Preparing Programmers for Quantum

J.N. Oliveira

Universidade do Minho
INESCTEC
TECHNOLOGY & SCIENCE
ASSOCIATE LABORATORY
PORTUGAL

Research context

QUANTA ⟨L|A|B⟩

- Bridging **U.Minho** / **INESC TEC** / **INL** (Braga, Portugal)
- Targeting **reversible** and **quantum programming** from a formal method's perspective.

Classical computing

Happy blend of diverse bodies of knowledge:

$$\begin{tikzcd}
\textit{Philosophy} \arrow[dr, "\left\{\begin{array}{l}\textit{Formal}\\ \textit{Logic}\end{array}\right." '] & & \textit{Maths} \arrow[dl, "\left\{\begin{array}{l}\textit{Automata}\\ \textit{Calculus}\end{array}\right."] \\
& \textbf{Computing} & \\
\textit{Physics} \arrow[ur, "\left\{\begin{array}{l}\textit{Semiconductor}\\ \textit{Electronics}\end{array}\right."] & & \textit{Linguistics} \arrow[ul, "\left\{\begin{array}{l}\textit{Grammars}\\ \textit{Languages}\end{array}\right." ']
\end{tikzcd}$$

(Source: Manchester University, UK)

Maths dreamed of it...

1936

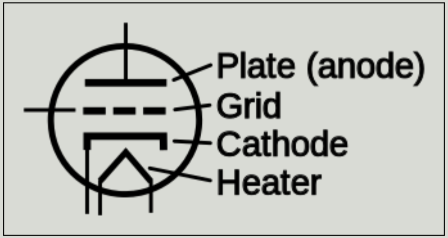
A. Turing — **abstract** *notion of what we now call a* **programmable computer** *— known as the* **Turing machine***.*

1936

A. Church — λ-calculus, the basis of **functional** *programming.*

A. Turing (1912-1954)

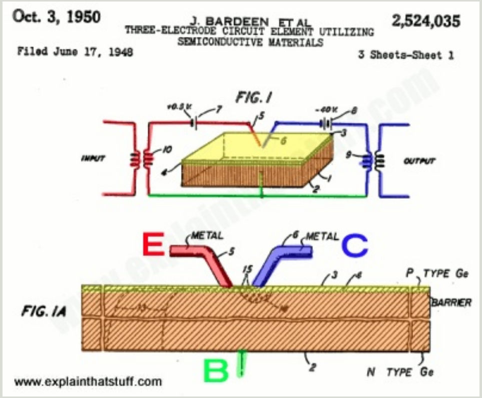
Physics made it happen...



Vacuum tubes, triodes (1912)

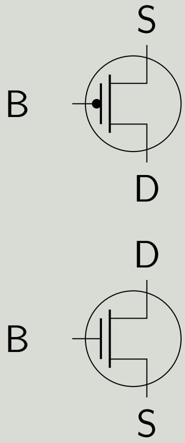
(Credits: `https://en.wikipedia.org/wiki/Triode`)

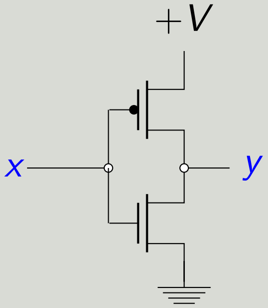
Physics made it happen...

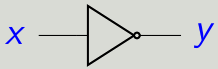


Transistors (1948)

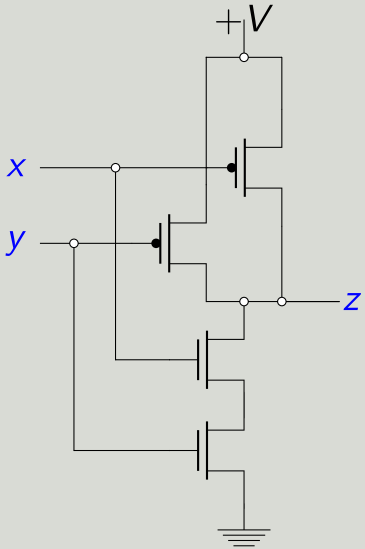
… but soon abstraction was needed



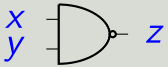




… first graphical, then formal



Graphical



Formal logic

$$\mathit{nand} : \{0,1\} \times \{0,1\} \to \{0,1\}$$
$$\mathit{nand}\,(x, y) = \neg\,(x \wedge y)$$

50 years later...

