИСТ

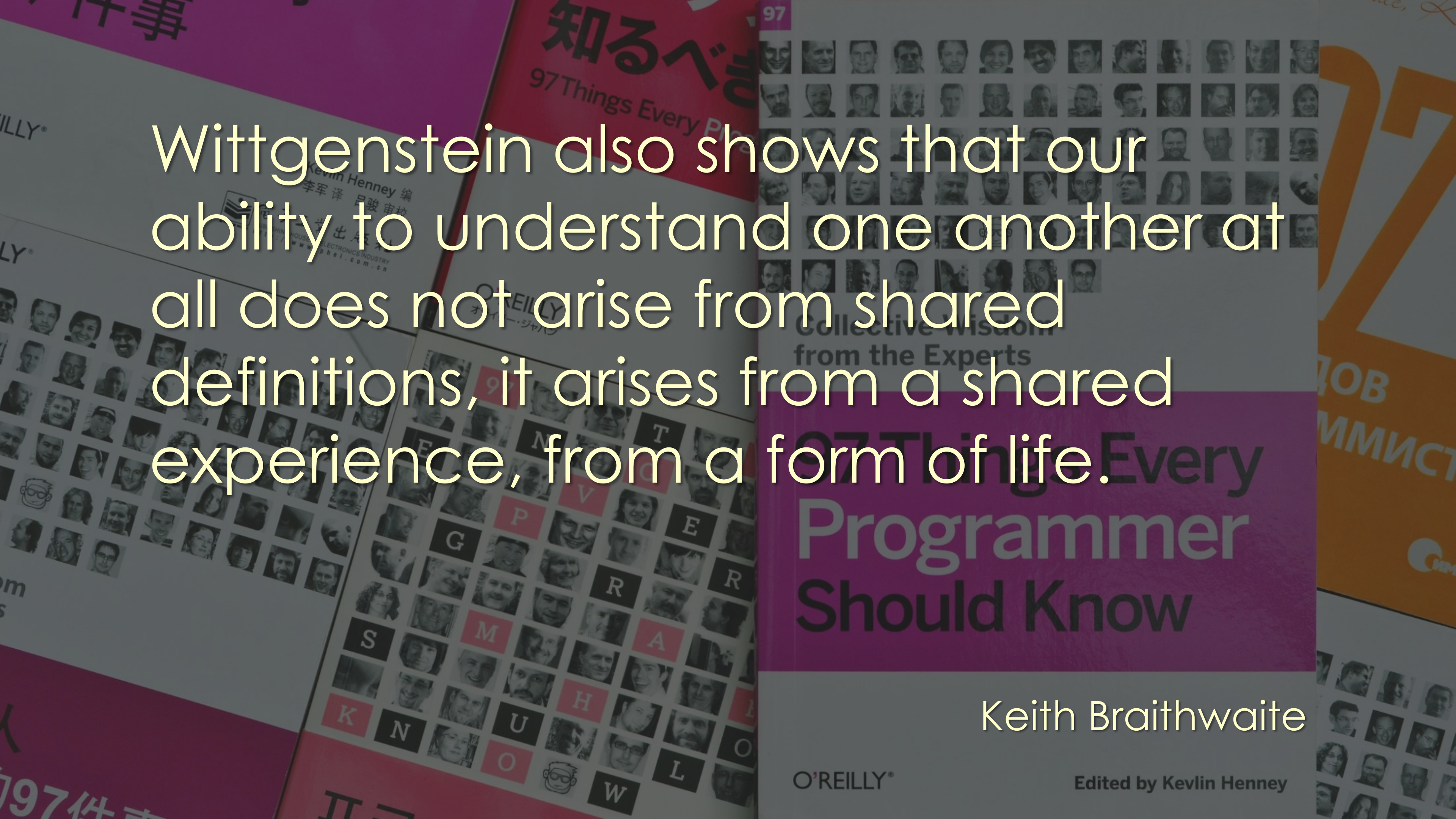
Read the Humanities

Keith Braithwaite


97
97 Things Every Programmer Should Know
Collective Wisdom from the Experts
O'REILLY
Edited by Kevlin Henney

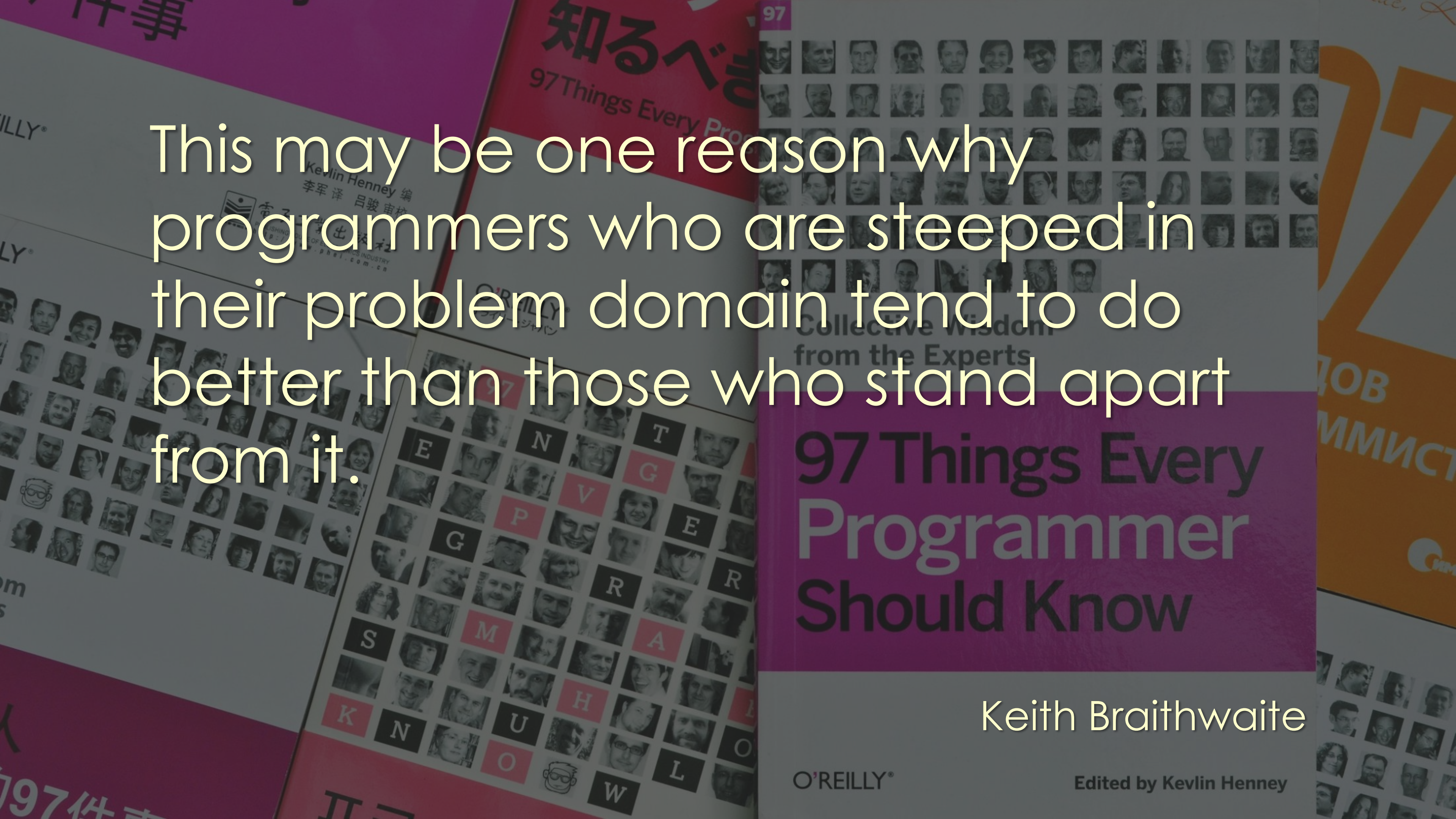
Ludwig Wittgenstein makes a very good case […] that any language we use to speak to one another is not—cannot be—a serialization format for getting a thought or idea or picture out of one person's head and into another's.

Keith Braithwaite


97 Things Every Programmer Should Know
Collective Wisdom from the Experts
O'REILLY
Edited by Kevlin Henney

Wittgenstein also shows that our ability to understand one another at all does not arise from shared definitions, it arises from a shared experience, from a form of life.

Keith Braithwaite



This may be one reason why programmers who are steeped in their problem domain tend to do better than those who stand apart from it.

