C++ język nie dla ludzi o słabych nerwach

Małgorzata Bieńkowska
malgorzata.bienkowska@gmail.com

9LivesData

9LivesData

HYDRAStor
Dla NEC Japan od ponad 10 lat
1,5 miliona linii kodu
większość rozwijana w Warszawie
www.9livesdata.com

```
typedef int MyIntType;
```

```
//typedef int MyIntType;

class MyIntType
{
public:
  MyIntType();
  MyIntType(int);
};
```

```
//typedef int MyIntType;

class MyIntType
{
public:
  MyIntType();
  MyIntType(int);
};

void foo()
{
  int max = numeric_limits<MyIntType>::max();
  ...
}
```

<limits>

```
template<class T> class numeric_limits {
public:

    static constexpr T max() noexcept { return T(); }
 ...
}
template<>
struct numeric_limits<int>
{
  static constexpr int max() noexcept { return __INT_MAX__; }
}
```

```
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;

private:
    int * ptr;
};
```

```
MyPointer ptr(NULL);
if (ptr) {
    cout << "MyPointer is true" << endl;
} else {
    cout << "MyPointer is false" << endl;
}
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11 -Weverything
sample2.cpp:40:9: error: value of type 'MyPointer' is not contextually convertible to 'bool'
        if (ptr) {
        ^~~
1 error generated.
```

```
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;
    operator bool() const {
        return ptr != NULL;
    }
private:
    int * ptr;
};
```

```
MyPointer ptr(NULL);
if (ptr) {
    cout << "MyPointer is true" << endl;
} else {
    cout << "MyPointer is false" << endl;
}
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11 -Weverything
$ ./sample-clang.out
MyPointer is false
```

```
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;
    operator bool() const {
        return ptr != NULL;
    }
private:
    int * ptr;
};
```

```
struct AnotherClass
{
    operator bool() const {
        return false;
    }
};
MyPointer ptr(NULL);
AnotherClass a;
if (a==ptr) {
    cout << "Uncomparable classes are
compared" << endl;
}
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11 -Weverything
$ ./sample-clang.out
Uncomparable classes are compared
```

```
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;

    typedef int*
(MyPointer::*UnspecifiedBoolType)()
const;
    operator UnspecifiedBoolType()
const
    {
        return NULL == this->get() ?
NULL : &MyPointer::get;
    }

private:
    int * ptr;
};
```

```
struct AnotherClass
{
    operator bool() const {
        return false;
    }
};
MyPointer ptr(NULL);
AnotherClass a;
if (ptr){
    cout << "MyPointer is true" << endl;
}
if (a==ptr) {
    cout << "Uncomparable" << endl;
}
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11 -Weverything
sample2.cpp:52:6: error: invalid operands to binary expression ('AnotherClass' and 'MyPointer')
if (a==ptr) {
      ~^ ~~~
1 error generated.
```

```cpp
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;

    typedef int*
(MyPointer::*UnspecifiedBoolType)()
const;
    operator UnspecifiedBoolType()
const
    {
        return NULL == this->get() ?
NULL : &MyPointer::get;
    }

private:
    int * ptr;
};
```

```cpp
MyPointer ptrNull(NULL);
MyPointer ptr1(new int(1));
MyPointer ptr2(new int(2));

if (ptrNull==ptr1) {
    cout << "ptrNull==ptr1" << endl;
}
if (ptr1==ptr2) {
    cout << "ptr1==ptr2" << endl;
}
```

```
domek@domek ~/Pulpit/cpp-meetup $ clang++ sample2.cpp -o sample-clang.out -std=c++11
-Weverything
domek@domek ~/Pulpit/cpp-meetup $ ./sample-clang.out
ptr1==ptr2
```

Safe bool solution in C++11

```
explicit operator bool();
```

```
class MyPointer
{
public:
    MyPointer(int *);
    int * get() const;

   explicit operator bool() const {
        return ptr != NULL;
    }

private:
    int * ptr;
};
```

```
MyPointer ptrNull(NULL);
MyPointer ptr1(new int(1));
MyPointer ptr2(new int(2));

if (ptrNull==ptr1) {
    cout << "ptrNull==ptr1" << endl;
}
if (ptr1==ptr2) {
    cout << "ptr1==ptr2" << endl;
}
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11
sample2.cpp:49:12: error: invalid operands to binary expression ('MyPointer' and 'MyPointer')
if (ptrNull==ptr1) {
        ~~~~~~~^ ~~~~
sample2.cpp:52:9: error: invalid operands to binary expression ('MyPointer' and 'MyPointer')
if (ptr1==ptr2) {
        ~~~~^ ~~~~
1 warning and 2 errors generated.
```