1985

David Deutsch (U. Oxford) describes the first universal **quantum computer***.*

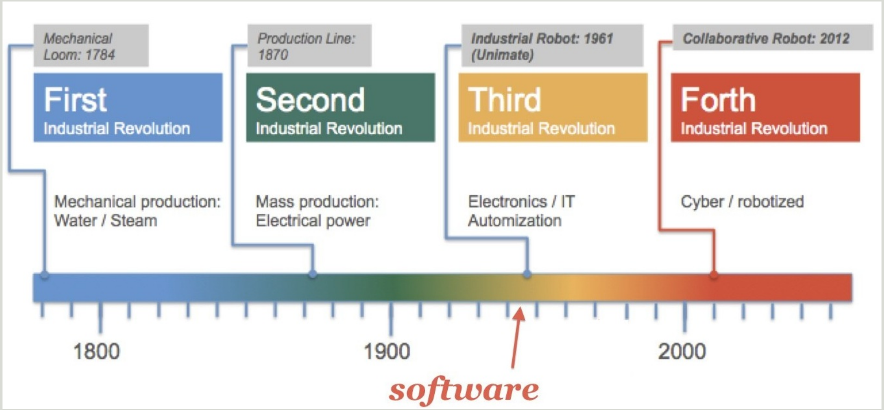
Nowadays

Physics making it happen, again...

History repeating itself?

Source: IBM Q Experience website

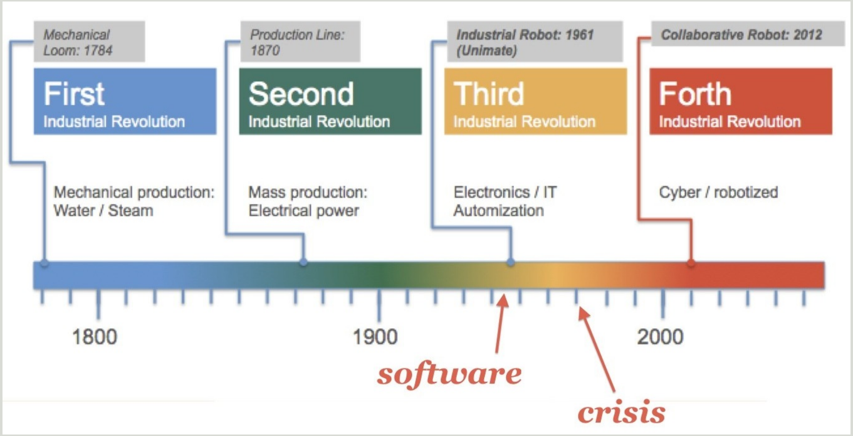
The big picture


Mechanical Loom: 1784
First Industrial Revolution
Mechanical production: Water / Steam
Production Line: 1870
Second Industrial Revolution
Mass production: Electrical power
Industrial Robot: 1961 (Unimate)
Third Industrial Revolution
Electronics / IT Automization
Collaborative Robot: 2012
Forth Industrial Revolution
Cyber / robotized
1800
1900
2000
software

1950s — *"L'enfant terrible"* is born

(Adapted from nelmia Robotics Insight 2015)

Crisis (1960s)


Mechanical Loom: 1784
Production Line: 1870
Industrial Robot: 1961 (Unimate)
Collaborative Robot: 2012
First Industrial Revolution
Second Industrial Revolution
Third Industrial Revolution
Forth Industrial Revolution
Mechanical production: Water / Steam
Mass production: Electrical power
Electronics / IT Automization
Cyber / robotized
1800
1900
2000
software
crisis

1st NATO Conference on **Software Engineering**, Garmisch, Oct. 1968

Software Engineering (1968-2018)

Phrase **software engineering** seems to date from the Garmisch NATO conference in 1968:

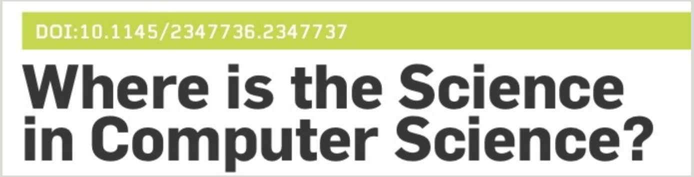
> *In late 1967 the Study Group recommended the holding of a working conference on* **Software Engineering***.*
>
> *The phrase 'software engineering' was deliberately chosen as being* **provocative***, in implying the need for software manufacture to be based on the types of* **theoretical foundations** *and practical disciplines, that are traditional in the established branches of engineering.*

How **scientific** is SE today?

