Tony Hoare:

A Life of Logic, Theory, and Practice

Prof. Jonathan P. Bowen FRSA FBCS

Emeritus Professor of Computing
London South Bank University, UK

Adjunct Professor, Southwest University, Chongqing, China

Chairman, Museophile Limited, Oxford, UK

jonathan.bowen@lsbu.ac.uk

www.jpbowen.com

EST 1892 LSBU

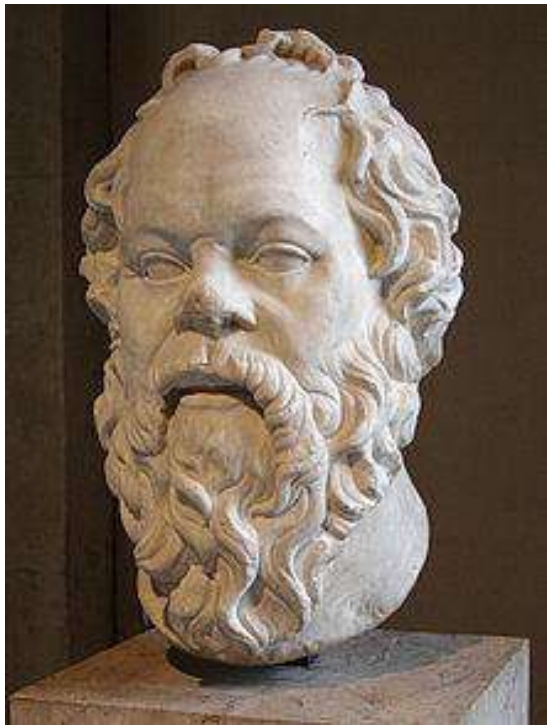
Museophile

SOUTHWEST UNIVERSITY 1906

Prologue

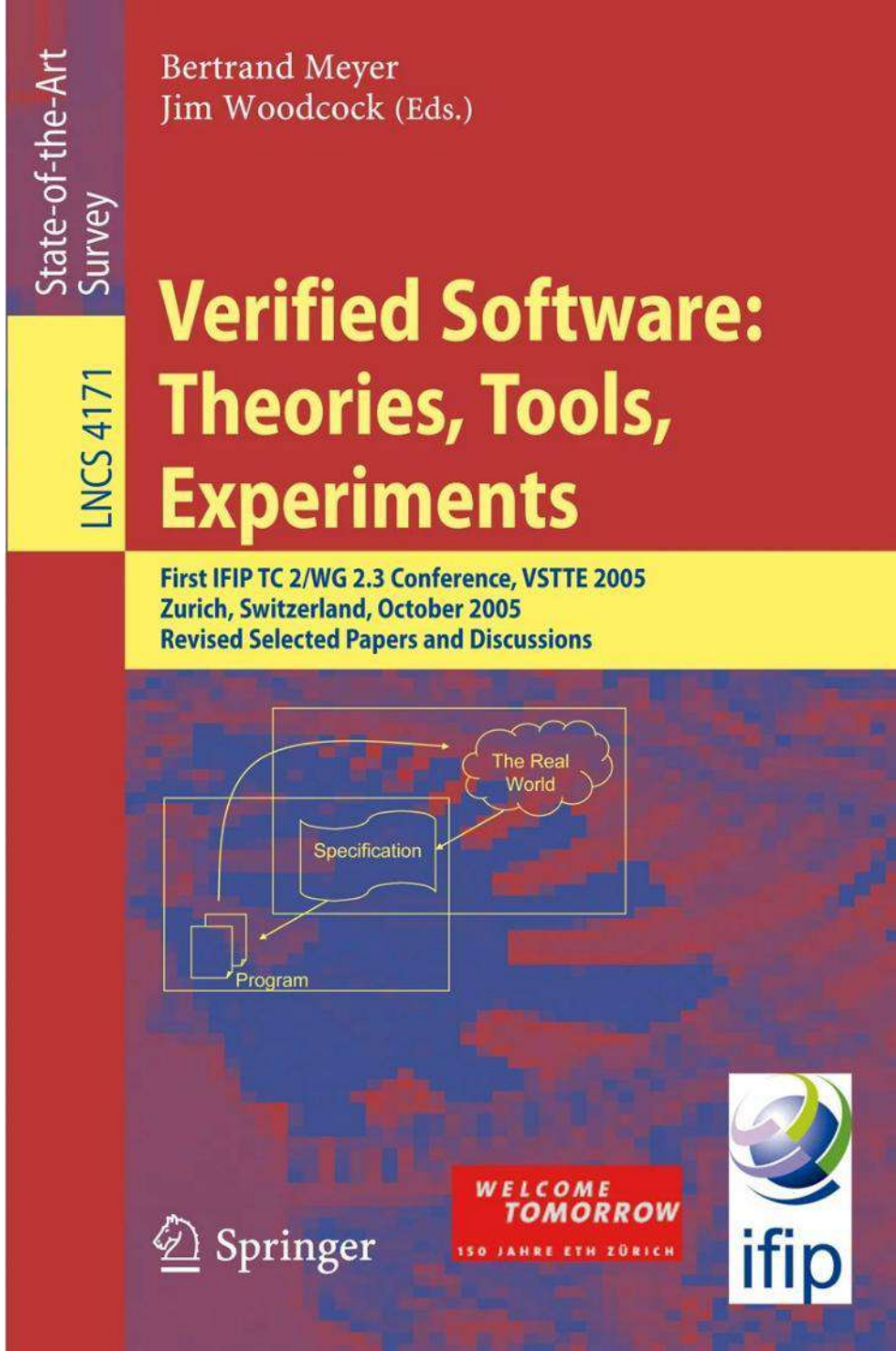
Socrates – Σωκράτης
(c.470 BC – 399 BC)

- ὁ δὲ ἀνεξέταστος βίος οὐ βιωτὸς ἀνθρώπῳ
- *“The unexamined life is not worth living for a human being”*



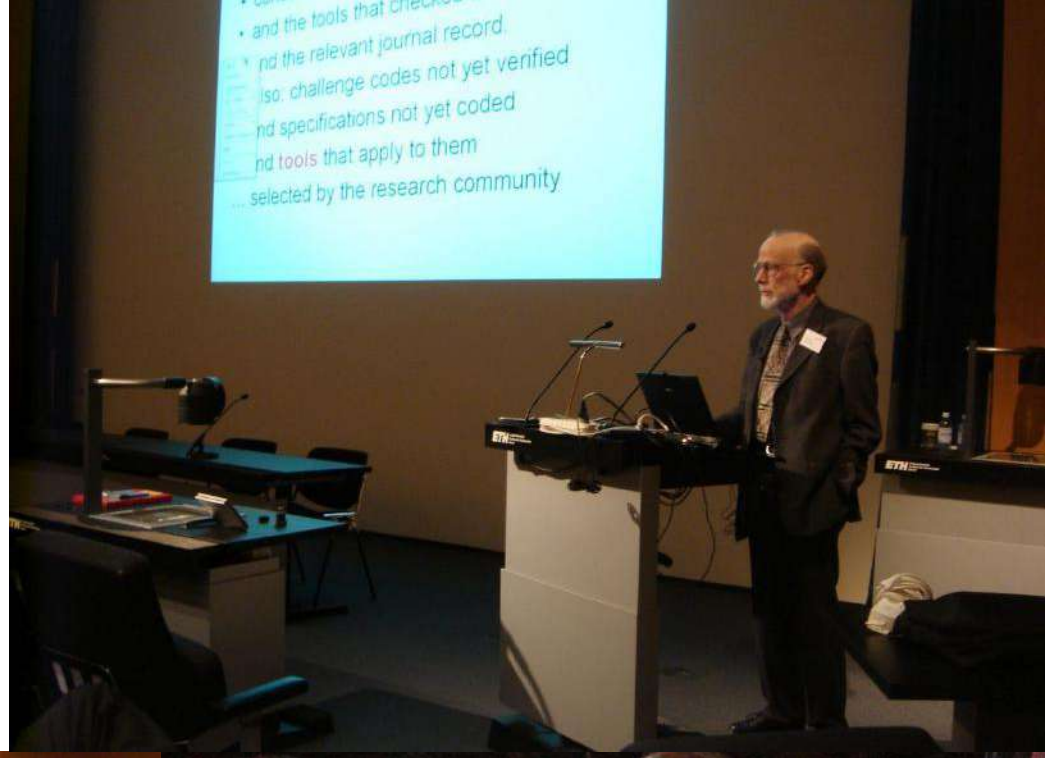
VSTTE 2005

- ***Verified Software: Theories, Tools, Experiments***: First IFIP TC2/WG2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 2005. Revised Selected Papers and Discussions (Springer LNCS 4171, 2008)
- ***Verified Software: Theories, Tools, Experiments – Vision of a Grand Challenge Project***. **Tony Hoare** and Jay Misra. In B. Meyer and J. Woodcock (eds.): *Verified Software*, LNCS 4171, pp. 1–18, 2008.



VSTTE 2005

- **Tony Hoare**, opening session
- Jay Misra, Niklaus Wirth, **Tony Hoare**, on the train to the conference dinner
- Closing session panel



Photographs by Bertrand Meyer

VSTTE 2005

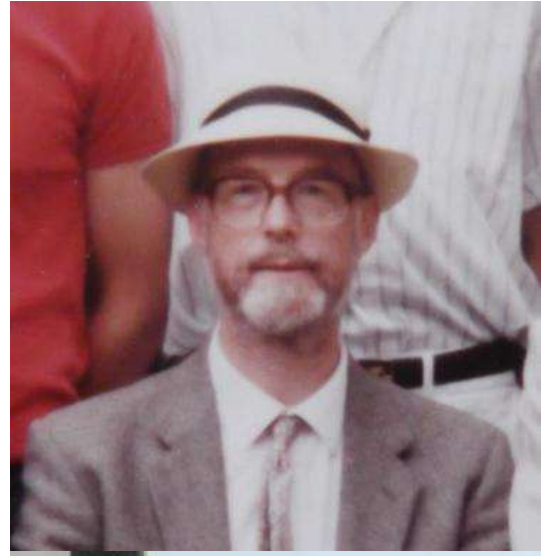
Welcome party in the historic Semper-Aula at ETH Zurich

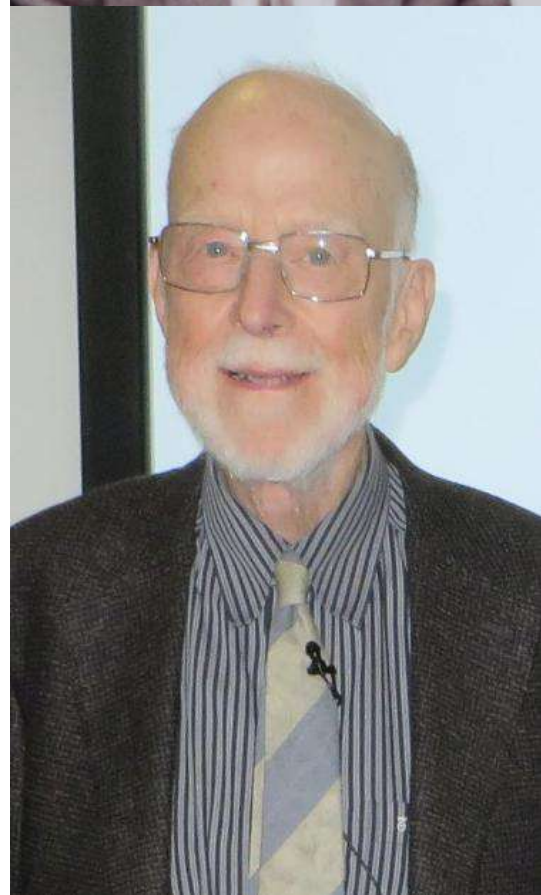
VSTTE 2005

Tony Hoare (1934–2026)

- Merton College, Oxford
- Moscow State University
- Elliott Brothers
- Queen’s University Belfast
- Oxford University Computing Laboratory
  - Programming Research Group
- Microsoft Research Cambridge

1986

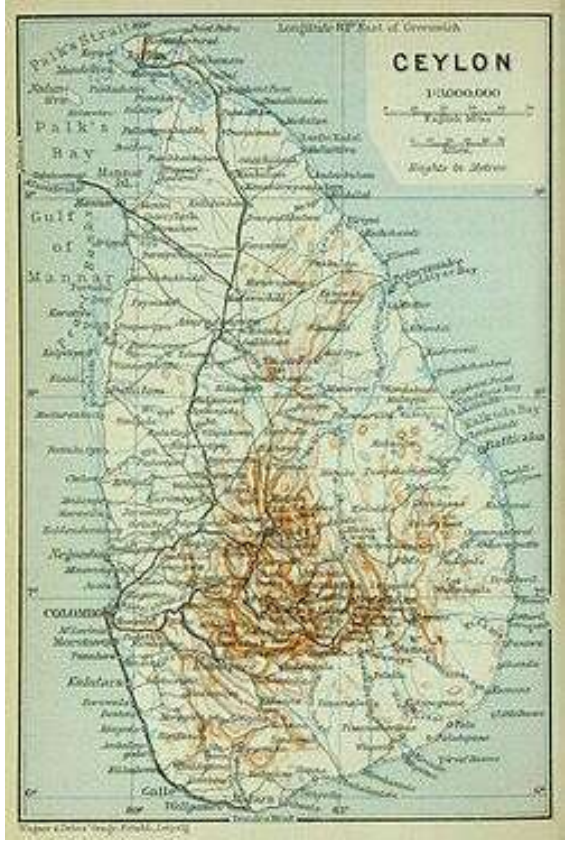




2018

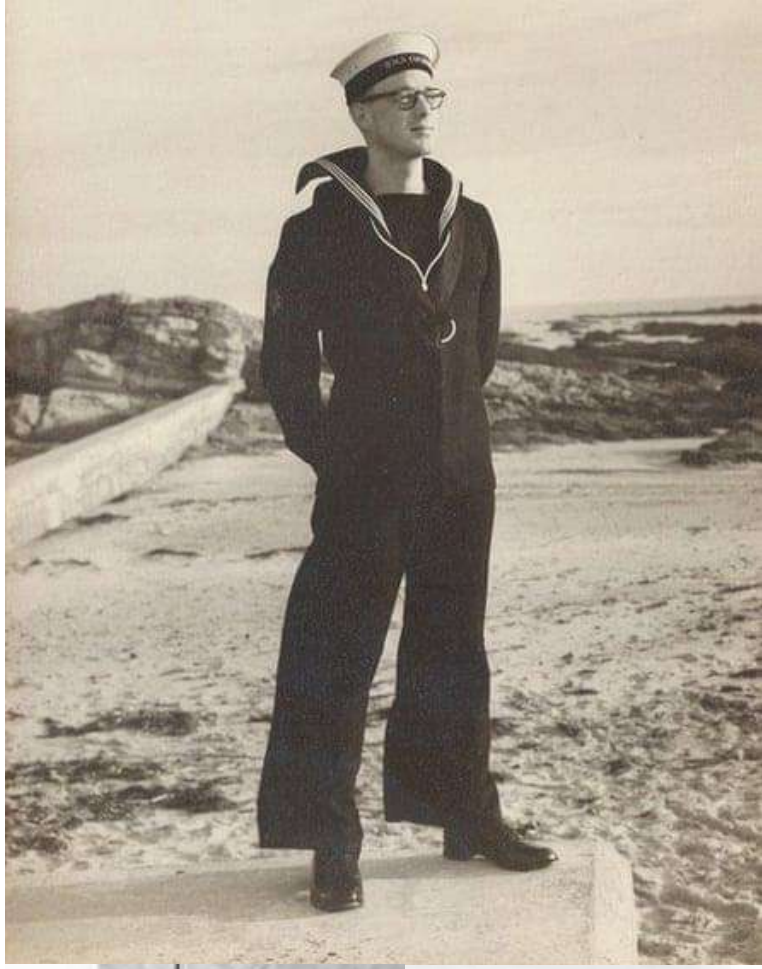
Tony Hoare – early life

- Born 11 January 1934 in Colombo, then Ceylon, now Sri Lanka, father a colonial civil servant
  - Full name: **C**harles **A**n**tony** **R**ichard **H**oare
- Dragon School, Oxford, England
- The King's School, Canterbury
- Merton College, Oxford
  - Studied "Greats" (classics), including logic
  - **John Lucas** (1929–2020), philosopher
    - *Minds, Machines and Gödel* (1959)
    - Penrose–Lucas argument (Gödel)
  - Graduated from Oxford in 1956



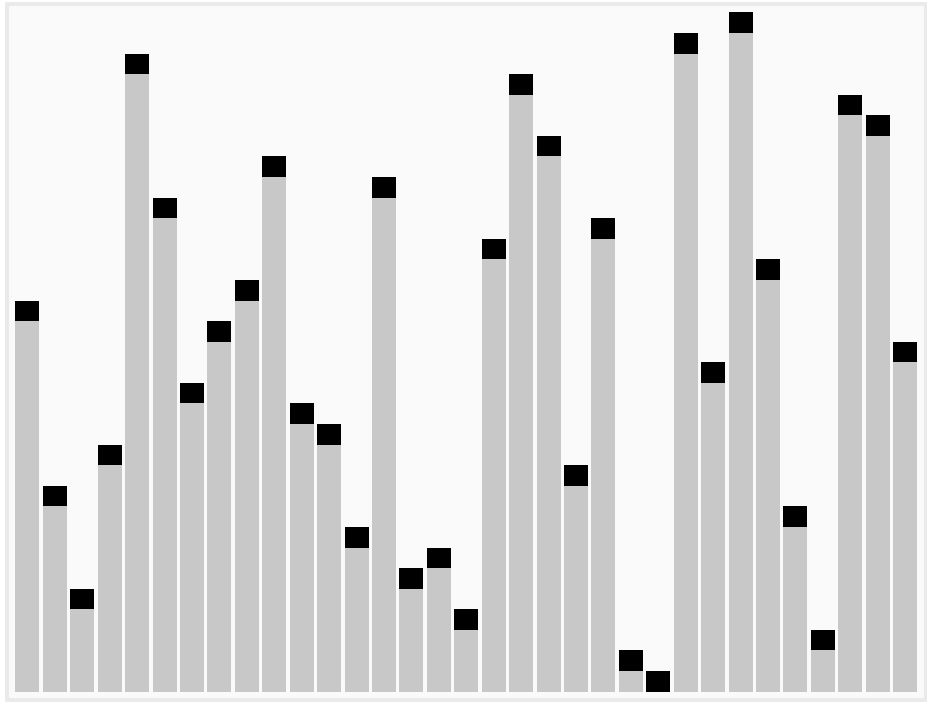
Tony Hoare – early life

- 1956: 18 months National Service in the Royal Navy – learnt Russian
- 1958: Returned to Oxford – postgraduate degree in statistics
  - Learned Autocode programming on the Ferranti Mercury computer
  - **Leslie Fox** (1918–1992), numerical analyst
- 1959–1960: Moscow State University
  - British Council exchange student
  - Studied machine translation under **Andrey Kolmogorov** (1903–1987), probability theorist
  - Developed Quicksort for efficient index sorting!



