LE LANGAGE C++
MASTER 1

Jean-Baptiste.Yunes@univ-paris-diderot.fr
U.F.R. d'Informatique
Université Paris Diderot - Paris 7

2019–2020

Quels sont les buts de ce cours ?

- Une connaissance de la problématique de l'analyse/conception objet
- La maîtrise des éléments de base d'UML
- La connaissance de l'implémentation en C++ de fragments UML
- La maîtrise du langage C++, de ses variantes et de ses particularités

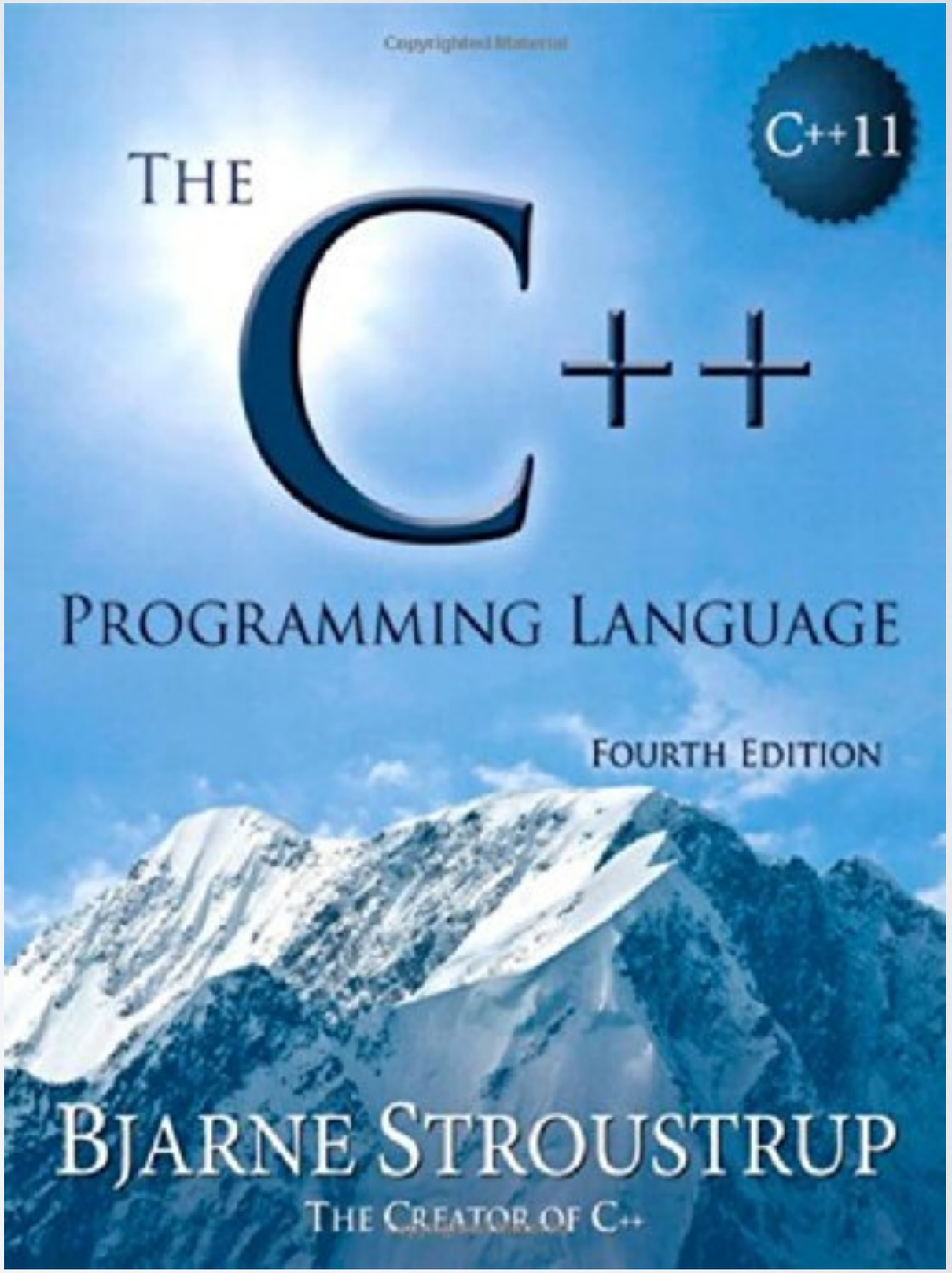
Éléments du cours (ordre non contractuel) :

1. Introduction
2. Un C plus plus ?
3. Le paradigme objet
4. Les classes
5. Du modèle aux classes
6. La spécialisation
7. La factorisation
8. L'héritage multiple
9. Les opérateurs
10. Les modèles
11. Les design patterns
12. Les exceptions
13. La Standard Template Library
14. La méta-programmation