Wirtualne destruktory

```
class A
{
  virtual void foo() = 0;
private:
   Foo a;
};
class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};

A* a = new AA();
delete a;
```

Wirtualne destruktory

```
class A
{
  virtual void foo() = 0;
private:
   Foo a;
};
class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};

A* a = new AA();
delete a;
```

```
$ clang++ sample.cpp -o sample-clang.out
sample.cpp:41:1: warning: delete called on 'A' that is abstract but has
non-virtual destructor
        [-Wdelete-non-virtual-dtor]
delete a;
^
1 warning generated.

$ g++ sample.cpp -o sample-gcc.out
$
```

Wirtualne destruktory

```
class A
{
  virtual void foo() = 0;
private:
   Foo a;
};
class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};

A* a = new AA();
delete a;
```

```
$ clang++ sample.cpp -o sample-clang.out
sample.cpp:41:1: warning: delete called on 'A' that is abstract but has
non-virtual destructor
        [-Wdelete-non-virtual-dtor]
delete a;
^
1 warning generated.

$ g++ sample.cpp -o sample-gcc.out
$ g++ sample.cpp -o sample-gcc.out -Wall
sample.cpp: In function ‘int main()’:
sample.cpp:106:8: warning: deleting object of abstract class type
‘main()::A’ which has non-virtual destructor will cause undefined
behaviour [-Wdelete-non-virtual-dtor]
 delete a;
```

Wirtualne destruktory

```
class A
{
  virtual void foo(){};
private:
   Foo a;
};
class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};

A* a = new AA();
delete a;
```

```
$ clang++ sample.cpp -o sample-clang.out
$ g++ sample.cpp -o sample-gcc.out
$
```

Wirtualne destruktory

```
class A
{
//  virtual void foo() = 0;
private:
   Foo a;
};

class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};
A* a = new AA();
delete a;
```

```
$ clang++ sample.cpp -o sample-clang.out -Wall
$ g++ sample.cpp -o sample-gcc.out -Wall -Wextra
$ ./sample-gcc.out
*** Error in `./sample-gcc.out': free(): invalid pointer:
0x000000000068fc28 ***
```

Wirtualne destruktory

```
class A
{
//  virtual void foo() = 0;
private:
   Foo a;
};

class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};
A* a = new AA();
delete a;
```

```
$ clang++ sample2.cpp -o sample-clang.out -std=c++11 -Weverything
sample2.cpp:23:7: warning: 'AA' has virtual functions but non-virtual
destructor
        [-Wnon-virtual-dtor]
class AA : public A
```

Wirtualne destruktory

```
class A
{
 // virtual void foo() = 0;
private:
   Foo a;
};

class AA : public A
{
public:
    virtual void foo(){}
    Foo aa;
};
A* a = new AA();
delete a;
```

```
$ clang++ sample.cpp -o sample-clang.out -Wall
$ g++ sample.cpp -o sample-gcc.out -Wall -Wextra
$ ./sample-gcc.out
*** Error in `./sample-gcc.out': free(): invalid pointer:
0x000000000068fc28 ***
```

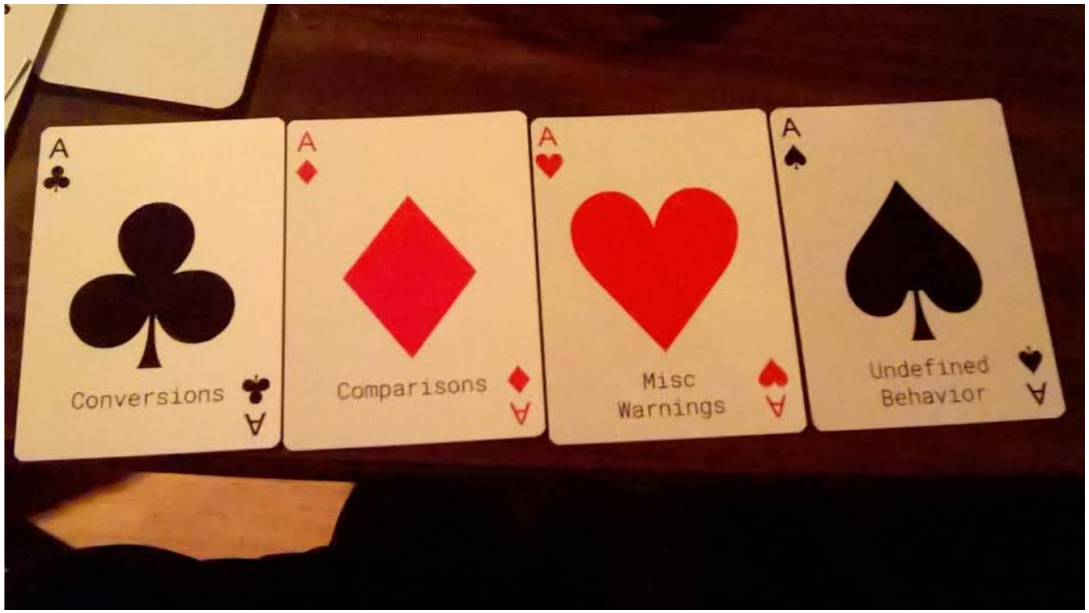
unique_ptr<A>?
shared_ptr<A>?