Not everybody looks happy...

Vinton Cerf (1943-)

ACM President



DOI:10.1145/2347736.2347737

Where is the Science in Computer Science?

"... we have a responsibility to pursue the **science** *in computer science. We must develop better tools and much deeper understanding of the systems we invent and a far greater ability to make* **predictions** *about the behavior of these complex, connected, and interacting systems."*.

(Vinton G. Cerf, Letter from the ACM President, CACM 55(10), Oct. 2012)

Software as a problem

$$\text{Software} \left\{ \begin{array}{l} \text{Process} \text{ — } ✓ \\[2em] \text{Product} \text{ — } ? \end{array} \right.$$

“L’enfant terrible”

Unlike **hardware**, **software** not governed by the laws of **physics**:

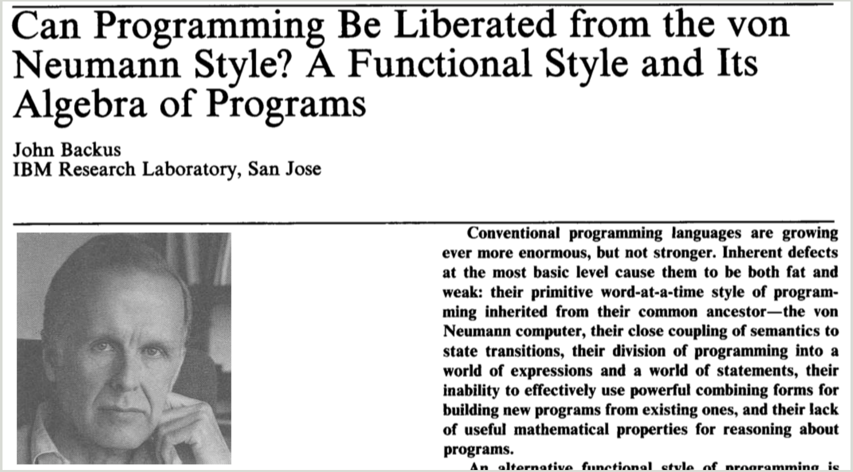
- does not weight, does not smell,
- it is chemically neutral ...

Anthony Oettinger (**ACM** President, 1967):

> *”(...) the scientific, rigorous component of computing, is more like* **mathematics** *than it is like* **physics”.**

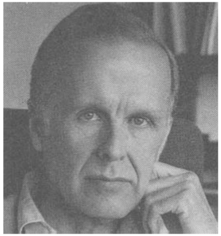
A. Oettinger (1929-)

J. Backus Turing Award (1978–2018)



Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is

Old concerns

Still Oettinger **(already** in 1967):

"It is a matter of **complexity***. Once you start putting thousands of these instructions together you create a* **monster** *which is* **unintelligible** *to anyone save its creator and, most of the time, unfortunately even to the creator."*

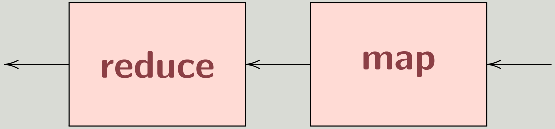
Old concerns

Still Backus (1978):

"Conventional **programming languages** *are growing ever more* **enormous***, but not stronger. Inherent defects (...) cause them to be both fat and weak: (...) their inhability to effectively use powerful* **combining forms** *for building programs from existing ones, and their lack of useful* **mathematical** *properties for reasoning about programs."*

Interestingly...