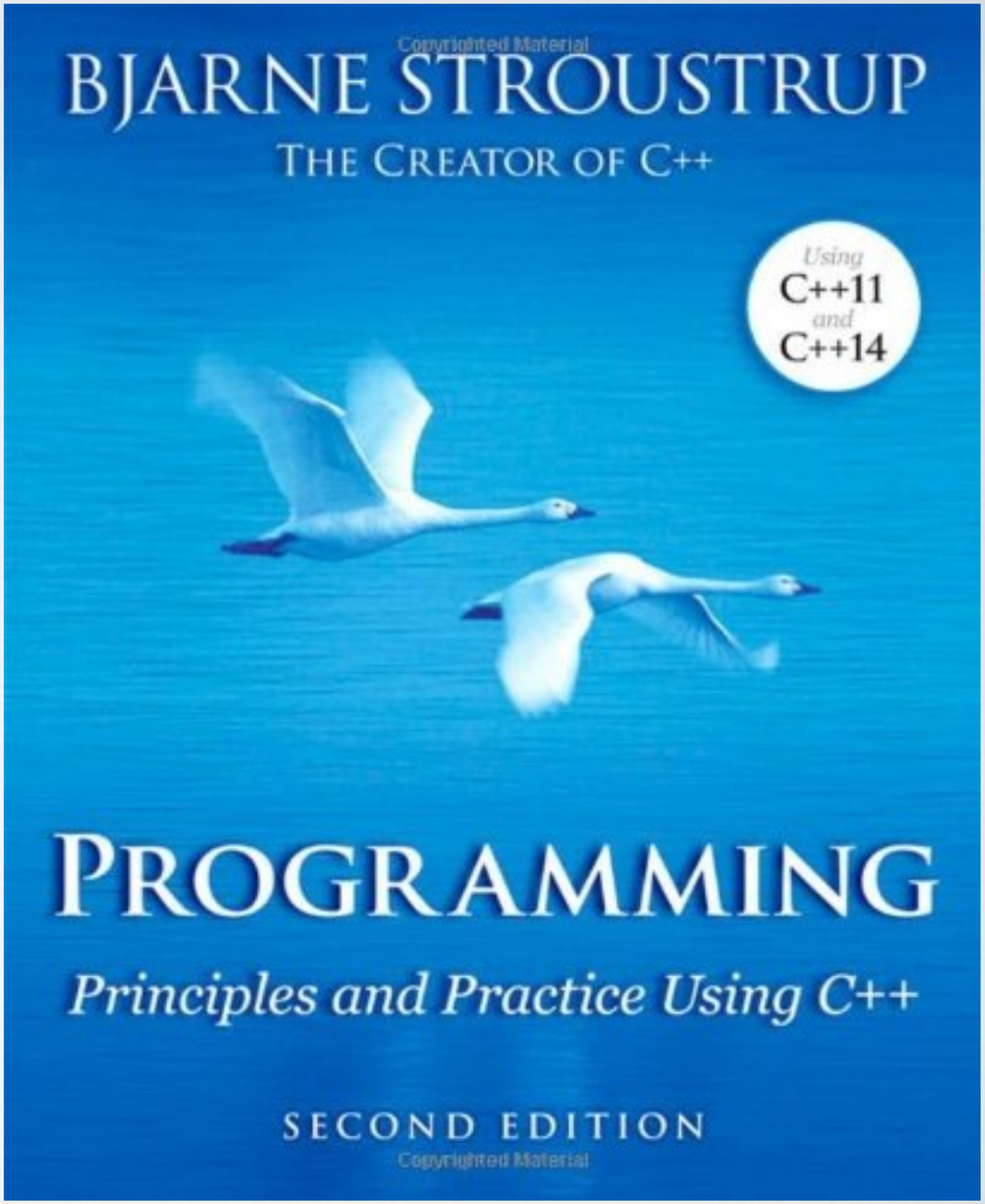
BIBLIOGRAPHIE

- Bjarne Stroutrup

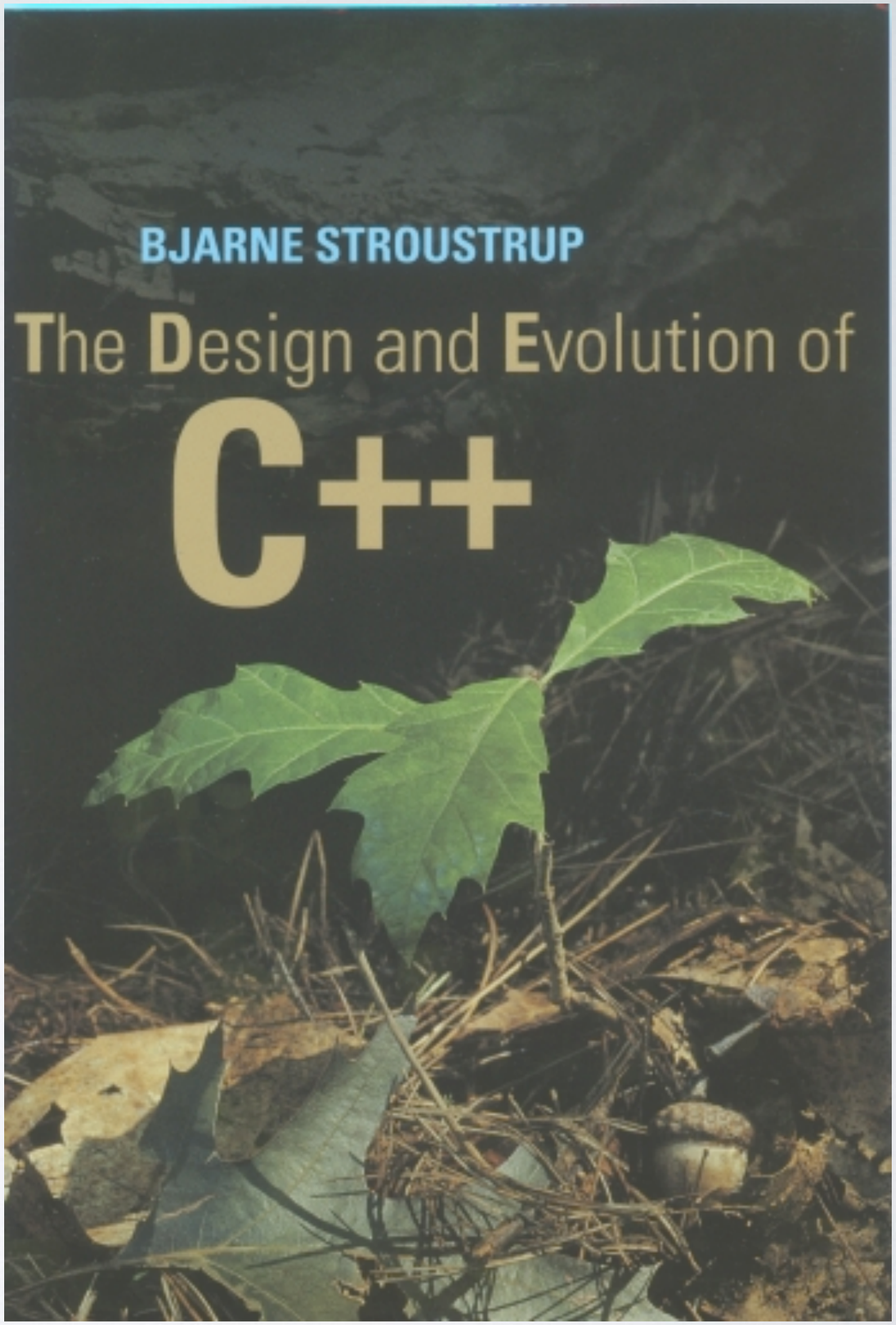
- La référence



