Software Engineering

A historical perspective and a current assessment

Niklaus Wirth

Zurich, 22.11.2010

Early days

1960: Language and compiler Neliac

- Overriding property: A mess
- Extending language, fixing compiler bugs: Black art

Missing a scientific basis

Therefore a unique subject for scientific research

The ESSENCE of SE: Cleaning the mess

How can you write “correct” programs, if there is no rigorous specification of your notation?

Algol 60 appeared as an excellent starting point

It provided structure, defined in a rigorous mathematical sense: Syntax

Giving a strong impetus to methods of syntactic analysis

EBNF: The Syntax of Syntax

- syntax = {rule}.
- rule = ident “:=“ expression “.” .
- expression = term {“|” term}.
- term = factor {factor}.
- factor = ident | string | “(“expression”)” |
- “[“expression”]” | “{“expression”}”.

Semantics

Syntax ok, but what about meaning?

The dominant idea: Operational semantics

Attach a meaning to each syntactic construct

Semantic of each construct defined in terms of a short program, specified in a simpler language

Garwick, Wirth: Euler

Did not solve the problem, simply moved it down to a lower plane

Axiomatic definition

- R. Floyd (1968)
- C.A.R.Hoare (1969) - Assertion
- E.W. Dijkstra (1973) - Predicate transformer
- Axiomatic defintion of the PL Pascal (1973)
- Programming = Constructing correctness proof
- The ESSENCE of SE: Not ceaning up, but avoiding the mess (Dijkstra)

State of the Art (2010)

- Correctness proofs are rarely used. Why?
- The proof is more complex than the program
- The theory requires a strong ability for formula manipulation. Most prog. lack it.
- Programming methodology has improved:

  - Structures, modularity, type checking, and languages supporting them.

  - Interactive tools to help locating errors

Consequences

- Industrial SE has produced a myriad of tools, emerging from early “symbolic debuggers”
- Resulting work mode: Try, test, fix.
- Faster computers promote this paradigm
- Fast compilers and interactive use also promote this paradigm
- Academia has embraced it and declared programming as trivial and no longer to be taught as a discipline
- SE has dwindled to a management science

My own recent experiences (working with Verilog)

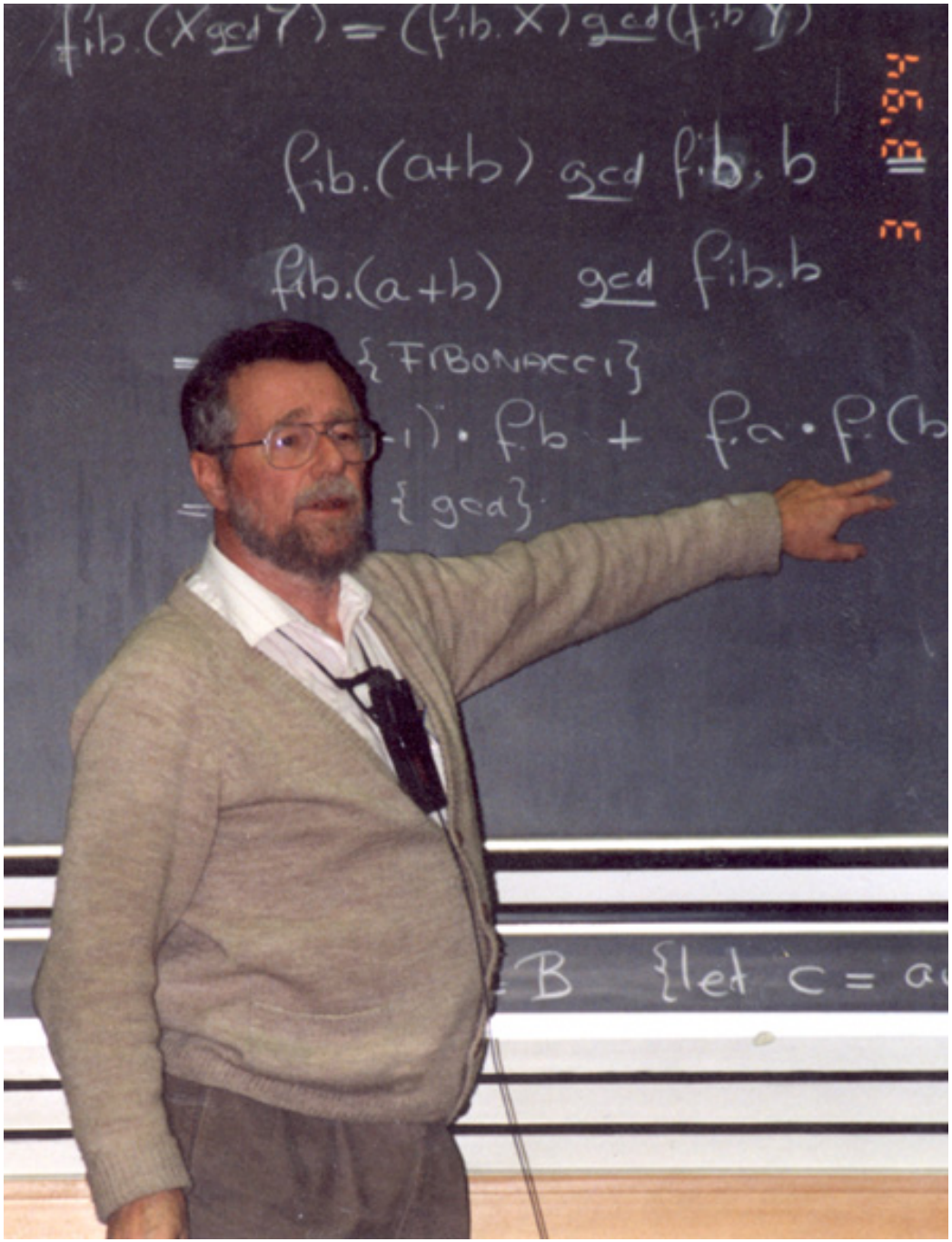
- Very slow compilation
- Inundation of unhelpful warnigs
- Errors in spite of “successfully completed”
- Unreliable type checking
- Archaic tools for assembling load files
- Small changes in the source may result in heavy changes in the generated circuit
- Malfunctioning without visible reasons

Facit

- I am very often frustrated, spending much more time in fighting the tools than on my design
- Often feel thrown back to the 1960s
- Insufficiently clear specifications of the language and the rules about “tools”
- The one item that has permeated the entire field is the ugly syntax of “C”: Java, C#, Go, and Verilog, all have adopted it.
- Complexity grows rather than shrinks

Good Design

- Programming is designing abstract machines
- The distillation of the art of designing
- SE is the discipline of designing in the large
- Courses on Good Design ?
- Literature on Good Design ?
- Good Design can emerge only from long and enduring exposure to practice


fib.(X gcd Y) = (fib.X) gcd (fib.Y)
fib.(a+b) gcd fib.b, b
fib.(a+b) gcd fib.b
{FIBONACCI}
{gcd}
B {let c = a