Backus (1978) predicted the age of **MapReduce**,

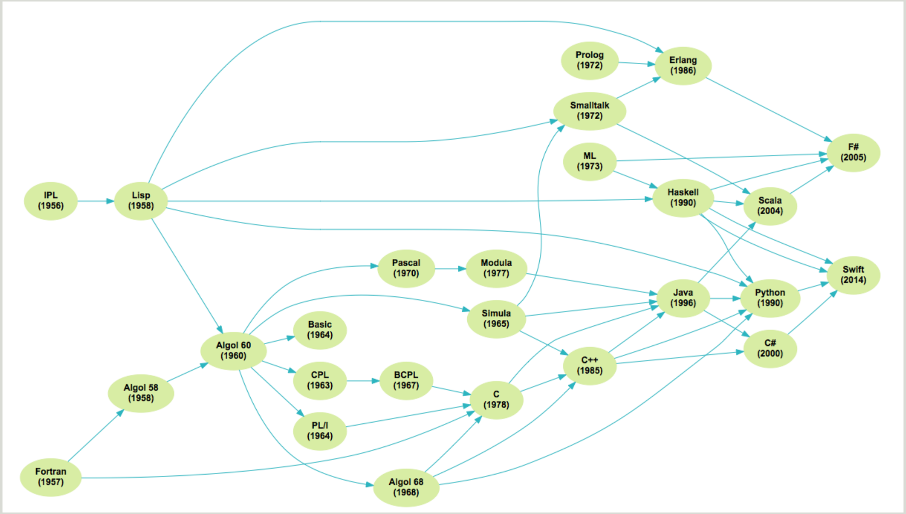


written

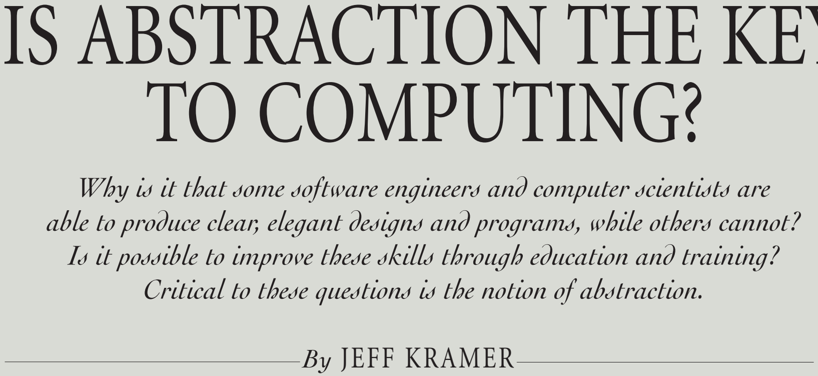
$$(/g) \cdot (\alpha\ f)$$

in his **FP** **combinator** notation.

Meanwhile...


Prolog (1972)
Erlang (1986)
Smalltalk (1972)
ML (1973)
F# (2005)
IPL (1956)
Lisp (1958)
Haskell (1990)
Scala (2004)
Pascal (1970)
Modula (1977)
Swift (2014)
Java (1996)
Python (1990)
Simula (1965)
Basic (1964)
Algol 60 (1960)
C# (2000)
C++ (1985)
CPL (1963)
BCPL (1967)
Algol 58 (1958)
C (1978)
PL/I (1964)
Fortran (1957)
Algol 68 (1968)

What should have happened?



IS ABSTRACTION THE KEY TO COMPUTING?

Why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot? Is it possible to improve these skills through education and training? Critical to these questions is the notion of abstraction.

By JEFF KRAMER

(Commun. ACM, 50:4, pages 37–42, April 2007)

Abstraction

Quoting Jeff Kramer:

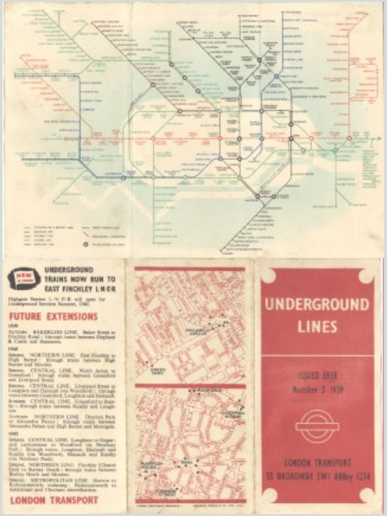
Abstraction *is widely used in other disciplines such as* **art** *and* **music***. (...) Henri Matisse manages to clearly represent the* **essence** *of his subject (...) using only simple lines or cutouts. His representation* **removes** *all detail yet* **conveys** *much.*

H MATISSE 52

Abstraction

The famous "abstract map" of London's Underground (1939).

The art of removing **unnecessary** detail.