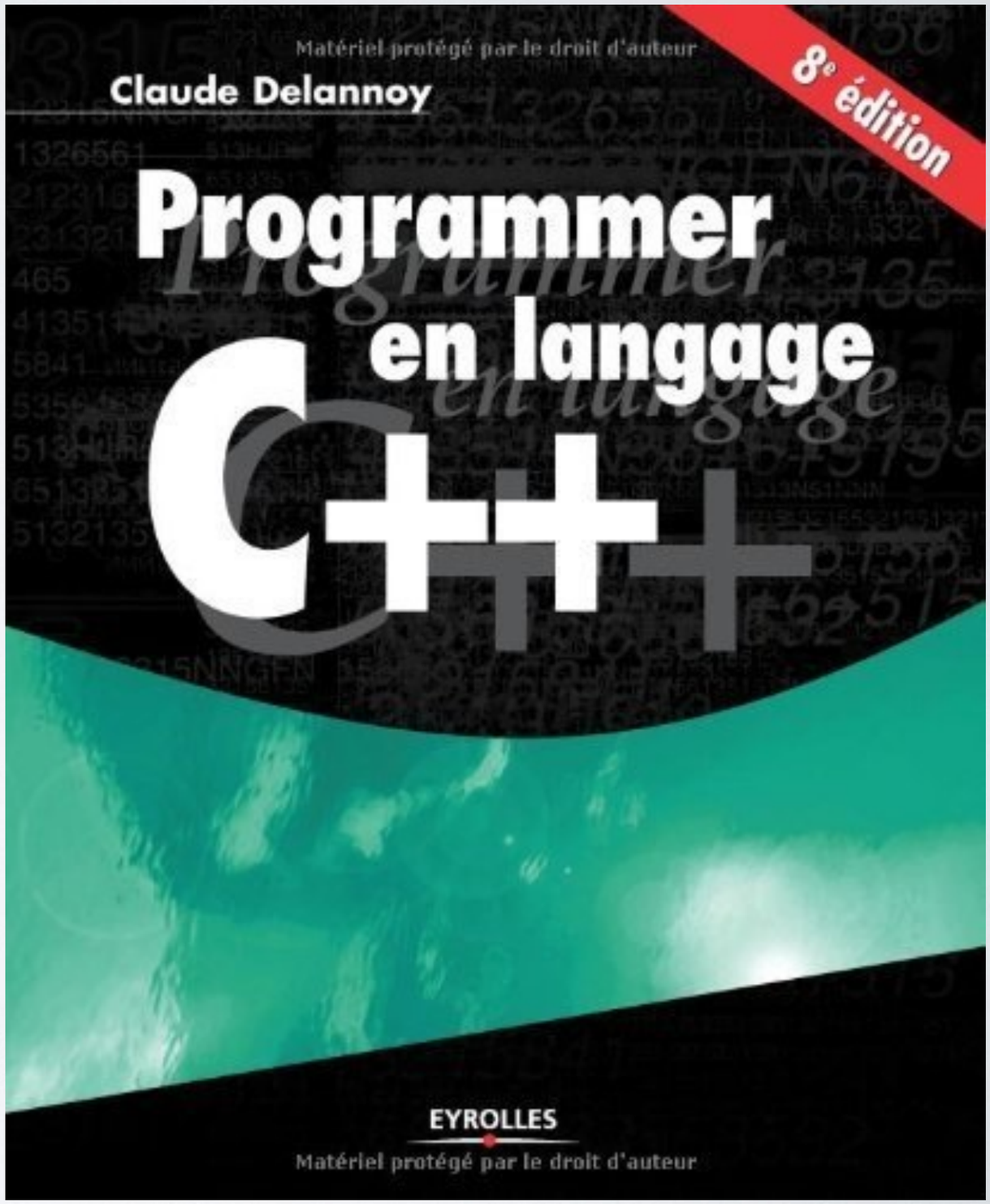
- Bjarne Stroustrup
- La pratique



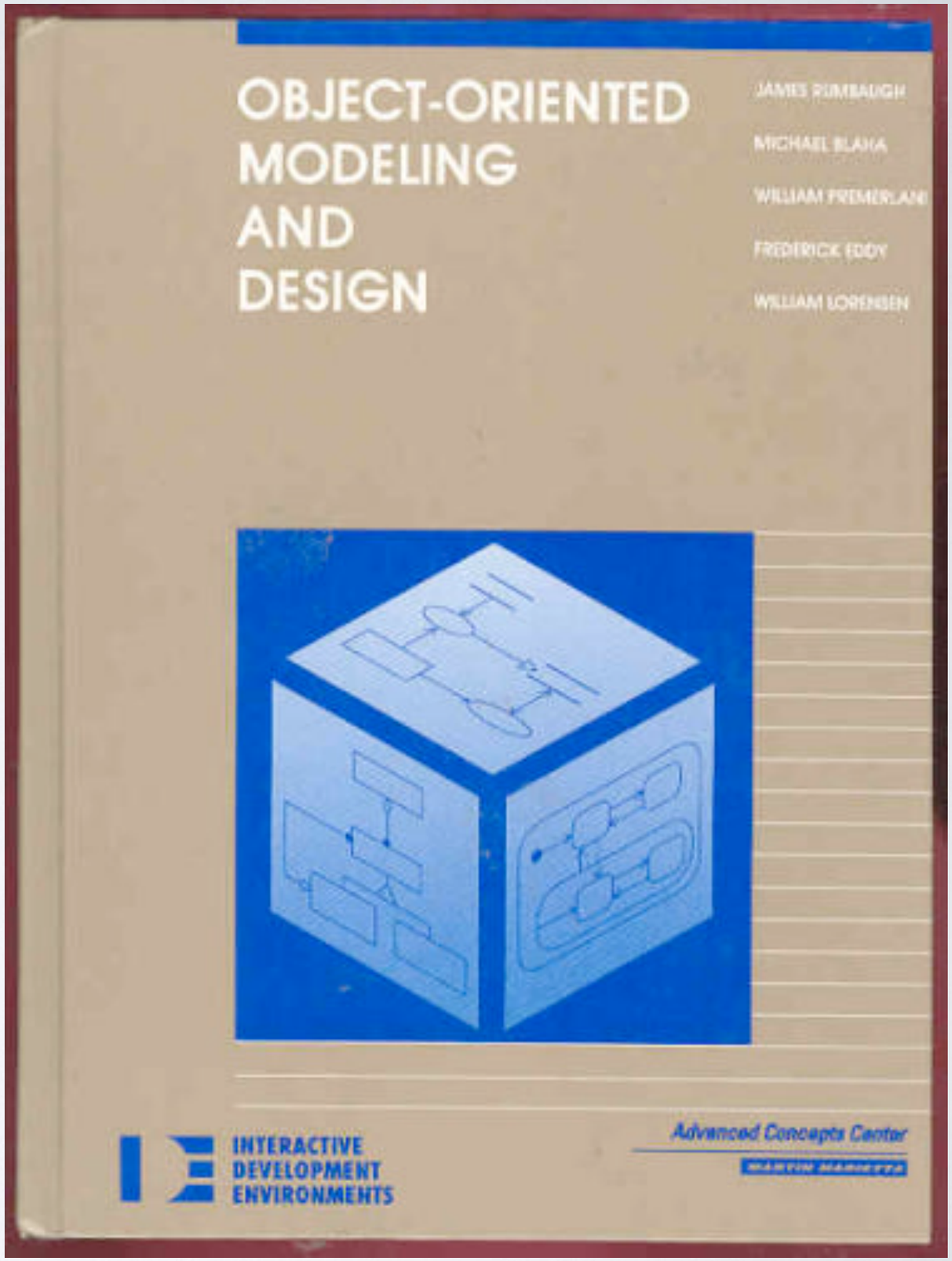
- Stroustrup
- L'histoire
- Ma vie, mon œuvre



