ThoughtWorks®

Teoria a serviço da prática

ALÉM DOS PARADIGMAS

Entenda as características da linguagem
e saiba escolher e adaptar os padrões

Luciano Ramalho
@ramalhoorg

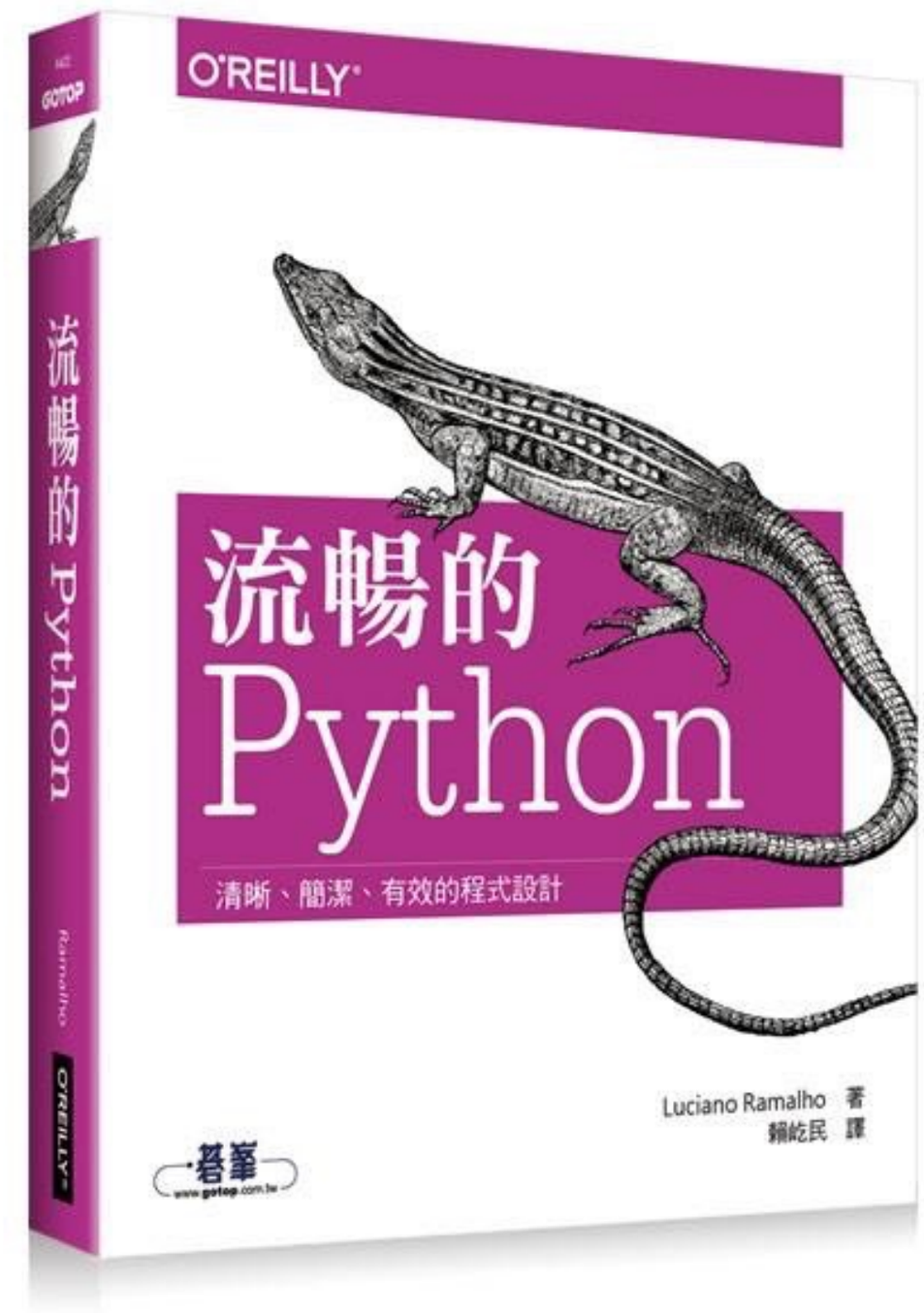
ThoughtWorks®

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com

FLUENT PYTHON, MEU 1º LIVRO



4.7 stars at Amazon.com

Fluent Python (O'Reilly, 2015)

Python Fluente (Novatec, 2015)

Python к вершинам мастерства (DMK, 2015)

流暢的 Python (Gotop, 2016)