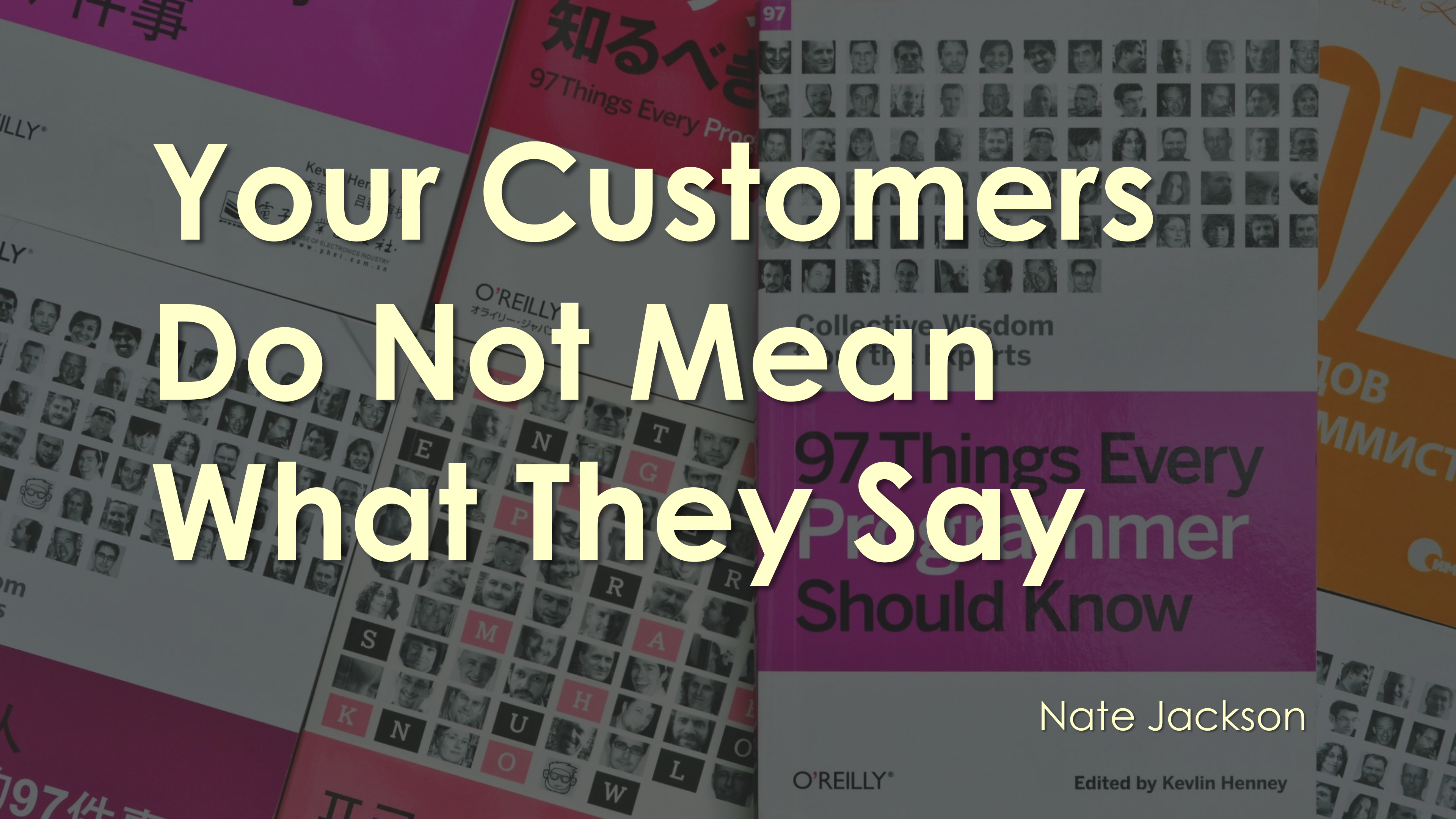
Keith Braithwaite

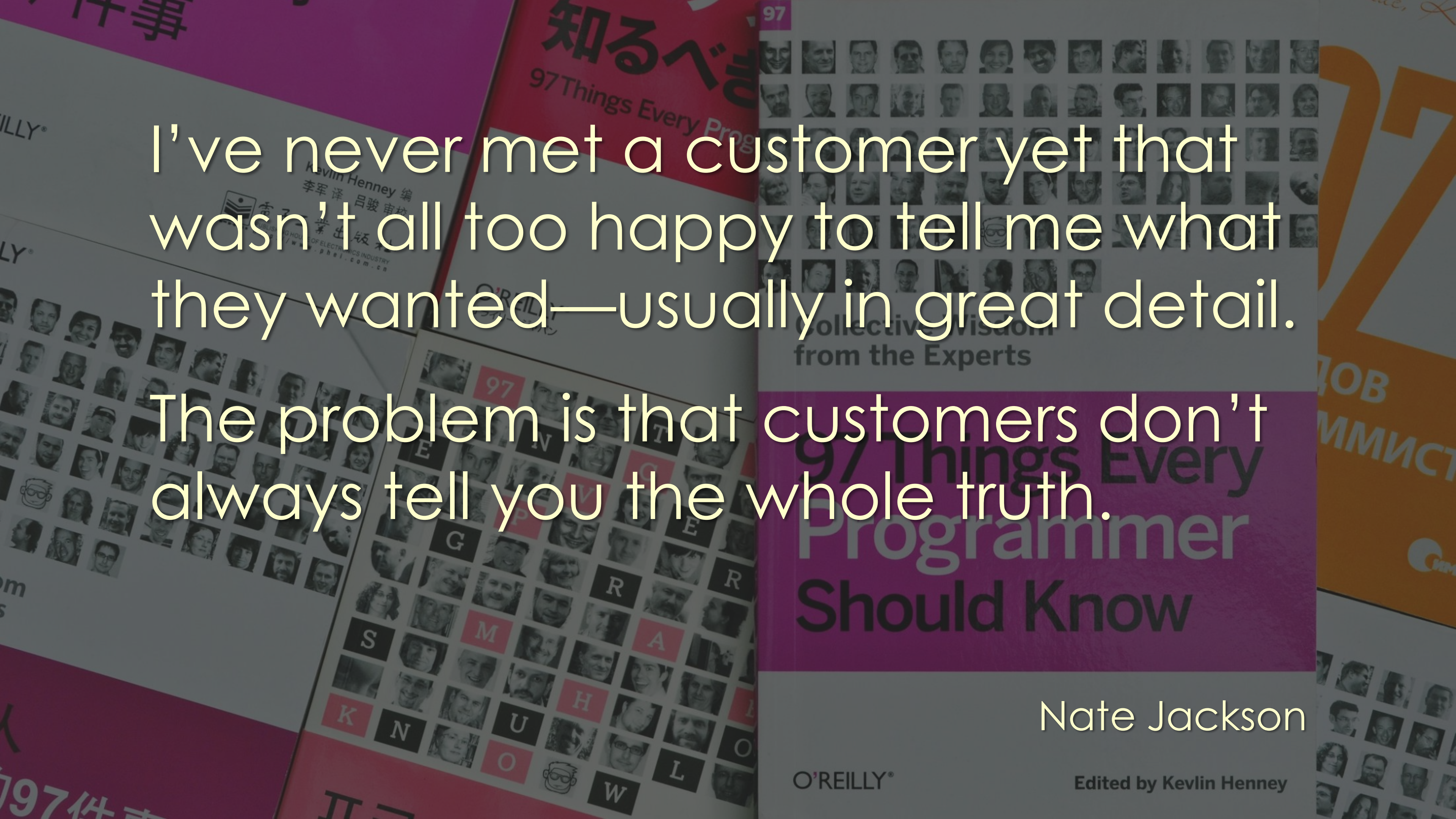
MIND THE GAP

件事
ILLY®
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
LY®
om
s
人
97件事
知るべき
97 Things Every Prog
O'REILLY®
オライリー・ジャパン
97
E N T G V P E R G R S M A K N U H O L W
97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY®
Edited by Kevlin Henney
97
дов
ммист

Your Customers Do Not Mean What They Say

Nate Jackson

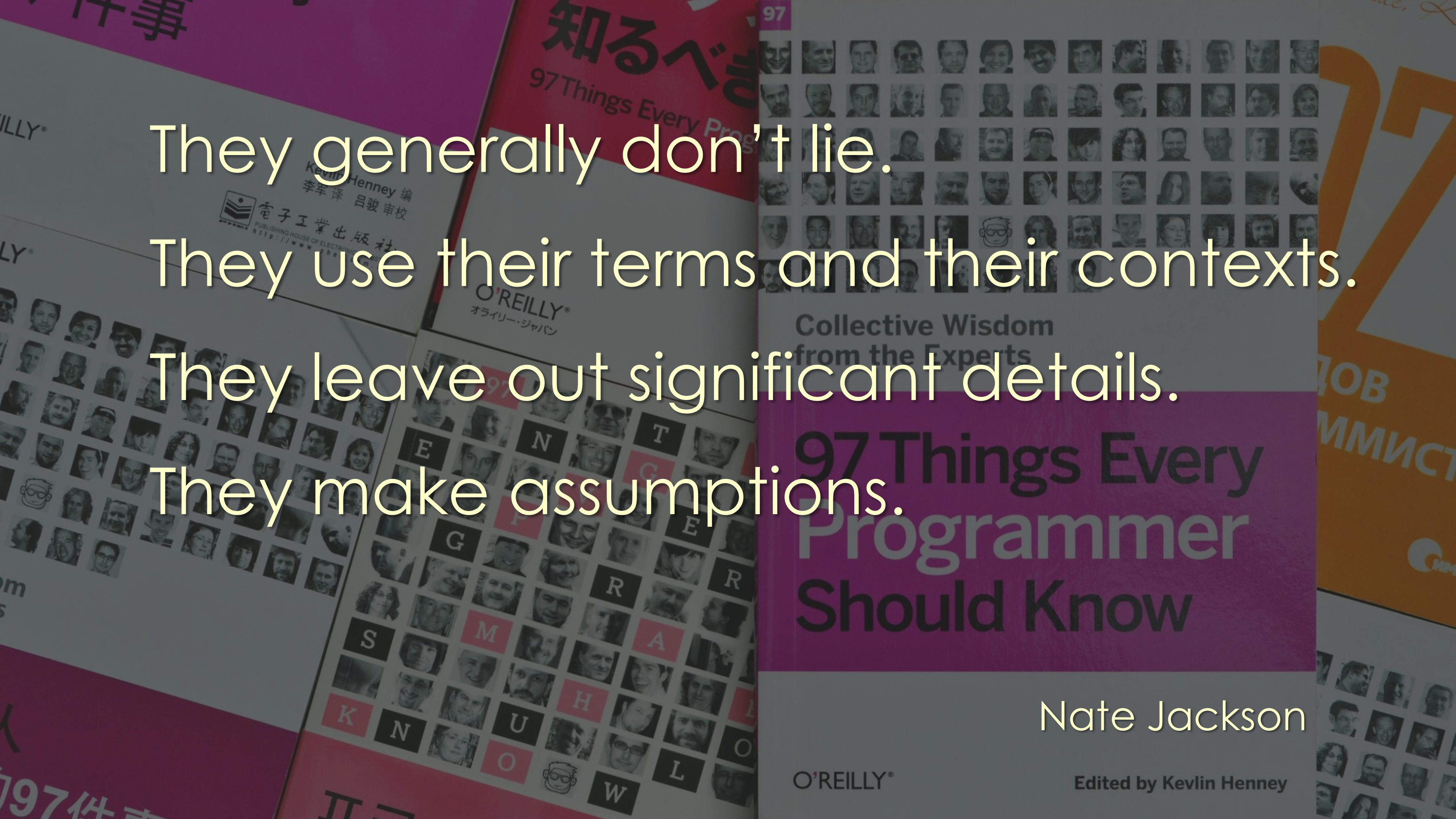




I’ve never met a customer yet that wasn’t all too happy to tell me what they wanted—usually in great detail.

The problem is that customers don’t always tell you the whole truth.

Nate Jackson


97 Things Every Programmer Should Know
Collective Wisdom from the Experts
O'REILLY
Edited by Kevlin Henney

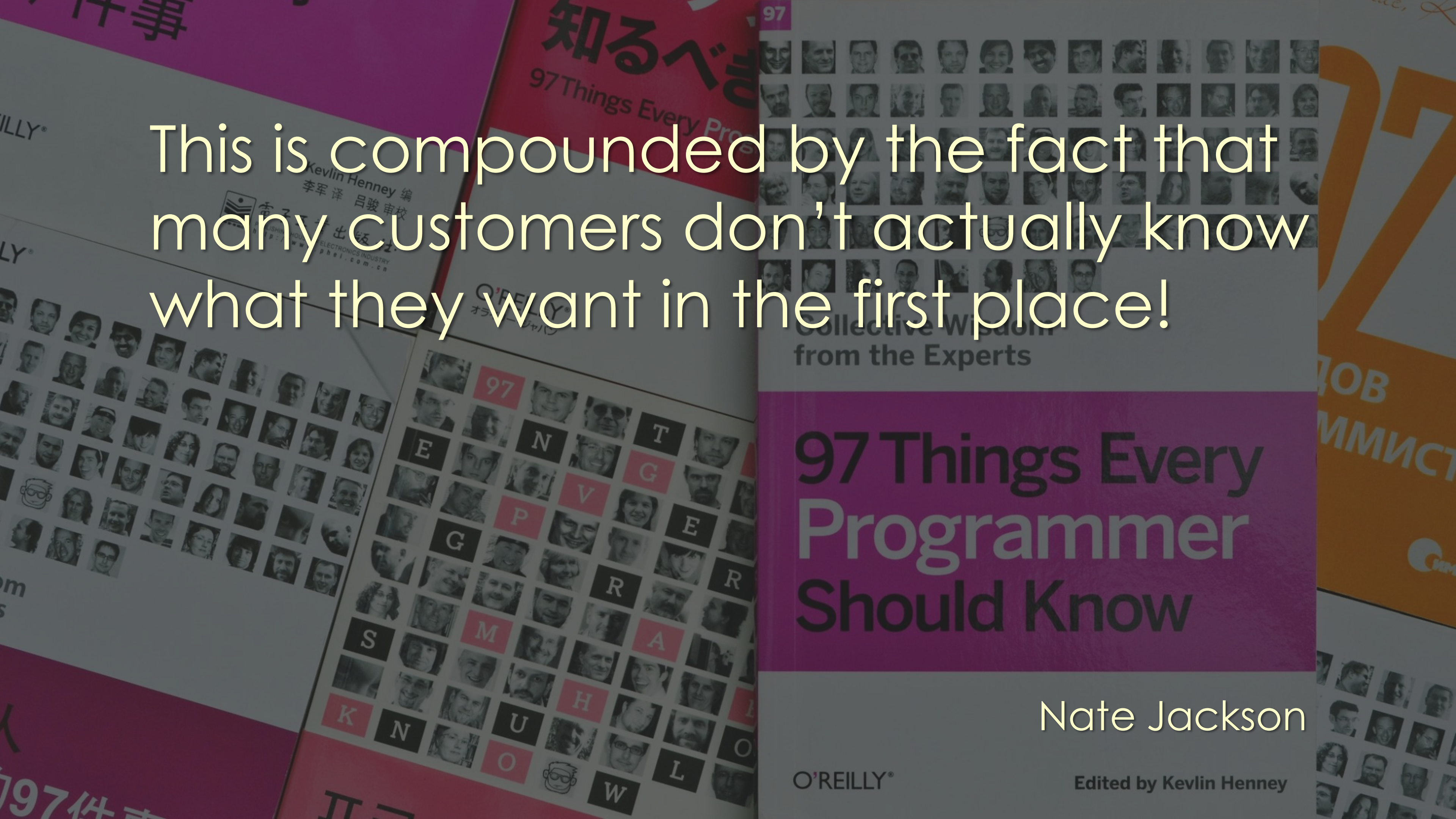
They generally don’t lie.