Quicksort

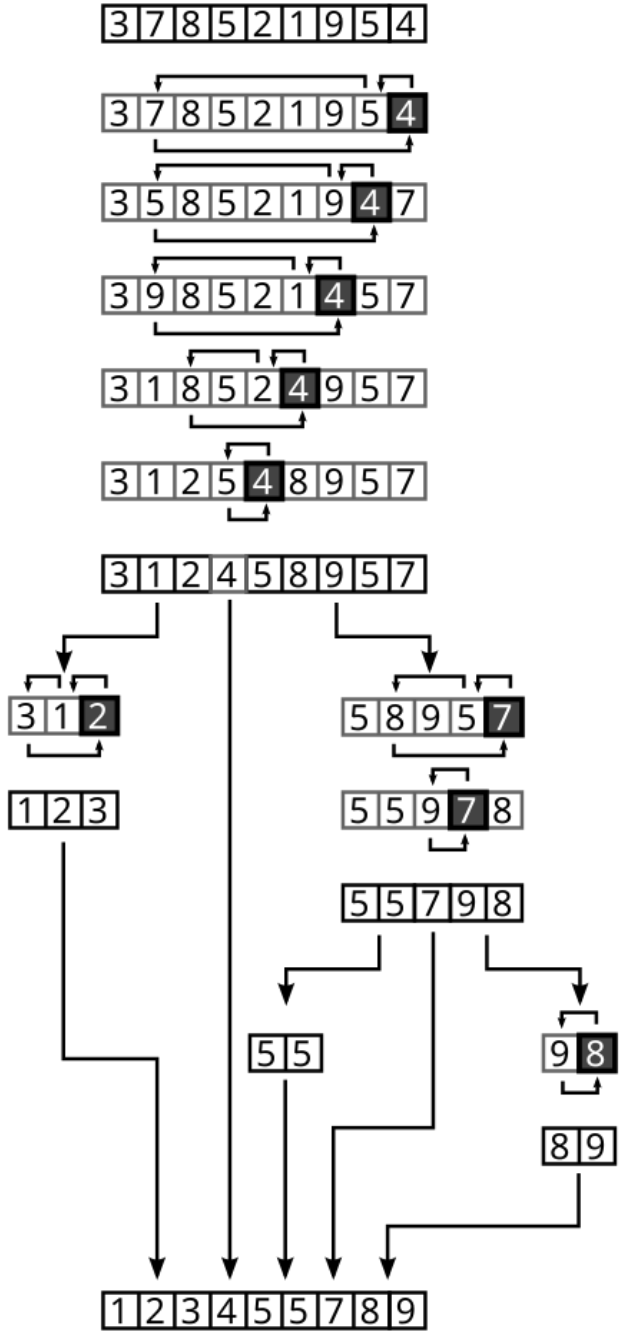
- 1959–1960: Developed for efficient sorting of words in (Russian) sentences dictionary between looking them up in a Russian-English dictionary
- Divide-and-conquer algorithm
- Best case: $O(n \log n)$; worst case $O(n^2)$ (rarely)
- Published in 1961, *Communications of the ACM* (Algorithm 63 and 64)
- Still commonly used algorithm for sorting
- Invented when aged 25!



Quicksort

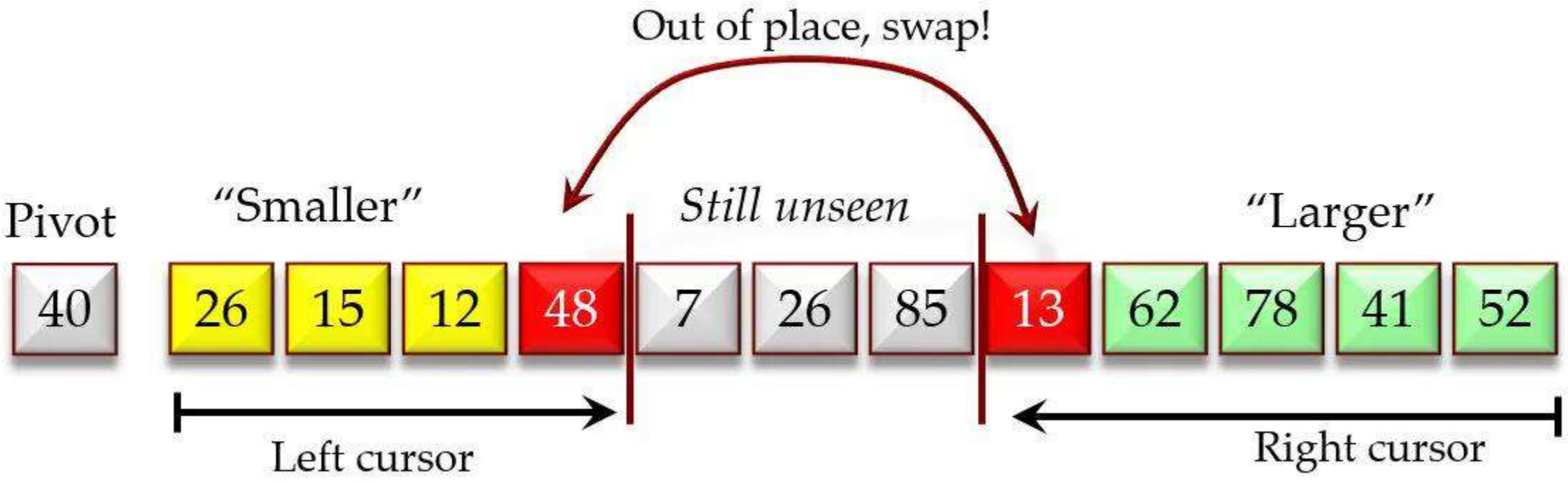
- Partition – choose a “pivot”
  - Can be random or first/middle/last element, for example
- Recursively sort
  - Furthest left and right entries inwards, swapping when out of order

6 5 3 1 8 7 2 4



Quicksort – example

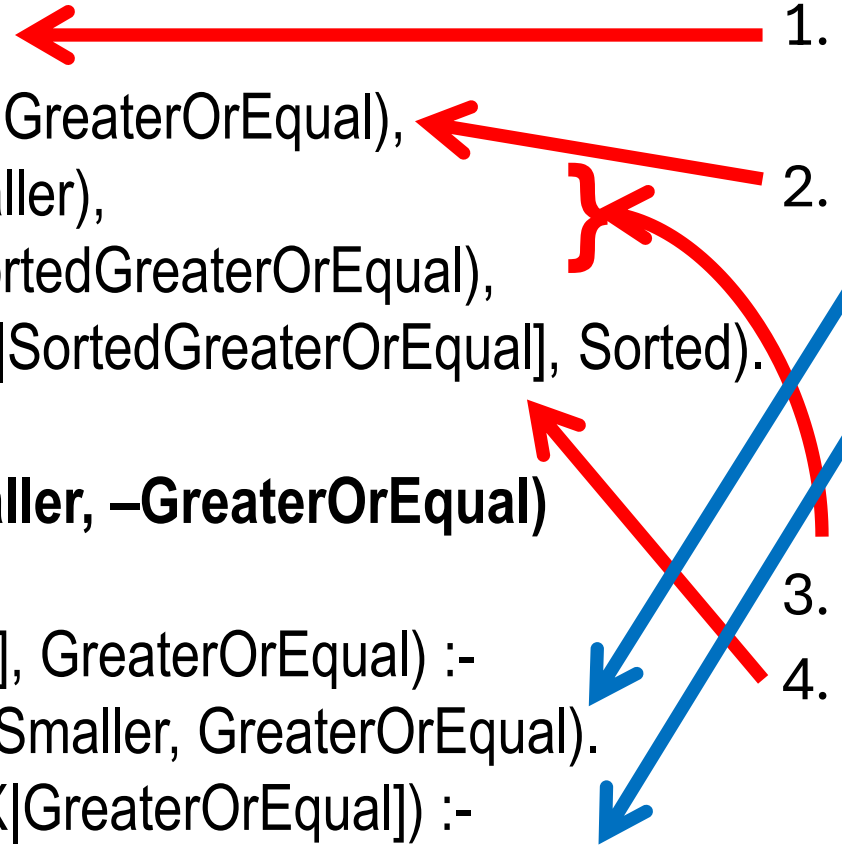
- Partition – choose a “pivot” (may be random or deterministic)
- Furthest left/right entries inwards, swapping when unordered
- Sort left and right parts recursively



Quicksort – Prolog logic program

```
% quicksort(+List, –Sorted)
quicksort([], []).
quicksort([Pivot|Rest], Sorted) :-
    partition(Pivot, Rest, Smaller, GreaterOrEqual),
    quicksort(Smaller, SortedSmaller),
    quicksort(GreaterOrEqual, SortedGreaterOrEqual),
    append(SortedSmaller, [Pivot|SortedGreaterOrEqual], Sorted).

% partition(+Pivot, +List, –Smaller, –GreaterOrEqual)
partition(_, [], [], []).
partition(Pivot, [X|Xs], [X|Smaller], GreaterOrEqual) :-
    X < Pivot, partition(Pivot, Xs, Smaller, GreaterOrEqual).
partition(Pivot, [X|Xs], Smaller, [X|GreaterOrEqual]) :-
    X >= Pivot, partition(Pivot, Xs, Smaller, GreaterOrEqual).
```

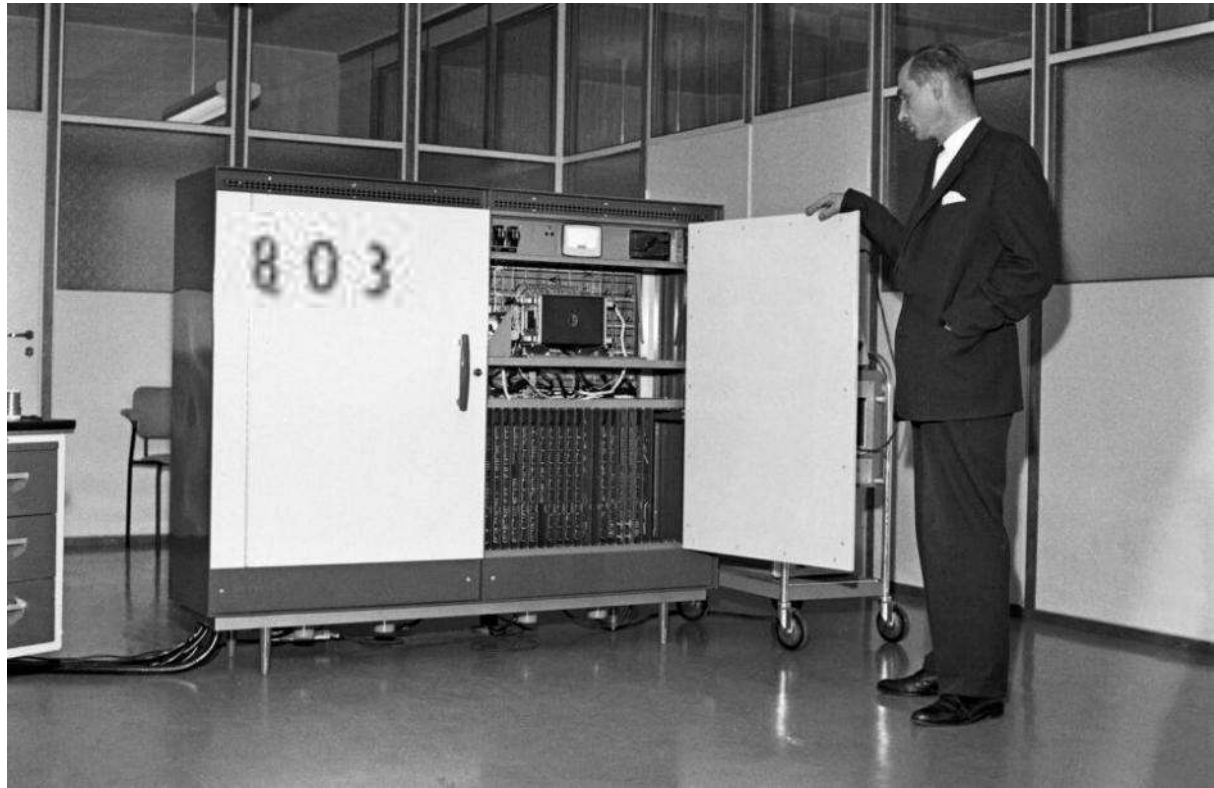


How it works:

1. The first element is chosen as the pivot.
2. The rest of the list is split.
   a. elements smaller than the pivot
   b. elements greater than or equal to the pivot
3. Each side is sorted recursively.
4. The results are joined back together.

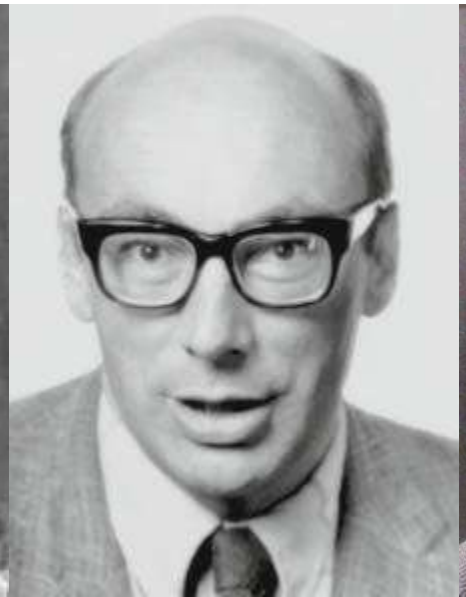
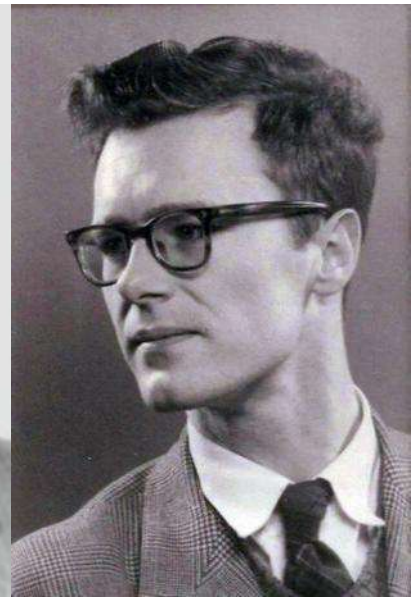
Elliott Brothers Ltd, UK (1960–1968)

- 1960: Sorting subroutine for the Elliott 803 computer
  - First Shellsort, then suggested Quicksort to his manager
  - Won 6d for being quicker!
- Then Elliott 503
  - 60 times faster



Elliott Brothers Ltd, UK (1960–1968)

- Read ALGOL 60 report
- 1961: Attended ALGOL 60 course, Brighton, UK
  - Course tutors: **Peter Naur** (1928–2016), **Peter Landin** (1930–2009), **Edsger Dijkstra** (1930–2002)
  - Attended with **Jill Pym**, Elliott programmer (married 1962)


Elliott Brothers Ltd, UK (1960–1968)

- Recursion included in ALGOL 60
- Wrote "**Quicksort**" procedure using recursion
- Decided to implement ALGOL 60 on Elliott computers
- Included runtime upper/lower bound array checking
  - Customers wanted this kept, even if less efficient
- Single-pass compiler, mutually recursive procedures
- Member of **IFIP Working Group 2.1** on *Algorithmic Languages and Calculi* (ALGOL 60 and ALGOL 68)

Partition and Quicksort (1961)

ALGORITHM 63
PARTITION
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  partition (A,M,N,I,J);  value M,N;
                array A;  integer M,N,I,J;
```

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin   real X;  integer F;
        F := random (M,N);  X := A[F];
        I := M;  J := N;
up:     for I := I step 1 until N do
                if X < A [I] then go to down;
        I := N;
down:   for J := J  step −1 until M do
                if A[J]<X then go to change;
        J := M;
change: if I < J then begin exchange (A[I], A[J]);
                        I := I + 1; J := J − 1;
                        go to up
                    end
else    if I < F then begin exchange (A[I], A[F]);
                        I := I + 1
                    end
else    if F < J then  begin exchange (A[F], A[J]);
                        J := J − 1
                    end;
end     partition
```