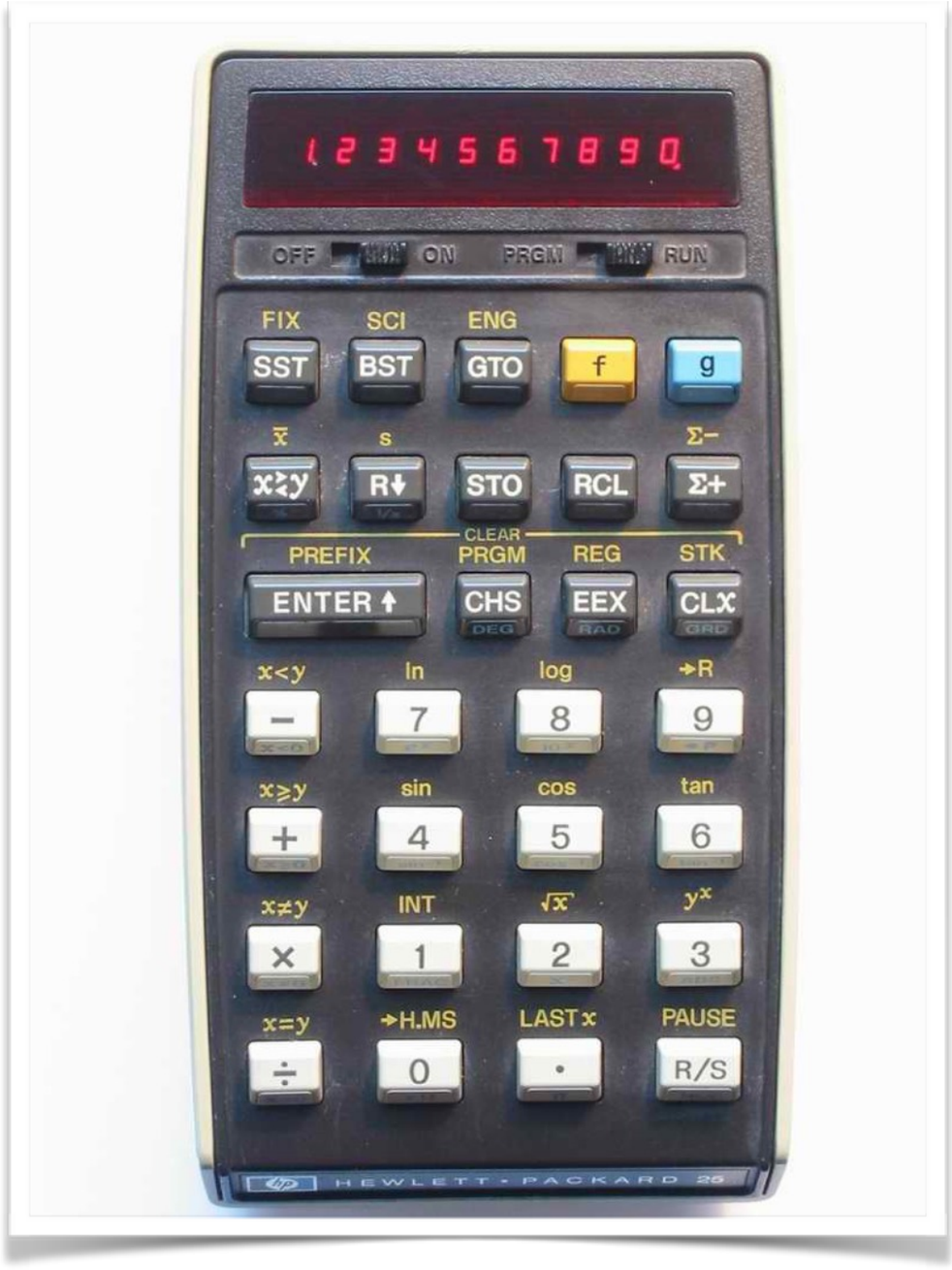
also in **Polish, Korean, etc...**

ThoughtWorks®

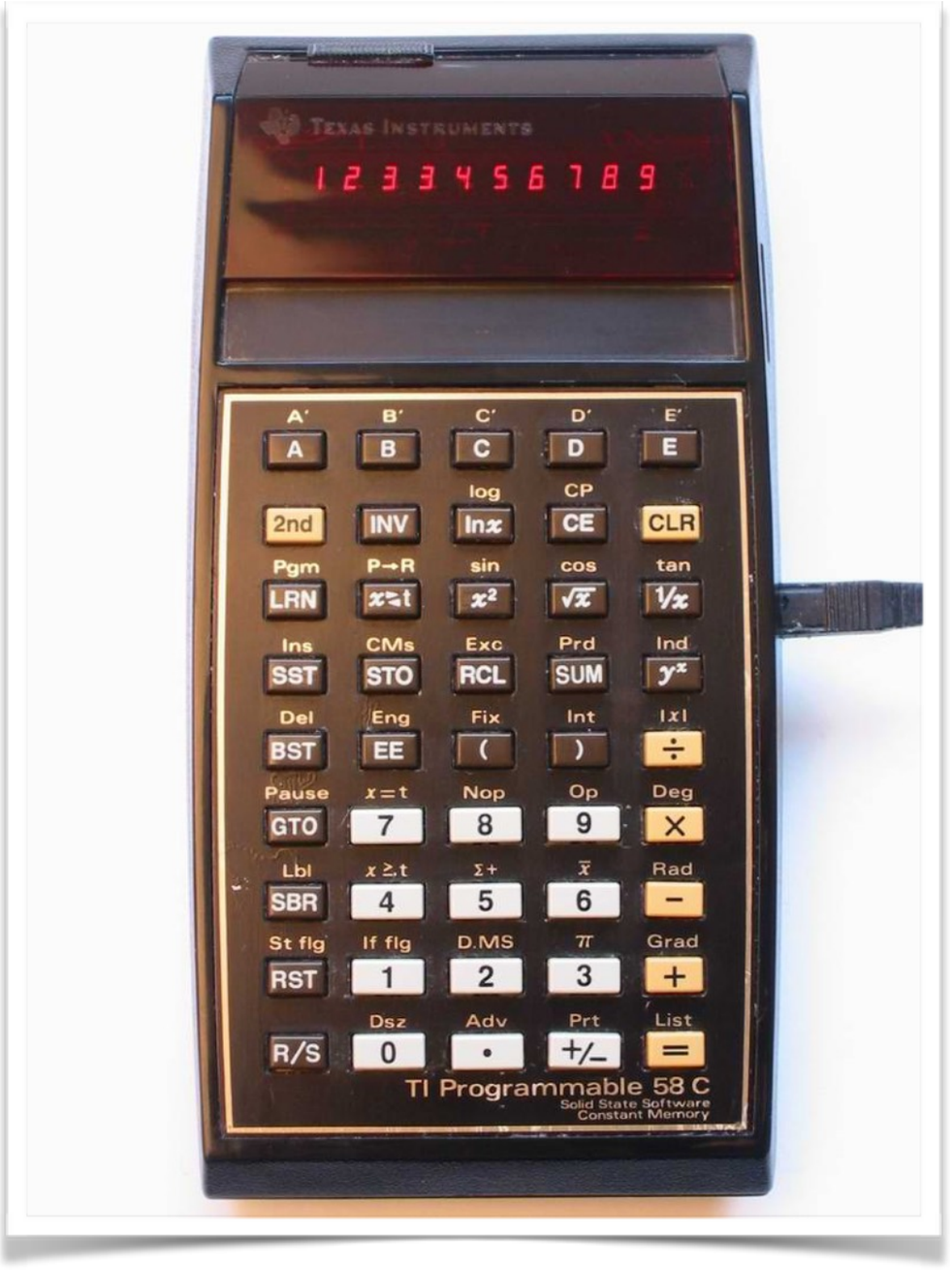
PARADIGMAS

Categorias de linguagens de programação

O PRIMEIRO “PARADIGMA" QUE APRENDI: IMPERATIVO



HP-25

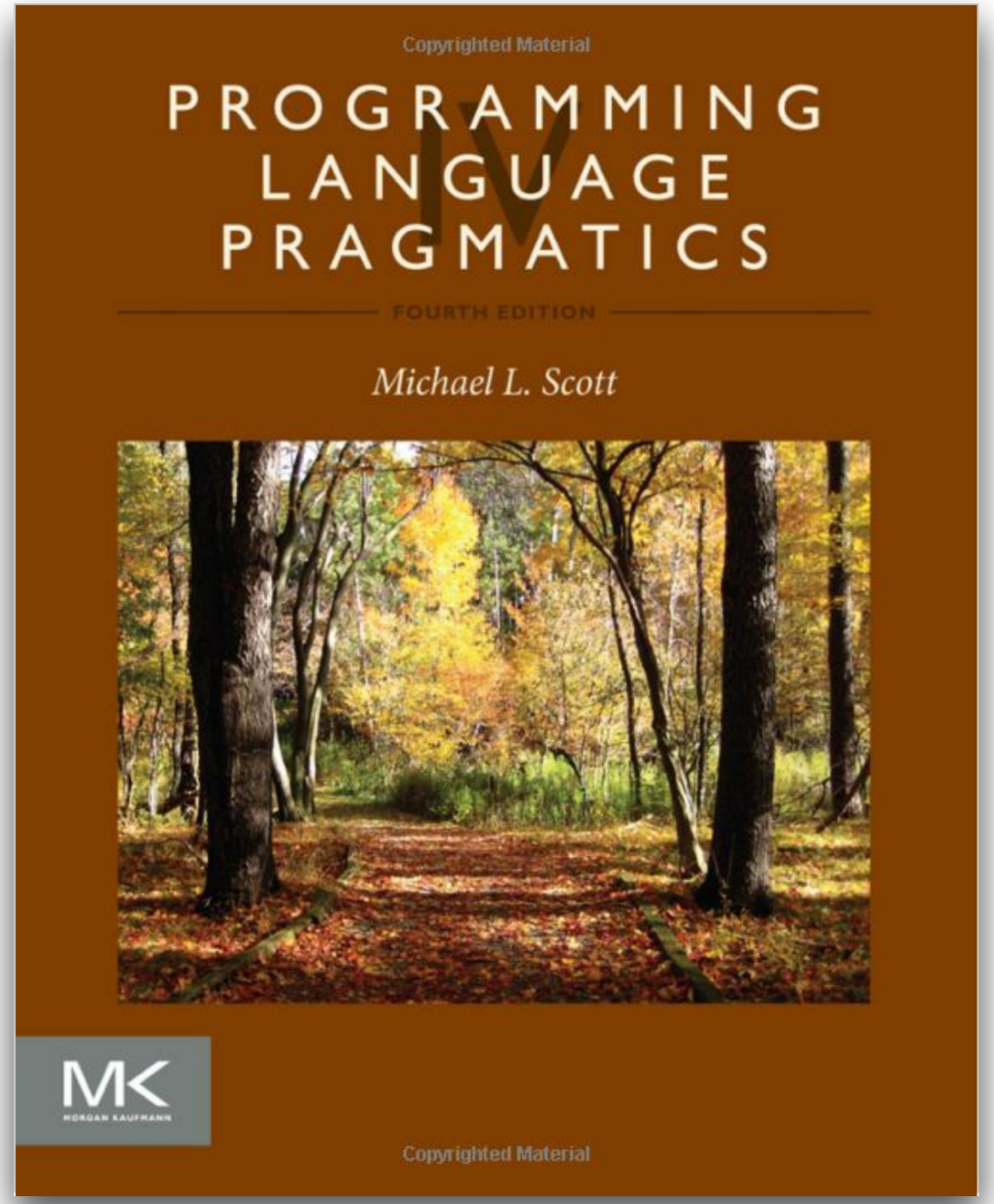


TI 58C

PROGRAMA PARA CALCULADORA HP-25

Visor		Intro-	X	Y	Z	T	
Linha	Código	dução					
00							
01	14 11 04	f FIX 4					Apres
02	24 00	RCL 0	X				Apres
03	33	EEX	1. 00	X			
04	04	4	1. 04	X			
05	71	÷	$X/10^4$				Divid
06	24 01	RCL 1	V	$X/10^4$			
07	15 41	g x<0	V	$X/10^4$			V é n
08	13 11	GTO 11	V	$X/10^4$			Sim,
09	51	+	$V + X/10^4$				Não,
10	13 13	GTO 13	$V + X/10^4$				
11	21	x⇄y	$X/10^4$	V			V < 0
12	41	–	$V - X/10^4$				
13	74	R/S	V.X				V.X =
14	24 02	RCL 2	F	B			Quein
15	14 41	f x<y	F	B			Comb
16	13 34	GTO 34	F	B			Sim,
17	22	R↓	B			F	Não,
18	23 41 02	STO – 2	B			F	Subtr
19	05	5	5	B			Gravi

PANORAMA SOBRE LINGUAGENS DE PROGRAMAÇÃO



Programming Language Pragmatics, 4th edition (2015)

Michael L. Scott

MDC: ASM X86

Máximo divisor comum em Assembly x86

(Scott, 2015)

```
        pushl   %ebp                # \
        movl    %esp, %ebp          #  ) reserve space for local variables
        subl    $16, %esp           # /
        call    getint              # read
        movl    %eax, -8(%ebp)      # store i
        call    getint              # read
        movl    %eax, -12(%ebp)     # store j
A:      movl    -8(%ebp), %edi      # load i
        movl    -12(%ebp), %ebx     # load j
        cmpl    %ebx, %edi          # compare
        je      D                   # jump if i == j
        movl    -8(%ebp), %edi      # load i
        movl    -12(%ebp), %ebx     # load j
        cmpl    %ebx, %edi          # compare
        jle     B                   # jump if i < j
        movl    -8(%ebp), %edi      # load i
        movl    -12(%ebp), %ebx     # load j
        subl    %ebx, %edi          # i = i - j
        movl    %edi, -8(%ebp)      # store i
        jmp     C
B:      movl    -12(%ebp), %edi     # load j
        movl    -8(%ebp), %ebx      # load i
        subl    %ebx, %edi          # j = j - i
        movl    %edi, -12(%ebp)     # store j
C:      jmp     A
D:      movl    -8(%ebp), %ebx      # load i
        push    %ebx                # push i (pass to putint)
        call    putint              # write
        addl    $4, %esp            # pop i
        leave                       # deallocate space for local variables
        mov     $0, %eax            # exit status for program
        ret                         # return to operating system
```

Figure 1.7 Naive x86 assembly language for the GCD program.

MDC EM C, OCAML, PROLOG

```c
int gcd(int a, int b) {                                  // C
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

```
let rec gcd a b =                                        (* OCaml *)
    if a = b then a
    else if a > b then gcd b (a - b)
         else gcd a (b - a)
```

```
gcd(A,B,G) :- A = B, G = A.                              % Prolog
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

Figure 1.2 The GCD algorithm in C (top), OCaml (middle), and Prolog (bottom). All three versions assume (without checking) that their inputs are positive integers.

MDC EM PYTHON

```python
def gcd(a, b):
    assert a > 0 and b > 0
    while a != b:
        if a > b:
            a -= b
        else:
            b -= a
    return a
```

Estilo imperativo

```python
def gcd(a, b):
    assert a > 0 and b > 0
    if a == b:
        return a
    elif a > b:
        return gcd(b, a - b)
    else:
        return gcd(a, b - a)
```

Estilo funcional

MDC EM PYTHON

Inadequado para Python que não faz otimização de chamada de cauda (TCO)

```python
def gcd(a, b):
    assert a > 0 and b > 0
    while a != b:
        if a > b:
            a -= b
        else:
            b -= a
    return a
```

Estilo imperativo

```python
def gcd(a, b):
    assert a > 0 and b > 0
    if a == b:
        return a
    elif a > b:
        return gcd(b, a - b)
    else:
        return gcd(a, b - a)
```

Estilo funcional

UMA CLASSIFICAÇÃO

1.2 The Programming Language Spectrum

Example 1.3

Classification of programming languages

The many existing languages can be classified into families based on their model of computation. Figure 1.1 shows a common set of families. The top-level division distinguishes between the *declarative* languages, in which the focus is on *what* the computer is to do, and the *imperative* languages, in which the focus is on *how* the computer should do it.■

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, …
object-oriented	Smalltalk, Eiffel, Java, …
scripting	Perl, Python, PHP, …

FIGURE 1.1 Classification of programming languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

Programming Language Pragmatics, 4th edition (2015)

Michael L. Scott

UMA CLASSIFICAÇÃO (ZOOM)

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, …
object-oriented	Smalltalk, Eiffel, Java, …
scripting	Perl, Python, PHP, …