They use their terms and their contexts.

They leave out significant details.

They make assumptions.

Nate Jackson


97 Things Every Programmer
Collective Wisdom from the Experts
97 Things Every Programmer Should Know
O'REILLY
Edited by Kevlin Henney
Kevlin Henney 编
李军 译 吕骏 审校
知るべき
97 Things Every Pro

This is compounded by the fact that many customers don't actually know what they want in the first place!

Nate Jackson

This is compounded by the fact that many humans don’t actually know what they want in the first place!

You have to finish things — that’s what you learn from, you learn by finishing things.

Neil Gaiman

SOFTWARE ENGINEERING

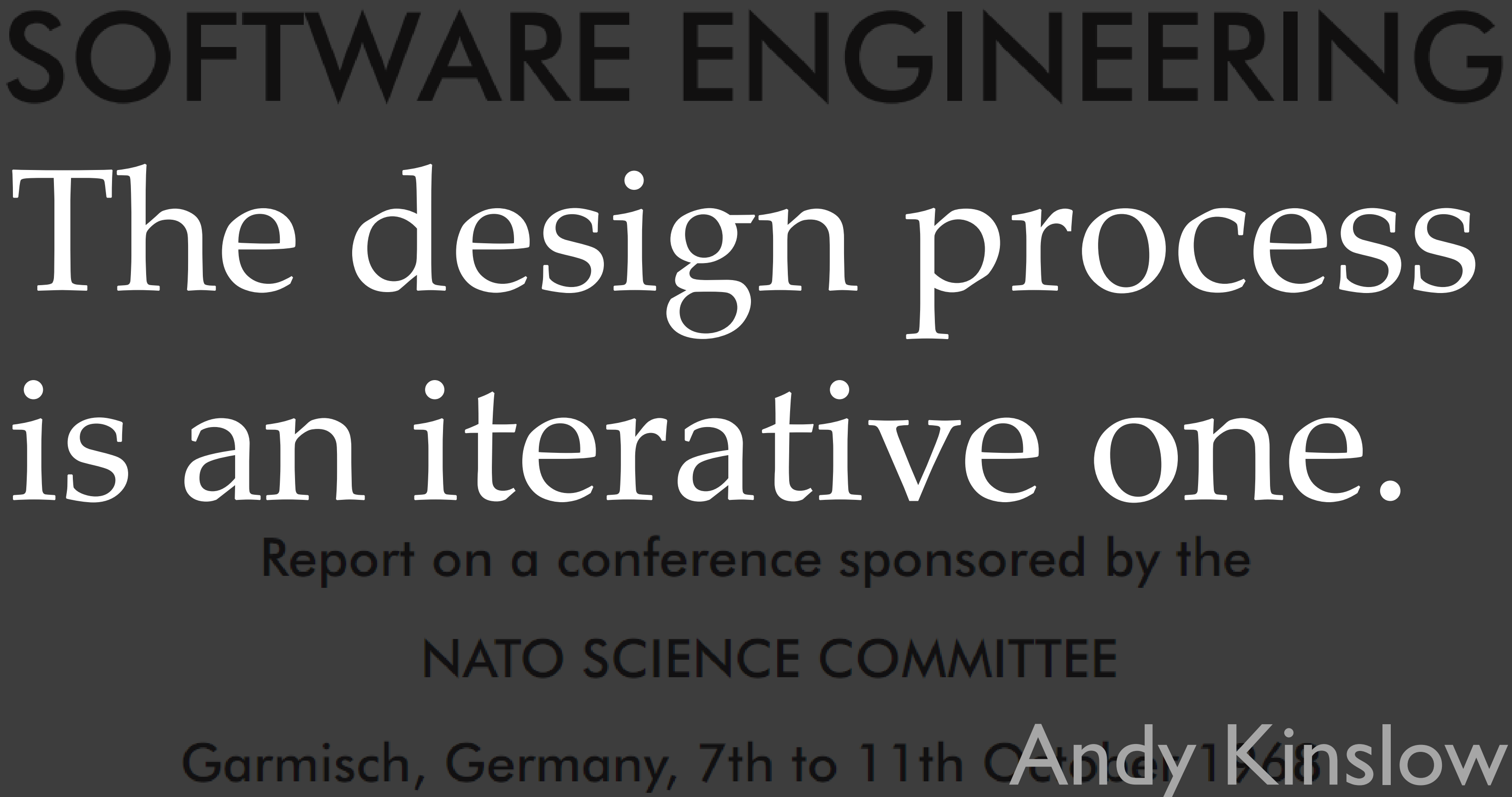
Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

The design process is an iterative one.

Andy Kinslow

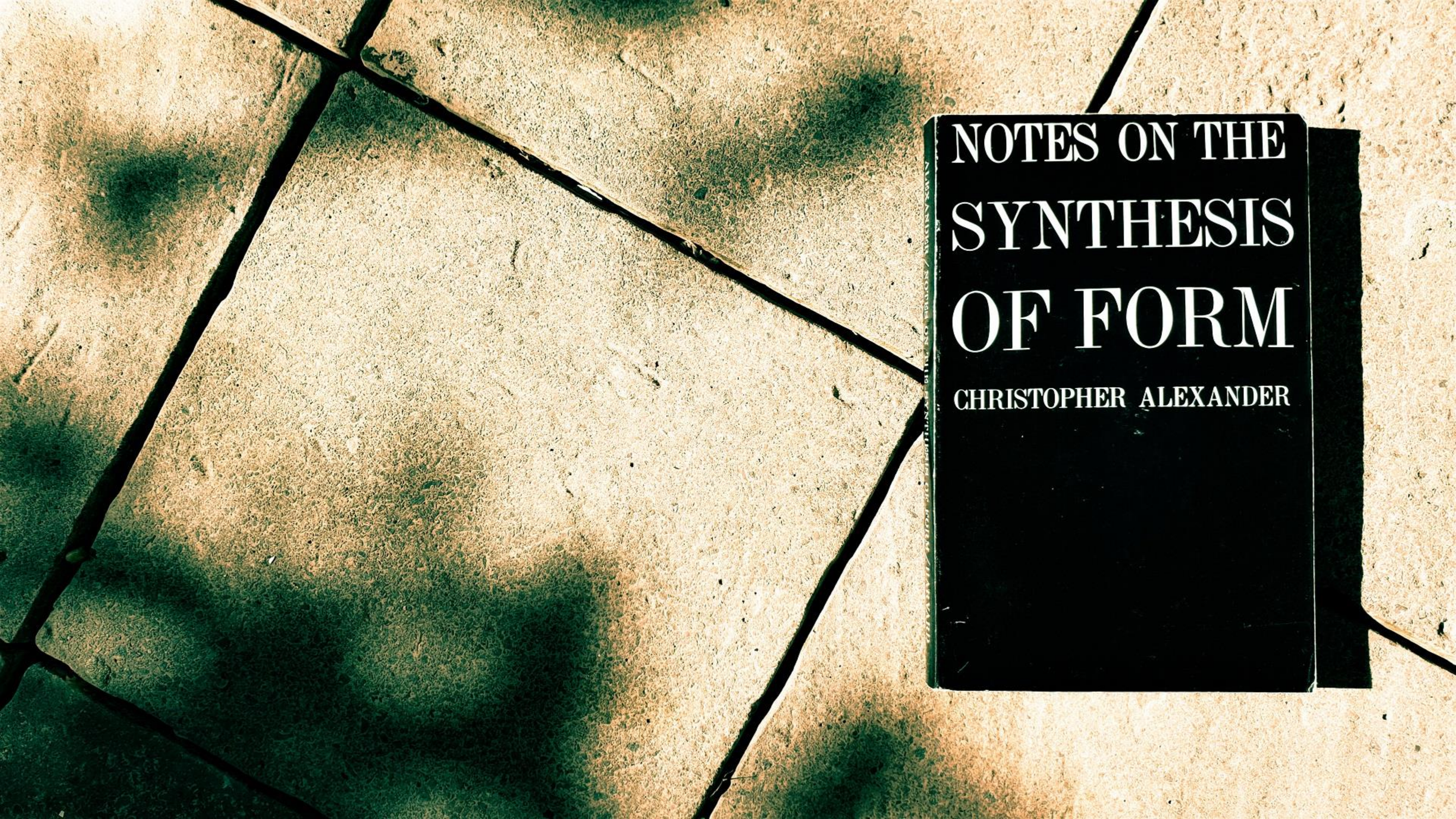


SOFTWARE ENGINEERING

Report on a conference sponsored by the

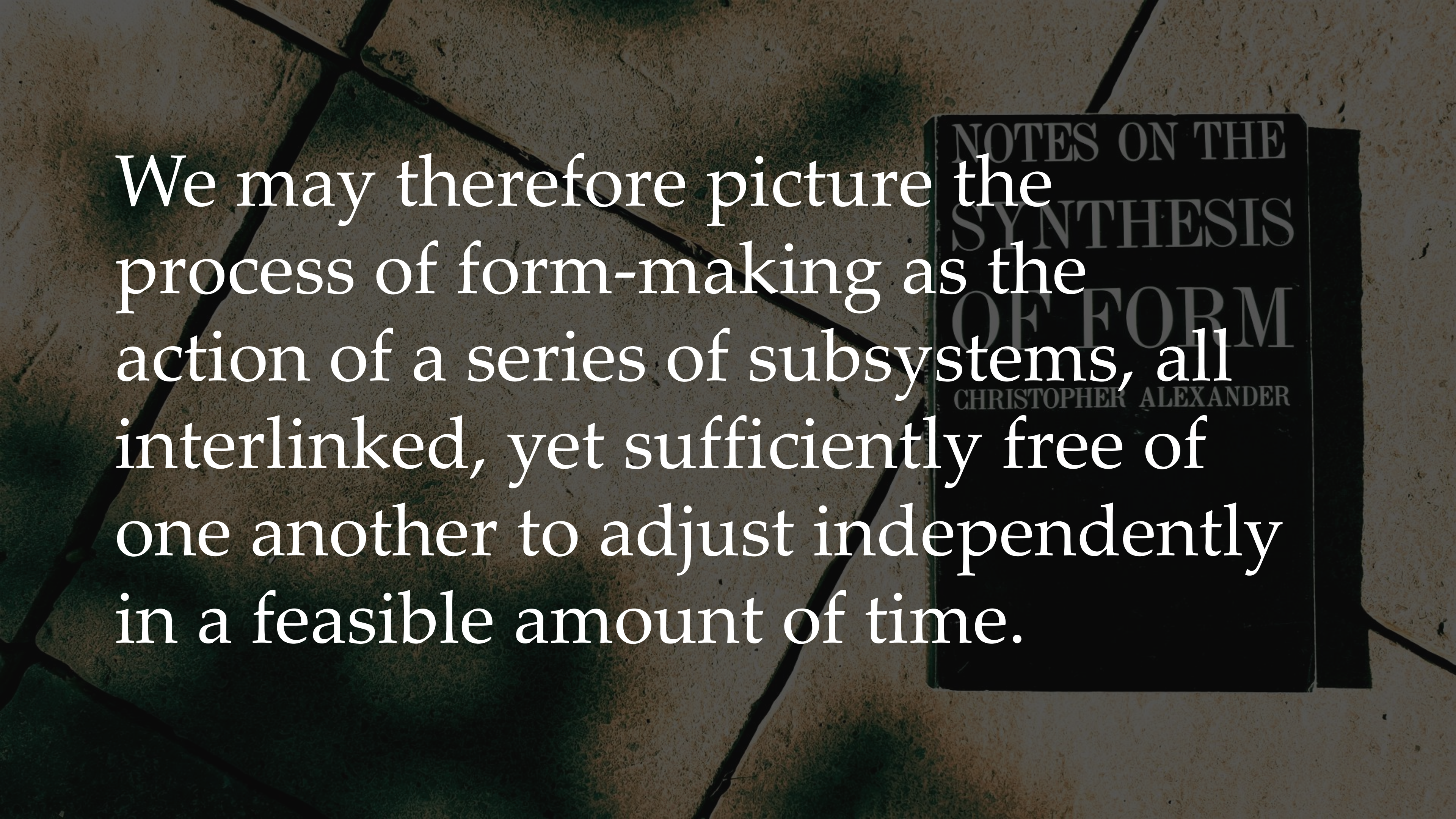
NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968