CACM, DOI: 10.1145/366622.366642

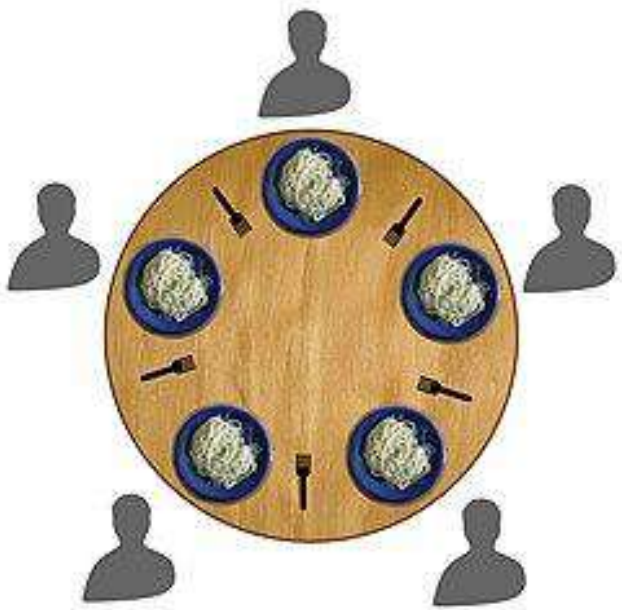
ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

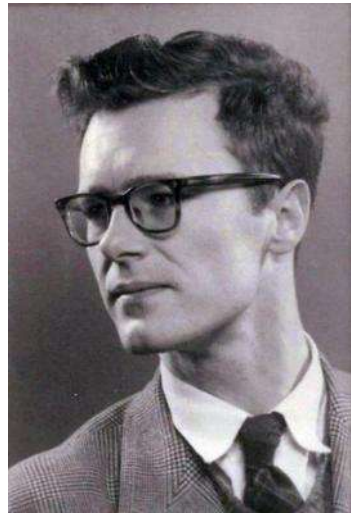
```
procedure  quicksort (A,M,N);  value M,N;
                array A;  integer M,N;
```

comment Quicksort is a very fast and convenient method of sorting an array in the random-access store of a computer. The entire contents of the store may be sorted, since no extra space is required. The average number of comparisons made is $2(M-N) \ln (N-M)$, and the average number of exchanges is one sixth this amount. Suitable refinements of this method will be desirable for its implementation on any actual computer;

```
begin       integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                                quicksort (A,M,J);
                                quicksort (A, I, N)
                          end
end         quicksort
```

Dining philosophers problem (1965)





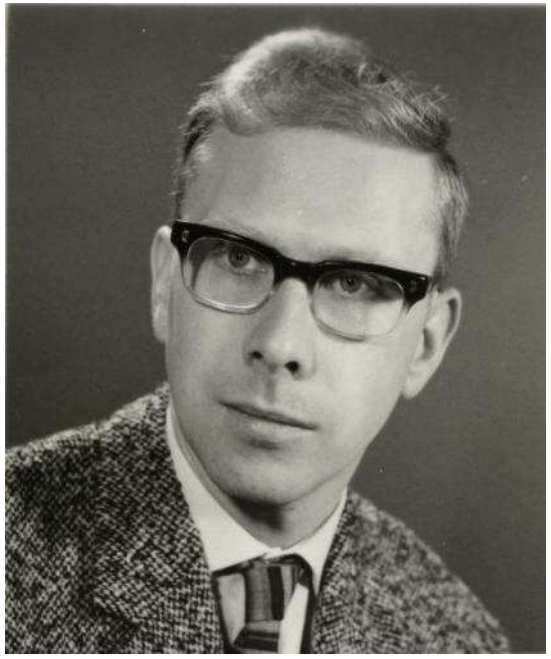
- First formulated by **Edsger Dijkstra** as an exercise of computers competing for tape drive access, then "dining quintet"
- Soon after, **Tony Hoare** renamed it as the "**dining philosophers problem**"
- Four conditions necessary for deadlock:

Negating any of these conditions avoids deadlock

1. *mutual exclusion* (no fork can be simultaneously used by multiple philosophers)
2. *resource holding* (the philosophers hold a fork while waiting for the second)
3. *non-preemption* (no philosopher can take a fork from another), and
4. *circular wait* (each philosopher may be waiting on the philosopher to their left)

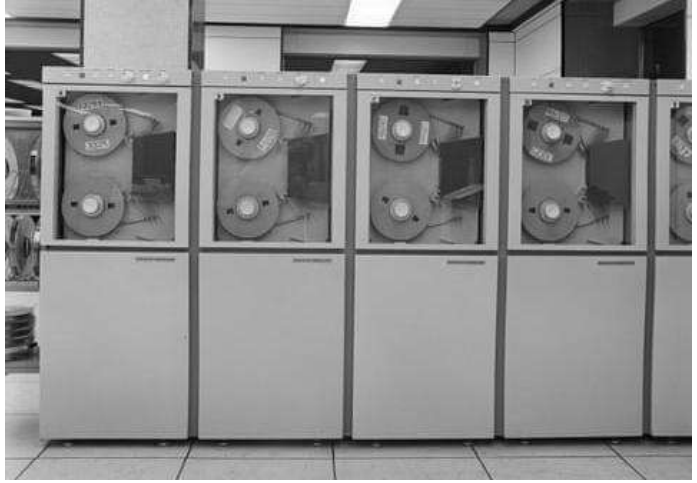
ALGOL X (1965) and ALGOL W (1966)

- **ALGOL X** proposed by **Niklaus Wirth** (1934–2024) and Tony Hoare
- **ALGOL W** then proposed: an upgrade from ALGOL X & ALGOL 60
  - Tony Hoare worked on record structures
  - Included the “null reference” (“billion-dollar mistake”! See later)
- **ALGOL 68** criticized by Tony Hoare and **Edsger Dijkstra** for its complexity



Wirth, Niklaus; Hoare, C. A. R. (1966).
“A contribution to the development of ALGOL”.
Communications of the ACM, 9(6):413–432.
DOI: 10.1145/365696.365702

Elliott Brothers (until 1968)



- 1966: Elliott 4100 operating system
  - Multi-user – implementation became too large
- 1967: Elliott Automation merged with English Electric
- 1968: Merger with International Computers and Tabulators (ICT)
  - Became International Computers Limited (ICL)
  - Tony Hoare moved to academia

Queen’s University Belfast (1968–1977)

- 1968: Professor of Computing Science
- 1969: Hoare logic

Hoare logic – Assertions

Hoare logic: {*pre*} *prog* {*post*}

Program *proving* with pre- and post-conditions as "***assertions***" (logical statements about the program)

30 years later: assertions widely used by programmers for testing and debugging rather than proof

An Axiomatic Basis for Computer Programming. *Communications of the ACM,* 26(10):576–580, 1969. (Republished 1983.) DOI: 10.1145/357980.358001

Aside: First formal methods paper?

Arguably the first “formal methods” paper ever:

Checking a Large Routine, Paper for the EDSAC Inaugural Conference, 24 June **1949**. In *Report of a Conference on High Speed Automatic Calculating Machines,* pp 67–69.

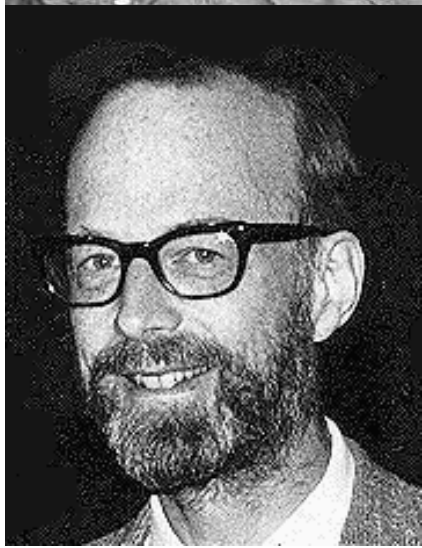
Reprinted with corrections and annotations in:
An early program proof by Alan Turing, L. Morris and C.B. Jones, IEEE Ann. Hist. Computing 6(2):129–143, 1984.

See also: *Turing and Software Verification,* C.B. Jones. Tech. Report CS-TR-1441, Newcastle University, UK, 2014.

— Alan Turing (1912–1954)

Turing's influence on program proving

- **Aad van Wijngaarden** (1916–1987) was at the Cambridge meeting – but no known influence
- **Robert Floyd** (1936–2001) rediscovered ideas similar to those of Turing (published 1967)
- **Tony Hoare** developed these further (published 1969)
- Had Turing lived longer, perhaps formal methods (in particular, program proving) would have developed more rapidly, rather than being rediscovered



REFERENCES

1976 Dijkstra, E. W. 1976. *A Discipline of Programming.* Englewood Cliffs, N.J., Prentice-Hall.

1967 Floyd, R. W. 1967. "Assigning Meaning to Programs." *Proc. of Symposia in Appl. Math. 19.* (Also in S. T. Schwartz (ed.), *Mathematical Aspects of Computer Science*, Providence, American Mathematical Society, 1967.)

1947 Goldstine, H. H., and J. von Neumann. 1947. "Planning and Coding of Problems for an Electronic Computing Instrument." Report of U.S. Ord. Dept. In A. Taub (ed.), *Collected Works* of J. von Neumann, New York, Pergamon, Vol. 5, 1965, pp. 80–151.

1969 Hoare, C. A. R. October 1969. An axiomatic basis for computer programming. *Comm. ACM 12*, 10, 576–580.

1963 McCarthy, J. 1963. "A Basis for a Mathematical Theory of Computation." In P. Braffort and D. Hirschberg (eds.), *Computer Programming and Formal Systems*, Amsterdam, North-Holland, 1967, pp. 33–70.

1966 Naur, P. 1966. Proof of algorithms by general snapshots. *BIT 6*, 4, 310–316.

1949 Turing, A. M. 1949. "Checking a Large Routine." In *Report of a Conference on High Speed Automatic Calculating Machines*, Univ. Math. Lab., Cambridge, pp. 67–69.

20 years after Turing

Hoare logic

Empty statement axiom schema

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

Assignment axiom schema

$$\frac{}{\{P[E/x]\}\ x{:=}E\ \{P\}}$$

where $P[E/x]$ denotes the assertion P in which each free occurrence of x has been replaced by the expression E.

Hoare logic

Composition rule

$$\frac{\{P\}\ S\ \{Q\} \quad , \quad \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

Conditional rule

$$\frac{\{B \wedge P\}\ S\ \{Q\} \quad , \quad \{\neg B \wedge Q\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{endif}\ \{Q\}}$$

Hoare logic

Consequence rule

$$\frac{P_1 \rightarrow P_2 \ , \ \{P_2\}\ S\ \{Q_2\} \ , \ Q_2 \rightarrow Q_1}{\{P_1\}\ S\ \{Q_1\}}$$

Strengthen precondition P_2 and/or weaken postcondition Q_2.

While rule

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\neg B \wedge P\}}$$

where P is the loop *invariant* preserved by the loop body S. After the loop ends, P still holds, and $\neg B$ must have caused the loop to end.

Hoare logic – extension

Hoare logic and total correctness

$[P]\ S\ [Q]$

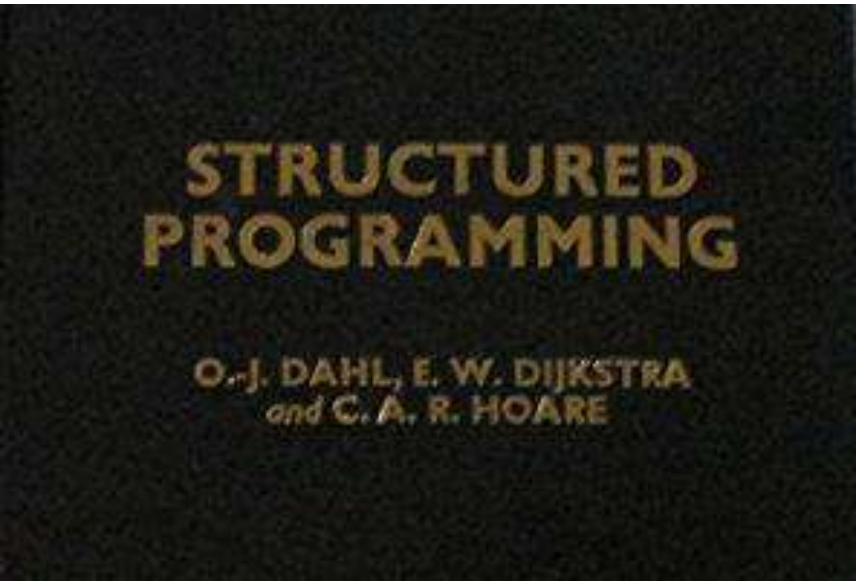
While rule, with total correctness

$$\frac{< \text{well-founded on set } D\ ,\ [P \wedge B \wedge t \in D \wedge t{=}z]\ S\ [P \wedge t \in D \wedge t{<}z]}{[P \wedge t \in D]\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ [\neg B \wedge P \wedge t \in D]}$$

where the loop invariant t strictly decreases with respect to $<$ (e.g., on positive integers).

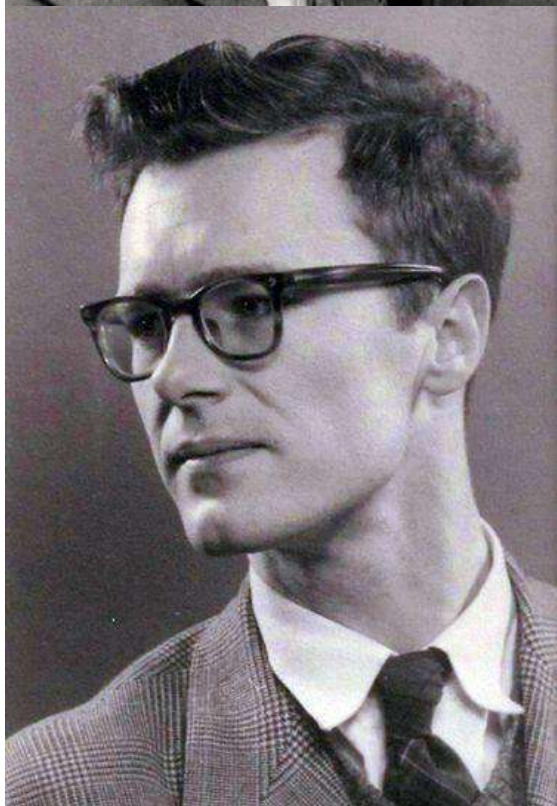
Structured programming

1972: Ole-Johan Dahl, Edsger W. Dijkstra, C. A. R. Hoare, *Structured Programming*. Academic Press. ISBN 978-0-12-200550-3.



1. *Notes on Structure Programming* (E.W.D.)
2. *Notes on Data Structuring** (C.A.R.H.)
3. *Hierarchical Structuring* (O.-J.D.)

* Based on lectures delivered at a Nato Summer School, Marktoberdorf, Germany, 1970.



Monitors (1973–74)

- Invented along with **Per Brinch-Hansen** (1938–2007)
- Synchronization construct in concurrent programming
- Allows waiting for locks – useful in operating systems

```
acquire(m);    // Acquire this monitor's lock.
while (!p) {   // While the predicate being waited for is not true.
        wait(m, cv); // Wait on this monitor's lock and condition variable.
}
// ... Critical section of code goes here ...
signal(cv2);   // cv2 might be the same as cv.
release(m);    // Release this monitor's lock.
```

Monitors: An Operating System Structuring Concept, C.A.R Hoare (1974). *Communications of the ACM*, 17(10):549–557. DOI: 10.1145/355620.361161

Oxford University Computing Laboratory (1977–1999)

- Professor of Computation
- Programming Research Group (PRG)
- Communicating Sequential Processes (CSP, 1978/85)
- European ProCoS projects (1989–1997)
- *Unifying Theories of Programming* (UTP, 1998)

Theory and practice

*"It has long been my personal view that the **separation of practical and theoretical work is artificial** and injurious. Much of the **practical work** done in computing, both in software and in hardware design, is **unsound and clumsy** because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and **theoretical work** is **sterile** because it has no point of contact with real computing."*

— Christopher Strachey (1916–1975)

First leader of the PRG (1963–1975)

Discovered by Tony Hoare on his arrival back in Oxford.

Early days at the PRG (staff and MSc students, 1981)

PhD students

Leading students:

- Cliff Jones (VDM)
- Bill Roscoe (CSP)

Mathematics Genealogy Project

Charles Antony Richard (Tony) Hoare

MathSciNet

PgDip University of Oxford 

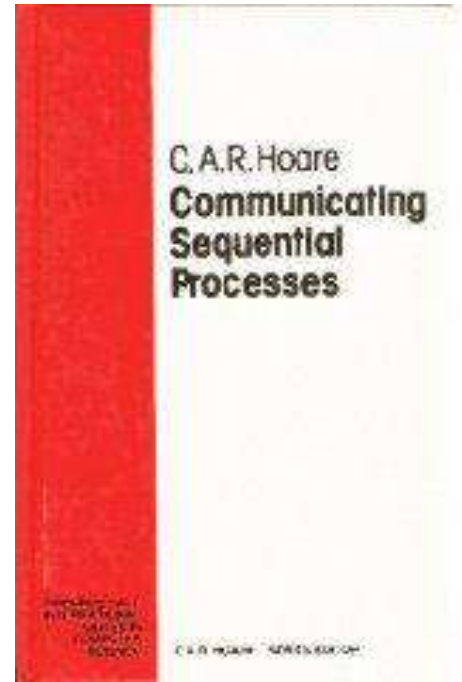
Dissertation:

Advisor 1: Andrei Nikolayevich Kolmogorov
Advisor 2: Leslie Fox

Name	School	Year	Descendants
Lauer, Peter	Queen's University of Belfast	1971	31
Page, Ian	City, University of London	1972	38
Stewart, William	Queen's University of Belfast	1974	16
Elder, John	Queen's University of Belfast	1975	
Malik, Masud	Queen's University of Belfast	1975	
Kaubisch, Jim	Queen's University of Belfast	1976	
Jones, Cliff	University of Oxford	1981	96
Kennaway, Richard	University of Oxford	1981	2
Sørensen, Ib	University of Oxford	1981	
Roscoe, Bill (A. W.)	University of Oxford	1982	101
Jones, Geraint	University of Oxford	1983	
Black, Andrew	University of Oxford	1984	12
Brookes, Stephen	University of Oxford	1984	19
Dollin, Christopher	University of Oxford	1984	
Teruel, Alex	University of Oxford	1985	
Kong, T. Yung	University of Oxford	1986	2
Page, Stephen	University of Oxford	1988	
Todd, Bryan	University of Oxford	1988	
Jacob, Jeremy	University of Oxford	1989	12
Martin, Clare	University of Oxford	1991	
Naumann, David	University of Texas at Austin	1992	6
Wood, Ken	University of Oxford	1992	
Sampaio, Augusto	University of Oxford	1993	7
Brien, Stephen	University of Oxford	1995	
Rudin, Paul	University of Oxford	2001	

Communicating Sequential Processes (CSP)

- **1978** paper: *Communications of the ACM*
  - Influenced by ALGOL 60 and Dijkstra's guarded commands
  - Concurrent programming language
- **1985** book: Prentice Hall International Series in Computer Science (also series editor)
  - Process calculus/algebra
  - Work with Stephen Brookes and Bill Roscoe
  - Textbook for computer science students
- Inspired the Occam programming language



Communicating Sequential Processes (CSP)

Syntax:

$$
\begin{array}{rcll}
Proc & ::= & \text{STOP} & \\
& | & \text{SKIP} & \\
& | & e \rightarrow Proc & \text{(prefixing)} \\
& | & Proc \,\Box\, Proc & \text{(external choice)} \\
& | & Proc \,\sqcap\, Proc & \text{(nondeterministic choice)} \\
& | & Proc \,|||\, Proc & \text{(interleaving)} \\
& | & Proc \,|[\{X\}]|\, Proc & \text{(interface parallel)} \\
& | & Proc \setminus X & \text{(hiding)} \\
& | & Proc; Proc & \text{(sequential composition)} \\
& | & \text{if } b \text{ then } Proc \text{ else } Proc & \text{(boolean conditional)} \\
& | & Proc \,\triangleright\, Proc & \text{(timeout)} \\
& | & Proc \,\triangle\, Proc & \text{(interrupt)}
\end{array}
$$

Semantics: compatible *denotational* semantics, *algebraic* semantics, and *operational* semantics.