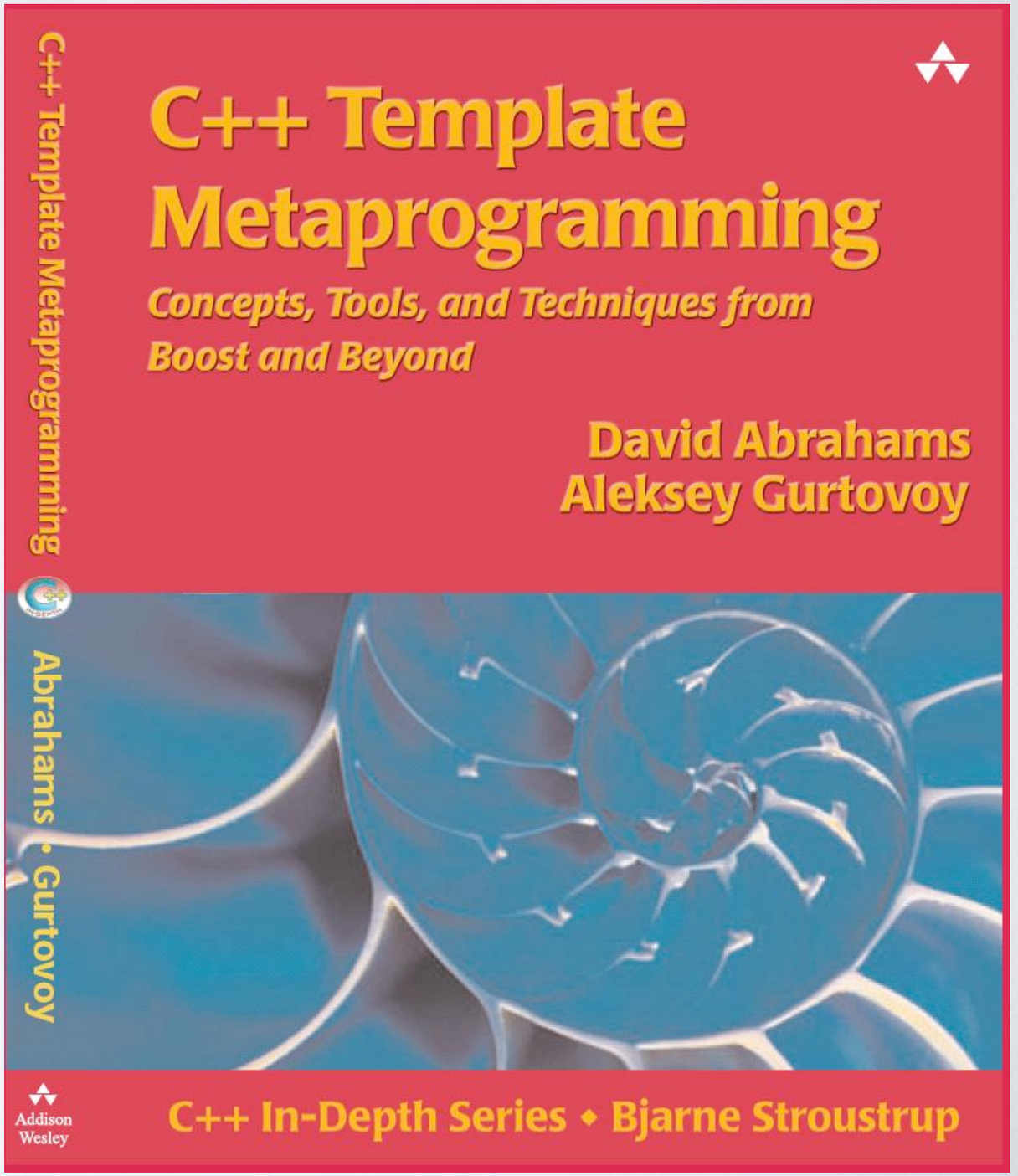
- Le livre en français
- La pratique



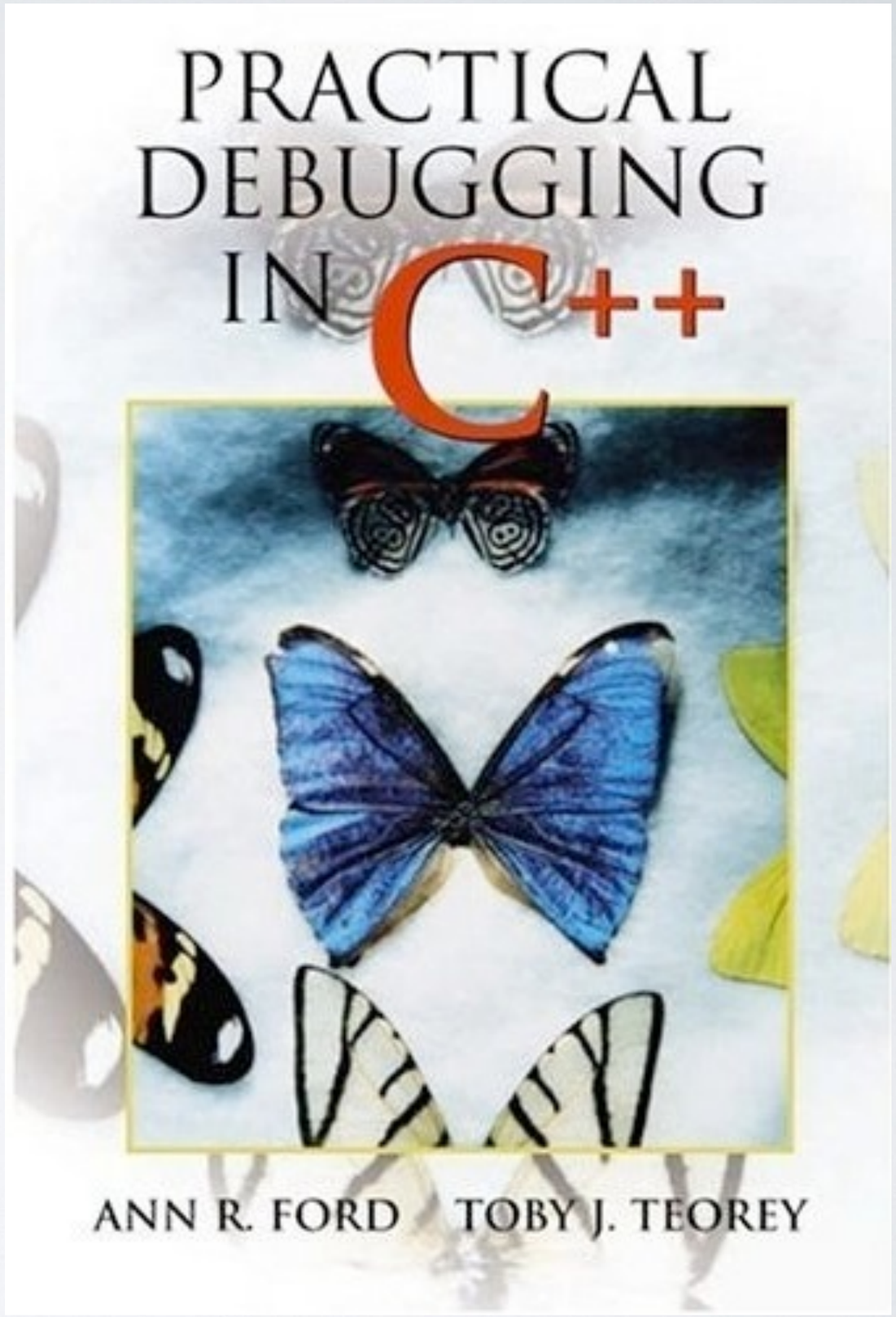
- La Pratique
- Modélisation
- Conception