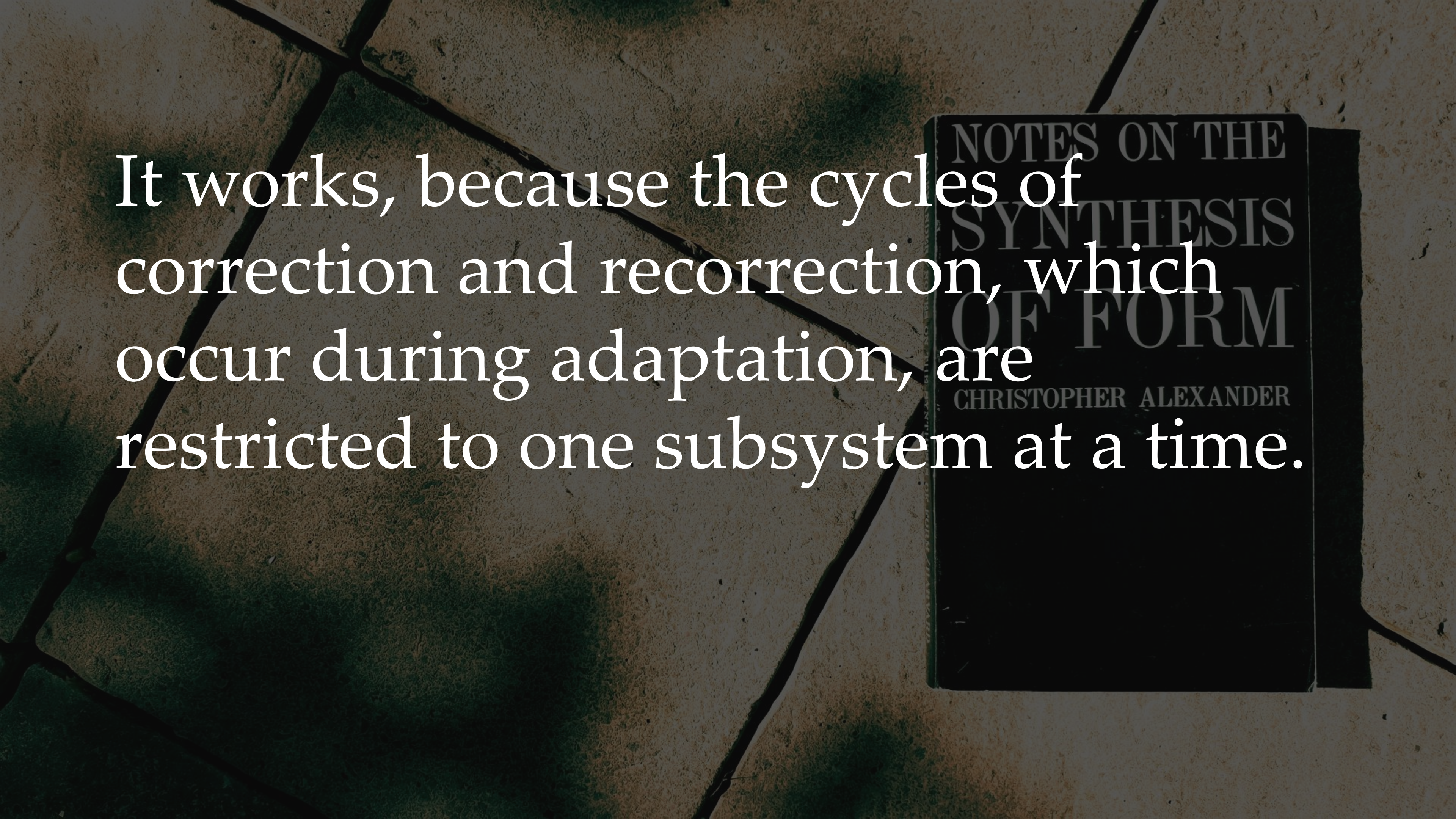
NOTES ON THE SYNTHESIS OF FORM

CHRISTOPHER ALEXANDER

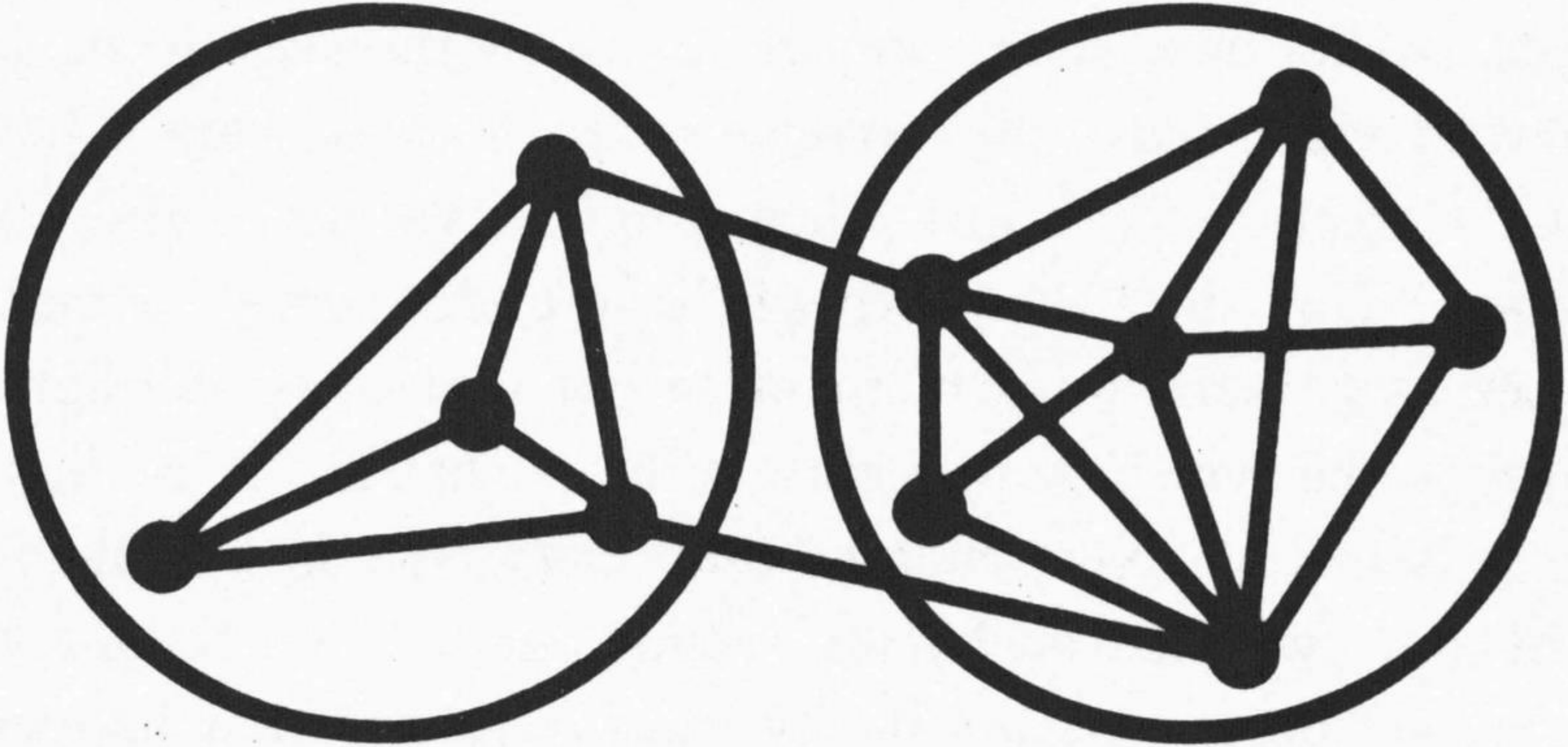


We may therefore picture the process of form-making as the action of a series of subsystems, all interlinked, yet sufficiently free of one another to adjust independently in a feasible amount of time.

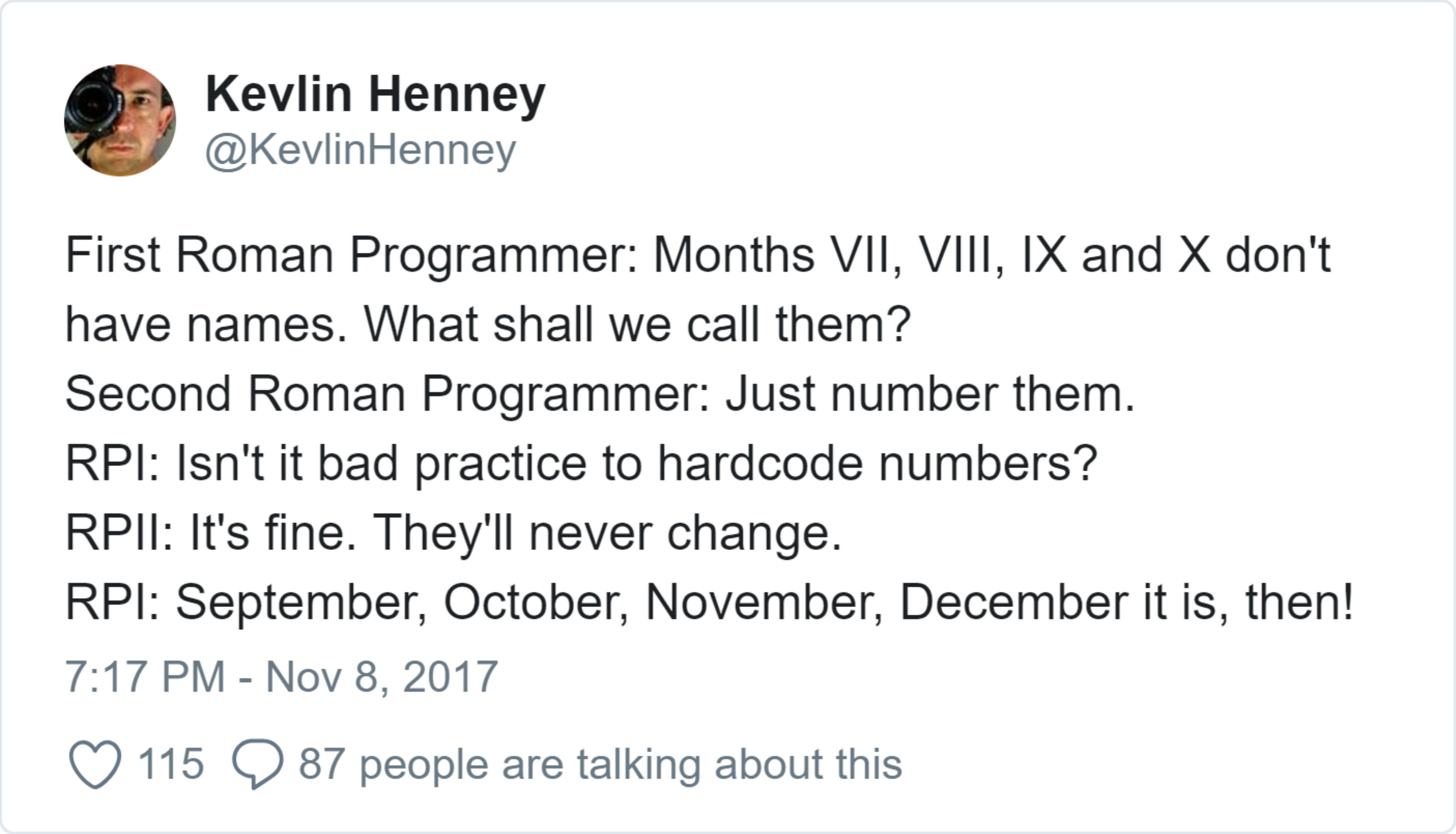
It works, because the cycles of correction and recorrection, which occur during adaptation, are restricted to one subsystem at a time.