Communicating Sequential Processes (CSP)

Denotational semantics: *traces* model, *stable failures* model, and *failures/divergences* model.

- Partial order of refinement on processes:
  $P \sqsubseteq Q$ denotes Q refines P
- *Traces* model: set of sequences of events (traces) that processes can be observed to perform
- *Stable failures* model: Traces model with refusal sets (events that a process will not perform)
- *Failures/divergences* model: Failures model with divergence (non-termination or termination in an exceptional state)

Communicating Sequential Processes (CSP)

Algebraic laws (with Bill Roscoe)

$$P \,\Box\, P = P$$
$$P \,\Box\, Q = Q \,\Box\, P$$
$$P \,\Box\, (Q \,\Box\, R) = (P \,\Box\, Q) \,\Box\, R$$
$$P \,\Box\, (Q \sqcap R) = (P \,\Box\, Q) \sqcap (P \,\Box\, R)$$
$$P \sqcap (Q \,\Box\, R) = (P \sqcap Q) \,\Box\, (P \sqcap R)$$
$$P \,\Box\, STOP = P$$
$$(a \rightarrow (P \sqcap Q)) = (a \rightarrow P) \sqcap (a \rightarrow Q)$$
$$(a \rightarrow P) \,\Box\, (a \rightarrow Q) = (a \rightarrow P) \sqcap (a \rightarrow Q)$$
$$P \sqcap P = P$$
$$P \sqcap Q = Q \sqcap P$$
$$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$$
$$P \parallel Q = Q \parallel P$$
$$P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$$
$$P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R)$$
$$(a \rightarrow P) \parallel (b \rightarrow Q) = STOP \qquad \text{if } a \neq b$$
$$= (a \rightarrow (P \parallel Q)) \qquad \text{if } a = b$$
$$P \parallel STOP = STOP \qquad \text{if } P \neq \bot$$
$$= \bot \qquad \text{if } P = \bot$$

$$P \,|||\, Q = Q \,|||\, P$$
$$(P \,|||\, Q) \,|||\, R = P \,|||\, (Q \,|||\, R)$$
$$P \,|||\, (Q \sqcap R) = (P \,|||\, Q) \sqcap (P \,|||\, R)$$
$$(a \rightarrow P) \,|||\, (b \rightarrow Q) = (a \rightarrow (P \,|||\, (b \rightarrow Q))) \,\Box\, (b \rightarrow ((a \rightarrow P) \,|||\, Q))$$
$$P ; (Q ; R) = (P ; Q) ; R$$
$$STOP \,|||\, Q = Q$$
$$SKIP ; Q = Q$$
$$STOP ; Q = STOP$$
$$P ; (Q \sqcap R) = (P ; Q) \sqcap (Q ; R)$$
$$(P \sqcap Q) ; R = (P ; R) \sqcap (Q ; R)$$
$$(a \rightarrow P) ; Q = (a \rightarrow (P ; Q)) \qquad \text{if } a \neq \surd$$
$$(P \setminus a) \setminus b = (P \setminus b) \setminus a$$
$$(P \setminus a) \setminus a = P \setminus a$$
$$(a \rightarrow P) \setminus b = (a \rightarrow (P \setminus b)) \qquad \text{if } a \neq b$$
$$= P \setminus b \qquad \text{if } a = b$$
$$(P \sqcap Q) \setminus a = (P \setminus a) \sqcap (Q \setminus a)$$

Trip to China (April 1983)

Tony and Jill Hoare, and family, sitting in front of the Nine-Dragon wall in Beijing

Arrival of He Jifeng from China (1983)

At the Chinese embassy in London:

He Jifeng,
Bernard Sufrin,
Tony and Jill Hoare

An important collaborator for Tony Hoare from 1983 to 1998

Inaugural lecture, Oxford (17 October 1985*)

The Mathematics of Programming, Clarendon Press 1986.

Axioms for programs in algebraic form. E.g.:

$$x \,;\, (y \,;\, z) \;=\; (x \,;\, y) \,;\, z$$

$$\text{SKIP} \,;\, x \;=\; x \;=\; x \,;\, \text{SKIP}$$

New undergraduate degree in Computing

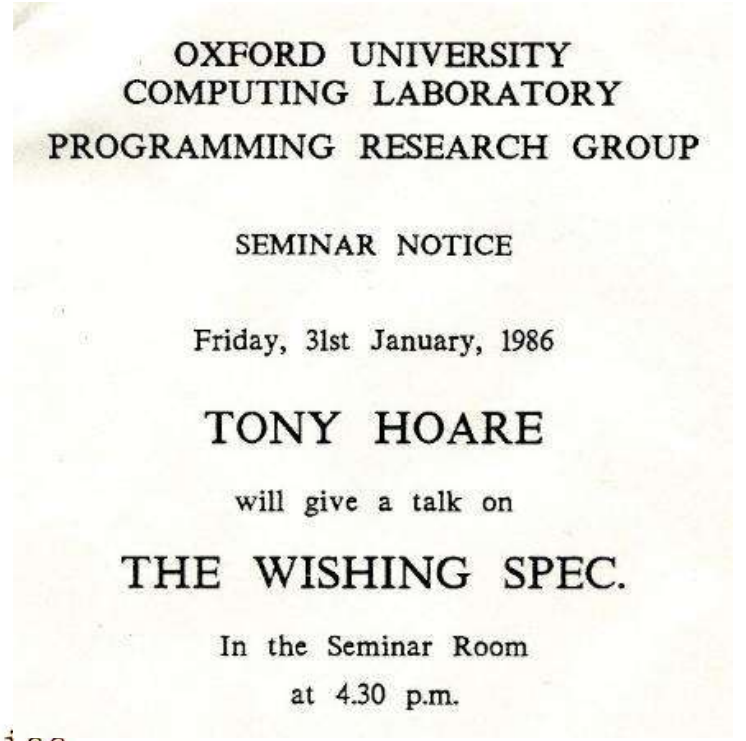
Mr Vice-Chancellor, Ladies and Gentlemen!

This is my inaugural lecture as Professor of Computation at Oxford University. I was appointed to this post just nine years ago, after the tragically premature death of its brilliant first occupant, Christopher Strachey. Nine years is a long delay for an inaugural lecture; but it has taken all those nine years to introduce an undergraduate curriculum in Computing at Oxford. Although many universities had been producing graduates in this subject for many years before I was appointed here, it is only this week that we welcome to Oxford and to this lecture the first entrants to our new Honour School in Mathematics and Computation.

So it is the new School rather than myself that I wish to inaugurate today. I shall do so by describing some of the research goals pursued by Christopher Strachey and his colleagues and successors in the Programming

* Slightly delayed, having joined in 1977!

The Wishing Spec (1986)



OXFORD UNIVERSITY
COMPUTING LABORATORY
PROGRAMMING RESEARCH GROUP

SEMINAR NOTICE

Friday, 31st January, 1986

TONY HOARE

will give a talk on

THE WISHING SPEC.

In the Seminar Room
at 4.30 p.m.

The Wishing Spec

C.A.R. Hoare. Jan. 1986

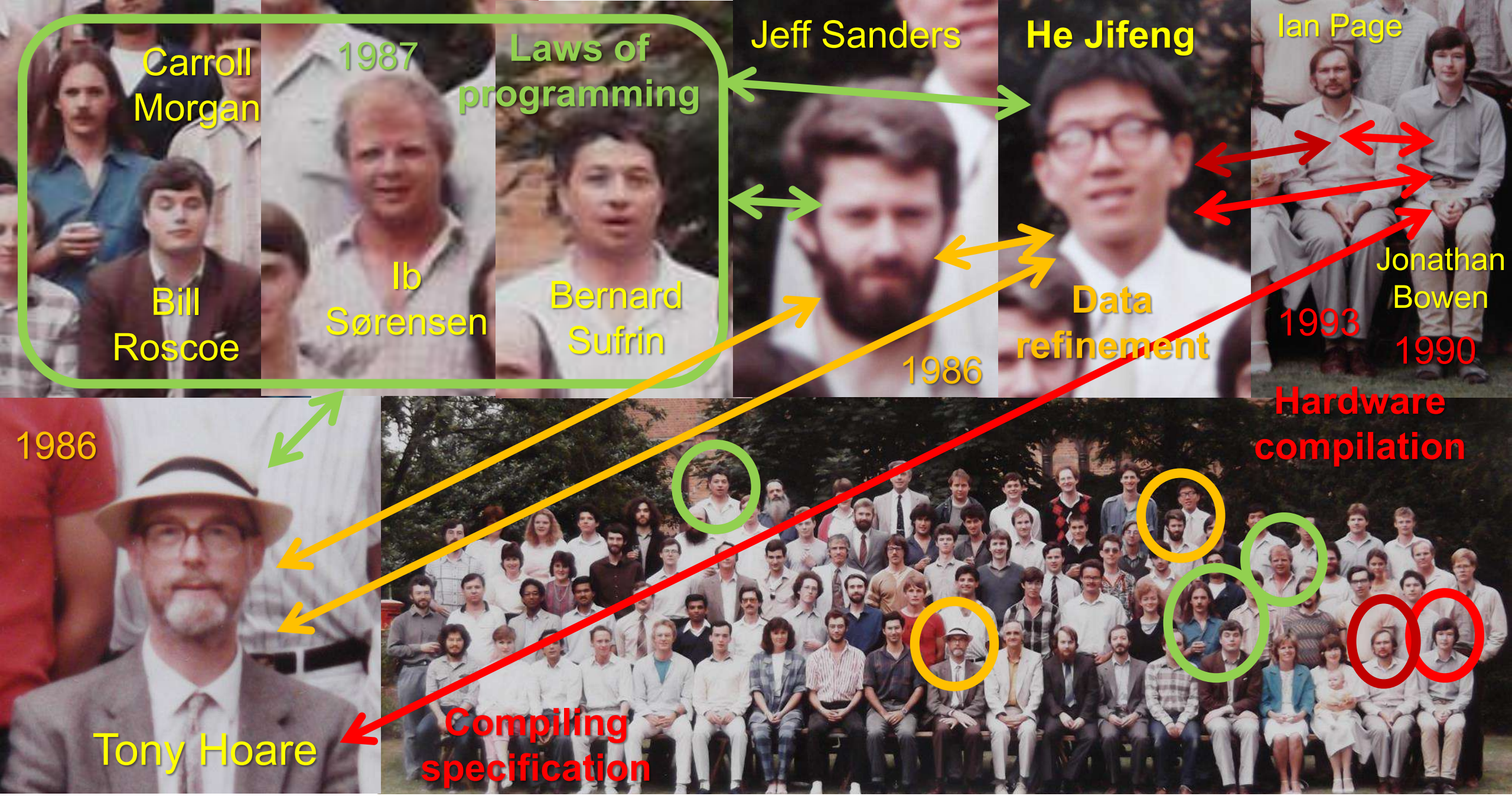
A wishing spec is a specification which pays no regard to feasibility or economics of implementation. Imagine an organisation in which every office worker uses a sophisticated graphics display connected to a network of computers which store and process all the information relevant to the organisation. The computer system is imagined to possess a number of desirable properties. For the logical behaviour of the system, these properties are

visibility, extensibility, correctibility, accountability, consistency, security, decision support and unprogrammability. For the implementation, desirable properties are

non-stop operation, extensibility, fireproofing, distribution, instantaneous response, reliability, economy and feasibility.

Cf. the World Wide Web, launched three years later!

Garden party, Oxford University Computing Laboratory, Keble Road, Oxford, July 1986



Co-authors, Oxford University Computing Laboratory

Laws of Programming (1987)

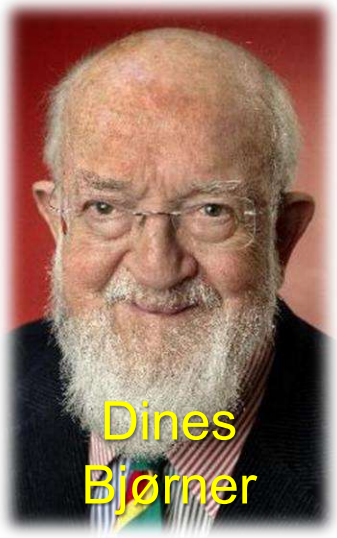
Hoare, C. A. R., Ian J. Hayes, He Jifeng, Carroll C. Morgan, A. William Roscoe, Jeff W. Sanders, Ib Holm Sørensen, J. Michael Spivey, and Bernard A. Sufrin. **Laws of programming**. *Communications of the ACM,* 30(8):672–686 (1987). DOI: 10.1145/27651.27653

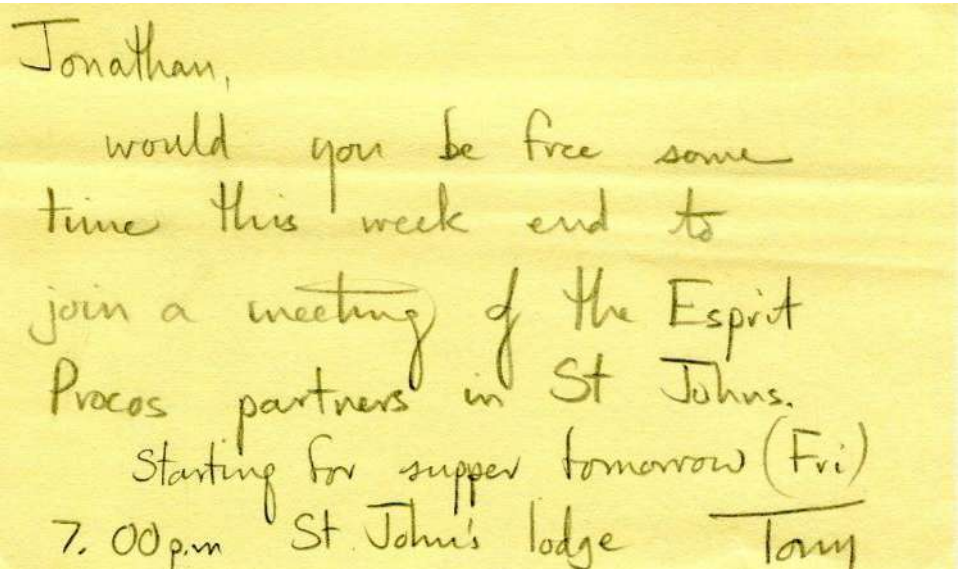
- Complete set of algebraic laws for Dijkstra's nondeterministic sequential programming language
- Interaction and recursion using Dana Scott's domain theory in terms of fixed points
- Calculus (cf. weakest preconditions) for deriving programs from specifications

ProCoS: Provably Correct Systems

European ESPRIT projects and Working Group (1989–1997)

- Requirements
- Specification
- Design
- Programming
- Compilation
- Hardware





Jonathan,
would you be free some time this week end to join a meeting of the Esprit Procos partners in St Johns.
Starting for supper tomorrow (Fri) 7.00 p.m St John's lodge Tony

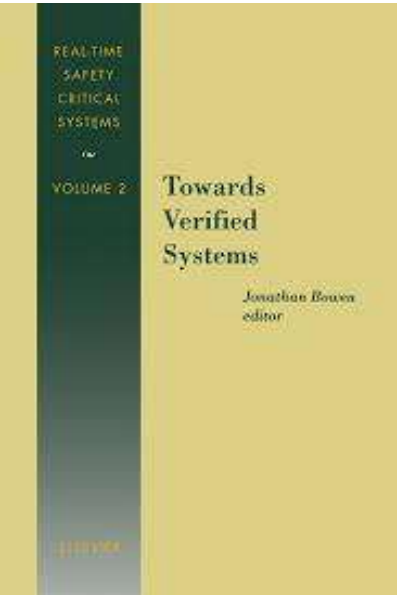
Stop press:
Retrospective multi-author paper in *Formal Aspects of Computing* journal (accepted March 2026).
DOI: 10.1145/3803555

safemos

SAFEMOS: Totally Verified Systems

UK IED (Information Engineering Directorate) project (1989–1993)

- Oxford (Tony Hoare)
- Cambridge (Mike Gordon, 1948–2017)
- INMOS, Bristol (David May)
- Based around Occam and the Transputer



Towards Verified Systems, Jonathan Bowen (ed.), Elsevier, Real-Time Safety Critical Systems, 1994. Foreword by Tony Hoare.