Clang and gcc warnings

```
domek@domek ~/Pulpit/cpp-meetup $ g++ --help=warnings | grep "-" | wc -l
218

domek@domek ~/Pulpit/cpp-meetup $ cat clang-warnings | wc -l
672
```

Clang obroną nam



```
T const & t = getA().getB();
```

T const & t = getA().getB();

Prawo Demeter

metoda danego obiektu może odwoływać się jedynie do metod należących do:

- tego samego obiektu,
- dowolnego parametru przekazanego do niej,
- dowolnego obiektu przez nią stworzonego,
- dowolnego składnika klasy, do której należy dana metoda.

https://pl.wikipedia.org/wiki/Prawo_Demeter

T const & t = getA().getB();

```
typedef Foo T;
struct A {
    T const & getB() const {
        return this->member;
    }
    T member;
};
A getA() {
    return A();
}
cout << "===== T const & t = getA().getB() =====" << endl;
T const & t = getA().getB();
cout << t << endl;
```

```
===== T const & t = getA().getB() =====
Foo
~Foo
<operator<< on Foo>
```

T const & t = getA().getB();

```
typedef Foo T;
struct A {
    T const & getB() const {
        return this->member;
    }
    T member;
};
A getA() {
    return A();
}
cout << "===== T const & t = getA().getB() =====" << endl;
T const & t = getA().getB();
cout << t << endl;
cout << "===== getA().getB() in function =====" << endl;
cout << getA().getB() << endl;
```

```
===== T const & t = getA().getB() =====
Foo
~Foo
<operator<< on Foo>
===== getA().getB() in function =====
Foo
<operator<< on Foo>
~Foo
```

zupa(f1(), f2(), unique_ptr<...>(new ...));

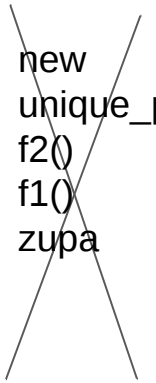
Kolejność ewaluacji wyrażeń

zupa(f1(), f2(), unique_ptr<...>(new ...));

new
unique_ptr<>
f2()
f1()
zupa

Kolejność ewaluacji wyrażeń

zupa(f1(), f2(), unique_ptr<...>(new ...));



new
f1()
f2()
unique_ptr<>
zupa

Np. MS VC++ 2005 wyliczał wyrażenia w innej kolejności

Pozostałości z C

```
int tab[5] = {1};
```

Pozostałości z C

```
int tab[5] = {1};          ->          int tab[5] = {1, 0, 0, 0, 0};
```

Pozostałości z C

```
char a = 65;
unsigned char b = 66;
signed char c = 67;

cout << a << " " << b << " " << c << endl;
cout << +a << " " << +b << " " << +c << endl;
```

Pozostałości z C

```
char a = 65;
unsigned char b = 66;
signed char c = 67;

cout << a << " " << b << " " << c << endl;
cout << +a << " " << +b << " " << +c << endl;
```

Output:
A B C
65 66 67

Pozostałości z C

```
char a = 65;
unsigned char b = 66;
signed char c = 67;

cout << a << " " << b << " " << c << endl;
cout << +a << " " << +b << " " << +c << endl;
```

Output:
A B C
65 66 67

Pozostałości z C

```
char a = 65;
unsigned char b = 66;
signed char c = 67;

cout << a << " " << b << " " << c << endl;
cout << +a << " " << +b << " " << +c << endl;
```

Zagadka:
c=a+++b;
++c=a+++b++;