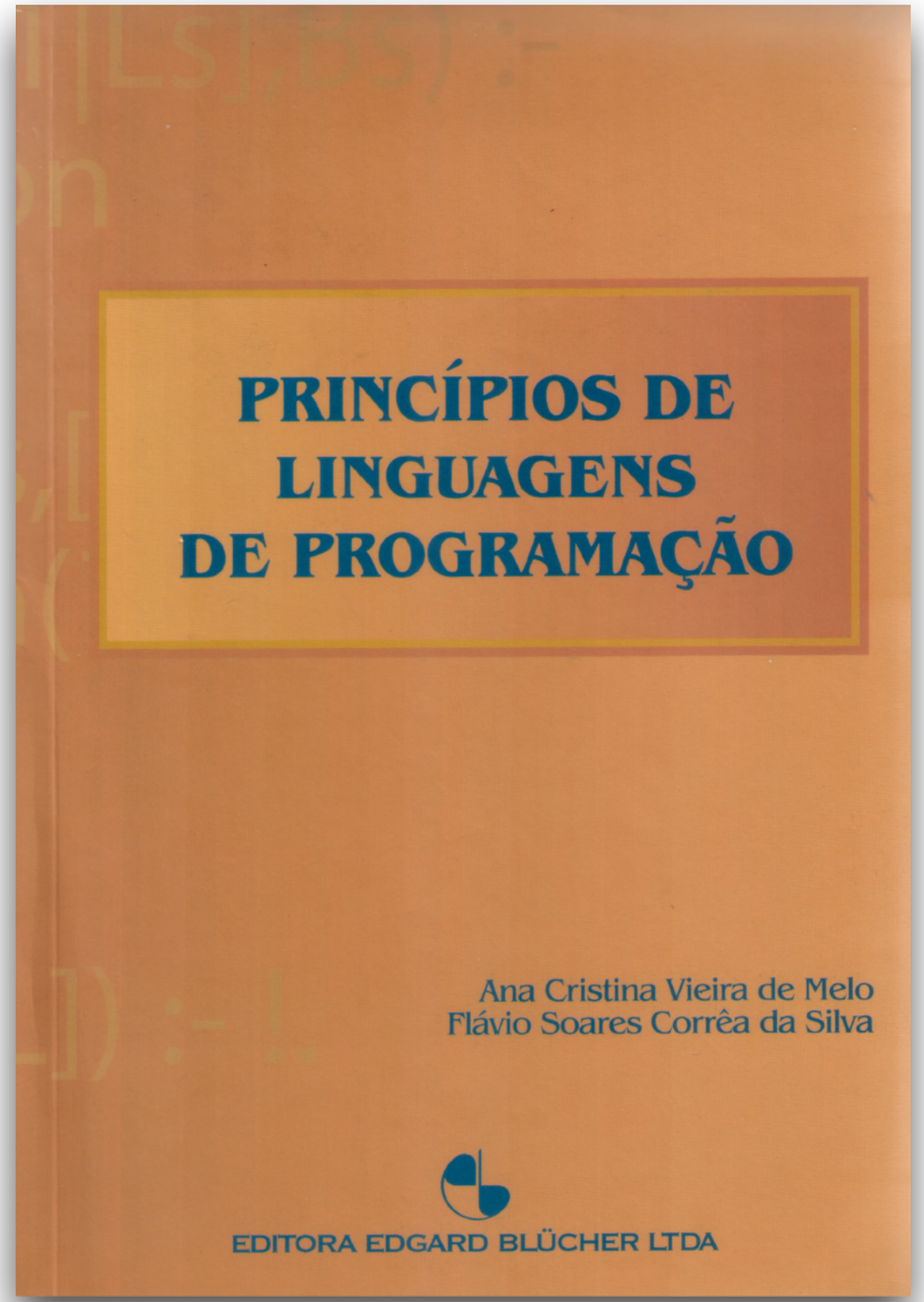
FIGURE 1.1 Classification of programming languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

UMA CLASSIFICAÇÃO (ZOOM)

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, …
object-oriented	Smalltalk, Eiffel, Java, …
scripting ???	Perl, Python, PHP, …

FIGURE 1.1 Classification of programming languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

OUTRO LIVRO, OUTRA CLASSIFICAÇÃO



Princípios de Linguagens de Programação
(2003)

Ana Cristina Vieira de Melo

Flávio Soares Corrêa da Silva

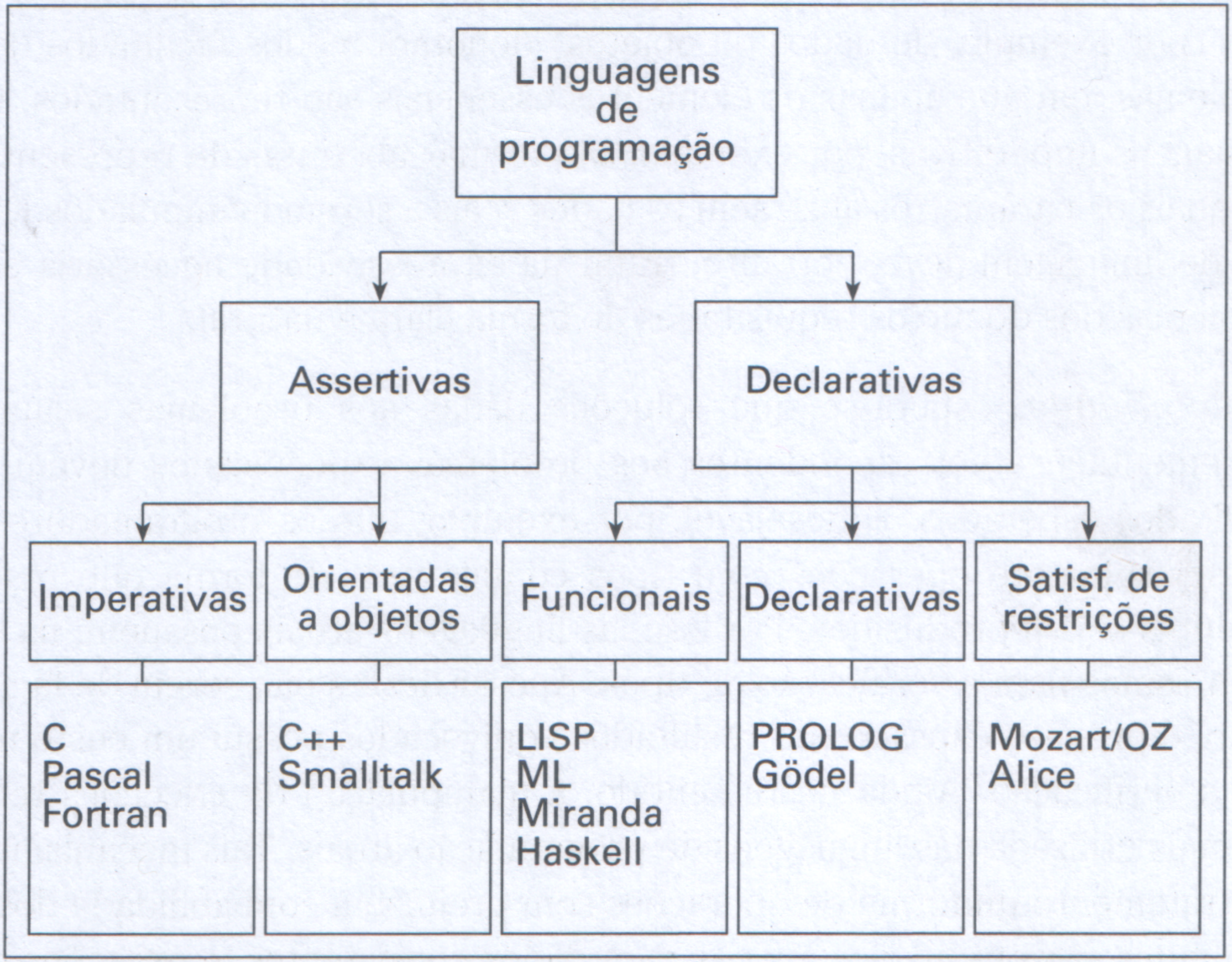


Figura 1 — Tipologia de linguagens de programação.

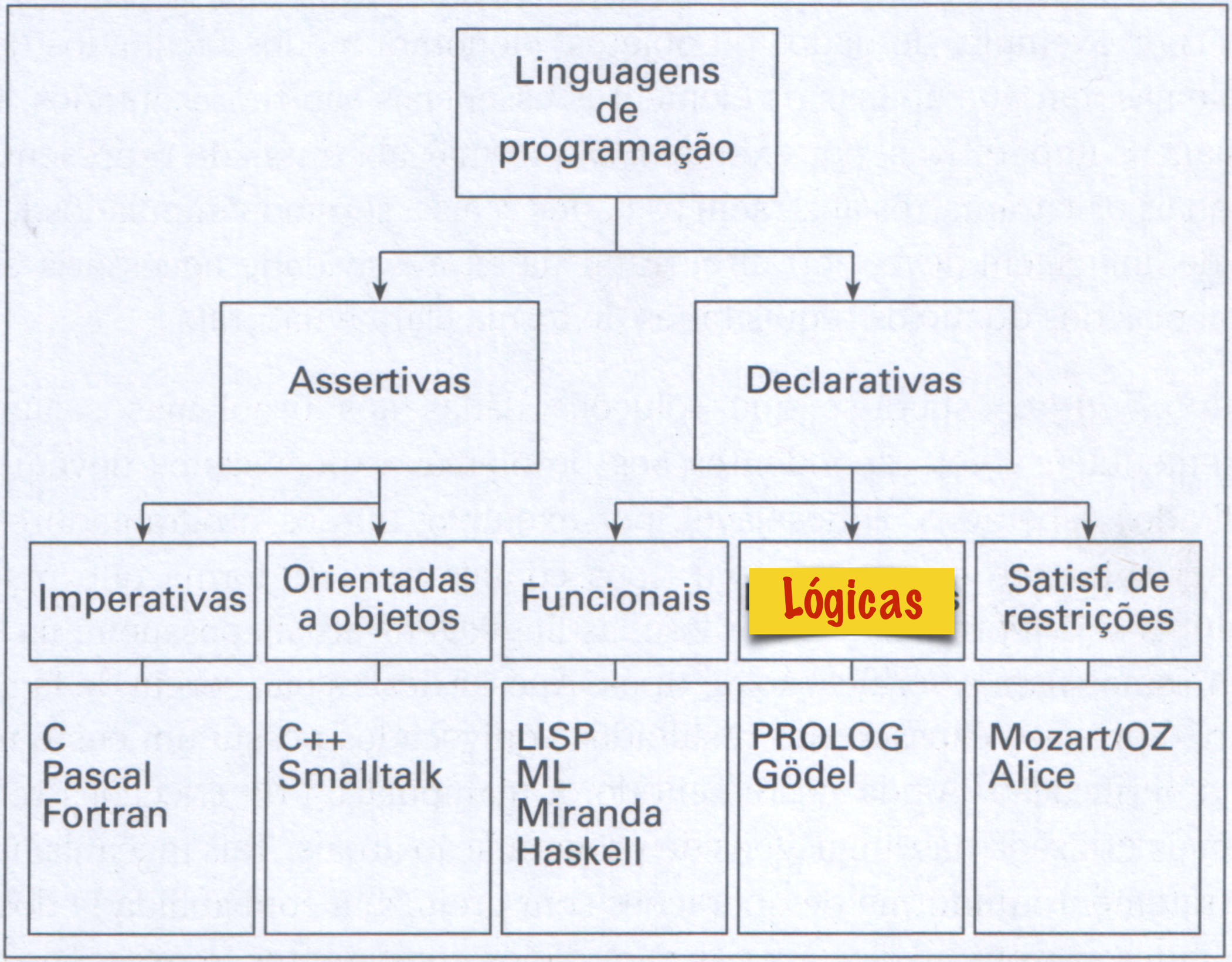


Figura 1 — Tipologia de linguagens de programação.

SITE: THE LANGUAGE LIST

Firefox File Edit View History Bookmarks Tools Window Help — Wed 19:56

The Language List

www.info.univ-angers.fr/~gh/hilapr/langlist/langlist.htm — 170% — Search

The Language List

Collected Information On About 2500 Computer Languages, Past and Present.

Maintained by Bill Kinnersley

Welcome to The Language List! Early versions of this list were posted to comp.lang.misc beginning in 1991. Now a web site, our intention remains the same -- to become one of the most complete sources of information on computer programming languages ever assembled (or compiled :-).

The list does not pretend to be a definitive scholarly work. Its purpose is to collect and provide timely information in a rapidly growing field. Its accuracy and completeness depend to a great extent on the users of the Internet. If you know about a language that should be added, please share your knowledge.