Queens Award for Technological Achievement (1990)

Occam transformation system and the verification of the T800 Transputer FPU (Floating Point Unit)

AN ALGEBRAIC APPROACH TO VERIFIABLE COMPILING SPECIFICATION AND PROTOTYPING OF THE PROCOS LEVEL 0 PROGRAMMING LANGUAGE

C.A.R. Hoare He Jifeng Jonathan Bowen Paritosh Pandya

Oxford University Computing Laboratory
Programming Research Group
Oxford OX1 3QD
England
Phone: +44-865-273838 Fax: +44-865-273839
E-mail: procos@prg.oxford.ac.uk

SUMMARY. A compiler is specified by a description of how each construct of the source language is translated into a sequence of object code instructions. The meaning of the object code can be defined by an interpreter written in the source language itself. A proof that the compiler is correct must show that interpretation of the object code is at least good (for any relevant purpose) as the corresponding source program. The proof is conducted using standard techniques of data refinement. All the calculations are based on algebraic laws governing the source language. The theorems are expressed in a form close to a logic program, which may used as a compiler prototype, or a check on the results of a particular compilation. A subset of the occam programming language and the transputer instruction set are used to illustrate the approach. An advantage of the method is that it is possible to add new programming constructs without affecting existing development work.

1. Introduction

Compilation is specified as a relation between a source program p and the corresponding object code c. Further details of compilation are given by a symbol table Ψ, mapping the global identifiers of p to storage locations of the target machine. This compilation relation will be abbreviated as a predicate

$$C\,p\,c\,\Psi \quad .$$

The internal structure of p, c and Ψ will be elaborated as the need arises.

Improvement is a relation between a product q and a product p that holds whenever for any purpose the observable behaviour of q is as good as or better than that of p. More precisely, if q satisfies every specification satisfied by p, and maybe more. For example, a

European ESPRIT ProCoS project on *Provably Correct Systems* (1989–92)

DOI: 10.1007/978-94-009-0705-8_65

Compiling specification (1990)

ESPRIT ProCoS Basic Research Action (1989–1992)

Compiling relation C: $C\,p\,c\,\Psi \stackrel{def}{=} \hat{\Psi}; \hat{p} \sqsubseteq \hat{c}; \hat{\Psi}$

- p is the high-level program
- c is the machine code
- Ψ is the mapping of variables (e.g., x) to store locations

Example theorems:

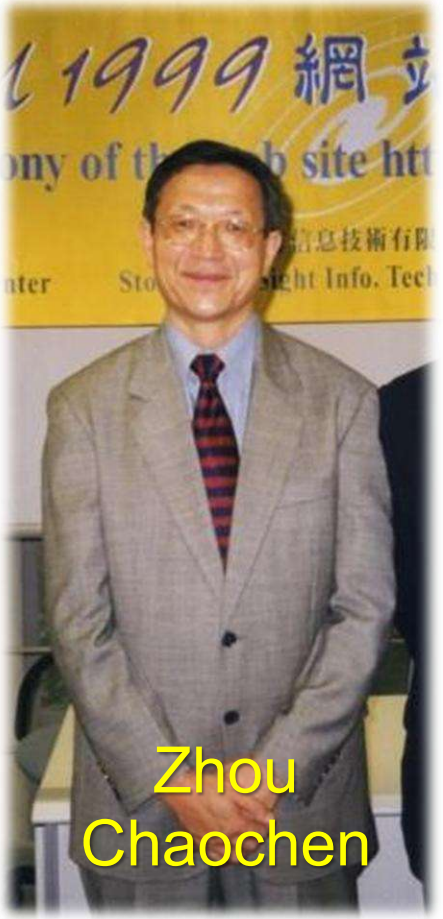
$C\,(x := y)\,\langle \mathtt{load}\Psi\, y, \mathtt{store}\Psi\, x\rangle\,\Psi$

$C\,\mathtt{SKIP}\,\langle\rangle\,\Psi$

$C\,(p1; p2)\,(c1 \frown c2)\,\Psi$ whenever $C\,p1\,c1\,\Psi$ and $C\,p2\,c2\,\Psi$

Duration Calculus (1991)

Zhou Chaochen, C. A. R. Hoare and Anders P. Ravn, **A Calculus of Durations**, *Information Processing Letters*, 40(5):269–276, 1991.
DOI: 10.1016/0020-0190(91)90122-X

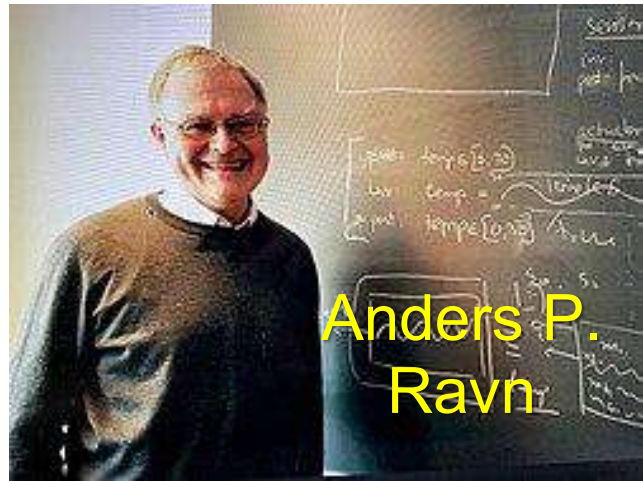


Gas burner example (requirement and design):

Req-1 $$e - b \geqslant 60 \text{ sec.} \Rightarrow 20 \int_b^e Leak(t)\ \mathrm{d}t \leqslant (e - b).$$

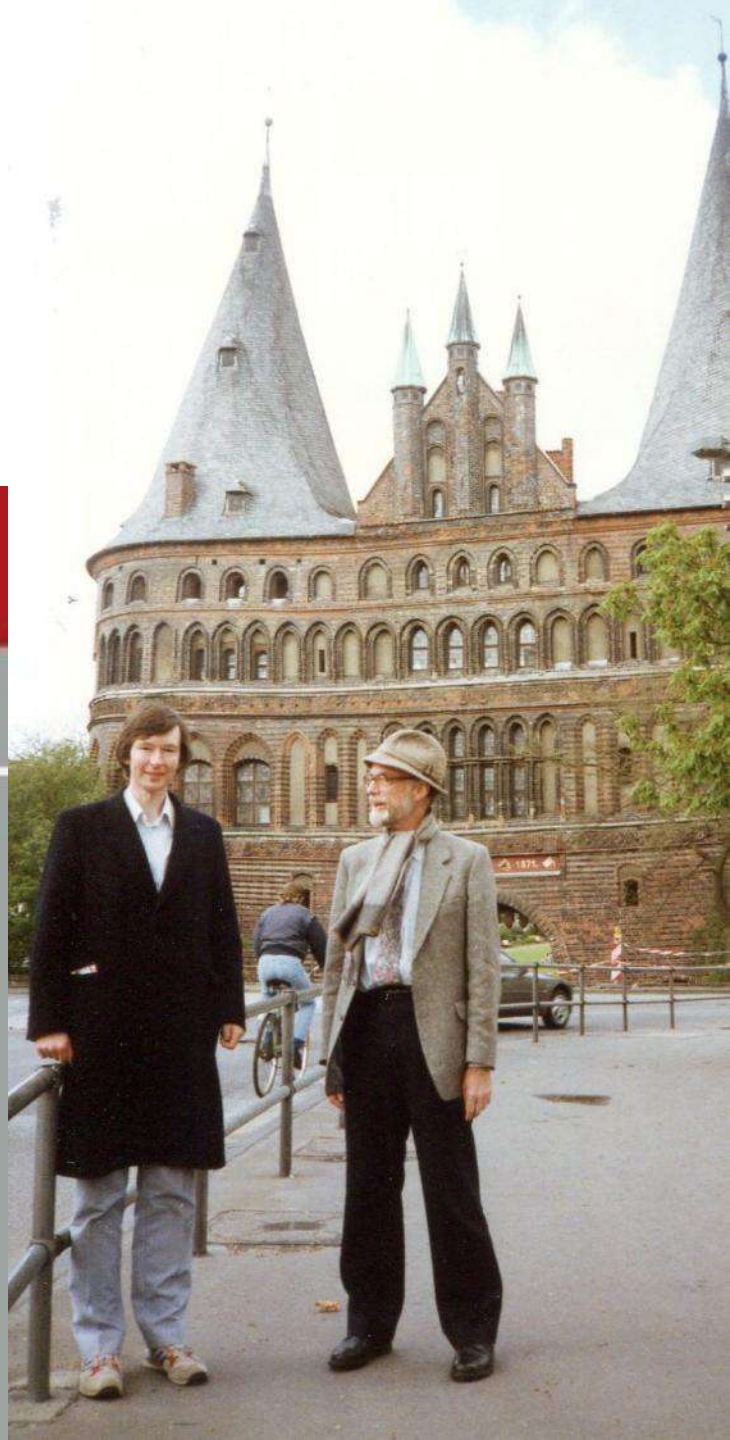
Des-1 $$\int_e^f Leak(t)\ \mathrm{d}t = (f - c) \Rightarrow f - c \leqslant 1 \text{ sec.}$$

Des-2 $$Leak(c) \wedge Leak(f) \wedge (\exists t \cdot c < t < f \wedge \neg Leak(t)) \Rightarrow 30 \text{ sec.} \leqslant f - c.$$



FTRTFT conference, Lübeck, Germany (1994)

European ESPRIT ProCoS II project (1992–1995)



Jifeng He, C.A.R. Hoare, *et al.* (1994). *Provably Correct Systems*. In: H. Langmaack, et al. (eds) *Formal Techniques in Real-Time and Fault-Tolerant Systems. FTRTFT ProCoS* 1994. LNCS, vol. 863. Springer. DOI; 10.1007/3-540-58468-4_171

ProCoS Working Group

University of Reading, UK, 1997

A ProCoS-WG Working Group Final Report: ESPRIT Working Group 8694, Jonathan P. Bowen, C.A.R. Hoare, Hans Langmaack, Ernst-Rüdiger Olderog and Anders P. Ravn. *Bulletin of the European Association for Theoretical Computer Science (EATCS),* **64**:63–72, February 1998.

Dines Bjørner

Tony
13.XI

GIVEN.

OCC a programming notation

spec: OCC → CSP its semantics

VLSI a VLSI design notation

behav: VLSI → CSP its semantics.

FIND

M: VLSI

and code: OCC → traces (behav (M))

such that for all D: OCC

behav (M)/code (D) ≤ spec (D)

(i.e. code is loaded only once at the beginning)

Hoare
87

Le Port d'Oye, Belgium
"Friday the 13th"

75

? Accountability.

· Machine must output current state on req.

Let tr = s^<req>^t^<qer> , ¬{req,qer} in t

then behav(M)/tr = behav(M)/s

≤ behav(M)/load(t)

where load = code∘dumpcrack.

FIND dumpcrack: dump → OCC.

"I'm still trying to understand this."

I'm still trying to understand this.

Tony Hoare 9 March 2015

ProCoS reunion meeting,
London, UK, 2015

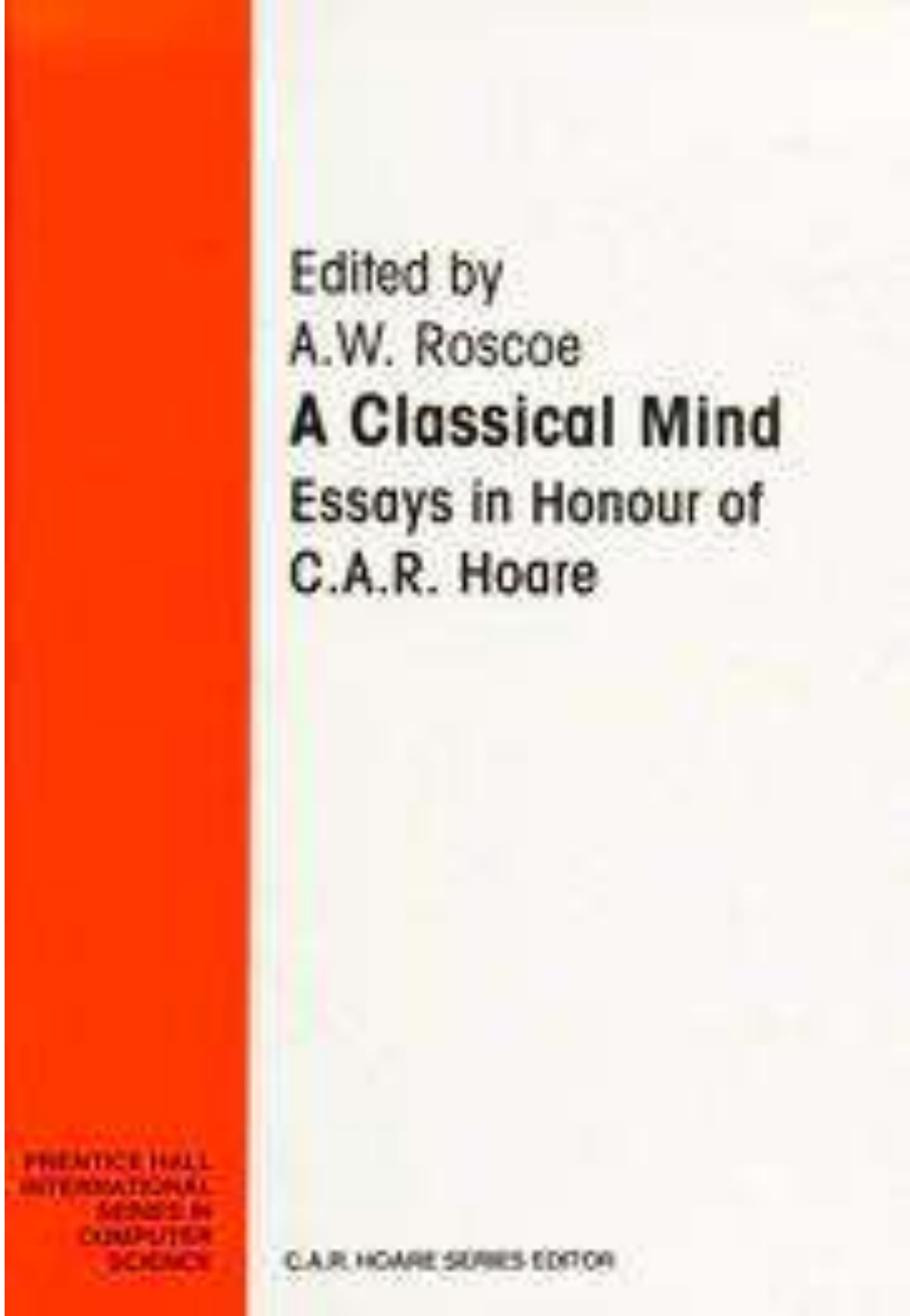
Tony Hoare – notebook

A Classical Mind (1994)

A Classical Mind: Essays in Honour of C.A.R. Hoare. **A.W. Roscoe** (ed). Prentice Hall International Series in Computer Science (1994).

Series editor: Tony Hoare.

25 contributed chapters



Unified theory? Cf. physics

"The construction of a single mathematical model obeying an elegant set of algebraic laws is a significant intellectual achievement; so is the formulation of a set of algebraic laws characterising an interesting and useful set of models."