Output:
A B C
65 66 67

Integer Promotion

Templates

```
void exit(int);

template <typename T>
class Base {
public:
        void exit();
};
template <typename T>
class Derived : Base<T> {
public:
        void foo() {
                exit();      // error: too few arguments to function ‘void exit(int)’
        }
};
```

Nicolai M. Josuttis, 2002, C++ Templates, The complete guide,

Templates - szukanie nazw

```
void exit(int);

template <typename T>
class Base {
public:
	void exit();
};
template <typename T>
class Derived : Base<T> {
public:
	void foo() {
		this->exit();
	}
};
```

Nicolai M. Josuttis, 2002, C++ Templates, The complete guide,

```
struct ClassToBind {
    void g(ClassToBind const &) {}
    ClassToBind() {}
    ClassToBind(ClassToBind const &) {
        cout << "copy of ClassToBind" << endl;
    }
};

typedef function<void (void) > Func;
void f(Func){}
```

```
struct ClassToBind {
    void g(ClassToBind const &) {}
    ClassToBind() {}
    ClassToBind(ClassToBind const &) {
        cout << "copy of ClassToBind" << endl;
    }
};

typedef function<void (void) > Func;
void f(Func){}
cout << "====boost::bind====" << endl;
ClassToBind toBind;
Func func1 = boost::bind(&ClassToBind::g, &toBind, toBind);
cout << "====func2====" << endl;
Func func2  = boost::bind(&f, func1);
cout << "====func3====" << endl;
Func func3  = boost::bind(&f, func2);
```

```
struct ClassToBind {
    void g(ClassToBind const &) {}
    ClassToBind() {}
    ClassToBind(ClassToBind const &) {
        cout << "copy of ClassToBind" << endl;
    }
};

typedef function<void (void) > Func;
void f(Func){}
cout << "====boost::bind====" << endl;
ClassToBind toBind;
Func func1 = boost::bind(&ClassToBind::g, &toBind, toBind);
cout << "====func2====" << endl;
Func func2  = boost::bind(&f, func1);
cout << "====func3====" << endl;
Func func3  = boost::bind(&f, func2);
```

```
====boost::bind====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
====func2====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
====func3====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
```

```
struct ClassToBind {
    void g(ClassToBind const &) {}
    ClassToBind() {}
    ClassToBind(ClassToBind const &) {
        cout << "copy of ClassToBind" << endl;
    }
};

typedef function<void (void) > Func;
void f(Func){}
cout << "====boost::bind====" << endl;
ClassToBind toBind;
Func func1 = boost::bind(&ClassToBind::g, &toBind, toBind);
cout << "====func2====" << endl;
Func func2  = boost::bind(&f, func1);
cout << "====func3====" << endl;
Func func3  = boost::bind(&f, func2);
```