circled below, which can, in principle, operate fairly independently.[32]



We may therefore picture the process of form-making as the action of a series of subsystems, all interlinked, yet suf-



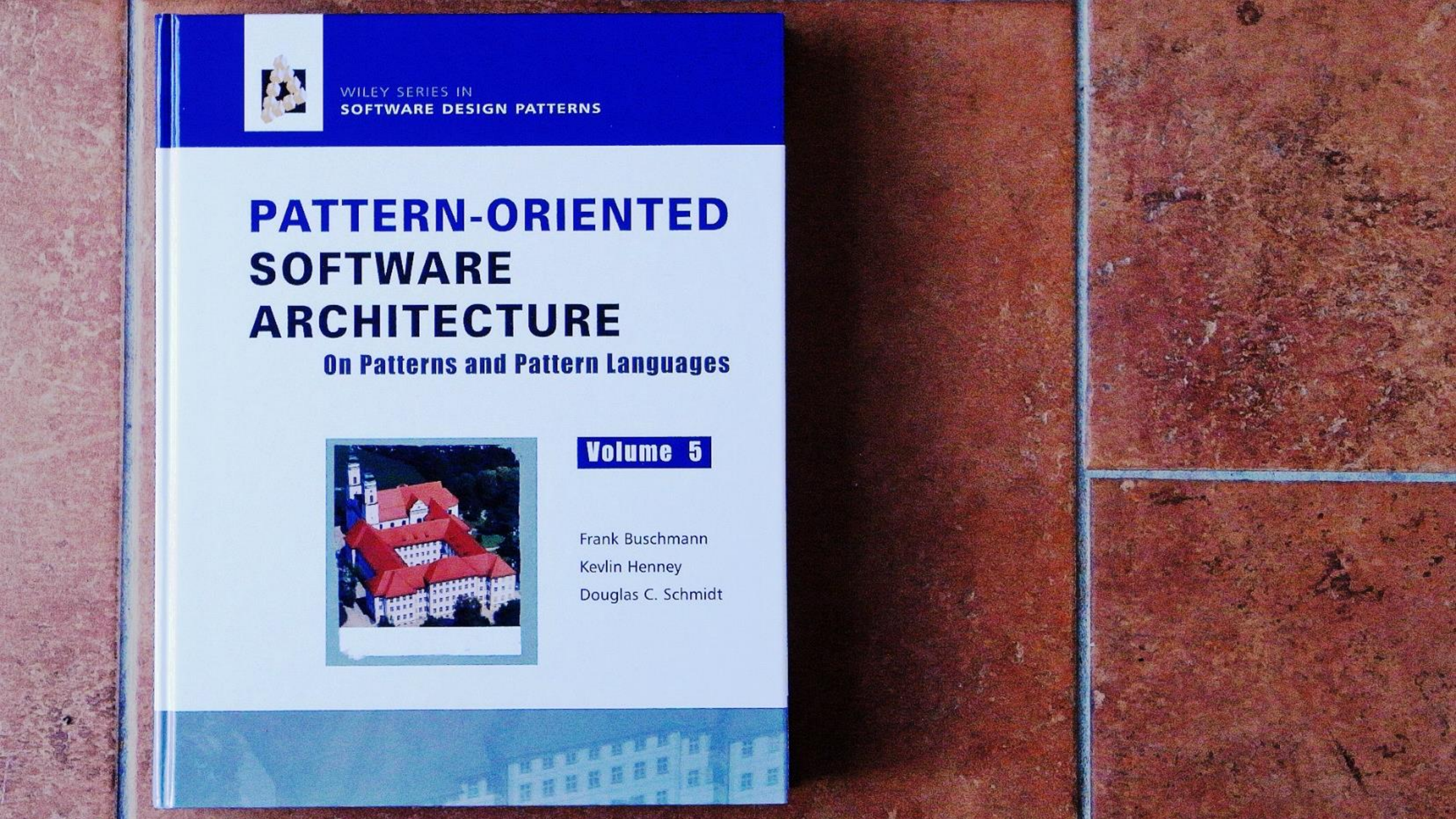
Kevlin Henney
@KevlinHenney

First Roman Programmer: Months VII, VIII, IX and X don't have names. What shall we call them?
Second Roman Programmer: Just number them.
RPI: Isn't it bad practice to hardcode numbers?
RPII: It's fine. They'll never change.
RPI: September, October, November, December it is, then!

7:17 PM - Nov 8, 2017

♡ 115 💬 87 people are talking about this

https://twitter.com/KevlinHenney/status/928310632830722048



WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

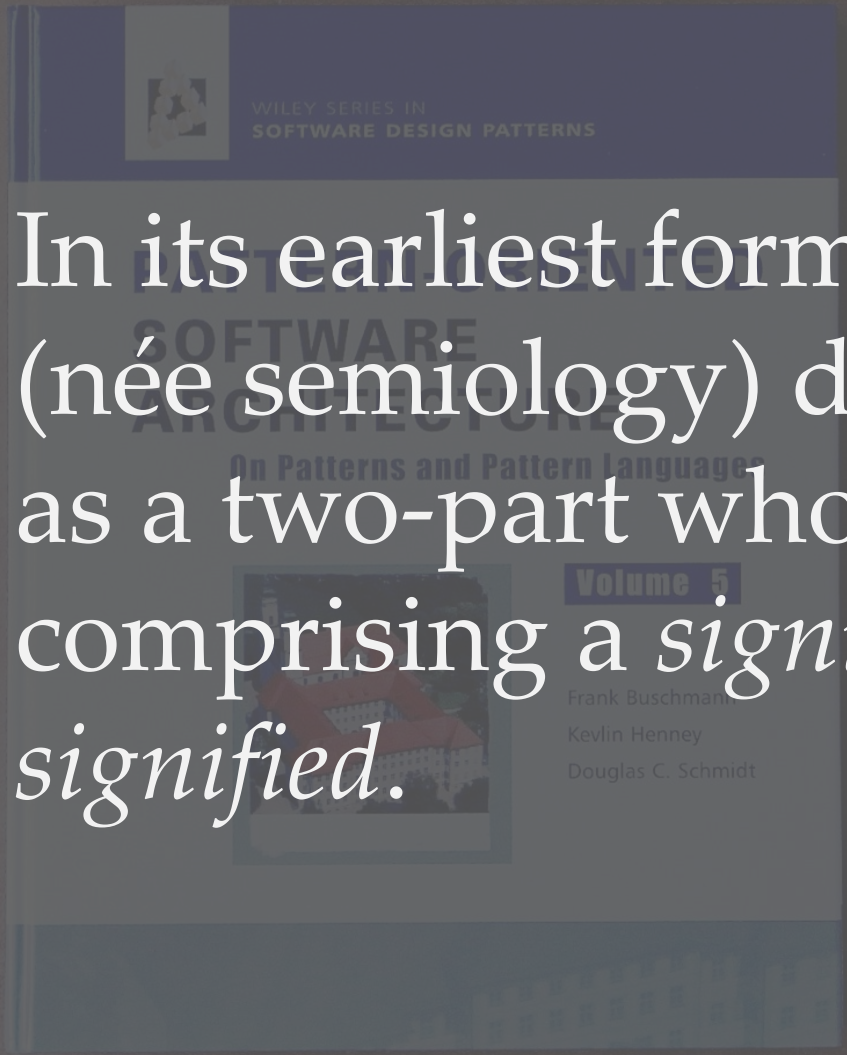
PATTERN-ORIENTED SOFTWARE ARCHITECTURE

On Patterns and Pattern Languages

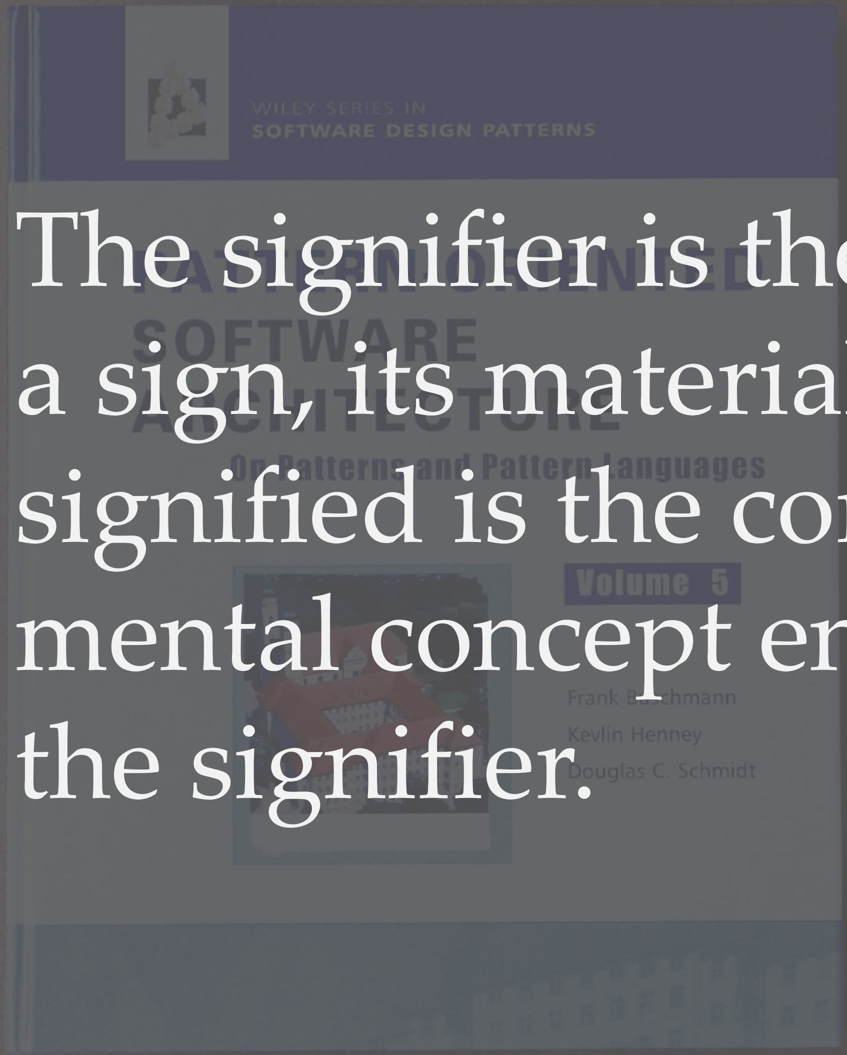
Volume 5

Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

In its earliest form, semiotics (née semiology) defines a sign as a two-part whole, a dyad, comprising a *signifier* and a *signified*.