Abstraction

Rather than "WYSIWYG", we want

$$\text{Expressiveness} = \frac{\text{What you get}}{\text{What you write}}$$

Expressiveness versus **productivity**:

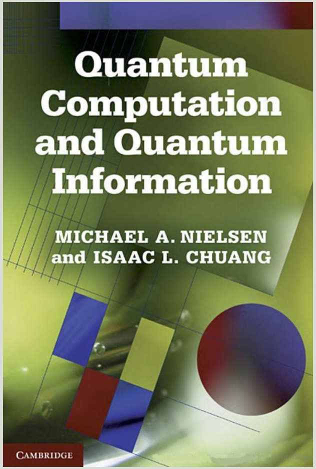
"(...) Readers of this book will enjoy a rare opportunity to learn how to write **less** *in order to say* **more***, without ambiguity. In short, to learn how to be* **productive***".*

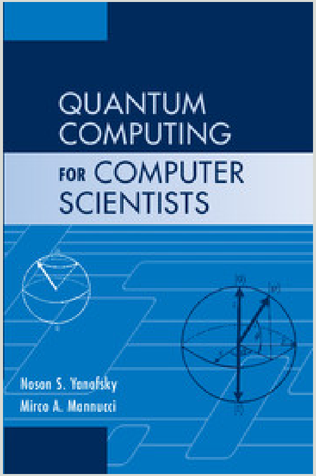
(endorsing D. Jackson's *Software Abstractions*, MIT Press, 2011)

Quantum abstraction

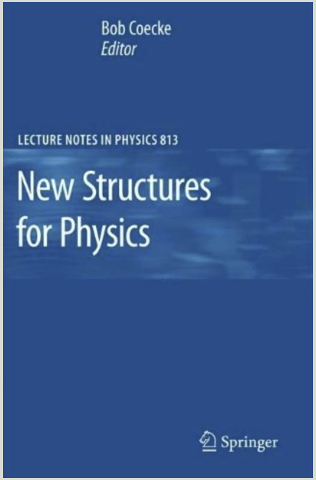
- Programmers to be challenged even further by **abstractions** in **quantum** programming
- Abstract from the physical layer! (**quantum physics** difficult and counter-intuitive)
- Get it **right** from the very beginning! (**debugging** nearly impossible)
- Avoid reading the state — **measuring** interferes with the quantum effect!

Quantum computing literature


Quantum Computation and Quantum Information
MICHAEL A. NIELSEN and ISAAC L. CHUANG
CAMBRIDGE



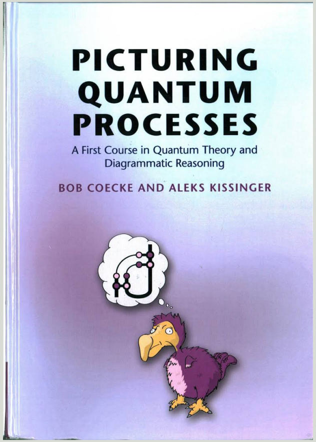
Quantum computing literature


Bob Coecke
Editor

LECTURE NOTES IN PHYSICS 813

New Structures for Physics

Springer

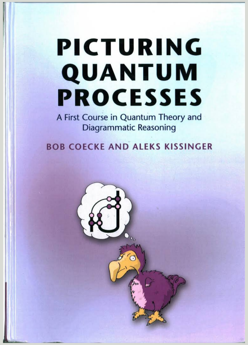

PICTURING QUANTUM PROCESSES

A First Course in Quantum Theory and Diagrammatic Reasoning

BOB COECKE AND ALEKS KISSINGER

Quantum computing literature

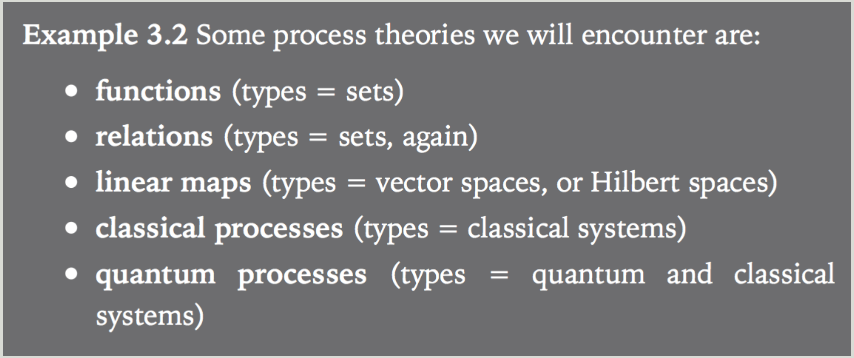
- Degree of sophistication varies
- Emphasis on the need to **change** one's **mindset** (as happened to physicists)
- Some pedagogical effort
- What kind of **abstractions** are involved?