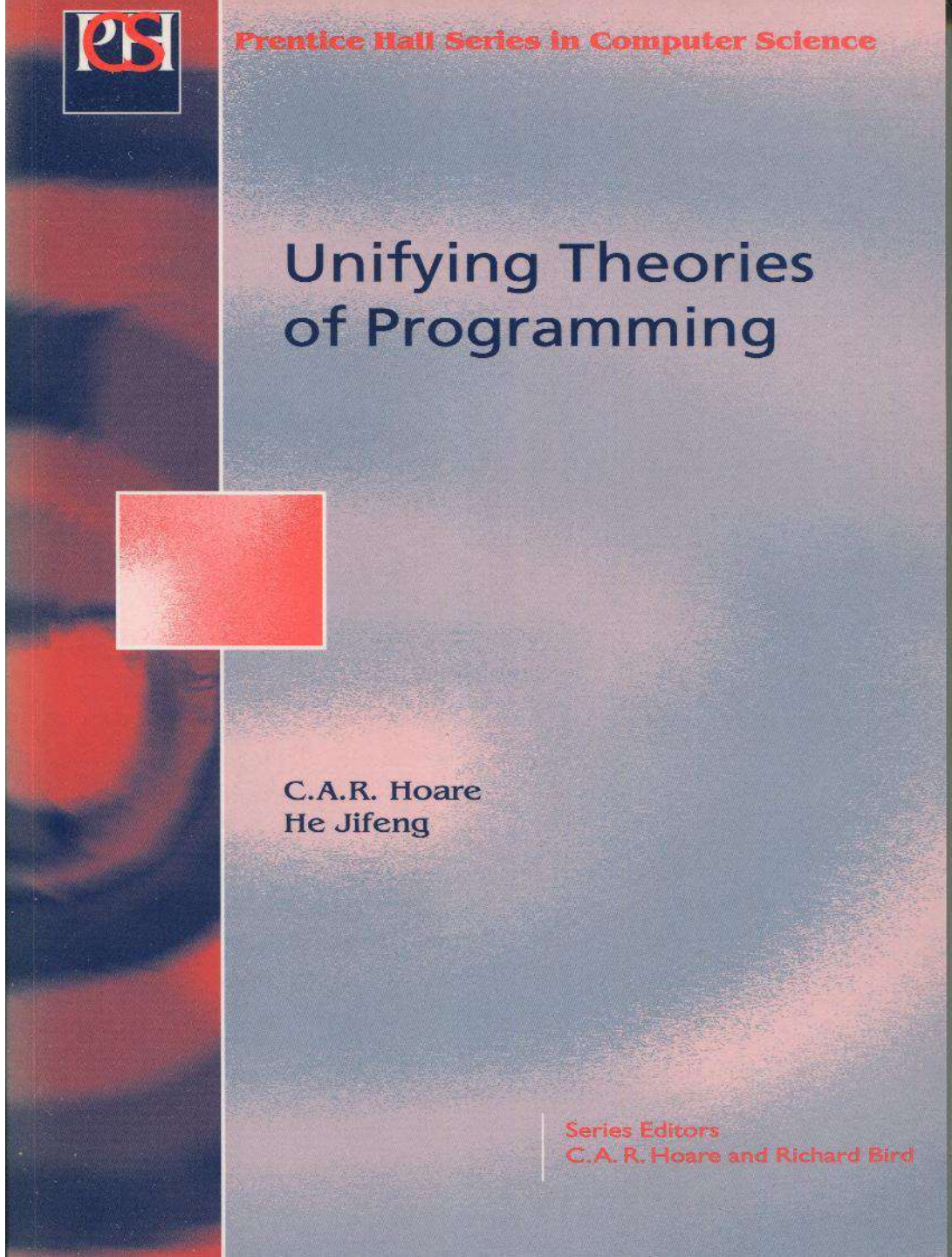
— Sir Tony Hoare, 1993

Operational, Denotational, Algebraic semantics

Unifying Theories of Programming (1998)

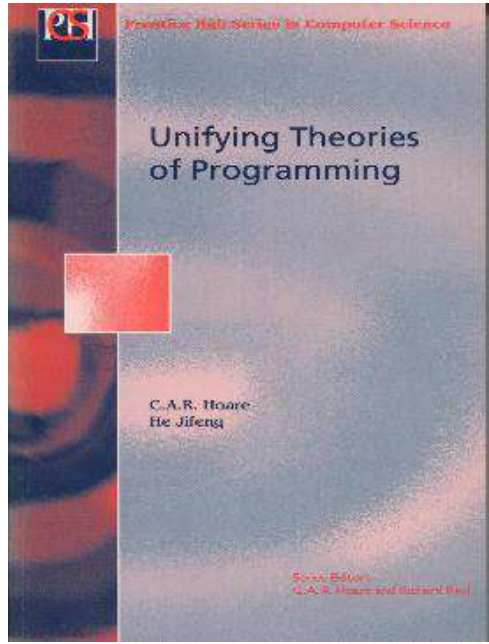
- Magnum opus
- Tony Hoare & Jifeng He
- From ideas during ProCoS

Tony Hoare: *"I must emphasise that all the effective research was conducted by Jifeng, who formalized the definitions, postulated the axioms, and proved the theorems. I enjoyed discussing the goal of research with him, and I wrote much of the English prose. But all of the new results were due to him."*



Unifying Theories of Programming (UTP)

- Combines *denotational*, *operational*, and *algebraic* semantics in a unified framework
- Formal specification, design, and implementation of programs and computer systems
- First-order predicate calculus, augmented with fixed-point constructs from second-order logic
- “Programs are predicates” (cf. Eric Hehner)
- A UTP *theory* has 1) an *alphabet*, 2) a *signature*, and 3) a collection of *healthiness conditions*

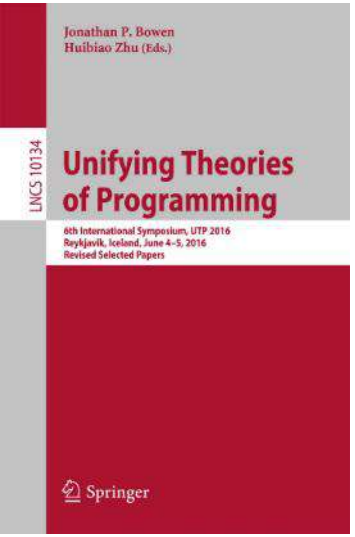


Unifying Theories of Programming

Refinement in all UTP theories:

$$P_1 \sqsubseteq P_2 \quad \text{if and only if} \quad [P_2 \Rightarrow P_1]$$

$[X]$ is the universal closure of all free variables



See right for common language constructs:

UTP symposia started 2006

- The skip statement, which does not alter the program state in any way, is modelled as the relational identity:

$$\mathbf{skip} \equiv v' = v$$

- The assignment of value E to a variable a is modelled as setting a' to E and keeping all other variables (denoted by u) constant:

$$a := E \equiv a' = E \wedge u' = u$$

- The sequential composition of two programs is just relational composition of intermediate state:

$$P_1 ; P_2 \equiv \exists v_0 \bullet P_1[v_0/v'] \wedge P_2[v_0/v]$$

- Non-deterministic choice between programs is their greatest lower bound:

$$P_1 \sqcap P_2 \equiv P_1 \vee P_2$$

- Conditional choice between programs is written using infix notation:

$$P_1 \lhd C \rhd P_2 \equiv (C \wedge P_1) \vee (\neg C \wedge P_2)$$

- A semantics for recursion is given by the least fixed point $\mu\mathbf{F}$ of a monotonic predicate transformer $\mathbf{F}$:

$$\mu X \bullet \mathbf{F}(X) \equiv \sqcap \{X \mid \mathbf{F}(X) \sqsubseteq X\}$$

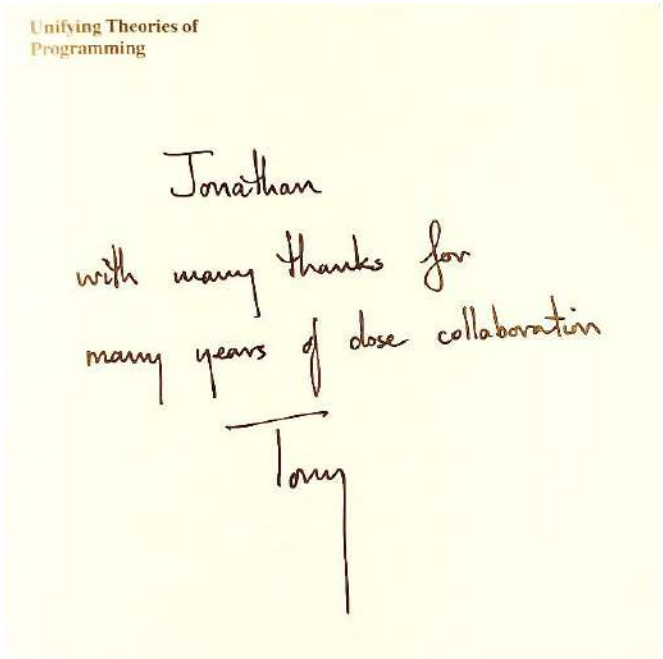
“Retirement” from Oxford (1999)

Many A.M. Turing Award winners attended (13–15 September 1999)

“Retirement” from Oxford (1999)

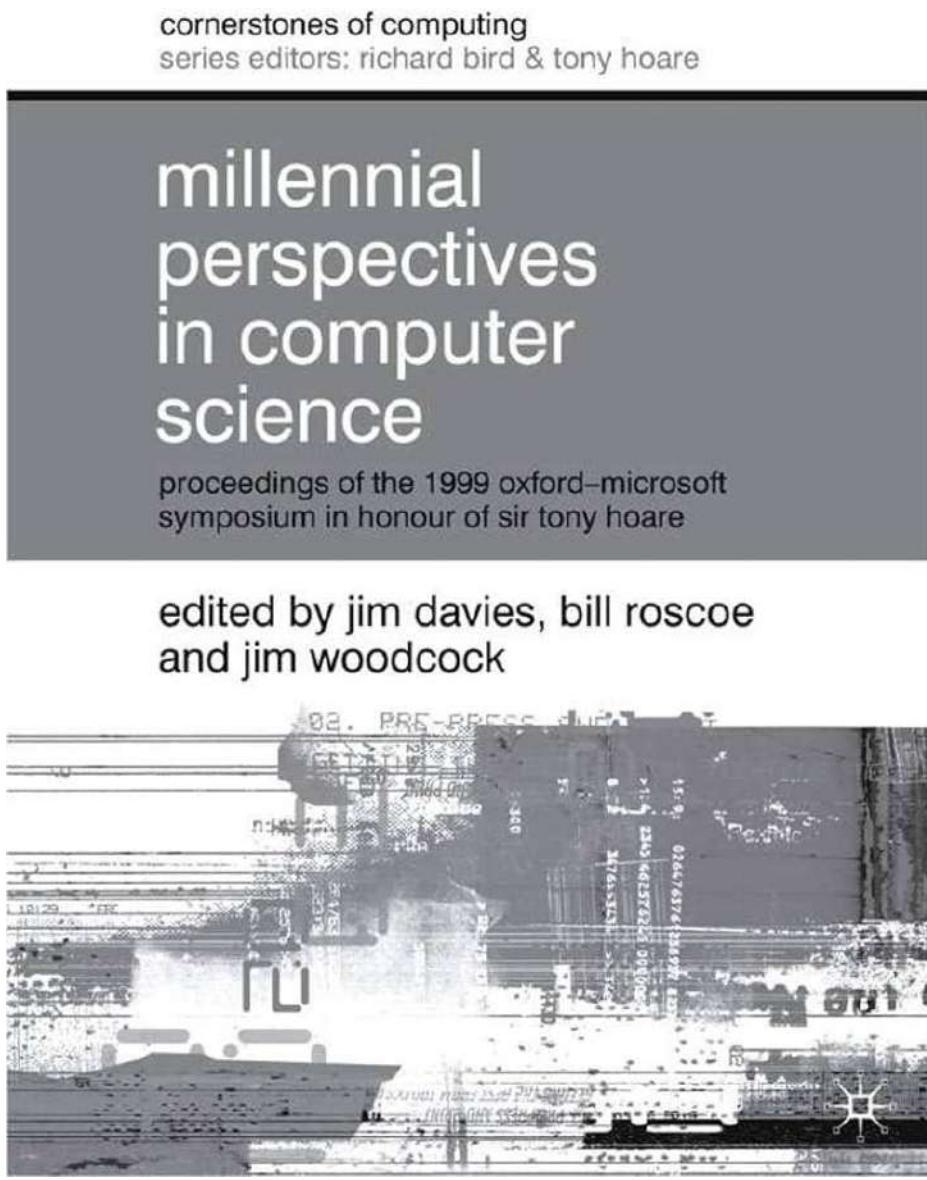
Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare. Jim Davies, Bill Roscoe, Jim Woodcock (eds). Palgrave, Cornerstones of Computing (2000).

Series editors: Richard Bird & Tony Hoare.

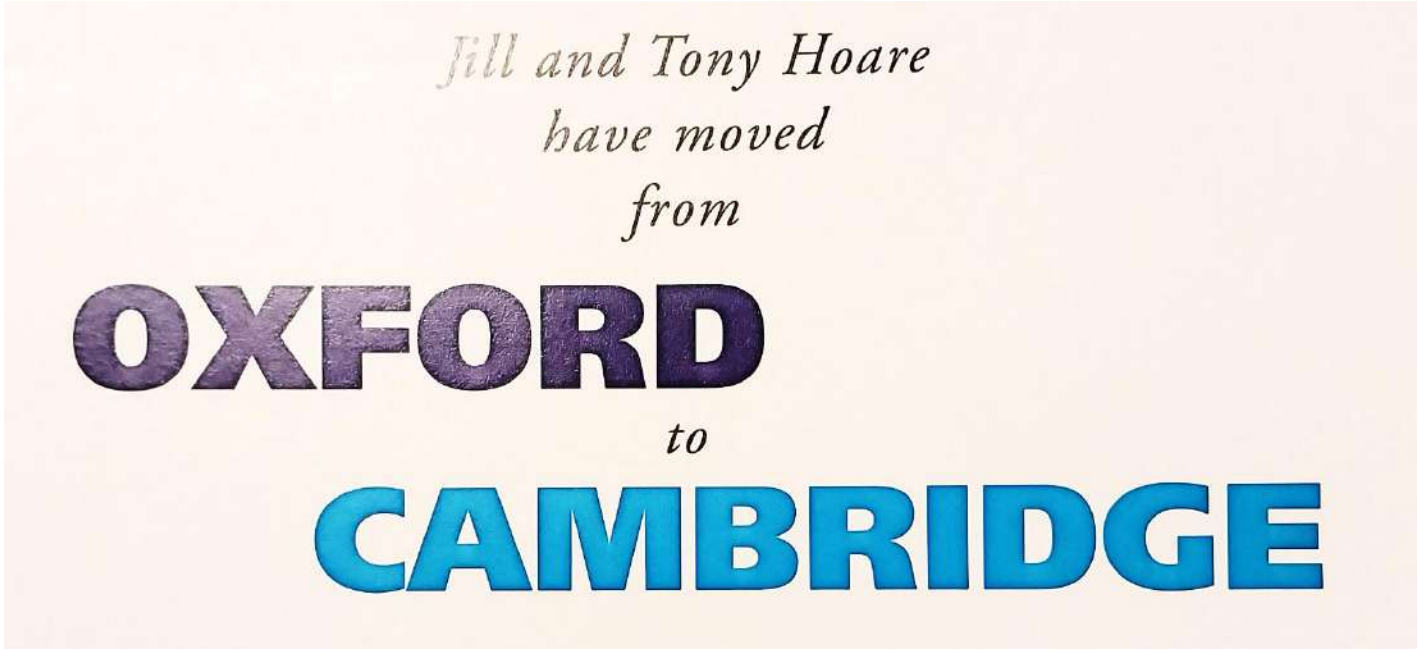


Many A.M. Turing Award winners contributed!

31 chapters



Microsoft Research Cambridge (1999)



Discovered that *assertions* from his 1969 paper were being widely used for program testing rather than proof

Had to start using email!

The Verifying Compiler: A Grand Challenge (2003)

Abstract. This contribution proposes a set of criteria that distinguish a grand challenge in science or engineering from the many other kinds of short-term or long-term research problems that engage the interest of scientists and engineers. As an example drawn from Computer Science, it revives an old challenge: the construction and application of a verifying compiler that guarantees correctness of a program before running it.

The Verifying Compiler: A Grand Challenge for Computing Research, C.A.R. Hoare (2003). *Journal of the ACM*, 50(1):63–69.
DOI: 10.1145/602382.602403

Interview with Tony Hoare (2006)

- Interview at Microsoft Research Cambridge for the Computer History Museum, California
- *Oral History of Sir Antony Hoare*. Interviewed by Jonathan P. Bowen. Recorded September 8, 2008, Cambridge, UK. https://archive.computerhistory.org/resources/text/Oral_History/Hoare_Sir_Antony/102658017.05.01.pdf
- An Interview With C.A.R. Hoare. *Communications of the ACM*, 52(3):38–41, 2009. DOI: 10.1145/1467247.1467261

CHM Computer History Museum

“Billion-dollar mistake” (1965 – 2009)

At a software conference in 2009:

I call it my **billion-dollar mistake**. It was the invention of the **null reference** in 1965. At that time, I was designing the first comprehensive type system for references in an object-oriented language (**ALGOL W**). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

BCS-FACS Christmas meeting (2012)

BCS-FACS: British Computer Society Formal Aspects of Computing Science Specialist Group

Peter Landin Annual Semantics Seminar by Tony Hoare, introduced by **Peter O'Hearn**, 3 December 2012

Frames rule for parallel computation

$$\frac{\{P\}\ Q\ \{R\}}{\{P \parallel F\}\ Q\ \{R \parallel F\}}$$

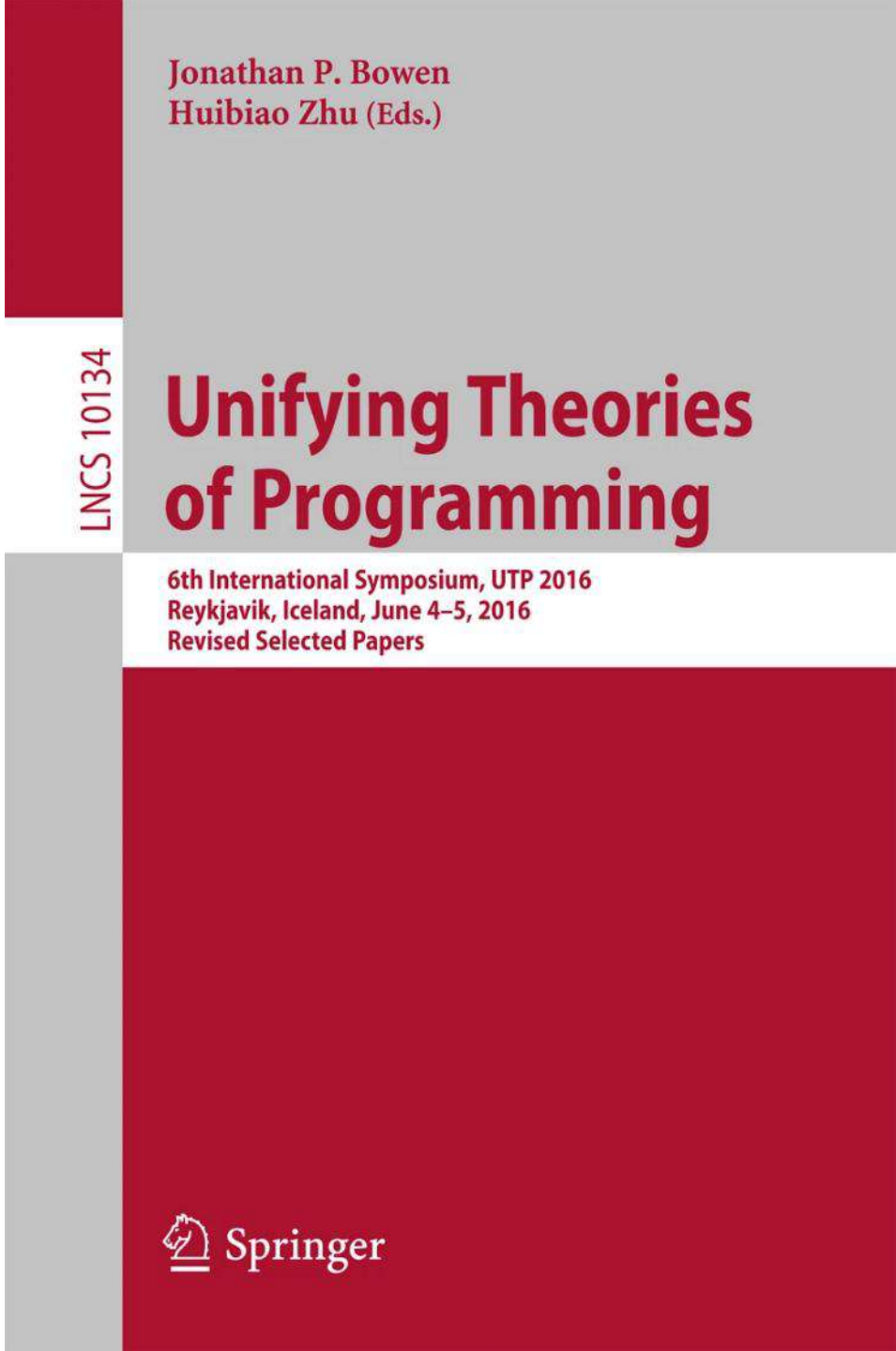
Unifying Theories of Programming symposia