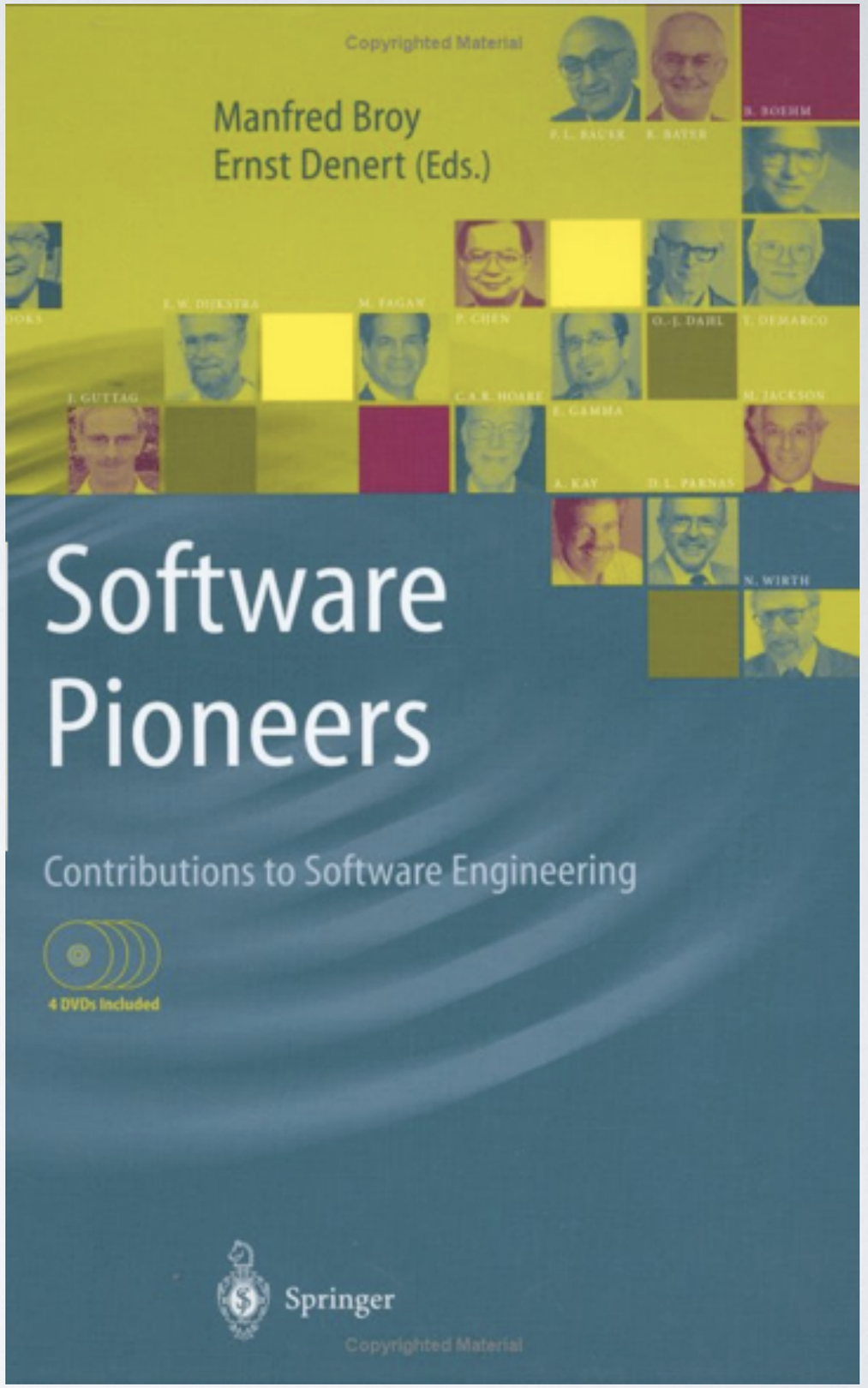
- C++ avancé
- L'expertise



- L’expertise



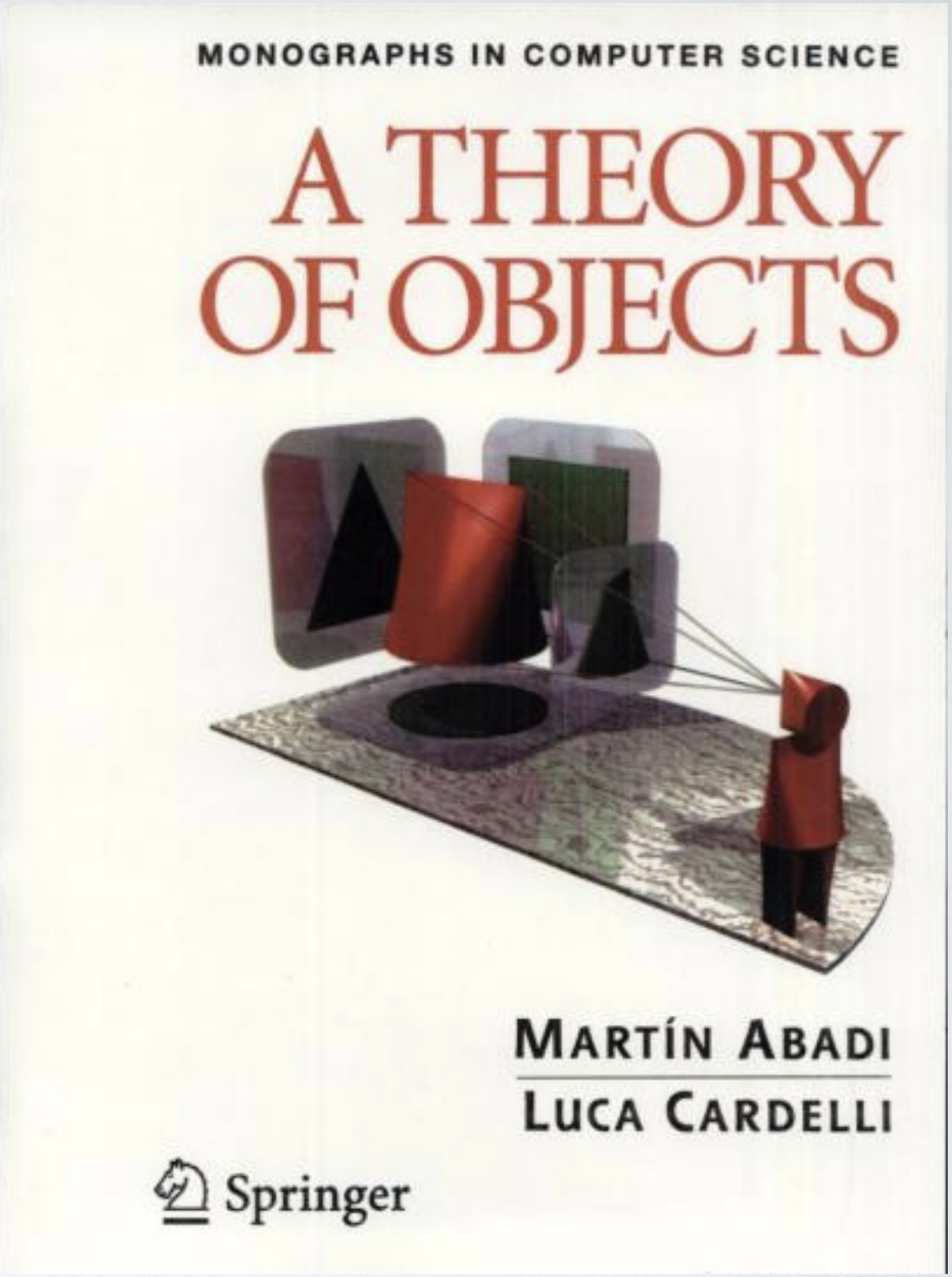
- De l'histoire



- UML


Deuxième édition
Pierre-Alain Muller
Nathalie Gaertner
Modélisation objet avec UML
Sur le CD offert avec ce livre :
Rational Rose 2000 pour Windows 95/98/NT4*
Outils de validation et de prototypage UML d'ObjeXion Software*
Modèles UML et prototypes des deux études de cas du livre
Spécifications UML 1.3**
* En version d'évaluation.
** En anglais au format pdf
Sur le site Web d'accompagnement :
Dialoguez avec les auteurs
Consultez les mises à jour et compléments
www.editions-eyrolles.com
Eyrolles

- La théorie


MONOGRAPHS IN COMPUTER SCIENCE
A THEORY OF OBJECTS
MARTÍN ABADI
LUCA CARDELLI
Springer

INTRODUCTION

Deux visions (compatibles) de la question

- Une extension du langage C ?
- Un langage orienté objet à la syntaxe inspirée du C ?

Deux approches pédagogiques

- *bottom-up*
- *top-down*

Le C++ repose sur un modèle d'objet

Encapsulation

- droits d'accès

Instanciation de classe

- construction/destruction
- statique vs dynamique

Messages

- invocation de méthodes

Généralisation

- héritage, redéfinition, extension
- types abstraits, factorisation, méthodes virtuelles

On y distingue trois polymorphismes

- *ad-hoc (la surcharge)*
- *par sous-typage (l'héritage)*
- *généralisé ou universel (les modèles)*

La généricité :

- fonctions/méthodes génériques
- types génériques (aka templates)

Rappels historiques à propos du langage C++ ou une ode à la gloire de Bjarne Stroustrup (Bell) :

- 1979-1982 : C avec classes
- 1983-1985 : C++, cfront, manuel de référence
- 1986-1988 : diffusion importante du langage
- 1989-1998 : cfront 2.0, cfront 3.0, STL (Alexander Stepanov - HP), ISO 14882:1998 aka C++98
- 2003 : ISO 14882:2003
- 2011 : ISO 14882:2011 aka C++11
- 2014 : C++14
- 2017 : C++17

Bjarne Stroustrup au travail - source Wikipédia

- isocpp.org (/std)

- Attention le standard est

un document de

1400 pages!!!

```
struct {
  char a;
  int b:5,
  c:11,
  :0,
  d:8;
  struct {int ee:8;} e;
}
```

contains four separate memory locations: The field `a` and bit-fields `d` and `e.ee` are each separate memory locations, and can be modified concurrently without interfering with each other. The bit-fields `b` and `c` together constitute the fourth memory location. The bit-fields `b` and `c` cannot be concurrently modified, but `b` and `a`, for example, can be. — *end example*]

1.8 The C++ object model [intro.object]