PICTURING QUANTUM PROCESSES

A First Course in Quantum Theory and Diagrammatic Reasoning

BOB COECKE AND ALEKS KISSINGER

Why do functions always come first?



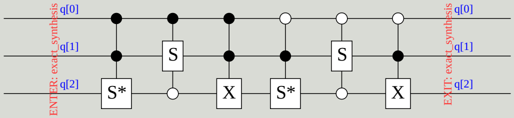
Example 3.2 Some process theories we will encounter are:

- **functions** (types = sets)
- **relations** (types = sets, again)
- **linear maps** (types = vector spaces, or Hilbert spaces)
- **classical processes** (types = classical systems)
- **quantum processes** (types = quantum and classical systems)

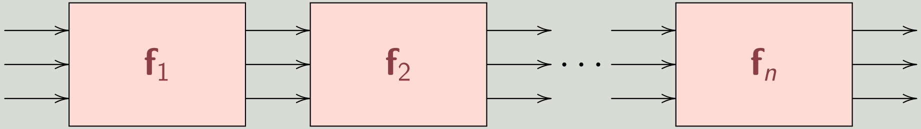
(Excerpt from chapter 3 of **Picturing quantum processes** by Coecke & Kissinger)

Quantum programs (circuits)

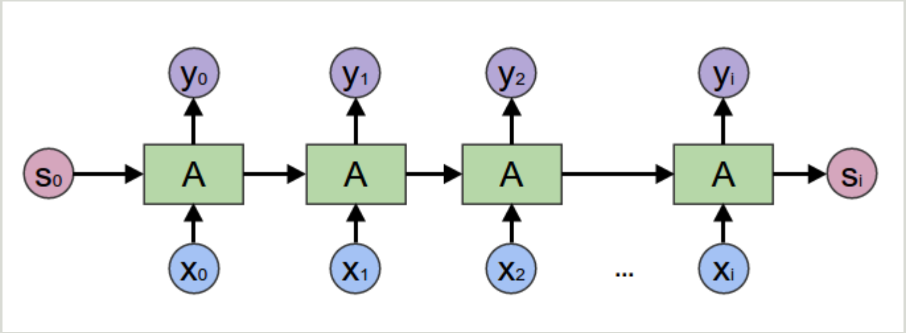
Quantum circuit generated using the QUIPPER tool:



Functional abstraction:



Similar abstractions in Neural Networks



(RNN = accumulating maps)

(Source: **Neural Networks, Types, and Functional Programming** by C. Olah, 2015)

A functional quantum program

$$
\begin{array}{l}
(\!\!|\, \cdot \,|\!\!) :: ((a, b) \to \mathit{Vec}\ (c, b)) \to ([a], b) \to \mathit{Vec}\ ([c], b) \\
(\!\!|\, f \,|\!\!)\ ([\,], b) = \mathbf{return}\ ([\,], b) \\
(\!\!|\, f \,|\!\!)\ (h : t, b) = \mathbf{do}\ \{ \\
\quad (t', b') \leftarrow (\!\!|\, f \,|\!\!)\ (t, b); \\
\quad (h'', b'') \leftarrow f\ (h, b'); \\
\quad \mathbf{return}\ (h'' : t', b'') \\
\quad \}
\end{array}
$$

It controls **qubit** b according to a list of classical bits using the **quantum** operator f (parameter).

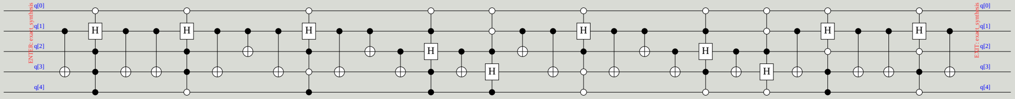
How does it compile?

Current experiments

Tool-chain:

$$\begin{tikzcd}
{} \arrow[r] & \boxed{\textbf{GHCi}} \arrow[r] & \boxed{\textbf{Quipper}} \arrow[r] & \boxed{\textbf{QISKit}^{\text{TM}}} \arrow[r] & \boxed{\textbf{IBM Q}}
\end{tikzcd}$$

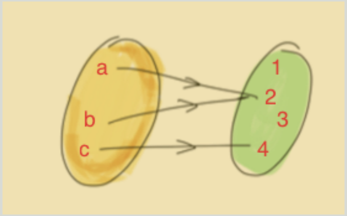
Example of quantum circuit generated from the given program:



(With thanks to: Ana Neri, Afonso Rodrigues, Rui S. Barbosa)

FP on the quantum way

- **Quantum programs** generalize (reversible) **functions**
- Quantum programs much closer to **functional programs** (FP) than to **imperative** ones.



