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
If a plot works out exactly as you first planned, you’re not working loosely enough to give room to your imagination and instincts.
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
procedure
The main difference is that the procedure can halt or need not halt. But the algorithm always halts and gives you the output.
algorithm
algorism
algorisme
algorismus
الخوارزمي
خوارزمی
algorithm
algorithm
std::sort
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:

\[ x = x + y \times 0 \]
\[ y \leq r \supset r + y \times q = (r - y) + y \times (1 + q) \]

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 \( y \leq r \)

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.
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:

\[ x = x + y \times 0 \]
\[ y \leq r \supset r + y \times q = (r - y) + y \times (1 + q) \]

The proof of the second of these is:

A5 \[ (r - y) + y \times (1 + q) \]

A9 \[ ((r - y) + y \times 1 + y \times q) \]

A3 \[ y \leq r \]

A6 \[ y \leq r \]

provided \( y \leq r \)

The axioms A1 to A9 are in practice true of the traditional infinte 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.

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

\[
\begin{align*}
x &= x + y \times 0 \\
y < r &\Rightarrow r + y \times q = (r - y) + y \times (1 + q)
\end{align*}
\]

The proof of the second of these is:

\[
\begin{align*}
&= (r - y) + (y \times 1 + y \times q) \\
&= (r - y) + (y + y \times q) \\
&= (r - y) + y \times q
\end{align*}
\]
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:

\[
x = x + y \times 0
\]

\[
y \leq r \Rightarrow r + y \times q = (r - y) + y \times (1 + q)
\]

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

\[
= r + y \times q \text{ provided } y \leq r
\]

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 there are no preconditions imposed, we write \( \text{true} \{ Q \} \ R. \)
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 \log n)
O\left( n^2 \right)
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!
```bash
$ 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
1
3
4
5
9
```
\( O(n) \)
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.
"We TOLD you it was hard."
"Yeah, but now that I'VE tried, we KNOW it's hard."
97 Things Every Programmer Should Know

Collective Wisdom from the Experts

Edited by Kevlin Henney
Read the Humanities
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.
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.
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.
97 Things Every Programmer Should Know

Collective Wisdom from the Experts
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
They generally don’t lie.
They use their terms and their contexts.
They leave out significant details.
They make assumptions.
This is compounded by the fact that many customers don’t actually know what they want in the first place!
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.
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.
We may therefore picture the process of form-making as the action of a series of subsystems, all interlinked, yet sufficiently different to be considered distinct.
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!
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
halv två
halb zwei
one
velocity
speed
\( \mathbf{v} = \mathbf{v}_x + \mathbf{v}_y \)
\[ v = |v| \]
\[ v = s' \]
v = \frac{ds}{dt}
v = \frac{S}{t}
This sentence no verb.
blanc
blanc
DO NOT CROSS THE RED MAN!
At any time on footway
red
green
green
green
black
green
value
business value
prioritise by business value
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.”
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

Always pay through Amazon.com’s Shopping Cart or 1-Click. Learn more about Safe Online Shopping and our safe buying guarantee.
<table>
<thead>
<tr>
<th>Date</th>
<th>Profnath</th>
<th>Bordeebk</th>
<th>Profnath Over Previous Bordeebk</th>
<th>Bordeebk Over Profnath</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Apr</td>
<td>$1,730,045.91</td>
<td>$2,198,177.95</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>9-Apr</td>
<td>$2,194,443.04</td>
<td>$2,788,233.00</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>10-Apr</td>
<td>$2,783,493.00</td>
<td>$3,536,675.57</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>11-Apr</td>
<td>$3,530,663.65</td>
<td>$4,486,021.69</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>12-Apr</td>
<td>$4,478,395.76</td>
<td>$5,690,199.43</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>13-Apr</td>
<td>$5,680,526.66</td>
<td>$7,217,612.38</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
</tbody>
</table>
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](http://www.michaeleisen.org/blog/?p=358) and our [safe buying guarantee](http://www.michaeleisen.org/blog/?p=358).

<table>
<thead>
<tr>
<th>Price + Shipping</th>
<th>Condition</th>
<th>Seller Information</th>
<th>Buying Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$18,651,718.08</strong>&lt;br/&gt;$3.99 shipping</td>
<td>New</td>
<td>Seller: profnath&lt;br/&gt;Seller Rating: 5 stars&lt;br/&gt;Seller Rating: 93% positive&lt;br/&gt;</td>
<td>Add to Cart or <a href="http://www.michaeleisen.org/blog/?p=358">Sign in</a> to turn on 1-Click ordering.</td>
</tr>
<tr>
<td><strong>$23,698,655.93</strong>&lt;br/&gt;$3.99 shipping</td>
<td>New</td>
<td>Seller: bordebook&lt;br/&gt;Seller Rating: 5 stars&lt;br/&gt;Seller Rating: 93% positive&lt;br/&gt;</td>
<td>Add to Cart or <a href="http://www.michaeleisen.org/blog/?p=358">Sign in</a> to turn on 1-Click ordering.</td>
</tr>
</tbody>
</table>
The pound has dived on Asian markets with automated trading being blamed for the volatility.
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

Software Requirements & Specifications

a lexicon of practice, principles and prejudices

MICHAEL JACKSON
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.