1 The constructs in a C++ program create, destroy, refer to, access, and manipulate objects. An *object* is a region of storage. [*Note:* A function is not an object, regardless of whether or not it occupies storage in the way that objects do. — *end note*] An object is created by a *definition* (3.1), by a *new-expression* (5.3.4) or by the implementation (12.2) when needed. The properties of an object are determined when the object is created. An object can have a *name* (Clause 3). An object has a *storage duration* (3.7) which influences its *lifetime* (3.8). An object has a *type* (3.9). The term *object type* refers to the type with which the object is created. Some objects are *polymorphic* (10.3); the implementation generates information associated with each such object that makes it possible to determine that object's type during program execution. For other objects, the interpretation of the values found therein is determined by the type of the *expressions* (Clause 5) used to access them.

2 Objects can contain other objects, called *subobjects*. A subobject can be a *member subobject* (9.2), a *base class subobject* (Clause 10), or an array element. An object that is not a subobject of any other object is called a *complete object*.

3 For every object `x`, there is some object called the *complete object of* `x`, determined as follows:

— If `x` is a complete object, then `x` is the complete object of `x`.

— Otherwise, the complete object of `x` is the complete object of the (unique) object that contains `x`.

4 If a complete object, a data member (9.2), or an array element is of class type, its type is considered the *most derived class*, to distinguish it from the class type of any base class subobject; an object of a most derived class type or of a non-class type is called a *most derived object*.

5 Unless it is a bit-field (9.6), a most derived object shall have a non-zero size and shall occupy one or more bytes of storage. Base class subobjects may have zero size. An object of trivially copyable or standard-layout type (3.9) shall occupy contiguous bytes of storage.

6 Unless an object is a bit-field or a base class subobject of zero size, the address of that object is the address of the first byte it occupies. Two objects that are not bit-fields may have the same address if one is a subobject of the other, or if at least one is a base class subobject of zero size and they are of different types; otherwise, they shall have distinct addresses.[4]

[*Example:*

```
static const char test1 = 'x';
static const char test2 = 'x';
const bool b = &test1 != &test2;        // always true
```

— *end example*]

7 [*Note:* C++ provides a variety of fundamental types and several ways of composing new types from existing types (3.9). — *end note*]

4) Under the "as-if" rule an implementation is allowed to store two objects at the same machine address or not store an object at all if the program cannot observe the difference (1.9).