The signifier is the expression of a sign, its material aspect. The signified is the corresponding mental concept engendered by the signifier.



dinner

half two

14:30

13:30

half twee

halv to

halv två

halb zwei

02:30

01:30

one

velocity

speed

$$\mathbf{v} = \mathbf{v}_x + \mathbf{v}_y$$

$$v = |\mathbf{v}|$$

$$v = s'$$

$$v = \frac{ds}{dt}$$

$$v = \frac{s}{t}$$

This sentence no verb.

blanc

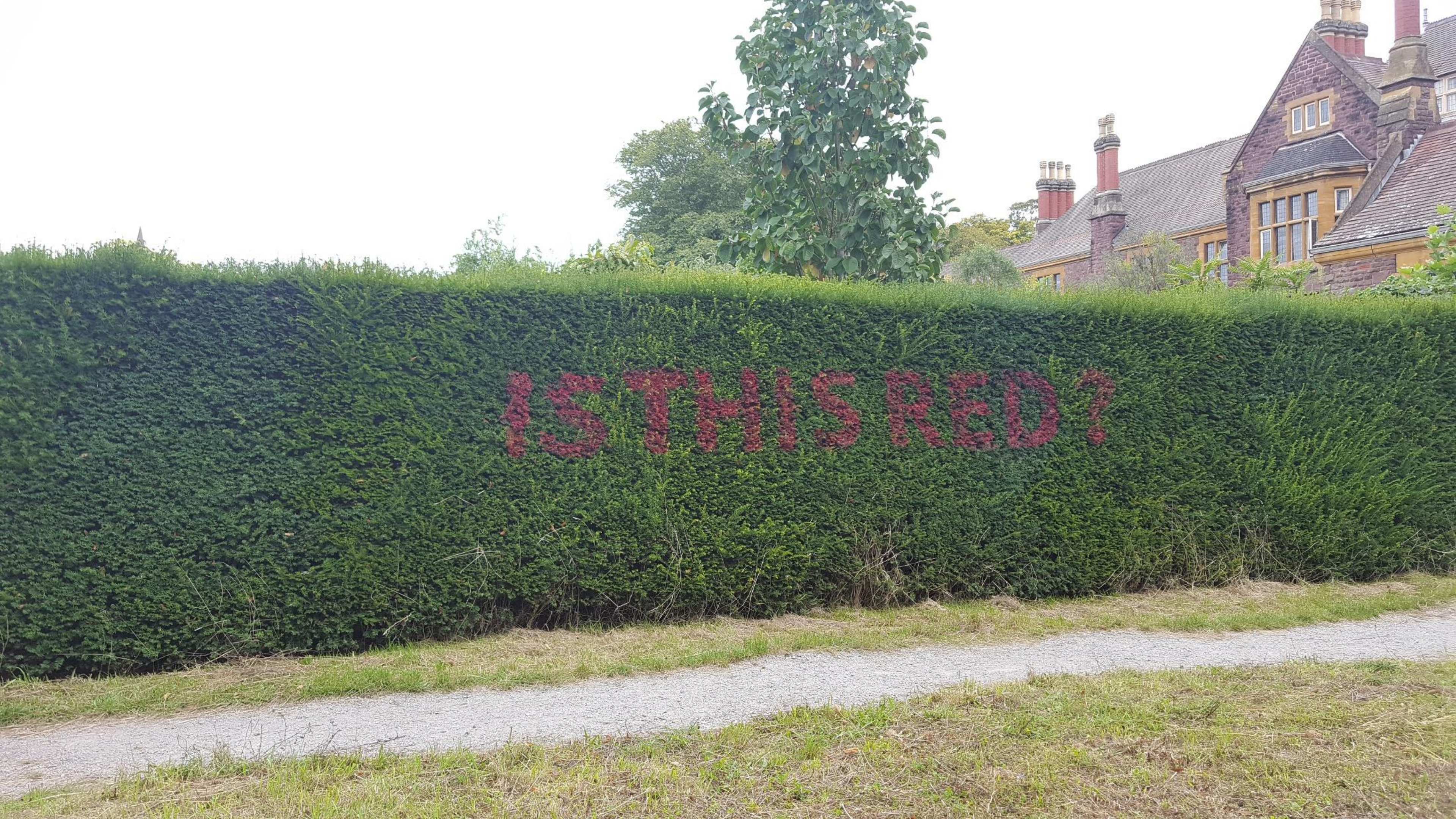
blanc

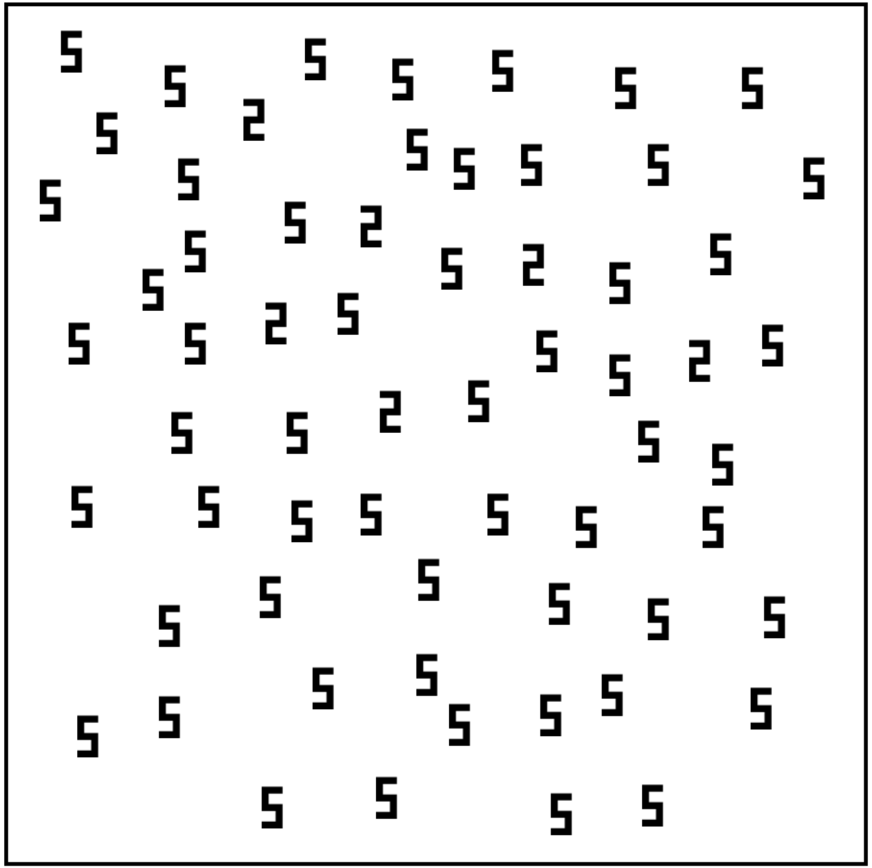
DO NOT
CROSS
THE
RED MAN!

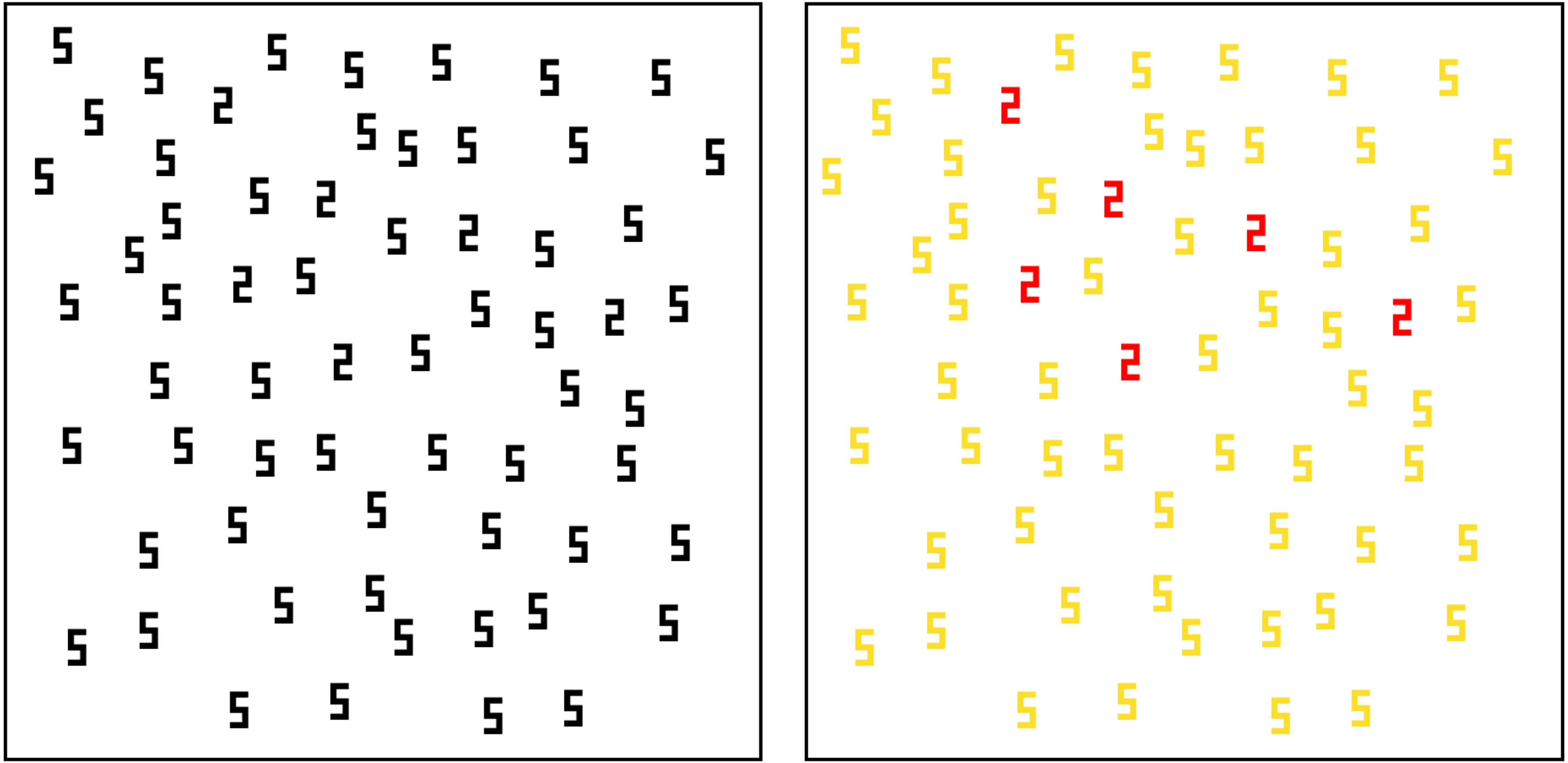
DEATH
IS
COMING
At
any
time
on footway

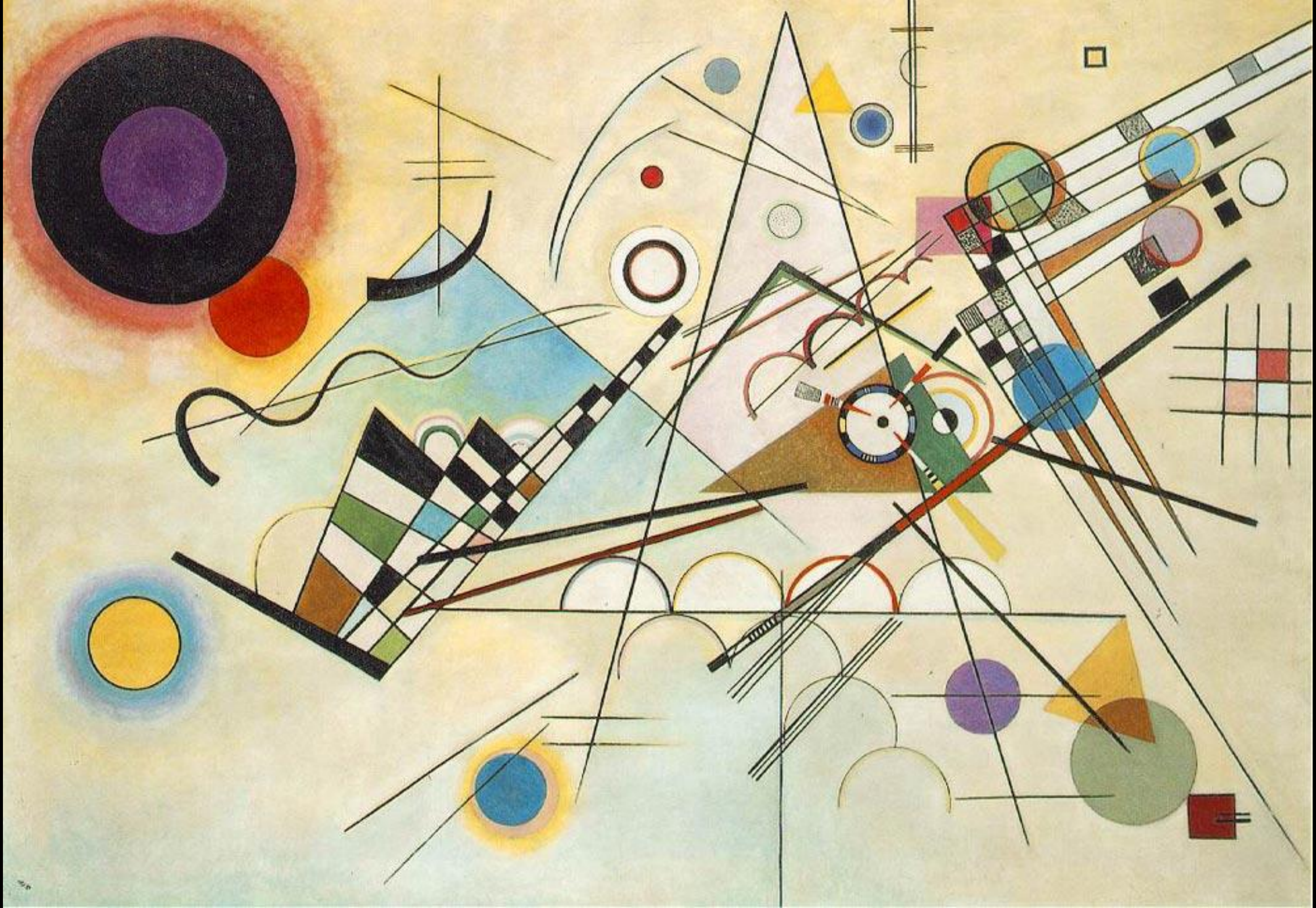
red

green


IS THIS RED?