UTP international symposium series

- First symposium 2006, Durham, UK

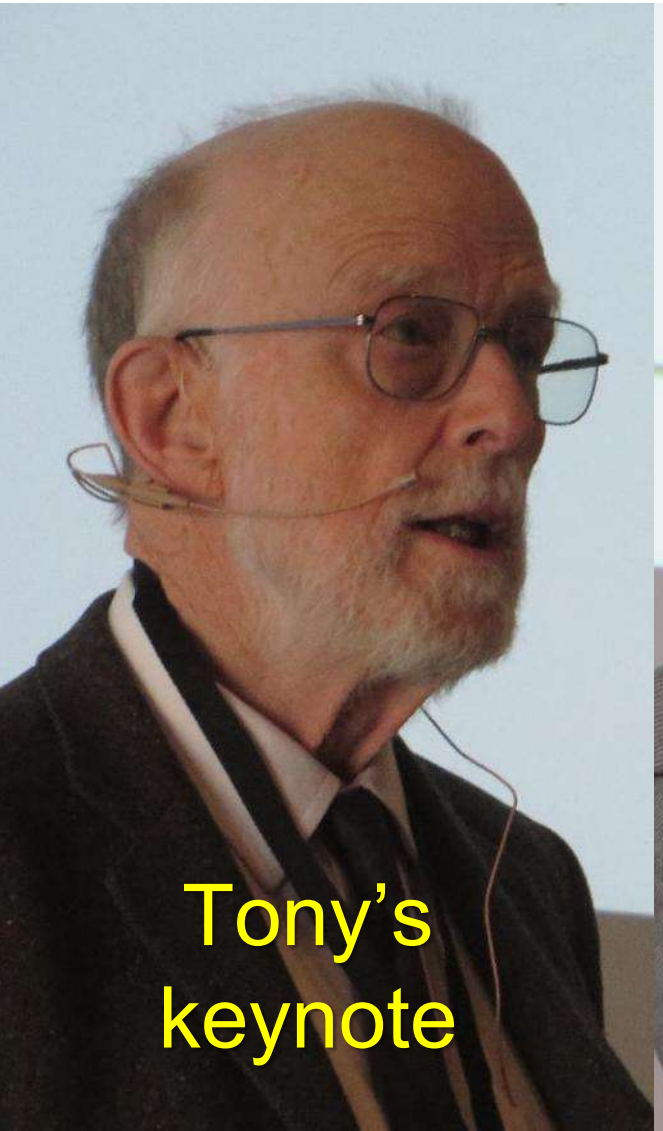
UTP 2016 in Reykjavik, Iceland

- Tony Hoare & Jifeng He both presented keynote talks
- DOI: 10.1007/978-3-319-52228-9



UTP 2016, Reykjavik – group photograph

UTP 2016, Reykjavik – photographic memories



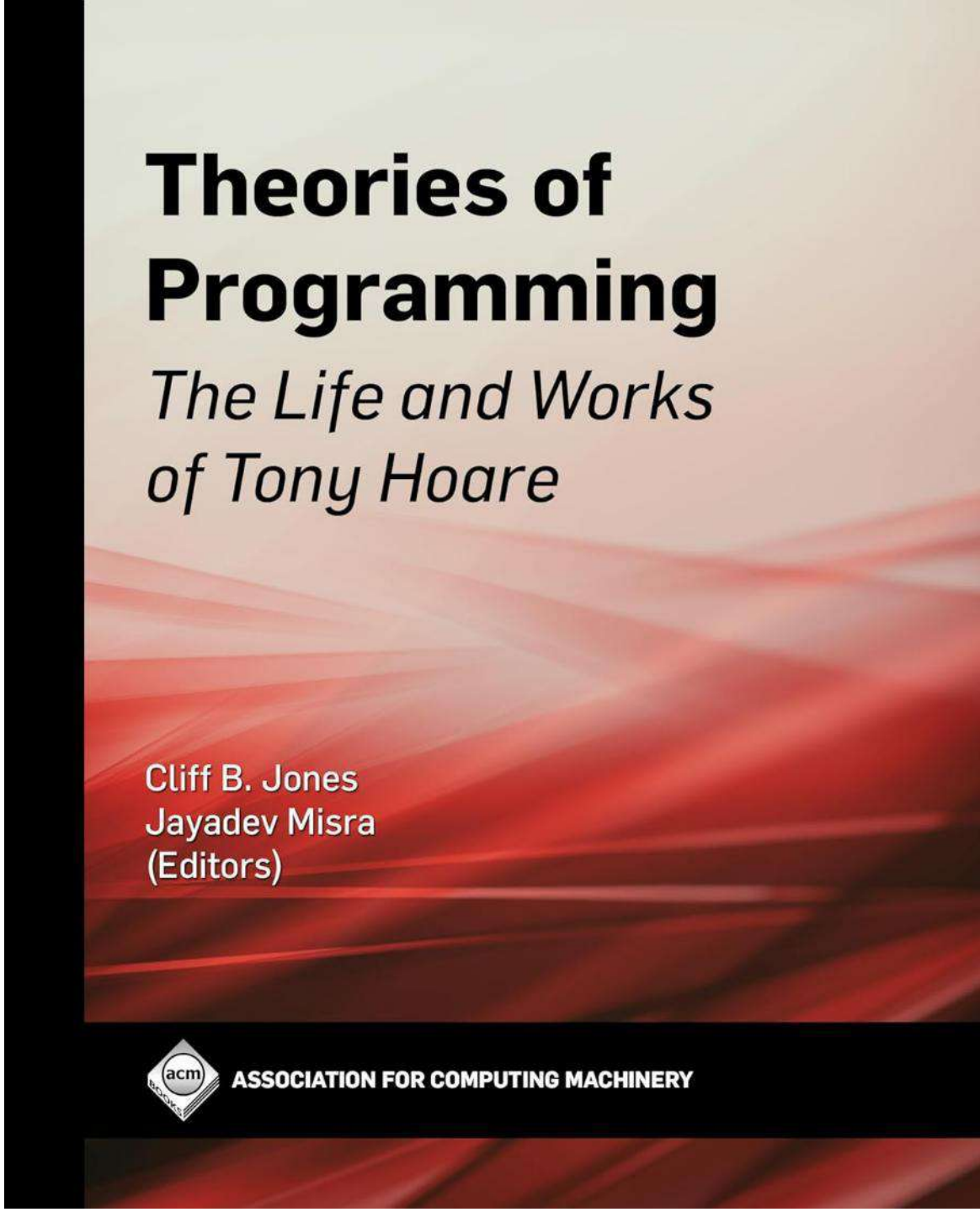
BCS-FACS, London, 2018

UTP @ 20 years old

Invited talk by Jifeng He
Commentary by Tony Hoare

Theories of Programming (2021)

Theories of Programming: The Life and Works of Tony Hoare. Cliff B. Jones & Jayadev Misra (eds). ACM Books (2021).



Tony Hoare – awards, prizes, and medals

- ACM Programming Systems and Languages Paper Award (1973) for the paper *Proof of correctness of data representations*
- **ACM A.M. Turing Award** for “fundamental contributions to the definition and design of programming languages” (1980)
- Harry H. Goode Memorial Award (1981)
- IEE Faraday Medal (1985)
- Computer Pioneer Award (1990)
- **Knighted** for services to education and computer science (2000)
- Kyoto Prize for Information Science (2000)
- Friedrich L. Bauer Prize, Technical University of Munich (2007)
- SIGPLAN Programming Languages Achievement Award (2011)
- IEEE John von Neumann Medal (2011)
- **Royal Medal of the Royal Society** (2023)

Tony Hoare – honorary fellowships

- Distinguished Fellow of the British Computer Society (1978)
- **Fellow of the Royal Society** (1982) – cf. **Chinese Academy of Sciences**
- Honorary Fellow, Kellogg College, Oxford (1998)
- Fellow of the Royal Academy of Engineering (2005)
- Member of the National Academy of Engineering (2006)
- Fellow of the Computer History Museum (CHM) in California, "for development of the Quicksort algorithm …" (2006)
- Fellow of the ACM (2020)

NULLIUS IN VERBA

THE ROYAL SOCIETY

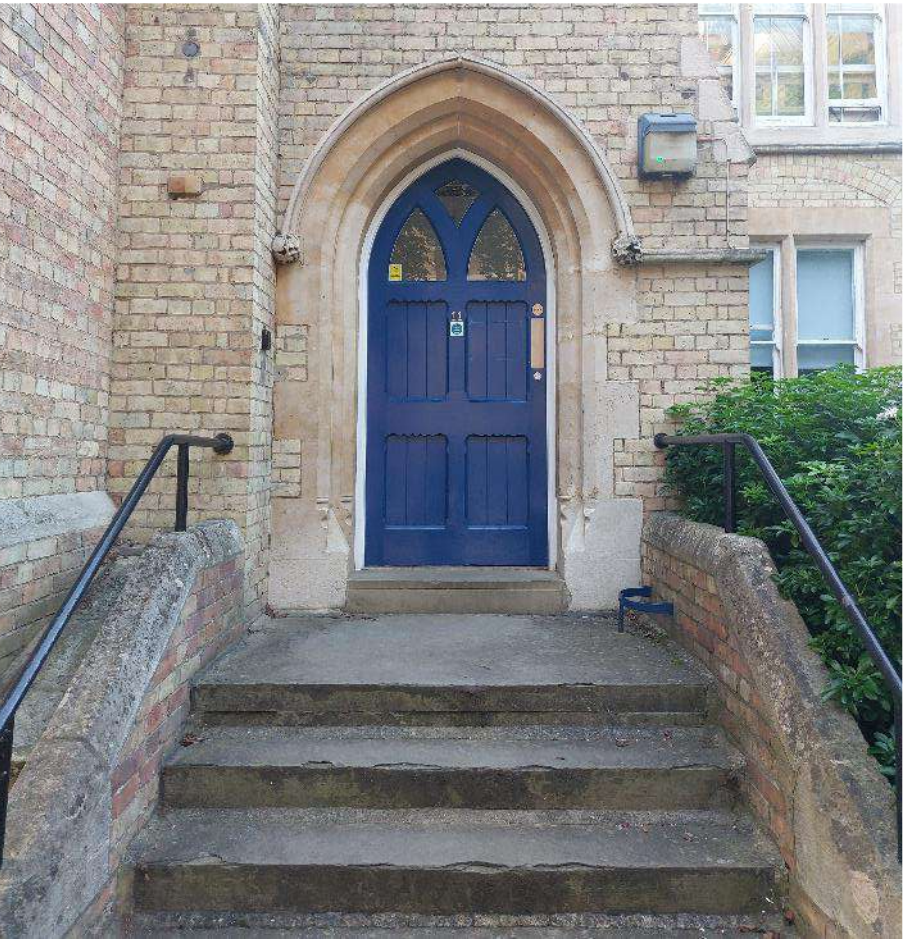
Tony Hoare – honorary degrees

- Honorary Doctorate of Science, **Queen's University Belfast** (1987)
- Honorary Doctorate of Science, University of Bath, UK (1993)
- Honorary Doctorate, Heriot-Watt University, UK (2007)
- Honorary Doctorate of Science, Department of Informatics of the Athens University of Economics and Business (AUEB), Greece (2007)
- Honorary Doctorate, University of Warsaw, Poland (2012)
- Honorary Doctorate, Complutense University of Madrid, Spain (2013)

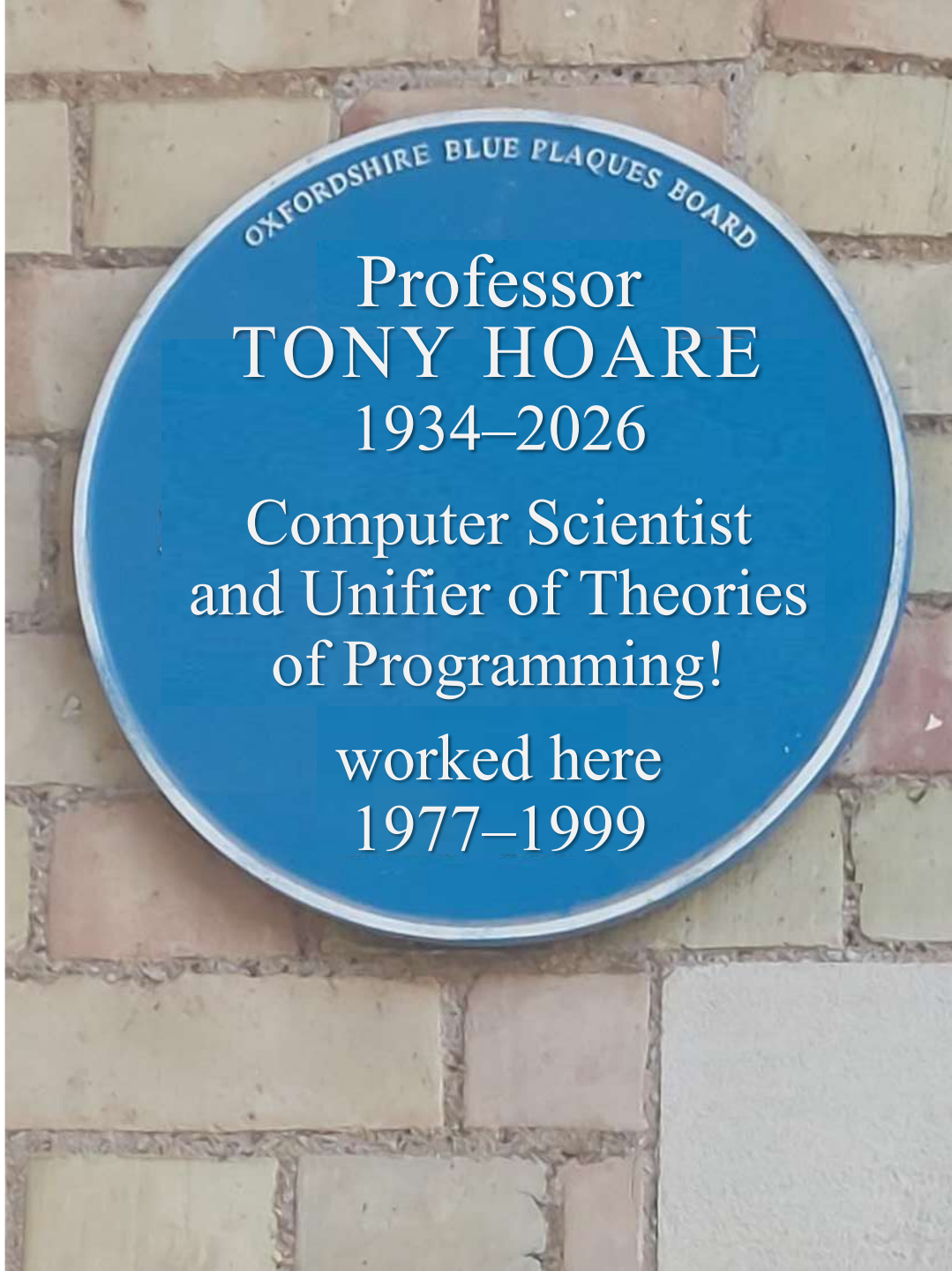
3 Keble Road, Oxford

Blue plaque for **James Legge** –
Missionary to China and first Professor of Chinese at Oxford University

Oxford University Computing Laboratory (11 Keble Road)



Oxford University Computing Laboratory (11 Keble Road)



Oxford University Computing Laboratory (11 Keble Road)

"Spoonerisms!"

"You have **h**issed all my **m**ystery lectures. You have **t**asted a whole **w**orm. Please leave Oxford on the next **t**own **d**rain."

("You have **m**issed all my **h**istory lectures. You have **w**asted a whole **t**erm. Please leave Oxford on the next **d**own **t**rain.")