ISO/IEC JTC1 SC22 WG21 N 3690
Date: 2013-05-15
ISO/IEC CD 14882
ISO/IEC JTC1 SC22
Secretariat: ANSI

predefine the `main` function. This function shall not be overloaded. It shall
, but otherwise its type is implementation-defined. All implementations shall

g `int` and

ter to pointer to `char`) returning `int`

. In the latter form, for purposes of exposition, the first function parameter is
nction parameter is called `argv`, where `argc` shall be the number of arguments
e environment in which the program is run. If `argc` is nonzero these arguments
hrough `argv[argc-1]` as pointers to the initial characters of null-terminated
17.5.2.1.4.2) and `argv[0]` shall be the pointer to the initial character of a
ne used to invoke the program or `""`. The value of `argc` shall be non-negative.
l be 0. [*Note:* It is recommended that any further (optional) parameters be
]
used within a program. The linkage (3.5) of `main` is implementation-defined.
as deleted or that declares `main` to be `inline`, `static`, or `constexpr` is ill-
otherwise reserved. [*Example:* member functions, classes, and enumerations
ities in other namespaces. — *end example*]
out leaving the current block (e.g., by calling the function `std::exit(int)`
objects with automatic storage duration (12.4). If `std::exit` is called to
ruction of an object with static or thread storage duration, the program has

the effect of leaving the main function (destroying any objects with automatic
`std::exit` with the return value as the argument. If control reaches the end
a `return` statement, the effect is that of executing

on-local variables [basic.start.init]

f named non-local variables: those with static storage duration (3.7.1) and
ation (3.7.2). Non-local variables with static storage duration are initialized
nitiation. Non-local variables with thread storage duration are initialized as a
n. Within each of these phases of initiation, initialization occurs as follows.
uration (3.7.1) or thread storage duration (3.7.2) shall be zero-initialized (8.5)
takes place. A *constant initializer* for an object `o` is an expression that is a
at it may also invoke `constexpr` constructors for `o` and its subobjects even
ral class types [*Note:* such a class may have a non-trivial destructor — *end*
is performed:
cluding implicit conversions) that appears in the initializer of a reference with
uration is a constant expression (5.19) and the reference is bound to an lvalue
th static storage duration or to a temporary (see 12.2);

or thread storage duration is initialized by a constructor call, and if the
ion is a constant initializer for the object;

thread storage duration is not initialized by a constructor call and if either the
or every full-expression that appears in its initializer is a constant expression.

Together, zero-initialization and constant initialization are called *static initialization*; all other initialization is *dynamic initialization*. Static initialization shall be performed before any dynamic initialization takes place. Dynamic initialization of a non-local variable with static storage duration is either ordered or

Langages ayant influencé la conception du C++

C (1972, Kernighan & Ritchie - Bell)

```
#include < stdio.h>

void main()
{
  int c, nc = 0, nl = 0;

  while ( (c = getchar()) != EOF )
  {
      nc++;
      if (c == '\n') nl++;
  }

  printf("Number of characters = %d, number of lines = %d\n",
       nc, nl);
}
```

Bulletin n°66 de SIF Société Informatique de France - In memoriam Dennis Ritchie

Langages ayant influencé la conception du C++

Simula (1960, Dahl & Nygaard - Oslo)

```
Class Rectangle (Width, Height); Real Width, Height;
                                 ! Class with two parameters;
Begin
   Real Area, Perimeter;  ! Attributes;
   Procedure Update;      ! Methods (Can be Virtual);
   Begin
     Area := Width * Height;
     Perimeter := 2*(Width + Height)
   End of Update;

   Boolean Procedure IsSquare;
     IsSquare := Width=Height;
   Update;                  ! Life of rectangle started at creation
   OutText("Rectangle created: "); OutFix(Width,2,6);
   OutFix(Height,2,6); OutImage
End of Rectangle;
```

Langages ayant influencé la conception du C++

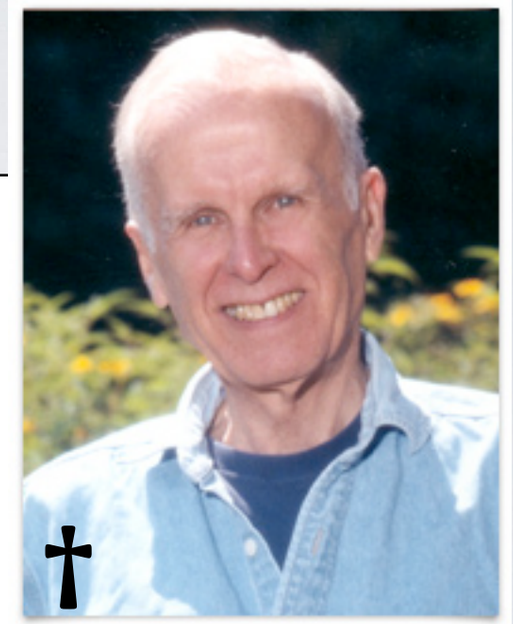
Ada 83 (1980, Ichbiah - CII-Honeywell Bull)

```
with Text_IO, Ada.Strings.Unbounded, Ustrings, World;
use  Text_IO, Ada.Strings.Unbounded, Ustrings;

with Parser;

procedure Small is
  Command : Unbounded_String; -- Contains user's current command.
  Quit    : Boolean := False;
begin
 Put_Line("Welcome to a Small World!");

 World.Setup;

 while not Quit loop
  New_Line;
  Put_Line("Your Command?");
  Get_Line(Command);
  Parser.Execute(Command, Quit);
 end loop;

 Put_Line("Bye!");
end Small;
```

Langages ayant influencé la conception du C++

Algol 68 (inspiré d'Algol 60, Backus & Naur)

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment Le plus grand élément en valeur absolue de la matrice
        a de taille n par m est transféré à y et les indices
        de cet élément à i et k ;
begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
        for q:=1 step 1 until m do
            if abs(a[p, q]) > y then
                begin
                    y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```



Langages ayant influencé la conception du C++

CLU (1975, Liskov - MIT)


```
sum_stream = proc (s: stream) returns (int)
signals (overflow, unrepresentable_integer(string),
bad_format(string)
) sum: int := 0;
while true do sum := sum + get_number(s)
resignal unrepresentable_integer, bad_format,
overflow;
end except when end_of_file: return (sum) end
end sum_stream;
```

Langages ayant influencé la conception du C++

ML (1980, Milner - Edimbourg)

```
fun factorial n = let
  fun fac (0, acc) = acc
    | fac (n, acc) = fac (n-1, n*acc)
  in
    if (n < 0) then raise Fail "negative argument"
    else fac (n, 1)
  end
```

Quelques caractéristiques :

- compatibilité presque totale avec le C (très rares exceptions bien connues, faciles à identifier et à réparer)
- performances comparables à celles du C (une obsession de conception) : on « colle » à la machine

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ sucks » :

> At a recent conference, a speaker asked for a show of hands and found that twice as many people claimed to hate C++ as had ever written even a single small C++ program. The only word for such behavior is bigotry. In dealing with the current wave of C++ bashing, we should remember that bigotry is bred by ignorance and fear.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ sucks » :

Lors d'une conférence, un orateur interrogea l'auditoire et observa qu'il y avait deux fois plus de gens qui disaient haïr le C++ que de gens qui avaient écrit ne serait-ce qu'une seule ligne de C+ +. La seule chose à dire à propose d'un tel comportement est qu'il s'agit de bigoterie. Ce qu'il faut se rappeler lorsqu'on est face au dénigrement du C++ est que la bigoterie est l'enfant de l'ignorance et de la peur.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is too complex » :

Experience shows that trying to learn OOP or C++ from The Annotated C++ Reference Manual is usually a BIG mistake. It is a nice book, I can recommend it (:-), but not as a tutorial. After all, most people wouldn't try learning a natural language exclusively from a grammar plus a dictionary. Few would succeed.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is too complex » :