(a **non-reversible** function!)

Meanwhile

- functional **flavour** is spreading across languages (F#, Swift, Java 8, Python, ...).

Too late

Question:

Should **FP** *be taught at universities as first language?*

Answer:

- Yes, it **should** (and this is the case in several places)

However:

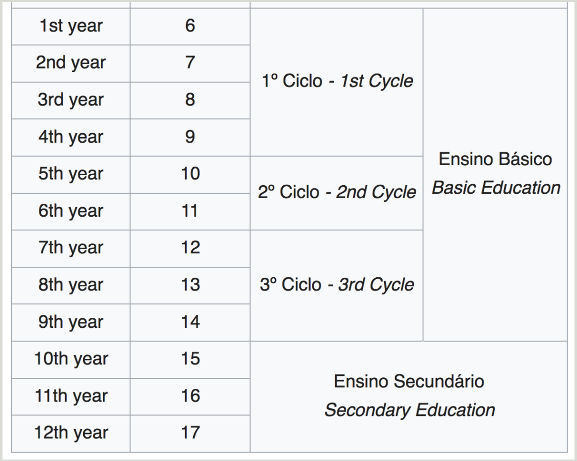
- That's simply... **too late**!

J. McCarthy (1927-2011)

Formal operational stage

"(...) from around **12** *to adulthood (...) individuals indicate an ability to* **think abstractly**, *systematically, and hypothetically, and to use* **symbols** *related to* **abstract concepts**. *This is the crucial stage at which individuals are capable of thinking abstractly and scientifically."*

(Kramer, 2007)



1st year	6	1º Ciclo - *1st Cycle*	Ensino Básico *Basic Education*
2nd year	7		
3rd year	8		
4th year	9		
5th year	10	2º Ciclo - *2nd Cycle*	
6th year	11		
7th year	12	3º Ciclo - *3rd Cycle*	
8th year	13		
9th year	14		
10th year	15	Ensino Secundário *Secondary Education*	
11th year	16		
12th year	17		

“Verão no Campus” (2007-2017)

VnC “junior university” courses, University of Minho, every July — introducing ISCED[1] level 3 students (15 to 18) to programming.

[1]International Standard Classification of Education.

“Verão no Campus” (2007-2017)

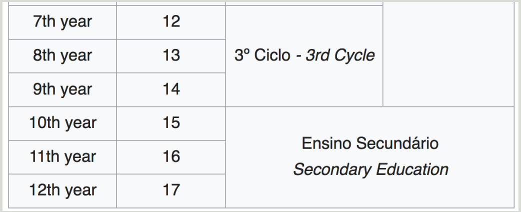
At our **VnC** module we have **10 years** of experience in teaching **FP** to ISCED level 3 students.

As a rule, students like the course.

(Even those who have programmed before.)

Formal operational stage

Alongside with **maths**, **physics** and the other subjects, **middle school** students should study basic **computer science** and **programming**.

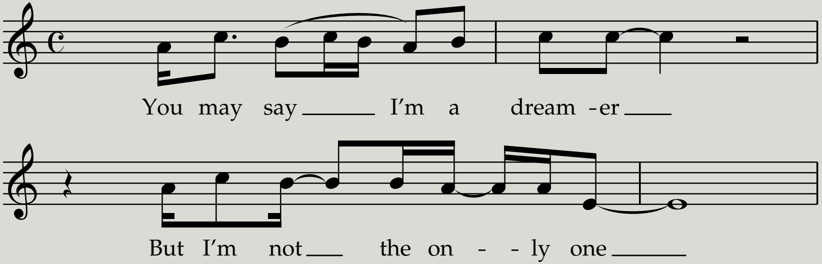


7th year	12	3º Ciclo - *3rd Cycle*	
8th year	13		
9th year	14		
10th year	15	Ensino Secundário *Secondary Education*	
11th year	16		
12th year	17		

FP blends very nicely with **maths** and **physics**.

Such has been our experience at **VnC** (2007-17).

Imagine...