Tony's office

Tributes

- “Tony Hoare @ 90”, *FACS FACTS*, 2024-2, pp. 5–42, July 2024
- “Tony Hoare and His Imprint on Computer Science”, *Blog@CACM*, Bertrand Meyer, 20 March 2026
- Cambridge celebration, Churchill College, 23 March 2026
- Obituaries: *The Guardian* (12 April 2026), *The Times* (28 April 2026)
- “In Memoriam: C.A.R. Hoare”, *CACM*, 69(5), May 2026. DOI: 10.1145/3802605
- Reminiscences to appear in *IEEE Annals*
- Oxford celebration, Wolfson College, Sept. 2026

ProCoS: Provably Correct Systems

European ESPRIT projects and Working Group (1989–1997)

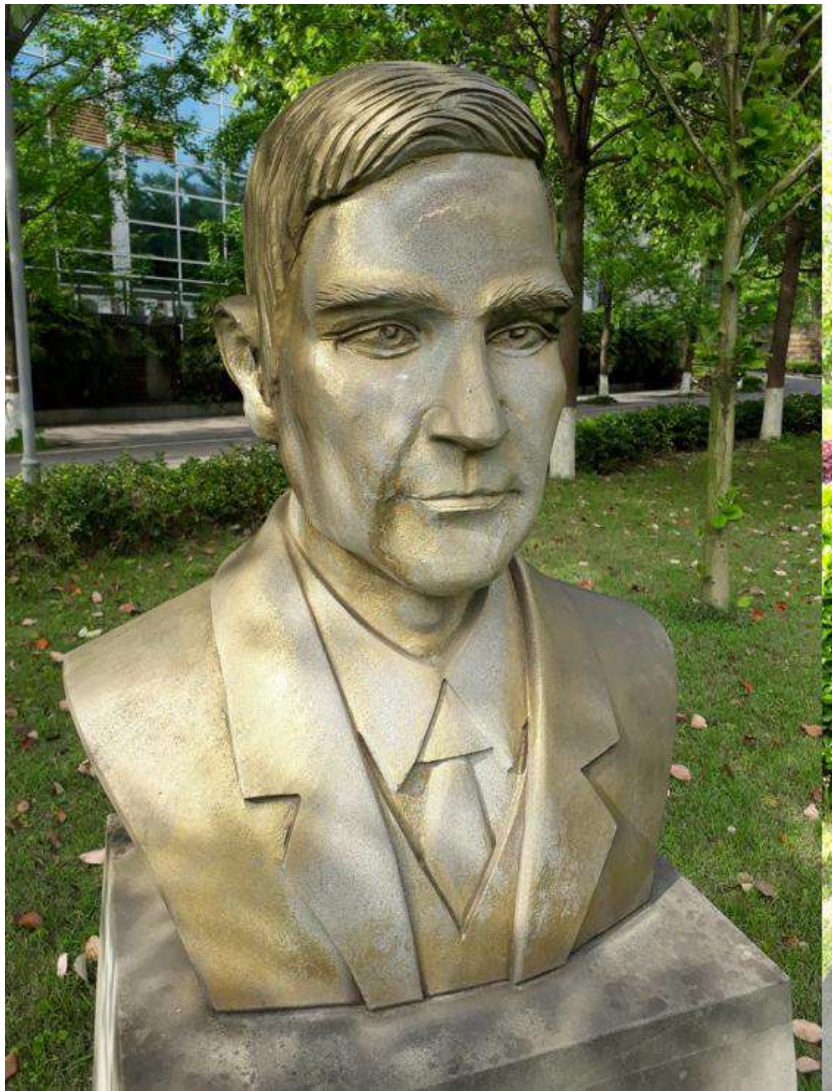
New: retrospective paper, especially Duration Calculus:

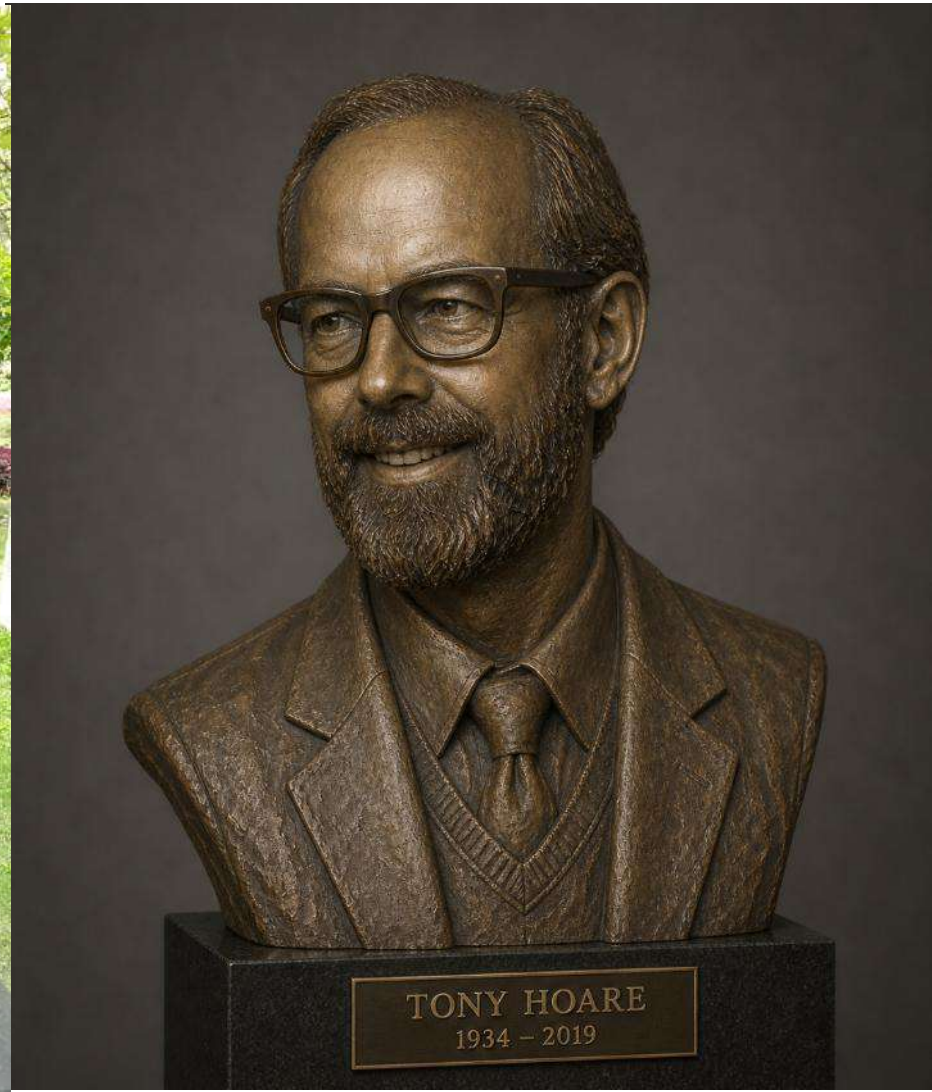
Bowen, Jonathan P.; Fränzle, Martin; Olderog, Ernst-Rüdiger; Bjørner, Dines; Hansen, Michael R.; Langmaack, Hans; Liu, Zhiming; Martin, Ursula (2026). "**Experiences from the European ProCoS Projects: Provably Correct Systems**". *Formal Aspects of Computing*. 38. DOI: 10.1145/3803555

Dedicated to **Tony Hoare** (1934–2026) and **Anders P. Ravn** (1947–2019)

A new bust at Southwest University, Chongqing?

1. Alan Turing 2. John von Neumann 3. Tony Hoare?

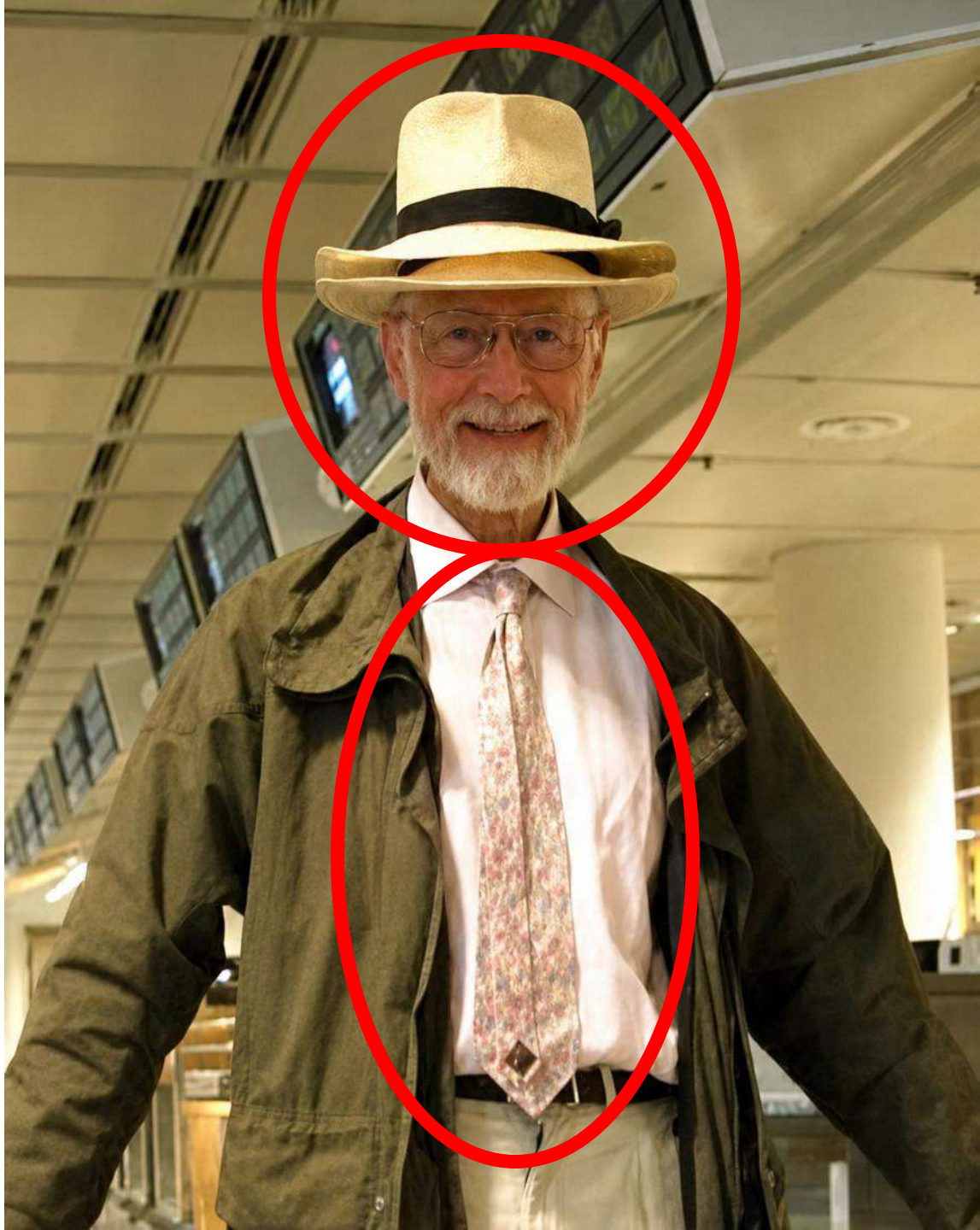




Proof by observation!

A visual proof that Tony Hoare owned at least two Panama hats!

He also owned a large collection of colourful ties!



Additional reflection: Jean-Raymond Abrial (1938–2025)

- Originator of three important formal methods: **Z notation**, **B-Method**, and **Event-B**
- "Tributes to Jean-Raymond Abrial", *FACS FACTS*, 2026-1, pp. 15–89, Jan. 2026 – by many colleagues www.bcs.org/media/veppnllv/facs-jan26.pdf
- J.P. Bowen, H. Habrias. "Jean-Raymond Abrial: A Scientific Biography of a Formal Methods Pioneer", *IEEE Annals of the History of Computing*, 48(2), 2026. DOI: 10.1109/MAHC.2026.3685515
- Springer LNCS "Gedenkschrift" memorial volume to be published

Tony Hoare (1934–2026)

Celebration at Cambridge

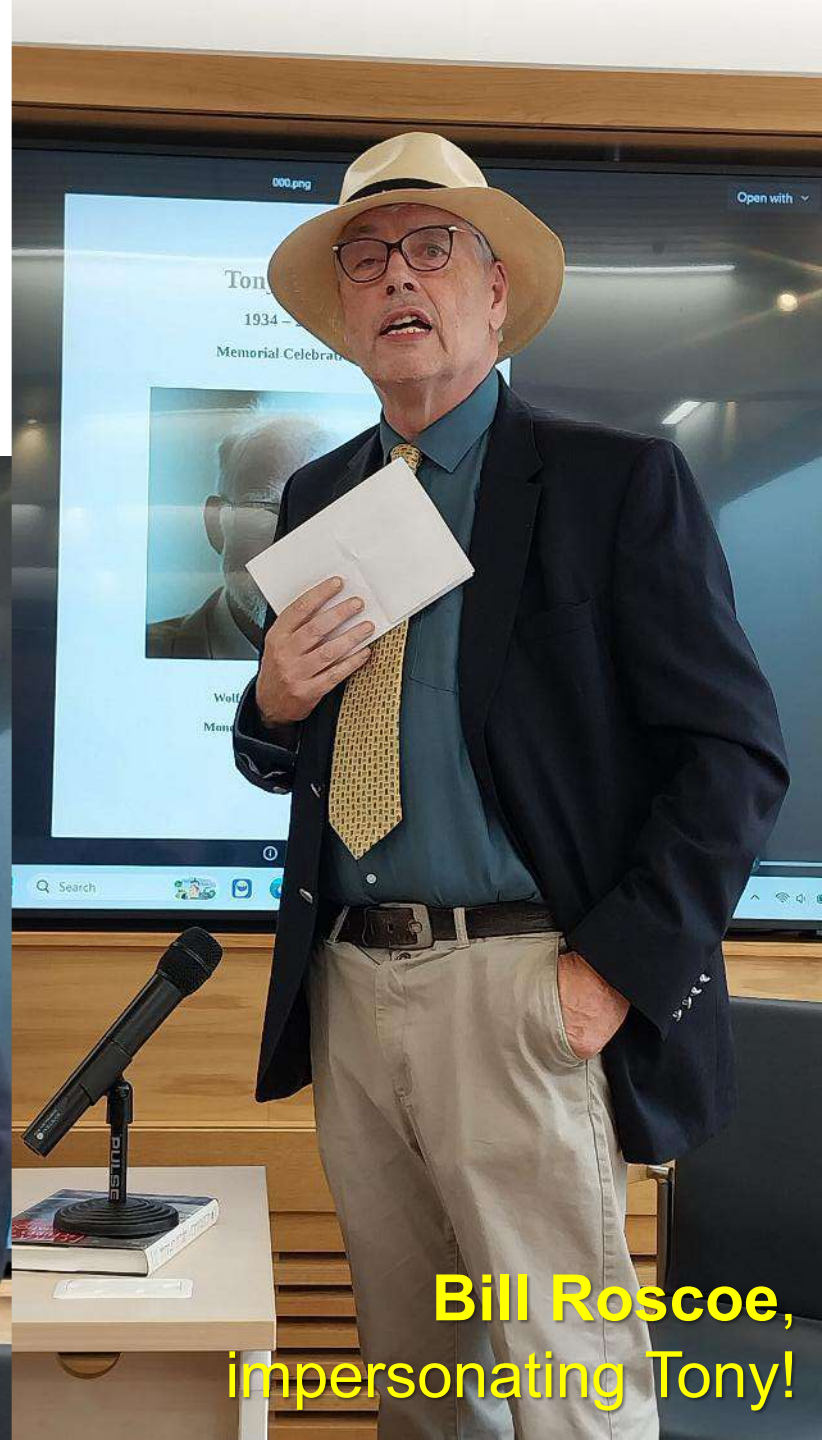
(23 March 2026)

Tony Hoare (1934–2026)

Celebration at Oxford

(7 September 2026)

Cliff Jones, chaired by Bernard Sufrin



Bill Roscoe, impersonating Tony!

Wikipedia

Created *template* before this talk!

v · t · e	Tony Hoare	[hide]
Software	**Programming**	ALGOL W · Monitors · Null references · occam · Structured programming
	Algorithms	Dining philosophers problem · Quickselect · Quicksort
	Formal methods	CSP · Duration calculus · Hoare logic · UTP
	Books	*Structured Programming* (1972) · *Communicating Sequential Processes* (1985) · *Unifying Theories of Programming* (1998)
Workplaces	Moscow State University · Elliott Brothers · Queen's University Belfast · University of Oxford · Microsoft Research Cambridge	
Collaborations	**Colleagues**	Richard Bird · Jonathan Bowen · Michael Butler · Ole-Johan Dahl · Edsger Dijkstra · Leslie Fox · Mike Gordon · He Jifeng · Eric Hehner · Andrey Kolmogorov · Gary T. Leavens · John Lucas · David May · Robin Milner · Jayadev Misra · Carroll Morgan · Peter O'Hearn · Ernst-Rüdiger Olderog · Jill Pym · Anders P. Ravn · Bill Roscoe · Natarajan Shankar · Mike Spivey · Joe Stoy · Bernard Sufrin · Niklaus Wirth · Zhou Chaochen
	Students	Cliff Jones · Bill Roscoe · Augusto Sampaio
	Groups	IFIP Working Group 2.1 · Programming Research Group · ProCoS
Awards	Distinguished Fellow of the British Computer Society (1978) · Turing Award (1980) · Harry H. Goode Memorial Award (1981) · Fellow of the Royal Society (1982) · IET Faraday Medal (1985) · Computer Pioneer Award (1990) · Knight Bachelor (2000) · Kyoto Prize (2000) · Fellow of the Royal Academy of Engineering (2005) · Fellow of the Computer History Museum (2006) · Friedrich L. Bauer Prize (2007) · IEEE John von Neumann Medal (2011) · Fellow of the ACM (2020) · Royal Medal (2023)	
Category · Wikiquote		

Epilogue

"There are **two ways** of constructing a **software design**; **one way** is to make it so **simple** that there are **obviously no deficiencies**, and **the other way** is to make it so **complicated** that there are **no obvious deficiencies**. *The first method is far more difficult*."

— **Tony Hoare** (1934–2026)

*Requiescat in pace.** The logic remains.

"If we knew what it was we were doing, it would not be called research, would it?"

— **Albert Einstein** (1879–1955) [apocryphal]

** May he rest in peace.*

Tony Hoare:

A Life of Logic, Theory, and Practice

Jonathan P. Bowen

London South Bank University, UK
Museophile Limited, Oxford, UK
Southwest University, Chongqing, China

jonathan.bowen@lsbu.ac.uk

www.jpbowen.com

EST 1892 LSBU

Museophile

SOUTHWEST UNIVERSITY