green

green

black

green

Agile Software Development with Scrum

red
yellow
green
blue
red
blue
yellow
green
blue

Color Test

Ken Schwaber Mike Beedle

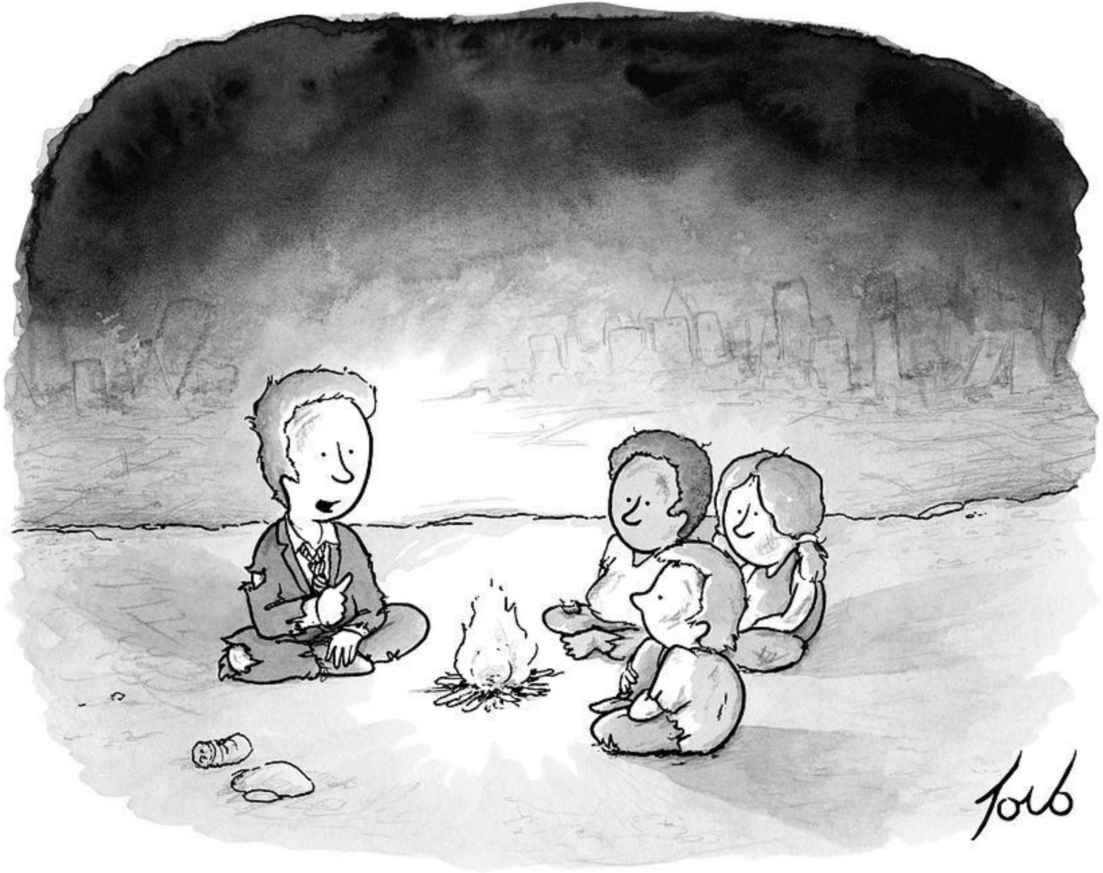
value

business value

prioritise by business value

POLICE
PUBLIC CALL
BOX
FREE
PUBLIC
PULL TO OPEN
SPACE
VADERS
PLAYER
01
NEW

prioritise by estimated business value



"Yes, the planet got destroyed, but for a beautiful moment in time we created a lot of value for shareholders."

http://tomtoro.com/cartoons/

S-Programs
P-Programs
E-Programs

Meir M Lehman

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

S-Programs

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

Meir M Lehman

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

P-Programs

Despite the fact that the problem to be solved can be precisely defined, 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"

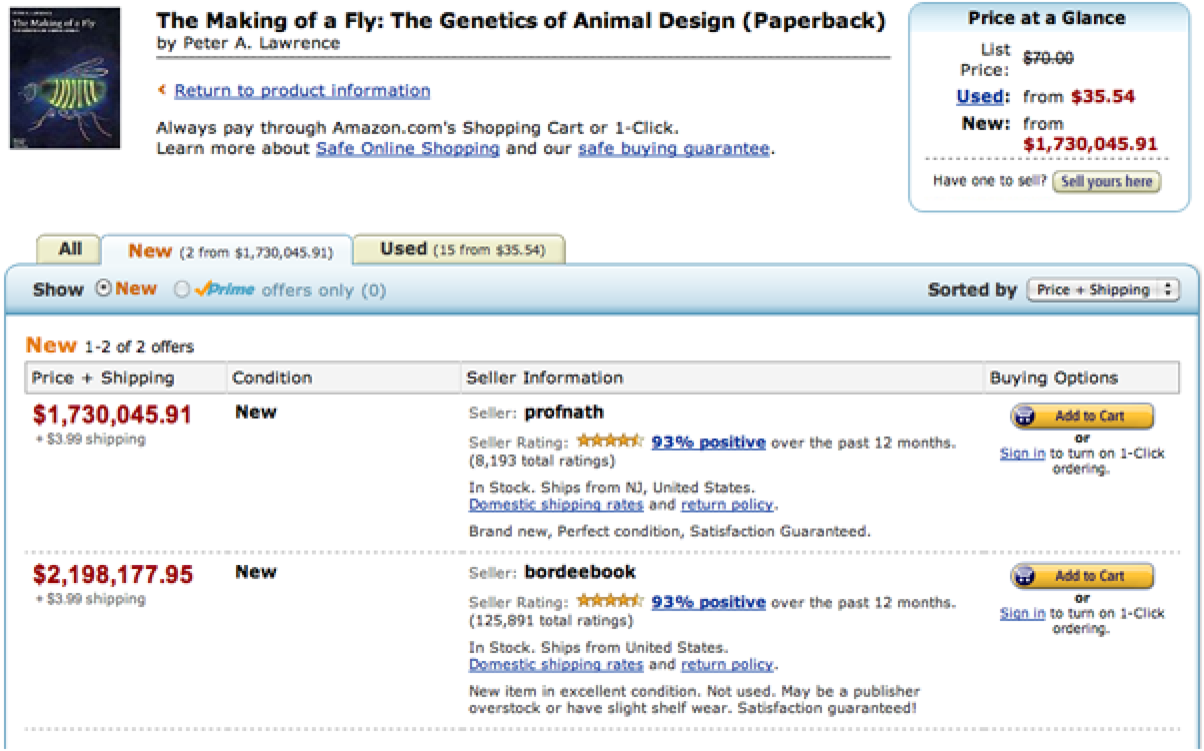
E-Programs

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**

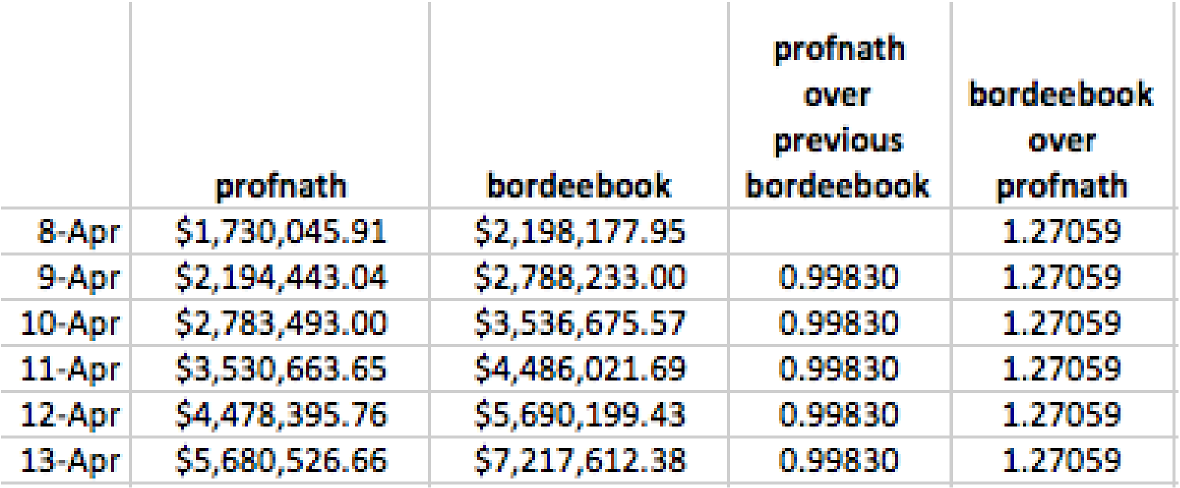
Have one to sell? Sell yours here

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

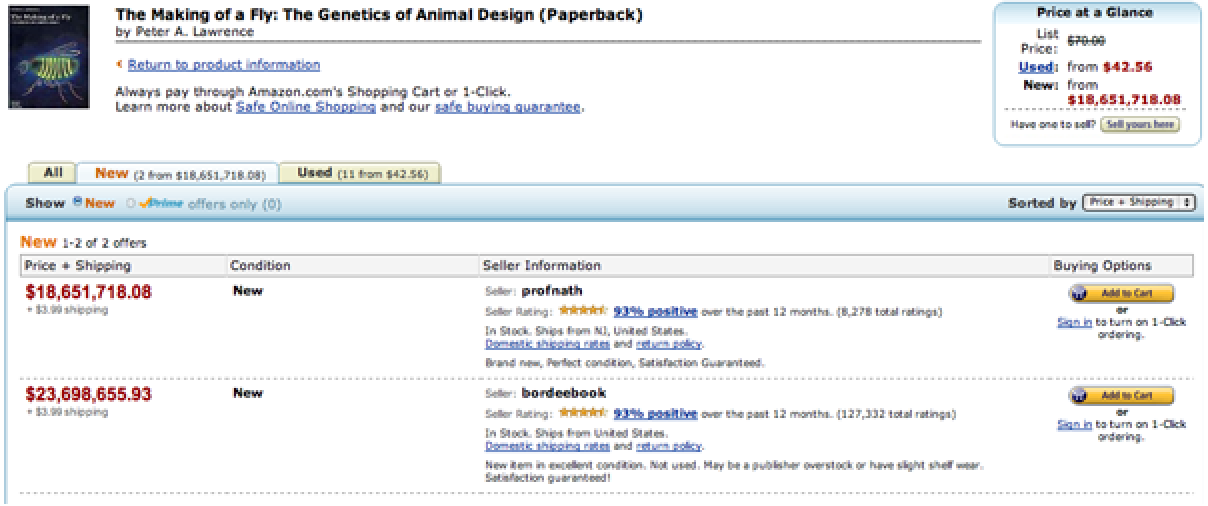
Price + Shipping	Condition	Seller Information	Buying Options
$1,730,045.91 + $3.99 shipping	**New**	Seller: **profnath** Seller Rating: ★★★★½ 93% positive over the past 12 months. (8,193 total ratings) In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect 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: ★★★★½ 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.

http://www.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

http://www.michaeleisen.org/blog/?p=358



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 $42.56
New: from $18,651,718.08