Start a Search

Contents of an Entry — What Languages Should be Included

Language Categories — Dialects, Variants, Versions and Implementations

References — A Chronology of Influential Languages

SITE QUEBRADO, LINGUAGENS FALTANDO...

Search the Language List

www.info.univ-angers.fr/~gh/hilapr/langlist/search.htm

Search for a particular language entry:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Home

Not Found

The requested URL /~nkinners/LangList/Indexes/gindex.htm was not found on this server.

Apache/2.2.15 (Red Hat) Server at people.ku.edu Port 80

CATEGORIAS DE LINGUAGENS

Language Categories

Procedural Language

A language which states how to compute the result of a given problem. This term encompasses both imperative and functional languages.

Imperative Language

A language which operates by a sequence of commands that change the value of data elements. Imperative languages are typified by assignments and iteration.

Declarative Language

A language which operates by making descriptive statements about data, and relations between data. The algorithm is hidden in the semantics of the language. This category encompasses both applicative and logic languages. Examples of declarative features are set comprehensions and pattern-matching statements.

Applicative Language

A language that operates by application of functions to values, with no side effects. A functional language in the broad sense.

Functional Language

In the narrow sense, a functional language is one that operates by use of higher-order functions, building operators that manipulate functions directly without ever appearing to manipulate data. Example: FP.

CATEGORIAS DE LINGUAGENS (2)

Constraint Language

A language in which a problem is specified and solved by a series of constraining relationships.

Object-Oriented Language

A language in which data and the functions which access it are treated as a unit.

Concurrent Language

A concurrent language describes programs that may be executed in parallel. This may be either

- Multiprogramming: sharing one processor
- Multiprocessing: separate processors sharing one memory
- Distributed

Concurrent languages differ in the way that processes are created:

- Coroutines - control is explicitly transferred - examples are Simula I, SL5, BLISS and Modula-2.
- Fork/join - examples are PL/I and Mesa.
- Cobegin/coend - examples are ALGOL 68, CSP, Edison, Argus.
- Process declarations - examples are DP, SR, Concurrent Pascal, Modula, PLITS and Ada.

and the ways in which processes interact:

- Semaphores - ALGOL 68
- Conditional critical regions - Edison, DP, Argus
- Monitors - Concurrent Pascal, Modula
- Message passing - CSP, PLITS, Gypsy, Actors
- Remote procedure calls - DP, *Mod
    - Rendezvous - Ada, SR

CATEGORIAS DE LINGUAGENS (3)

- Message passing - CSP, PLITS, Gypsy, Actors
- Remote procedure calls - DP, *Mod
  - Rendezvous - Ada, SR
  - Atomic transactions - Argus

Fourth Generation Language (4GL)

A very high-level language. It may use natural English or visual constructs. Algorithms or data structures may be selected by the compiler.

Query Language

An interface to a database.

Specification Language

A formalism for expressing a hardware or software design.

Assembly Language

A symbolic representation of the machine language of a specific computer.

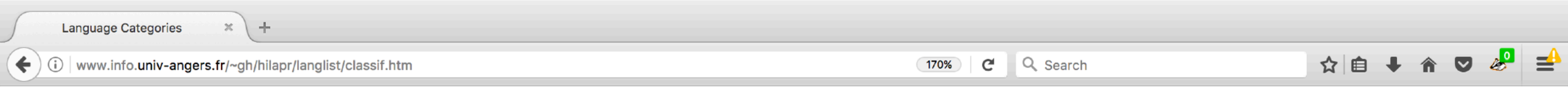
Intermediate Language

A language used as an intermediate stage in compilation. May be either text or binary.

Metalanguage

A language used for the formal description of another language.

CATEGORIAS DE LINGUAGENS (4)



Definitional Language

An applicative language containing assignments interpreted as definitions. Example: Lucid.

Single Assignment Language

An applicative language using assignments, with the convention that a variable may appear on the left side of an assignment only once within the portion of the program in which it is active.

Dataflow Language

A language suitable for use on a dataflow architecture. Necessary properties include freedom from side effects, and the equivalence of scheduling constraints with data dependencies. Examples: Val, Id, SISAL, Lucid.

Logic Language

A logic language deals with predicates or relationships p(X,Y). A program consists of a set of Horn clauses which may be:

- facts - p(X,Y) is true
- rules - p is true if q1 and q2 and ... qn are true
- queries - is g1 and g2 and ... gn true? (gi's are the goals.)

Further clauses are inferred using resolution. One clause is selected containing p as an assumption, another containing p as a consequence, and p is eliminated between them. If the two p's have different arguments they must be unified, using the substitution with the fewest constraints that makes them the same. Logic languages try alternative resolutions for each goal in succession, backtracking in a search for a common solution.

- OR-parallel logic languages try alternative resolutions in parallel
- AND-parallel logic languages try to satisfy several goals in parallel.

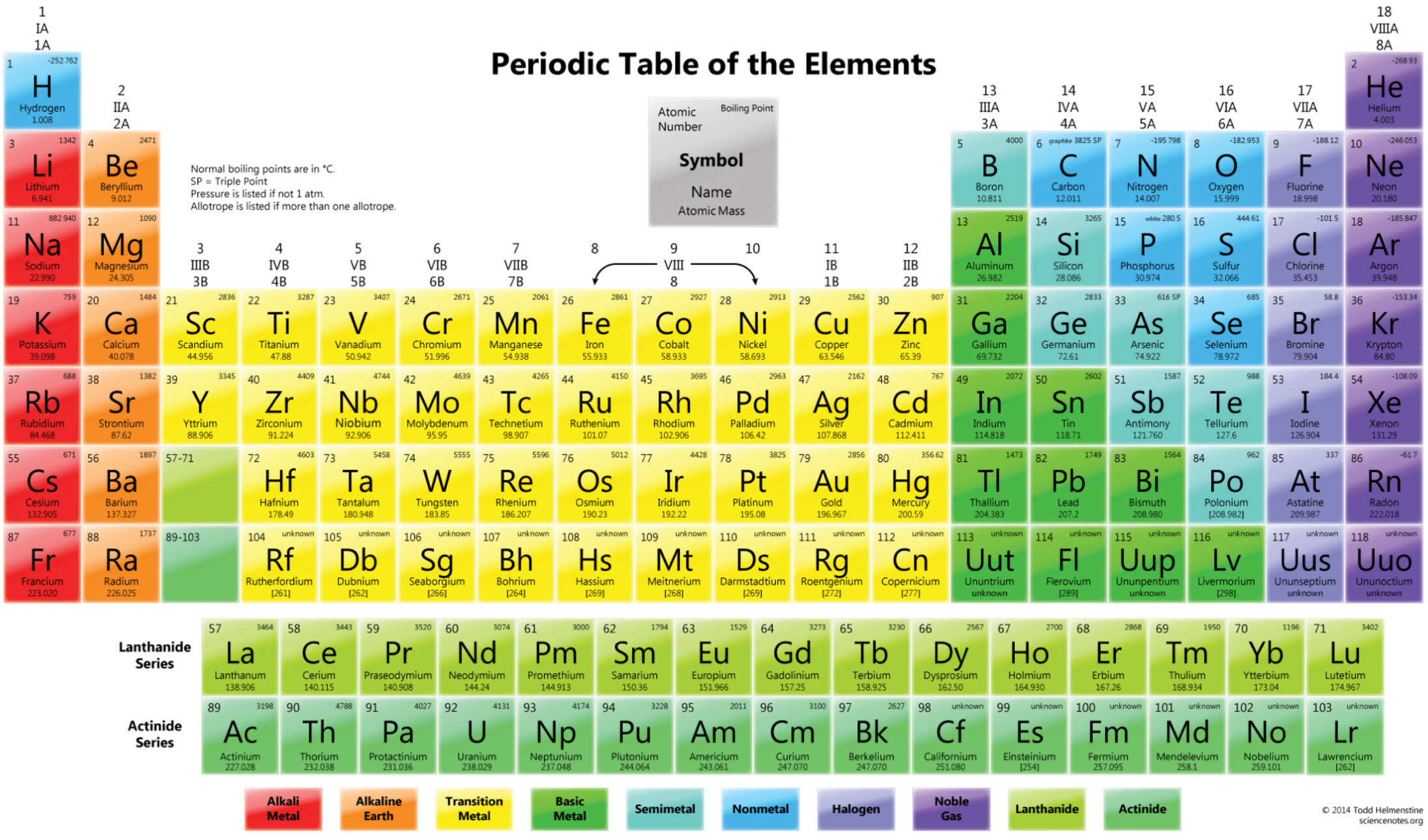
Constraint Language

ThoughtWorks®

CATEGORIAS?

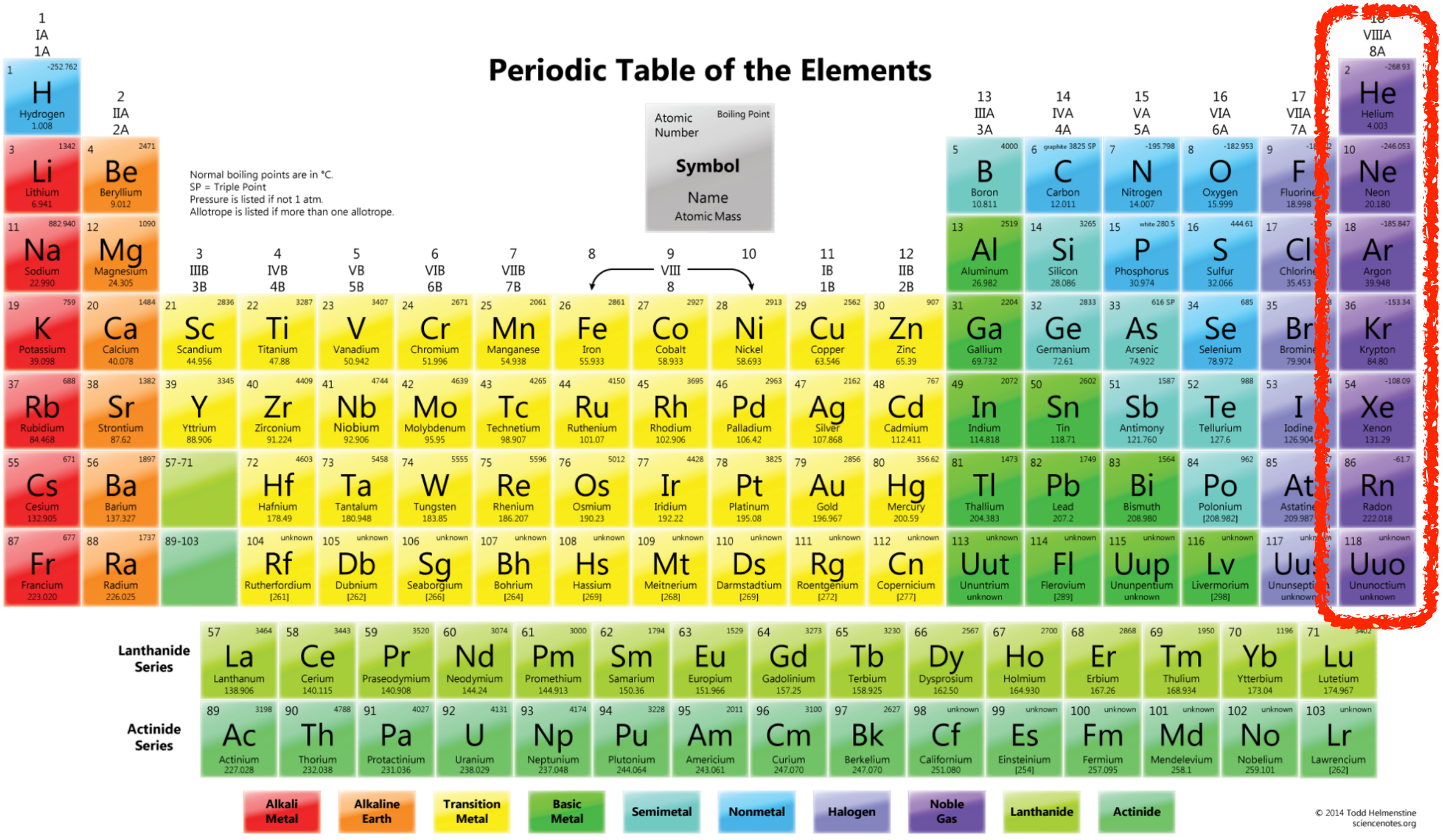
Ontologia é tão anos 90...