(Lennon, 1971)

FP @ U.Minho (1998-2018)

Our department has long been an advocate of the **functional-first** *school of programming and has been teaching* **Haskell** *as a first language (...) for* **20 years**. *(...) We have been using* **game programming** *to keep students motivated (...) We summarise (...) our experience [in developing] a model for comprehensive and interactive functional game programming assignments (...)*

Quoted from: *J.B. Almeida, A. Cunha, N. Macedo, H. Pacheco, J. Proença* - **Experience Report: Teaching how to program using automated assessment and functional Glossy games**, *ICFP 2018, St. Louis, USA (forthcoming).*

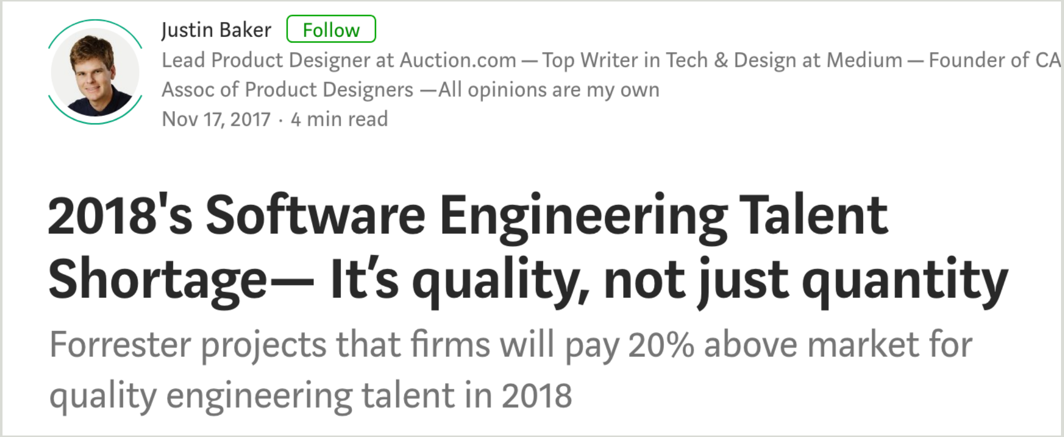
“Computing & Schools” (CAS) trend

Computing as a mandatory subject on the EU (2016)

- **2014, UK**: covering Primary and Secondary school
- **2016, France**: covering 2nd, 3rd and 4th cycles
- **2016-18, Finland**: covering Primary and Secondary school
- **2016-17, Poland, Malta and Croatia**: covering Primary and Secondary school
- **2017, Denmark**: covering Secondary school
- **Spain, Belgium and Germany**: introduced at a regional level

(Credits: N.F. Rodrigues)

Future jobs...



Justin Baker Follow

Lead Product Designer at Auction.com — Top Writer in Tech & Design at Medium — Founder of CA

Assoc of Product Designers —All opinions are my own

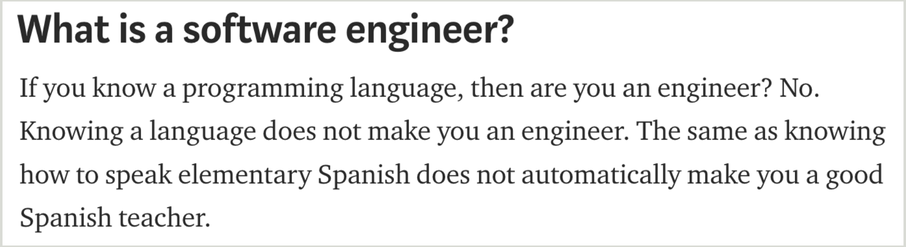
Nov 17, 2017 · 4 min read

2018's Software Engineering Talent Shortage— It's quality, not just quantity

Forrester projects that firms will pay 20% above market for quality engineering talent in 2018

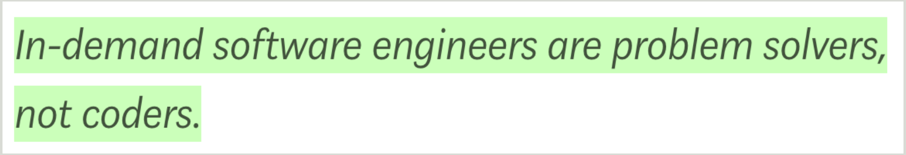
(`https://hackernoon.com`)

Future jobs…



What is a software engineer?

If you know a programming language, then are you an engineer? No. Knowing a language does not make you an engineer. The same as knowing how to speak elementary Spanish does not automatically make you a good Spanish teacher.



In-demand software engineers are problem solvers, not coders.

(`https://hackernoon.com`)

Thanks