Have one to sell? Sell yours here

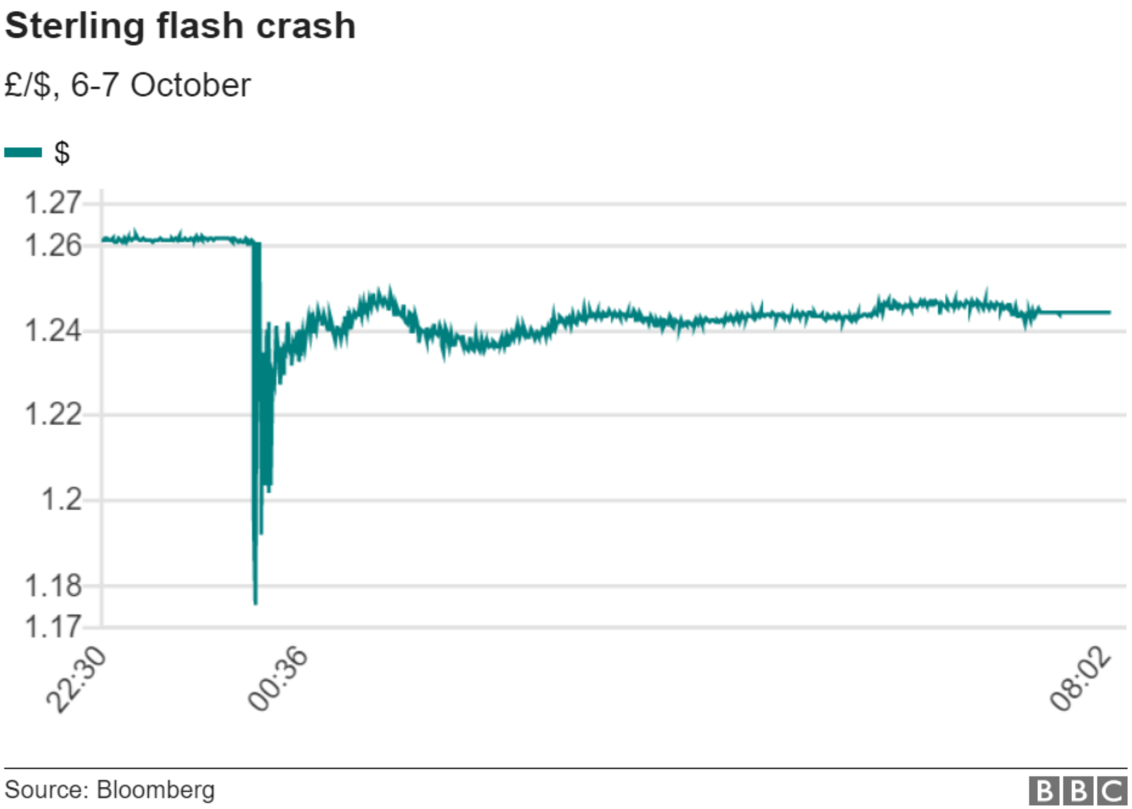
All | New (2 from $18,651,718.08) | Used (11 from $42.56)

Show ● New ○ ✓Prime offers only (0)   Sorted by Price + Shipping

New 1-2 of 2 offers

Price + Shipping	Condition	Seller Information	Buying Options
$18,651,718.08 + $3.99 shipping	New	Seller: **profnath** Seller Rating: ★★★★½ 93% positive over the past 12 months. (8,278 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.
$23,698,655.93 + $3.99 shipping	New	Seller: **bordeebook** Seller Rating: ★★★★½ 93% positive over the past 12 months. (127,332 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.

http://www.michaeleisen.org/blog/?p=358



Sterling flash crash

£/$, 6-7 October

Source: Bloomberg

BBC

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

http://www.bbc.co.uk/news/business-37582150

101 Things I Learned
in Architecture School
Matthew Frederick

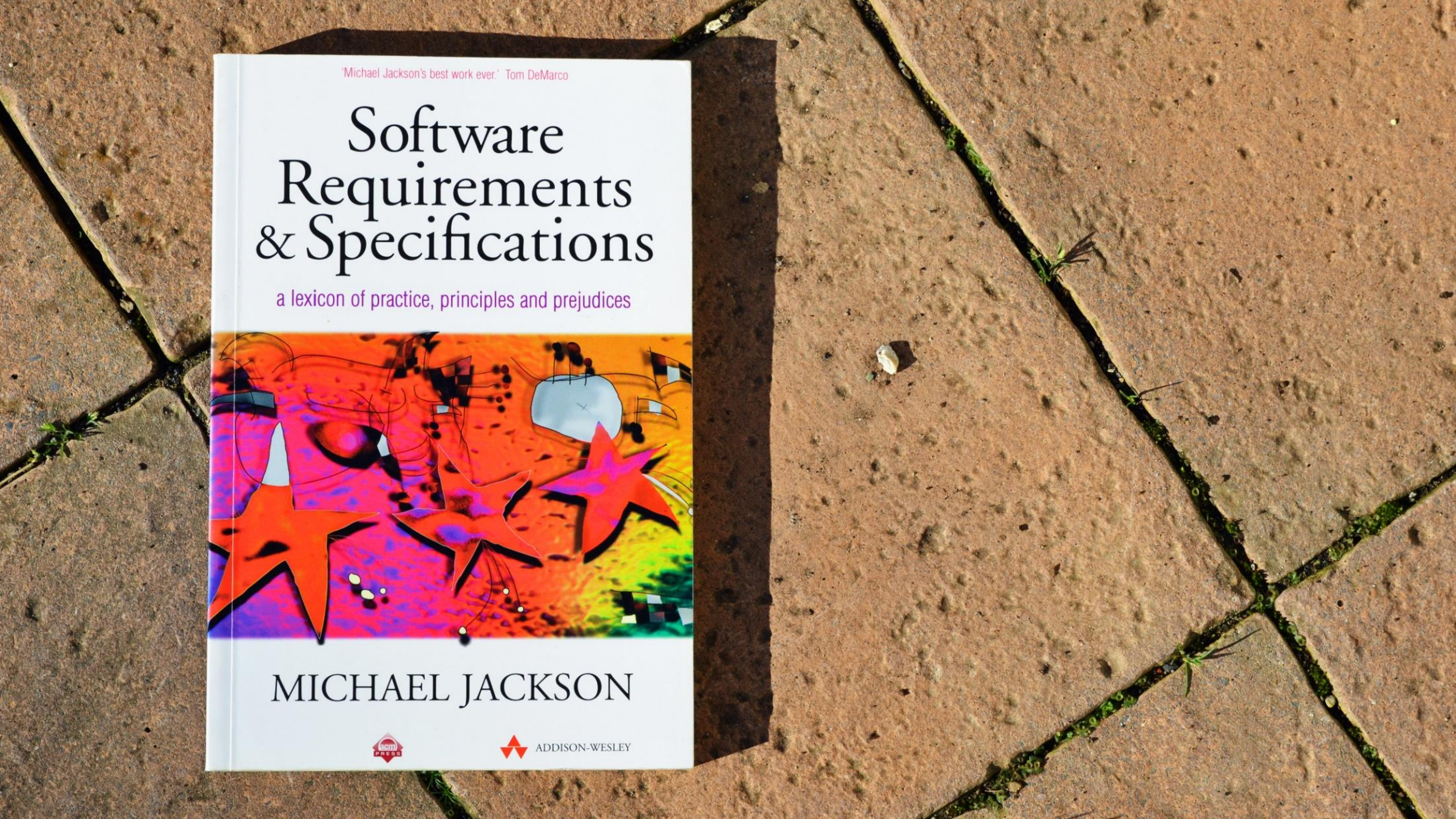
Always design a thing by considering it in its next larger context.

Eliel Saarinen

Development needs to go further than the technical stack; the full stack includes the world and people around the software.

Kevlin Henney

https://jaxlondon.com/blog/java-core-languages/the-error-of-our-ways-kevlin-henney/



'Michael Jackson's best work ever.' Tom DeMarco

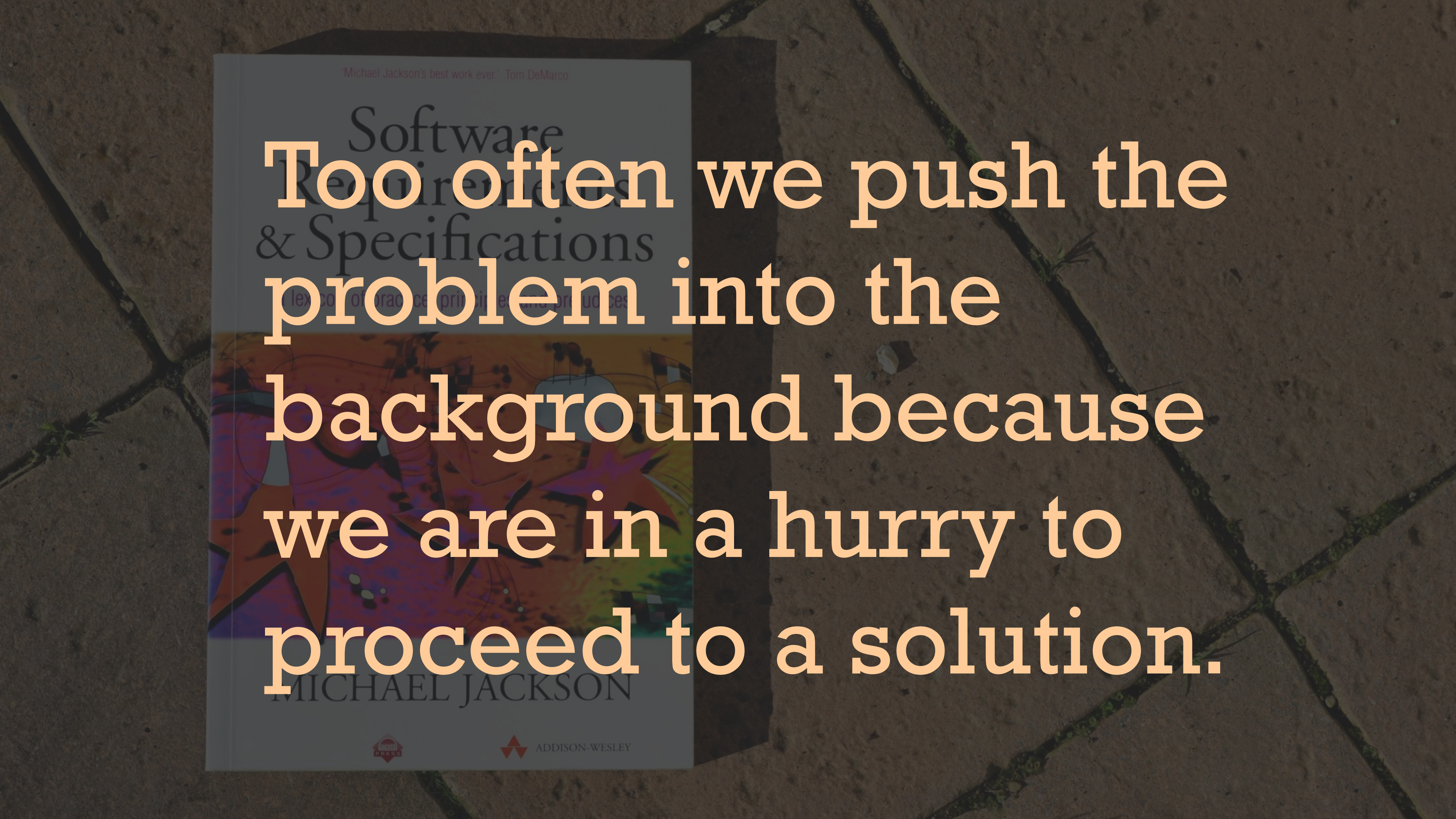
Software Requirements & Specifications

a lexicon of practice, principles and prejudices

MICHAEL JACKSON

acm PRESS

ADDISON-WESLEY


'Michael Jackson's best work ever.' Tom DeMarco
Software
Requirements
& Specifications
lexicon of practice, principles and prejudices
MICHAEL JACKSON
ADDISON-WESLEY

Too often we push the problem into the background because we are in a hurry to proceed to a solution.

It’s just semantics.

It’s just meaning.