UMA CLASSIFICAÇÃO BASEADA EM FATOS CIENTÍFICOS



UMA CLASSIFICAÇÃO BASEADA EM FATOS CIENTÍFICOS?

"Gases" "nobres"!?



“Ontology is overrated.”

Clay Shirky

Ontologias são sobrevalorizadas.
Pesquise o artigo em inglês!

ThoughtWorks®

UMA ABORDAGEM MELHOR

Fundamental Features of Programming Languages

ENSINO MODERNO DE TEORIA DE LINGUAGENS

Teaching Programming Langua ×

cs.brown.edu/~sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming

Ensinando linguagens de programação na era pós-Lineu

Abstract

> Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

Comment

The book discussed in this paper is available here.

Paper

PDF

These papers may differ in formatting from the versions that appear in print. They are made available only to support the rapid dissemination of results; the printed versions, not these, should be considered

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly
abandon this method of
reconciles an important split in programming language education, and describes a textbook that explores these matters.

"Paradigmas" de linguagens de programação são um legado moribundo e tedioso...

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching
reconciles an important split in p
describes a textbook that explores these matters.

Criadores de linguagens modernas não têm o menor respeito por eles...

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

...devemos abandonar esse método no ensino de linguagens...

do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM

Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language "paradigms" are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method o
reconciles an important split in programming language education, and describes a textbook that explores these matters.

...um livro-texto que explora esses assuntos.

TEORIA NA PRÁTICA COM RACKET (DIALETOS DE SCHEME)

TEORIA NA PRÁTICA COM PASCAL (1990)

Programming Languages: Sam × | Goodreads | Programming Lan × | site:norvig.com kamin - Google ×

Secure | https://www.goodreads.com/book/show/2082799.Programming_Languages?from_search=true

goodreads Home My Books Browse Community Search books

Read

My rating:

Programming Languages: An Interpreter-Based Approach

by Samuel N. Kamin

5.0 · Rating details · 1 Rating · 0 Reviews

GET A COPY

Amazon BR | Online Stores | Book Links

Hardcover, 640 pages
Published January 1st 1990 by Addison Wesley Publishing Company
More Details... edit details

Recommend It | Stats | Recent Status Updates

FRIEND REVIEWS

Recommend This Book None of your friends have reviewed this book yet.

READER Q&A

Ask the Goodreads community a question about Programming Languages

Ask anything about the book

Be the first to ask a question about Programming Languages

LISTS WITH THIS BOOK

GENRES

Science › Computer Science 1 user

See top shelves...

BOOKS BY SAMUEL N. KAMIN

More...

ThoughtWorks®

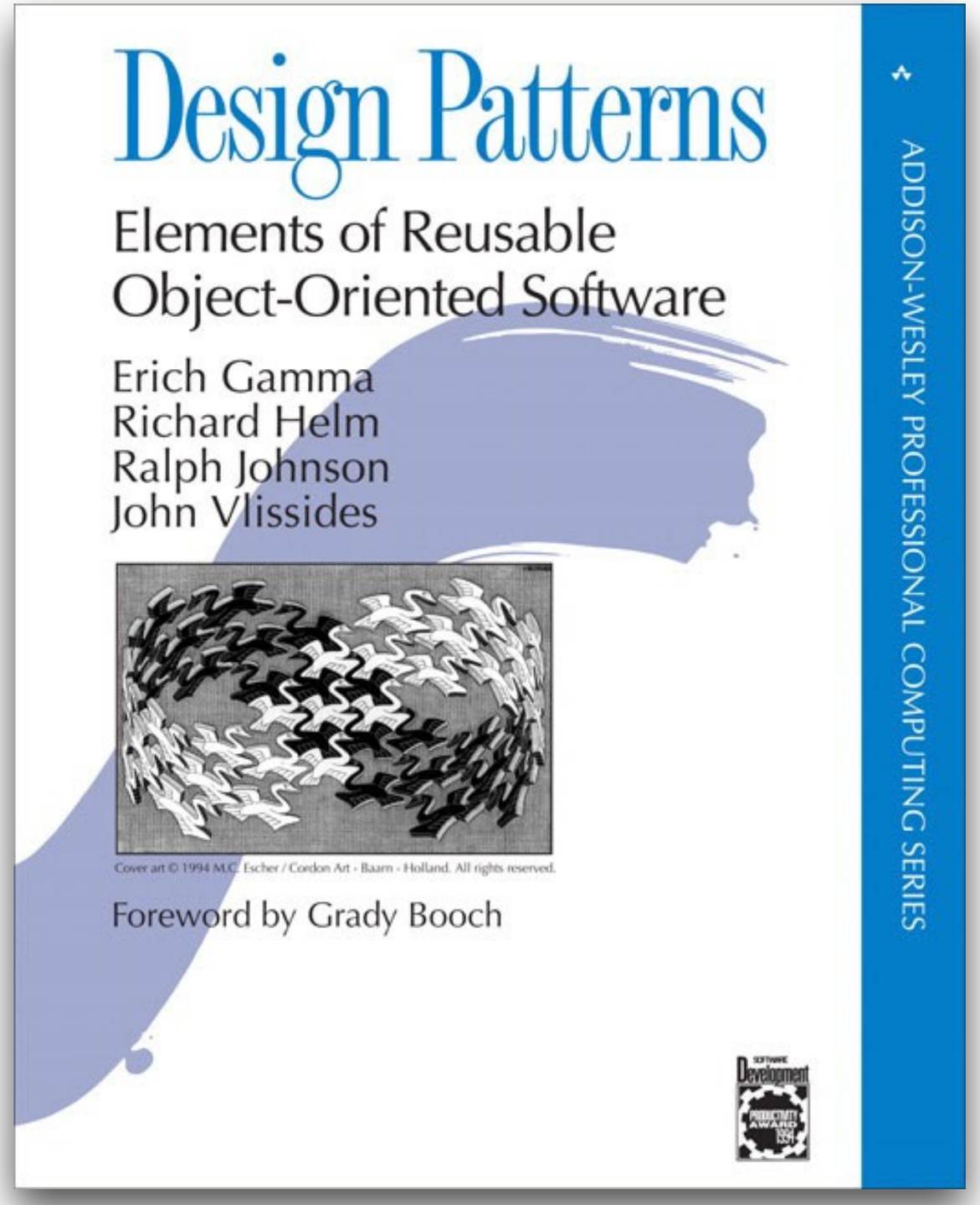
PADRÕES DE PROJETO

Quando as linguagens deixam a desejar

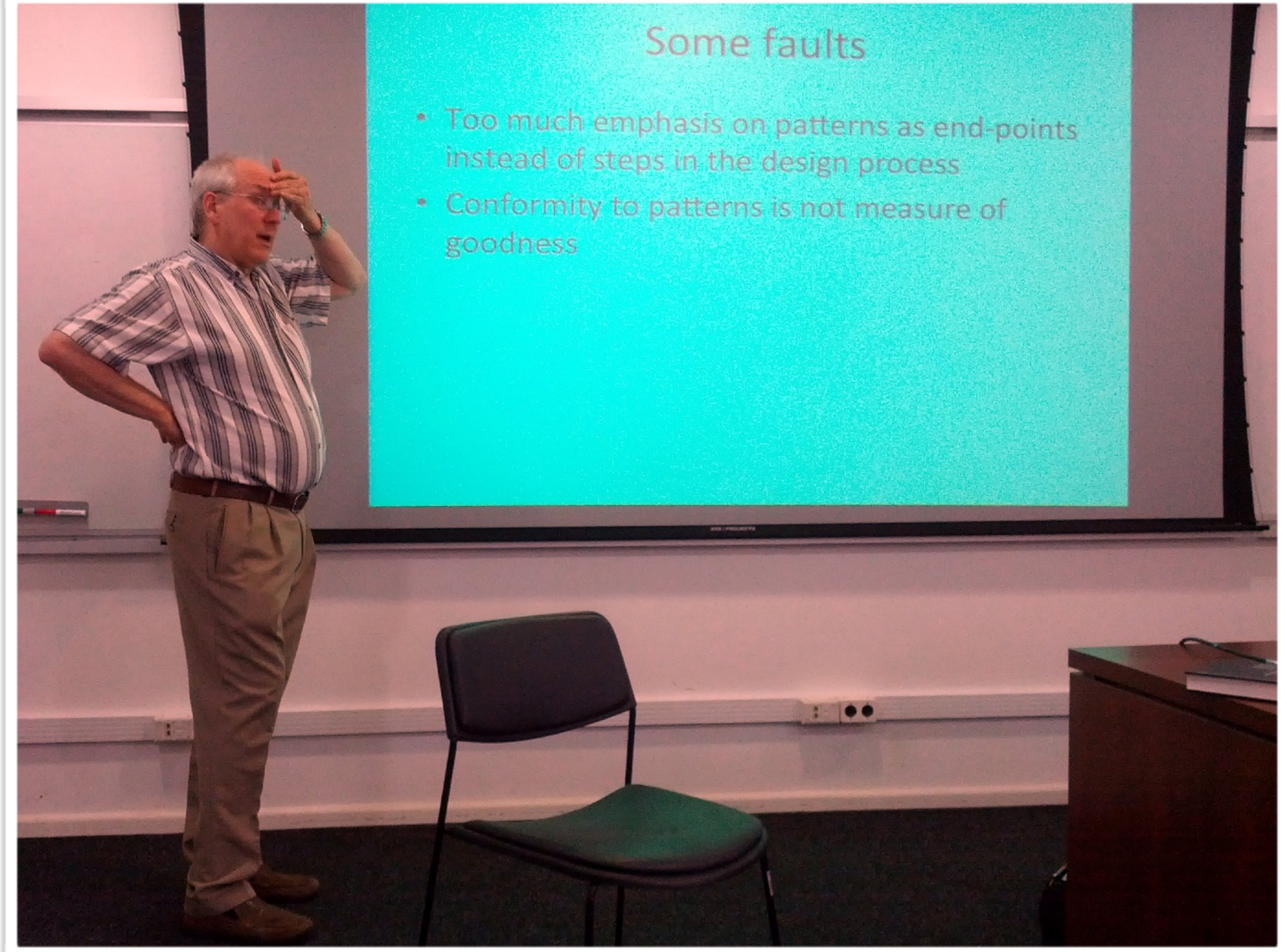
GOF: LIVRO CLÁSSICO DA “GANG OF FOUR”