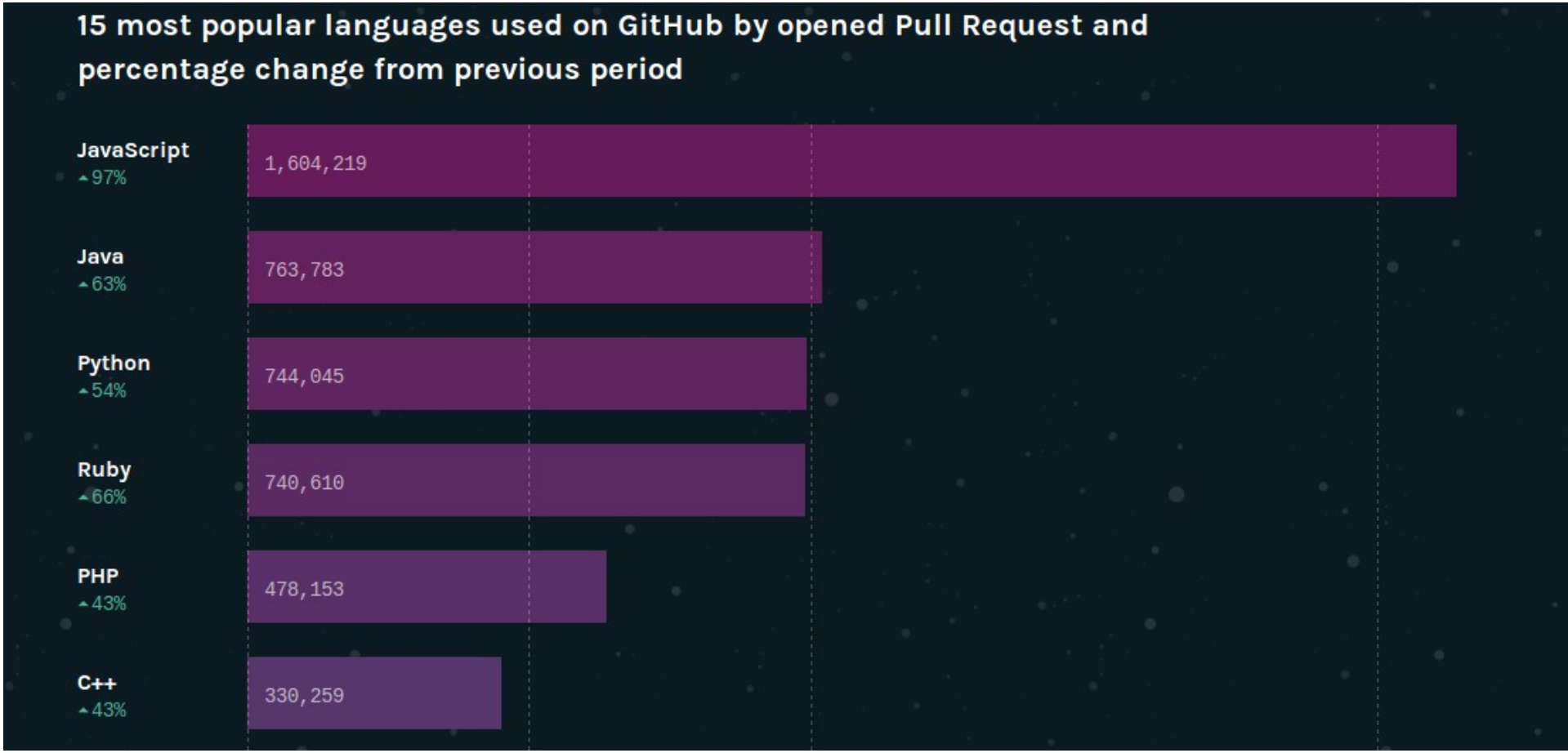
```
====boost::bind====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
====func2====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
====func3====
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
copy of ClassToBind
```

Każdy boost::bind woła 5 razy kontruktor kopiujący dla każdego parametru
Każda std::function to jedna alokacja

```
struct ClassToBind {
    void g(ClassToBind const &) {}
    ClassToBind() {}
    ClassToBind(ClassToBind const &) {
        cout << "copy of ClassToBind" << endl;
    }
};

typedef function<void (void) > Func;
void f(Func){}
cout << "====std::bind====" << endl;
ClassToBind toBind;
Func func1 = std::bind(&ClassToBind::g, &toBind, toBind);
cout << "====func2====" << endl;
Func func2  = std::bind(&f, func1);
cout << "====func3====" << endl;
Func func3  = std::bind(&f, func2);
```

```
====std::bind====
copy of ClassToBind
copy of ClassToBind
====func2====
copy of ClassToBind
====func3====
copy of ClassToBind
```



https://octoverse.github.com/

C++ zarządzanie pamięcią

Wycieki pamięci
Segmentation fault
Memory corruption
Null pointer exception
Zapobieganie fragmentacji pamięci

C++ kompatybilny z C

Pozostałości w składni sprzed 35 lat
Integer promotion

C++ język wieloplatformowy

Rozmiar longa może być różny
Standard nie precyzuje wszystkiego
Kompilatory różnią się od siebie np kolejnością ewaluacji wyrażeń

Bogactwo C++

Wybierasz czego chcesz użyć
std::bind czy boost::bind?
Łatwo napisać nieczytelny kod

C++ wydajność

Wymagane zarządzanie pamięcią
Bardziej wydajny niż inne języki?
Czy wszystkie miejsca w kodzie muszą być tak samo wydajne?
Przedwczesna optymalizacja?

Testy

“Writing unit tests in Go or Java is quite easy and natural, whereas in C++ it can be very difficult (and it isn’t exactly ingrained in C++ culture to write unit tests)” - https://testing.googleblog.com/2016/08/hackable-projects.html *By: Patrik Höglund*

Compilation time, IDE support, test frameworks

2004, Michael Feathers, Working Effectively with Legacy Code

CppUnit, CppUnitLite, Google Test

Szybkość pisania kodu

IDE
Produkt w 2 tygodnie?

Wnioski

C++ to trudny język
Jeśli można to wybierz język z zarządzaniem pamięcią
Nie optymalizuj za wcześnie
Pisz ładny i prosty kod

Dziękuję.