L'expérience montre qu'essayer d'apprendre la POO ou le c++ à partir du manuel de référence annoté du C++ est une grossière erreur. C'est un livre merveilleux, je le recommande, mais ce n'est pas un manuel d'apprentissage. Après tout, personne n'essaierait d'apprendre une langue en utilisant uniquement un dictionnaire et une grammaire. Très peu y arriveraient.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ programmers are idiots »

Many of the most experienced individuals and organizations in the industry (and academia) use C++. Ascribing their choice of C++ over alternatives to stupidity, ignorance, or inexperience is arrogant and insulting - and in many cases simply wishful thinking based on a limited field of knowledge. I still think the average C++ programmer is pretty smart, but rarely a fanatic, and usually too busy getting the work done to bother with language laws or the latest academic advances in OO type theory. Fortunately, one doesn't need to be a genius to write good C++.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ programmers are idiots »

Beaucoup d'experts, d'industriels (même le monde académique) utilisent le C++. Attribuer leur choix d'utiliser C++ plutôt qu'une autre langage à la stupidité, l'ignorance ou l'inexpérience est arrogant et insultant; il ne s'agit la plupart du temps que d'une réflexion assez pauvre basée sur un champ de savoir limité. Je continue à penser que le programmeur C++ moyen est plutôt assez malin, rarement fanatique, et habituellement trop occupé à faire son travail que de se soucier des dernières avancées académiques en matière de théorie des types objets et de langages. Heureusement, il n'est pas nécessaire d'être un génie pour écrire du bon code C++.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is useless/unreliable/ dangerous because it lacks feature/ property X » :

My firm conclusion is that no single feature is truly necessary. Much more successful software has been written in languages proclaimed BAD, than has been written in languages acclaimed as saviors of suffering programmers; much more.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is useless/unreliable/ dangerous because it lacks feature/ property X » :

Ma conclusion définitive à ce sujet est qu'aucune fonctionnalité à elle seule est véritablement nécessaire. Il y a bien plus de bons logiciels qui ont été écrits dans des langages considérés comme **mauvais** que de logiciels écrits dans des langages déclarés comme pouvant affranchir les pauvres programmeurs de leurs souffrances; bien plus.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is not Object-Oriented »

C++ supports object-oriented programming (as defined by most people). C++ supports it pretty well for real-world applications. Its type system and general model of the world are coherent and relevant to real applications.

The proof of the pudding is in the eating, so I consider the fact that many large projects have been completed using C++ much more significant than the fact that C++ doesn't conform to the latest fashion of OO theology.

Have a look at the "Design Patterns" book by Gamma et. al. for examples of elegant designs clearly expressed in relatively straightforward C++. I observe that there are a lot of myths about what features C++ has and doesn't have.

http://www.research.att.com/~bs/blast.html

Des critiques. « C++ is not Object-Oriented »

Le C++ permet de programmer dans le style de la programmation orientée objet (telle que définie par la plupart des gens). Le C++ le permet plutôt efficacement pour des applications du monde réel. Son système de type et son modèle d'objets sont à la fois cohérents et utiles à de telles applications.

C'est au fruit qu'on juge l'arbre, alors je considère que le fait que de nombreux grands projets ont été menés à bien en employant le C++ est bien plus significatif que le fait que le C++ ne soit pas à la dernière mode de la théologie orientée objet.

Jetez un œil au « Design Patterns » de Gamma et. al., vous y trouverez des exemples de conceptions élégantes, clairement exprimées et de façon assez directe en C++. Je constate qu'il y a de nombreux mythes à propos de ce que le C++ permet ou non.

Un exemple original :

```
#include <iostream>
using namespace std;

int main() {
  cout << "Bonjour tout le monde" << endl;
  return 0;
}
```

Un exemple :

surcharge

```
#include <iostream>
using namespace std;

void afficher(int n) {
  cout << "int:" << n << endl;
}
void afficher(float f) {
  cout << "float:" << f << endl;
}

int main() {
  int i=13; float f=2.14159;
  afficher(i);
  afficher(f);
  return 0;
}
```

Un exemple :

classes

```
#include <iostream>
using namespace std;

class Point {
 private:
  int abs, ord;
 public:
  Point(int x,int y) { abs = x; ord = y; }
  int getAbscisse() const { return abs; }
  int getOrdonnee() const { return ord; }
  void deplacer(int dx,int dy) { abs+=dx; ord+=dy; }
};

int main() {
 const Point origine(0,0);
 Point p(5,10);
 p.deplacer(10,20);
 return 0;
}
```

Un exemple :

surcharge d'opérateur

```
#include <iostream>
using namespace std;

class Compteur {
 private:
  int v;
 public:
  Compteur() { v = 0; }
  int getCompteur() const { return v; }
  Compteur &operator+=(int i) { v+=i; return *this; }
};

int main() {
 Compteur c;
 c += 15;
 return 0;
}
```

Un exemple :

généricité

```
#include <iostream>
using namespace std;
template <class T> class Wrap {
 private:
  T valeur;
 public:
  Wrap(T v) { valeur = v; }
  T get() { return valeur; }
};
int main() {
 Wrap<int> wi = 13;
 cout << wi.get() << endl;
 Wrap<float> wf = 9.9;
 cout << wf.get() << endl;
}
```

Un exemple :

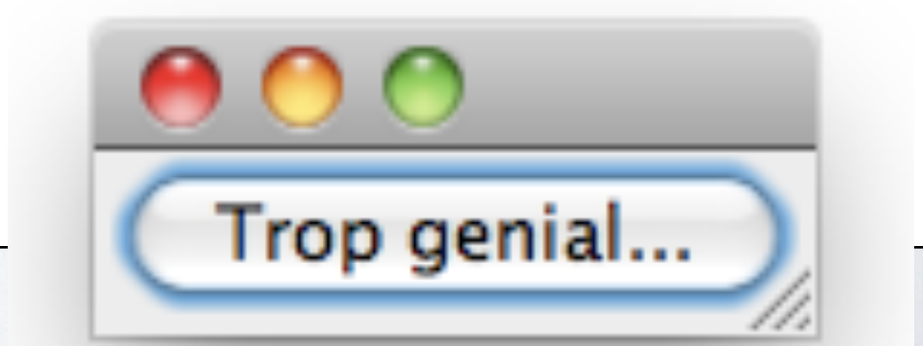
bibliothèque

```
#include <iosteam>
#include <vector>
using namespace std;
#include “Point.hpp“

int main() {
 vector<Point> tableau(50);
 tableau[2] = Point(10,20);
 return 0;
}
```

Un exemple :

bibliothèques graphiques (non normalisées)

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[]) {
     QApplication app(argc, argv);

     QPushButton bouton("Trop genial...");

     bouton.show();
     return app.exec();
}
```



- Pourquoi C++ ?
  - Toujours utilisé (top ten / top five)
  - Connaître plusieurs langages à objets est une bonne chose pour n'en maîtriser vraiment même qu'un seul…