Design Patterns:
Elements of Reusable
Object-Oriented
Software (1995)

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

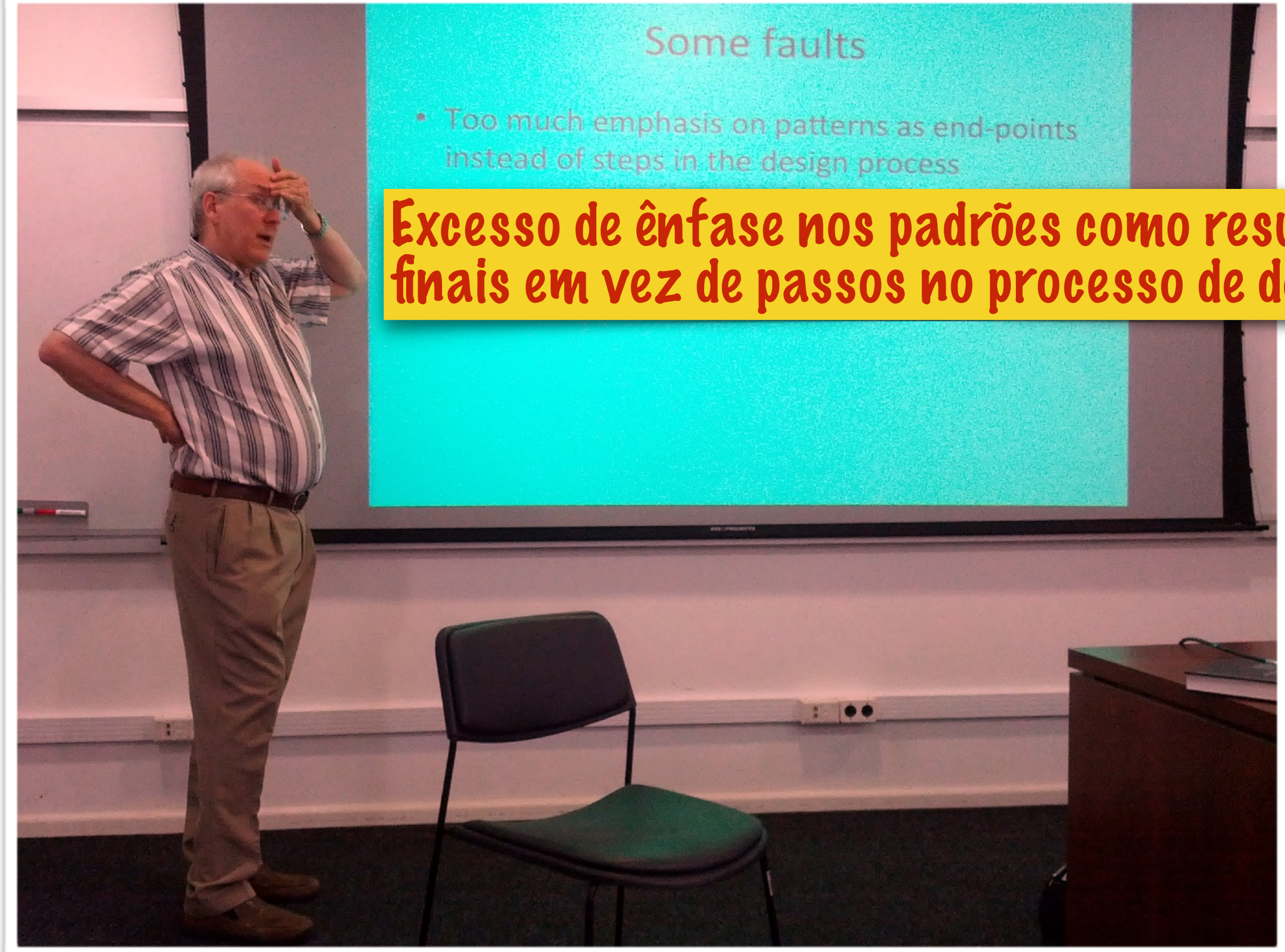


PROBLEMAS NA DIFUSÃO DOS PADRÕES



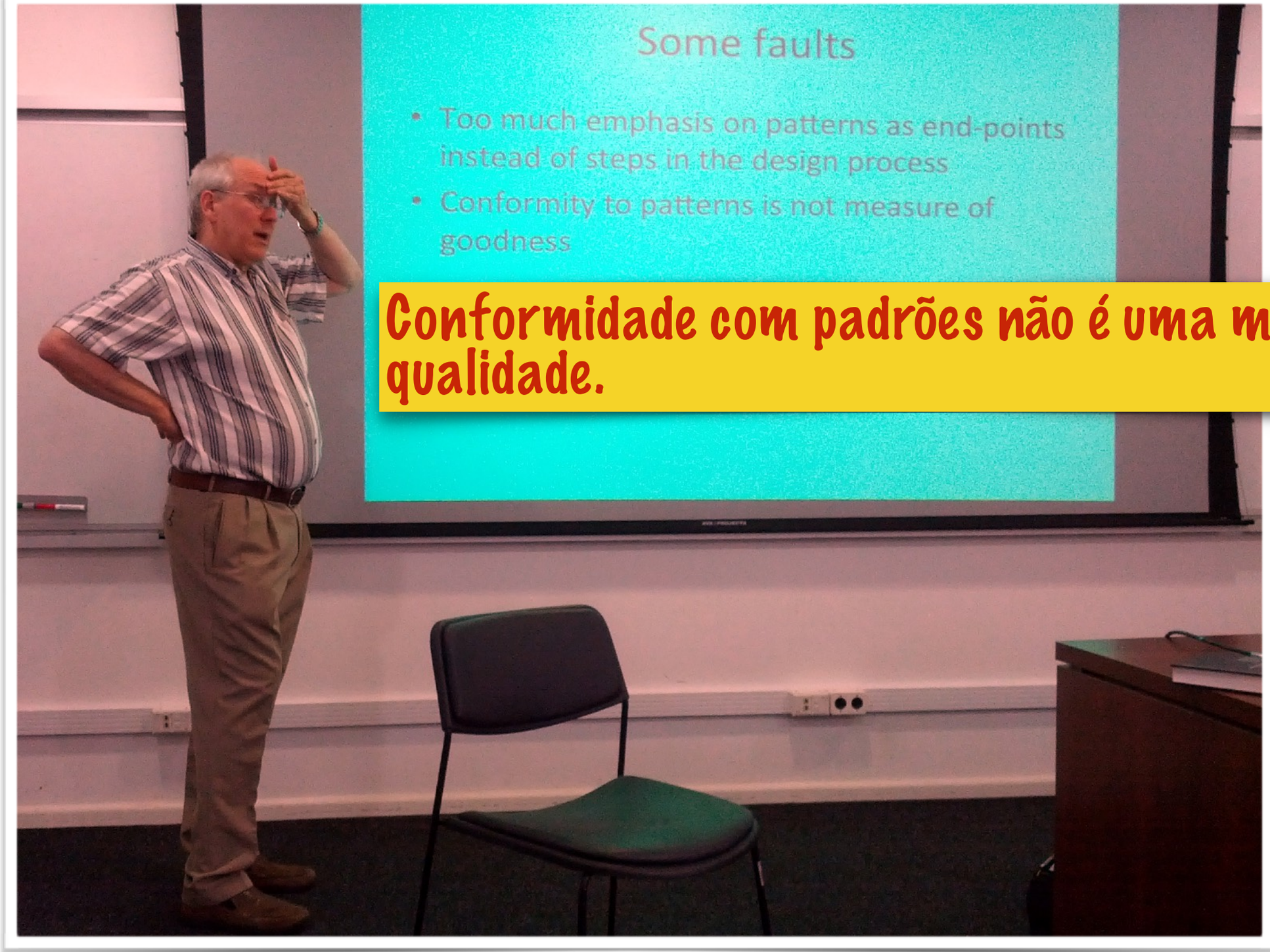
Ralph Johnson no IME/USP (2014?)

PROBLEMAS NA DIFUSÃO DOS PADRÕES



Ralph Johnson no IME/USP (2014?)

PROBLEMAS NA DIFUSÃO DOS PADRÕES



Ralph Johnson no IME/USP (2014?)

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included design patterns called "Inheritance," "Encapsulation," and "Polymorphism." Similarly, some of our patterns are supported directly by the less common object-oriented languages. CLOS has multi-methods, for example, which lessen the need for a pattern such as Visitor.[2]

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995), p. 4.

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have inc
and "Polymorphism." Sim
common object-oriented
lessen the need for a pattern such as Visitor.[2]

Nossos padrões assumem características no nível de Smalltalk/C++...

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995), p. 4.

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included design patterns called "Inheritance," "Encapsulation," and "Polymorphism." Similarly, some of our patterns are supported directly by the less common object-oriented languages. CLOS has multi-methods, for example, which lessen the need for a pattern such as Visitor.[2]

Se tivéssemos assumido linguagens procedurais, talvez tivéssemos incluído padrões como "Herança", "Encapsulamento" e "Polimorfismo".

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995), p. 4.

Design Patterns in Dynamic Programming

Peter Norvig
Chief Designer, Adaptive Systems
Harlequin Inc.

(2) Design Patterns in Dynamic Languages

- Dynamic Languages have fewer language limitations
  Less need for bookkeeping objects and classes
  Less need to get around class-restricted design
- Study of the *Design Patterns* book:
  16 of 23 patterns have qualitatively simpler implementation in Lisp or Dylan than in C++ for at least some uses of each pattern
- Dynamic Languages encourage new designs
  We will see some in Part (3)

(2) Design Patterns in Dynamic Languages

- Dynamic Languages have fewer language limitations
  Less need for bookkeeping objects and classes
  Less need to get around class-restricted design
- Study of the *Design Patterns* book:
  16 of 23 patterns have qualitatively simpler implementation in Lisp or Dylan than in C++
  for at least some uses of each pattern
- Dynamic Languages encourage new designs
  We will see sc

Estudo do livro "Padrões de Projeto": 16 dos 23 padrões têm implementações qualitativamente mais simples em Lisp ou Dylan do que em C++, ao menos para alguns usos de cada padrão.

Design Patterns in Dylan or Lisp

16 of 23 patterns are either invisible or simpler, due to:

- First-class types (6): `Abstract-Factory`, `Flyweight`, `Factory-Method`, `State`, `Proxy`, `Chain-Of-Responsibility`
- First-class functions (4): `Command`, `Strategy`, `Template-Method`, `Visitor`
- Macros (2): `Interpreter`, `Iterator`
- Method Combination (2): `Mediator`, `Observer`
- Multimethods (1): `Builder`
- Modules (1): `Facade`

16 dos 23 padrões são invisíveis ou mais simples, devido a essas características

ThoughtWorks®

CARACTERÍSTICAS

Características essenciais, não apenas açúcar sintático.

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

	Common Lisp
Funções de 1ª classe	✔
Tipos de 1ª classe	✔
Iteradores	*
Modelo de variáveis	referência
Checagem de tipos	dinâmica
Expressão de tipos	estrutural

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

	Common Lisp
Funções são objetos	✔
Classes são objetos	✔
Iteradores	*
Modelo de variáveis	referência
Checagem de tipos	dinâmica
Expressão de tipos	estrutural

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

	Common Lisp	C
Funções de 1ª classe	✔	*
Tipos de 1ª classe	✔	
Iteradores	*	
Modelo de variáveis	referência	valor*
Checagem de tipos	dinâmica	estática
Expressão de tipos	estrutural	nominal

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

	Common Lisp	C	Java
Funções de 1ª classe	✔	*	✔
Tipos de 1ª classe	✔		
Iteradores	*		✔
Modelo de variáveis	referência	valor*	valor e referência
Checagem de tipos	dinâmica	estática	estática
Expressão de tipos	estrutural	nominal	nominal

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

	Common Lisp	C	Java	Go
Funções de 1ª classe	✔	*	✔	✔
Tipos de 1ª classe	✔			
Iteradores	*		✔	*
Modelo de variáveis	referência	valor*	valor e referência	valor* e referência
Checagem de tipos	dinâmica	estática	estática	estática
Expressão de tipos	estrutural	nominal	nominal	estrutural

AMOSTRA DE CARACTERÍSTICAS ✖ LINGUAGENS

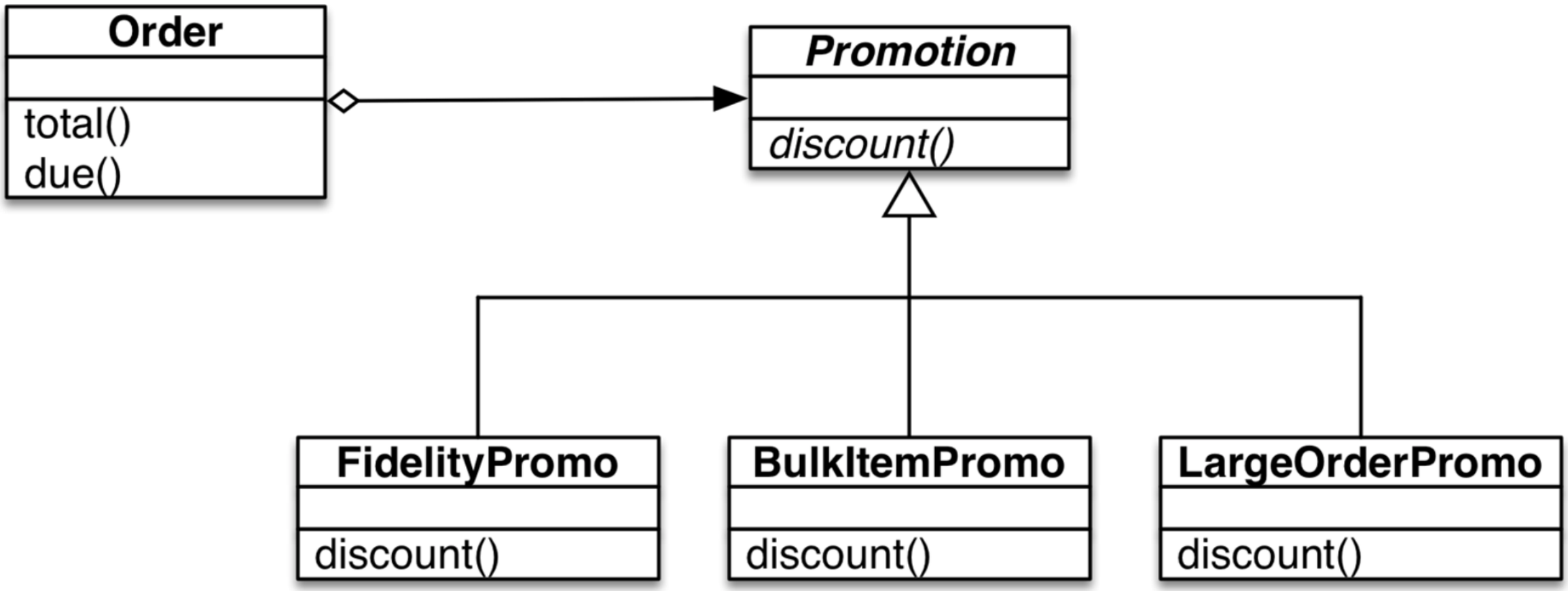
	Common Lisp	C	Java	Go	Python
Funções de 1ª classe	✔	*	✔	✔	✔
Tipos de 1ª classe	✔				✔
Iteradores	*		✔	*	✔
Modelo de variáveis	referência	valor*	valor e referência	valor* e referência	referência
Checagem de tipos	dinâmica	estática	estática	estática	dinâmica
Expressão de tipos	estrutural	nominal	nominal	estrutural	estrutural

ThoughtWorks®

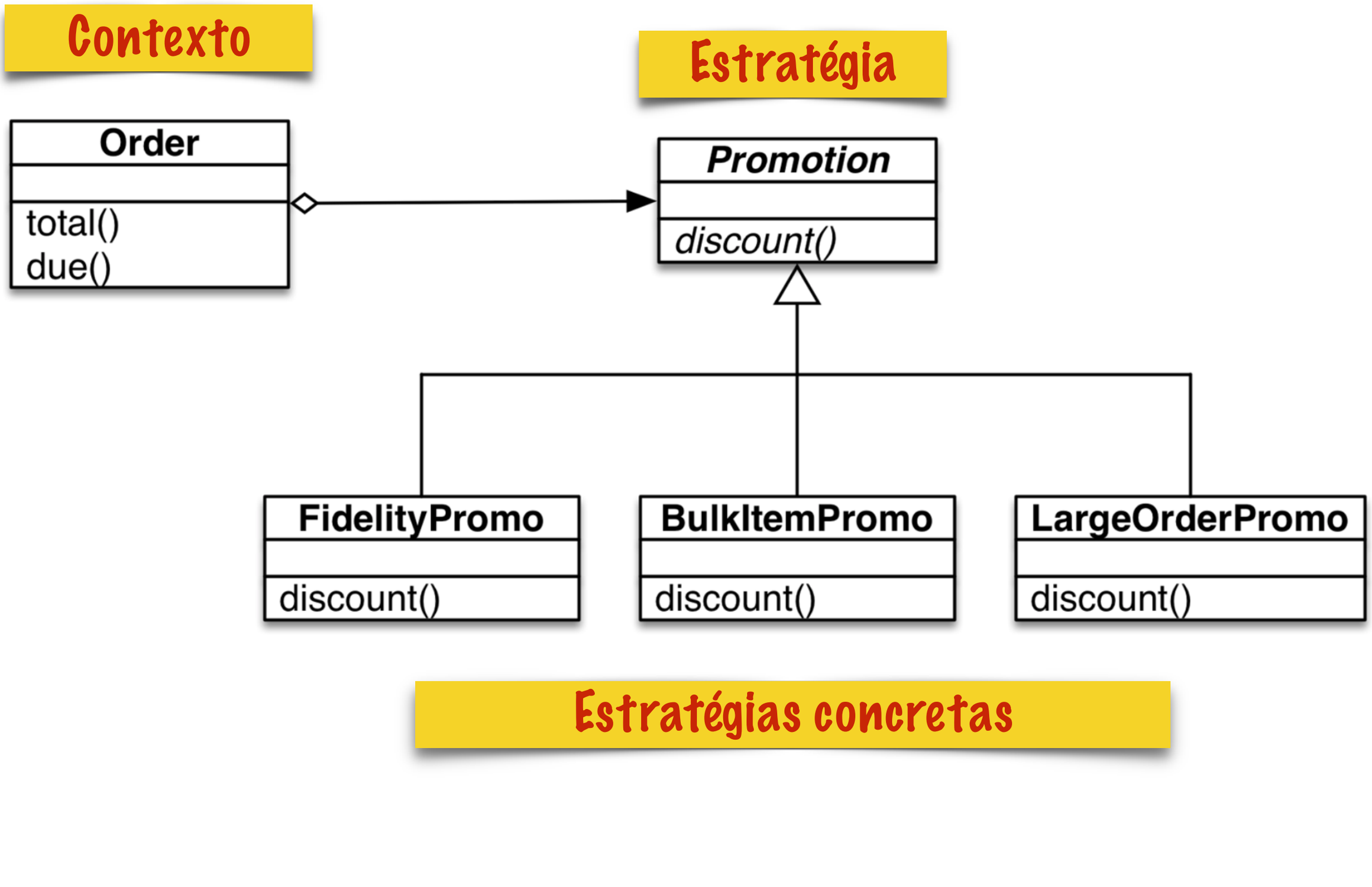
STRATEGY EM PYTHON

Aproveitando características de Python

SELEÇÃO DE UM ALGORITMO EM TEMPO DE EXECUÇÃO



SELEÇÃO DE UM ALGORITMO EM TEMPO DE EXECUÇÃO



DOCTESTS: CONTEXTO E UMA ESTRATÉGIA CONCRETA

```
Create two customers, with and without "fidelity points":

    >>> ann = Customer('Ann Smith', 1100)  # ❶
    >>> joe = Customer('John Doe', 0)

Create a shopping cart with some fruits:

    >>> cart = [LineItem('banana', 4, .5),  # ❷
    ...         LineItem('apple', 10, 1.5),
    ...         LineItem('watermellon', 5, 5.0)]

The `FidelityPromo` only gives a discount to Ann:

    >>> Order(joe, cart, FidelityPromo())
    <Order total: 42.00 due: 42.00>
    >>> Order(ann, cart, FidelityPromo())
    <Order total: 42.00 due: 39.90>
```

Instância de Estratégia (Promo) é passada ao construtor do Contexto (Order: o pedido)

DOCTESTS: MAIS DUAS ESTRATÉGIAS CONCRETAS

```
The `BulkItemPromo` gives a discount for items with 20+ units:

    >>> banana_cart = [LineItem('banana', 30, .5),  # ❺
    ...                LineItem('apple', 10, 1.5)]
    >>> Order(joe, banana_cart, BulkItemPromo())  # ❻
    <Order total: 30.00 due: 28.50>

`LargeOrderPromo` gives a discount for orders with 10+ items:

    >>> long_order = [LineItem(str(item_code), 1, 1.0) # ❼
    ...               for item_code in range(10)]
    >>> Order(joe, long_order, LargeOrderPromo())  # ❽
    <Order total: 10.00 due: 9.30>
```

VARIAÇÕES DE STRATEGY EM PYTHON

Implementação **clássica** usando ABC

Implementação com **funções de 1ª classe**

Implementação parametrizada com **closure**

Implementação parametrizada com **classe invocável**

STRATEGY CLÁSSICO: A CLASSE DO CONTEXT

```
class Order:  # the Context

    def __init__(self, customer, cart, promotion=None):
        self.customer = customer
        self.cart = list(cart)
        self.promotion = promotion

    def total(self):
        if not hasattr(self, '__total'):
            self.__total = sum(item.total() for item in self.cart)
        return self.__total

    def due(self):
        if self.promotion is None:
            discount = 0
        else:
            discount = self.promotion.discount(self)
        return self.total() - discount

    def __repr__(self):
```

Estratégia é passada para o construtor

Estratégia é usada aqui

STRATEGY: CLASSE ABSTRATA E SUBCLASSE CONCRETA

```
class Promotion(ABC):  # the Strategy: an Abstract Base Class

    @abstractmethod
    def discount(self, order):
        """Return discount as a positive dollar amount"""
```

```
class FidelityPromo(Promotion):  # first Concrete Strategy
    """5% discount for customers with 1000 or more fidelity points"""

    def discount(self, order):
        return order.total() * .05 if order.customer.fidelity >= 1000 else 0
```

MAIS DUAS ESTRATÉGIAS CONCRETAS

```
class BulkItemPromo(Promotion):  # second Concrete Strategy
    """10% discount for each LineItem with 20 or more units"""

    def discount(self, order):
        discount = 0
        for item in order.cart:
            if item.quantity >= 20:
                discount += item.total() * .1
        return discount


class LargeOrderPromo(Promotion):  # third Concrete Strategy
    """7% discount for orders with 10 or more distinct items"""

    def discount(self, order):
        distinct_items = {item.product for item in order.cart}
        if len(distinct_items) >= 10:
            return order.total() * .07
        return 0
```

ThoughtWorks®

STRATEGY COM FUNÇÃO DE 1ª CLASSE

CONTEXTO: FUNÇÃO DE ESTRATÉGIA COMO ARGUMENTO

Função de estratégia é passada ao construtor de Order

```
>>> banana_cart = [LineItem('banana', 30, .5),
...                LineItem('apple', 10, 1.5)]
>>> Order(joe, banana_cart, bulk_item_promo)
<Order total: 30.00 due: 28.50>
>>> long_order = [LineItem(str(item_code), 1, 1.0)
...                for item_code in range(10)]
>>> Order(joe, long_order, large_order_promo)
<Order total: 10.00 due: 9.30>
>>> Order(joe, cart, large_order_promo)
<Order total: 42.00 due: 42.00>
```

CONTEXTO: USO DA FUNÇÃO DE ESTRATÉGIA

classic_strategy.py ↔ strategy.py

```
        self.customer = customer
        self.cart = list(cart)
        self.promotion = promotion

    def total(self):
        if not hasattr(self, '__total'):
            self.__total = sum(item.total() for item
        return self.__total

    def due(self):
        if self.promotion is None:
            discount = 0
        else:
            discount = self.promotion.discount(self)
        return self.total() - discount

    def __repr__(self):
        fmt = '<Order total: {:.2f} due: {:.2f}>'
        return fmt.format(self.total(), self.due())


ss Promotion(ABC):  # the Strategy: an Abstract B

 @abstractmethod
```

```
        self.customer = customer
        self.cart = list(cart)
        self.promotion = promotion

    def total(self):
        if not hasattr(self, '__total'):
            self.__total = sum(item.total()
        return self.__total

    def due(self):
        if self.promotion is None:
            discount = 0
        else:
            discount = self.promotion(self)
        return self.total() - discount

    def __repr__(self):
        fmt = '<Order total: {:.2f} due: {:.
        return fmt.format(self.total(), self.

fidelity_promo(order):
```

ESTRATÉGIAS CONCRETAS COMO FUNÇÕES

classic_strategy.py ↔ strategy.py

```
-class Promotion(ABC):  # the Strategy: an Abstrac    +def fidelity_promo(order):
-
-    @abstractmethod
-    def discount(self, order):
-        """Return discount as a positive dollar a
-
-
-class FidelityPromo(Promotion):  # first Concrete        """5% discount for customers with 100
-    """5% discount for customers with 1000 or mor
-
-    def discount(self, order):
         return order.total() * .05 if order.custo        return order.total() * .05 if order.c


-class BulkItemPromo(Promotion):  # second Concret    +def bulk_item_promo(order):
     """10% discount for each LineItem with 20 or        """10% discount for each LineItem wit
-
-    def discount(self, order):
         discount = 0                                    discount = 0
         for item in order.cart:                         for item in order.cart:
             if item.quantity >= 20:                         if item.quantity >= 20:
                 discount += item.total() * .1                   discount += item.total() * .1
         return discount                                 return discount
```

ESTRATÉGIAS CONCRETAS COMO FUNÇÕES (2)

classic_strategy.py ↔ strategy.py

```
-class BulkItemPromo(Promotion):  # second Concret
     """10% discount for each LineItem with 20 or
-
-    def discount(self, order):
         discount = 0
         for item in order.cart:
             if item.quantity >= 20:
                 discount += item.total() * .1
         return discount


-class LargeOrderPromo(Promotion):  # third Concre
     """7% discount for orders with 10 or more dis
-
-    def discount(self, order):
         distinct_items = {item.product for item i
         if len(distinct_items) >= 10:
             return order.total() * .07
         return 0
```

```
+def bulk_item_promo(order):
     """10% discount for each LineItem wit
     discount = 0
     for item in order.cart:
         if item.quantity >= 20:
             discount += item.total() * .1
     return discount


+def large_order_promo(order):
     """7% discount for orders with 10 or
     distinct_items = {item.product for it
     if len(distinct_items) >= 10:
         return order.total() * .07
     return 0
```

ThoughtWorks®

ESTRATÉGIA PARAMETRIZADA COM CLOSURE

ESTRATÉGIA PARAMETRIZADA COM CLOSURE

função promo é invocada com porcentagem de desconto

```
>>> banana_cart = [LineItem('banana', 30, .5),
...                LineItem('apple', 10, 1.5)]
>>> Order(joe, banana_cart, bulk_item_promo(10))
<Order total: 30.00 due: 28.50>
>>> long_order = [LineItem(str(item_code), 1, 1.0)
...                for item_code in range(10)]
>>> Order(joe, long_order, large_order_promo(7))
<Order total: 10.00 due: 9.30>
>>> Order(joe, cart, large_order_promo(7))
<Order total: 42.00 due: 42.00>
```

IMPLEMENTAÇÃO COM CLOSURE

Função externa recebe argumento 'percent'

```
def bulk_item_promo(percent):
    """discount for each LineItem with 20 or more units"""
    def discounter(order):
        discount = 0
        for item in order.cart:
            if item.quantity >= 20:
                discount += item.total() * percent/100.0
        return discount
    return discounter
```

Função interna leva associação de `percent` em sua closure

LAMBDA: ATALHO PARA DEFINIR A FUNÇÃO INTERNA

strategy.py ↔ strategy_param.py

```
-def fidelity_promo(order):
-    """5% discount for customers with 1000 or mor
-    return order.total() * .05 if order.customer.
+def fidelity_promo(percent):
+    """discount for customers with 1000 o
+    return lambda order: (order.total() *
+                          if order.custom


-def bulk_item_promo(order):
-    """10% discount for each LineItem with 20 or
+def bulk_item_promo(percent):
+    """discount for each LineItem with 20
     discount = 0
     for item in order.cart:
         if item.quantity >= 20:
-            discount += item.total() * .1
+            discount += item.total()
     return discount
+    return discounter


-def large_order_promo(order):
-    """7% discount for orders with 10 or more dis
+def large_order_promo(percent):
+    """discount for orders with 10 or mor
     distinct_items = {item.product for item in or
     if len(distinct_items) >= 10:
-        return order.total() * .07
+        return order.total() * percen
     return 0
```

ThoughtWorks®

ESTRATÉGIA PARAMETRIZADA COM INVOCÁVEL

ESTRATÉGIA PARAMETRIZADA COM INVOCÁVEL

objeto promo é instanciado com porcentagem de desconto

```
>>> banana_cart = [LineItem('banana', 30, .5),
...                LineItem('apple', 10, 1.5)]
>>> Order(joe, banana_cart, BulkItemPromo(10))
<Order total: 30.00 due: 28.50>
>>> long_order = [LineItem(str(item_code), 1, 1.0)
...                for item_code in range(10)]
>>> Order(joe, long_order, LargeOrderPromo(7))
<Order total: 10.00 due: 9.30>
>>> Order(joe, cart, LargeOrderPromo(7))
<Order total: 42.00 due: 42.00>
```

CLASSE IMPLEMENTADA COMO INVOCÁVEL

strategy_param.py ↔ strategy_param2.py

```
- def fidelity_promo(percent):
-     """discount for customers with 1000 or m
-     return lambda order: (order.total() * pe
-                           if order.customer.
-
-
- def bulk_item_promo(percent):
-     """discount for each LineItem with 20 or
-     def discounter(order):
-         discount = 0
```

```
+ class Promotion():
+     """compute discount for order"""
+
+     def __init__(self, percent):
+         self.percent = percent
+
+     def __call__(self, order):
+         raise NotImplementedError("Subclass re
+
+
+ class FidelityPromo(Promotion):
+     """discount for customers with 1000 or mor
+
+     def __call__(self, order):
+         if order.customer.fidelity >= 1000:
+             return order.total() * self.percen
+         return 0
+
+
+ class BulkItemPromo(Promotion):
+     """discount for each LineItem with 20 or m
+
+     def __call__(self, order):
+         discount = 0
```

SUB-CLASSES IMPLEMENTADAS COMO INVOCÁVEIS

strategy_param.py ↔ strategy_param2.py

```
- def bulk_item_promo(percent):                    + class BulkItemPromo(Promotion):
      """discount for each LineItem with 20 or more       """discount for each LineItem with 20
-     def discounter(order):                        +
                                                   +     def __call__(self, order):
          discount = 0                                         discount = 0
          for item in order.cart:                              for item in order.cart:
              if item.quantity >= 20:                              if item.quantity >= 20:
-                 discount += item.total() * percer    +                   discount += item.total()
          return discount                                      return discount
-     return discounter

- def large_order_promo(percent):                  + class LargeOrderPromo(Promotion):
      """discount for orders with 10 or more distir       """discount for orders with 10 or mor
-     def discounter(order):                        +
                                                   +     def __call__(self, order):
          distinct_items = {item.product for item i            distinct_items = {item.product fo
          if len(distinct_items) >= 10:                        if len(distinct_items) >= 10:
-             return order.total() * percent / 100.    +               return order.total() * self.p
          return 0                                             return 0
-     return discounter
```

ThoughtWorks®

QUAL A SOLUÇÃO MAIS IDIOMÁTICA?

Classes x funções

QUAL É MAIS IDIOMÁTICA?

Estratégia clássica *parece* muito verbosa em Python*

Funções de 1ª classe são muito comuns na biblioteca padrão

- Exemplo: argumento **key** nas funções embutidas **sorted**, **max**, **min**

** Sim, isto é subjetivo. Estamos falando de estilo!*

QUAL É MAIS IDIOMÁTICA COM PARÂMETROS?

Use de closures é comum em Python

- Python 3 ganhou a declaração **nonlocal** para suportar melhor essa prática

Objetos invocáveis são característica típica de Python

- Graham Dumpleton recomenda invocáveis para construir **@decorators**

ThoughtWorks®

CONCLUINDO

Aprenda as características fundamentais

PORQUE APRENDER AS CARACTERÍSTICAS FUNDAMENTAIS

Aprender novas linguagens mais rápido

Aproveitar melhor as características de cada linguagem

Escolher entre implementações alternativas

Fazer bom uso de padrões de projeto

Depurar problemas difíceis

Emular características que fazem falta

* Inspirado por *Programming Language Pragmatics*

  Michael L. Scott

MUITO GRATO!

Luciano Ramalho
luciano.ramalho@thoughtworks.com

Twitter: @ramalhoorg
Github: github.com/standupdev/paradigm-free
Slides: speakerdeck.com/ramalho

ThoughtWorks®