Design, Evolve and Optimize the code

吴钊
IBM

Legal Disclaimer

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM, PowerPC and the IBM logo are trademarks or registered trademarks of IBM or its subsidiaries in the United States and other countries.
- Other company, product, and service names may be trademarks or service marks of others.

主要内容

- 代码设计 (Design)
- 代码迁移与进化 (Evolve)
- 代码优化 (Optimize)

Design (Part 1)

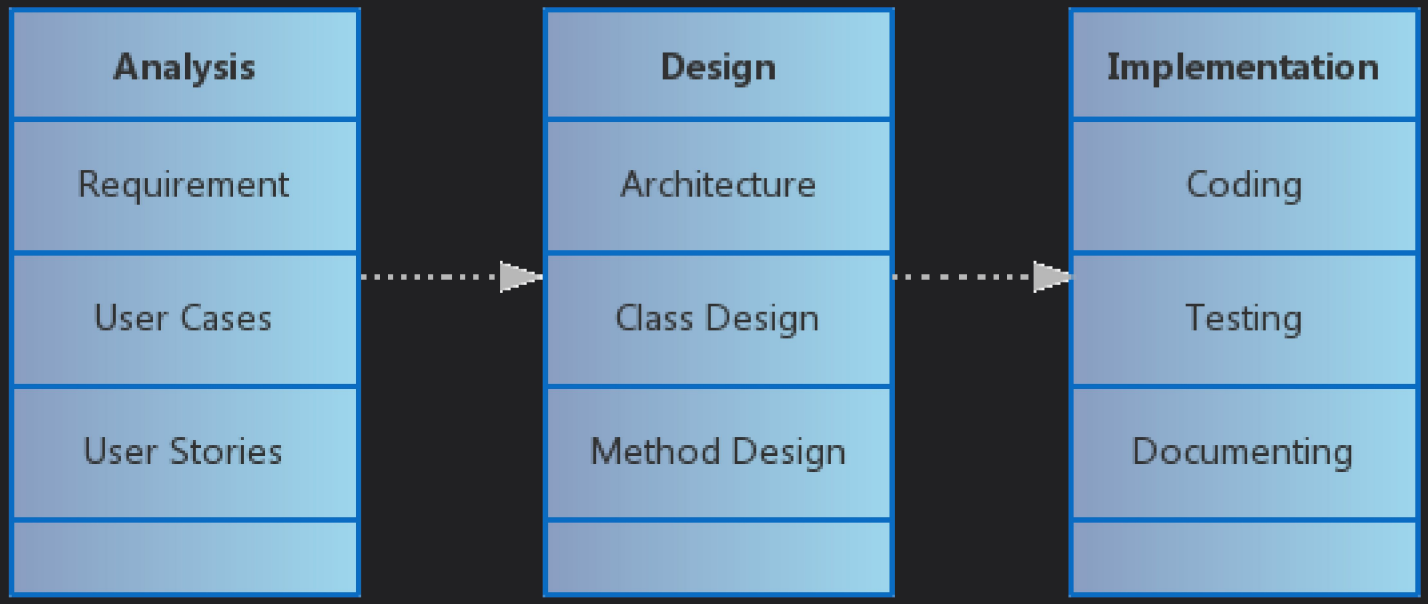
设想场景

目前我们的APP用户很多，横跨iOS，Android，WP...

问题

如何让APP跨平台？？？

软件工程的观点


Analysis
Requirement
User Cases
User Stories
Design
Architecture
Class Design
Method Design
Implementation
Coding
Testing
Documenting

头脑风暴、UML ...

Native UI， X-platform Same Core

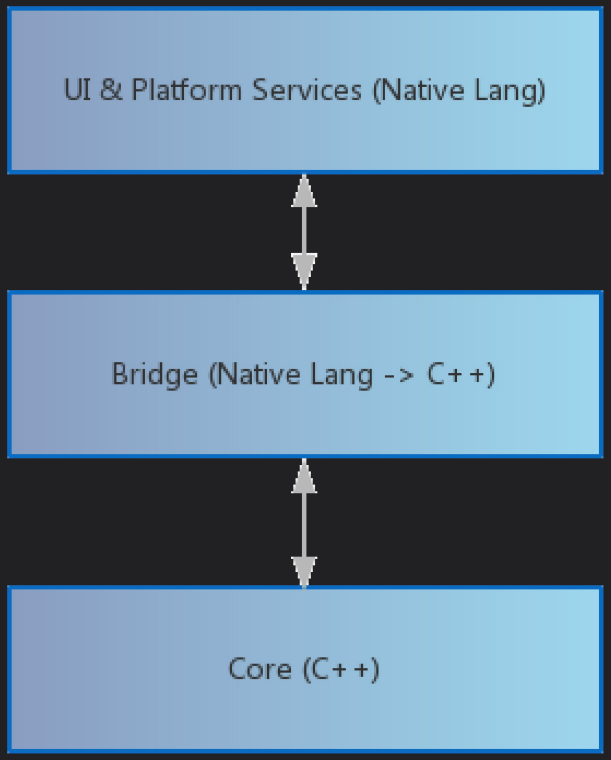
需求

- APP跨平台
- APP具有Native UI与Native Platform Services
- APP性能高效
- ...

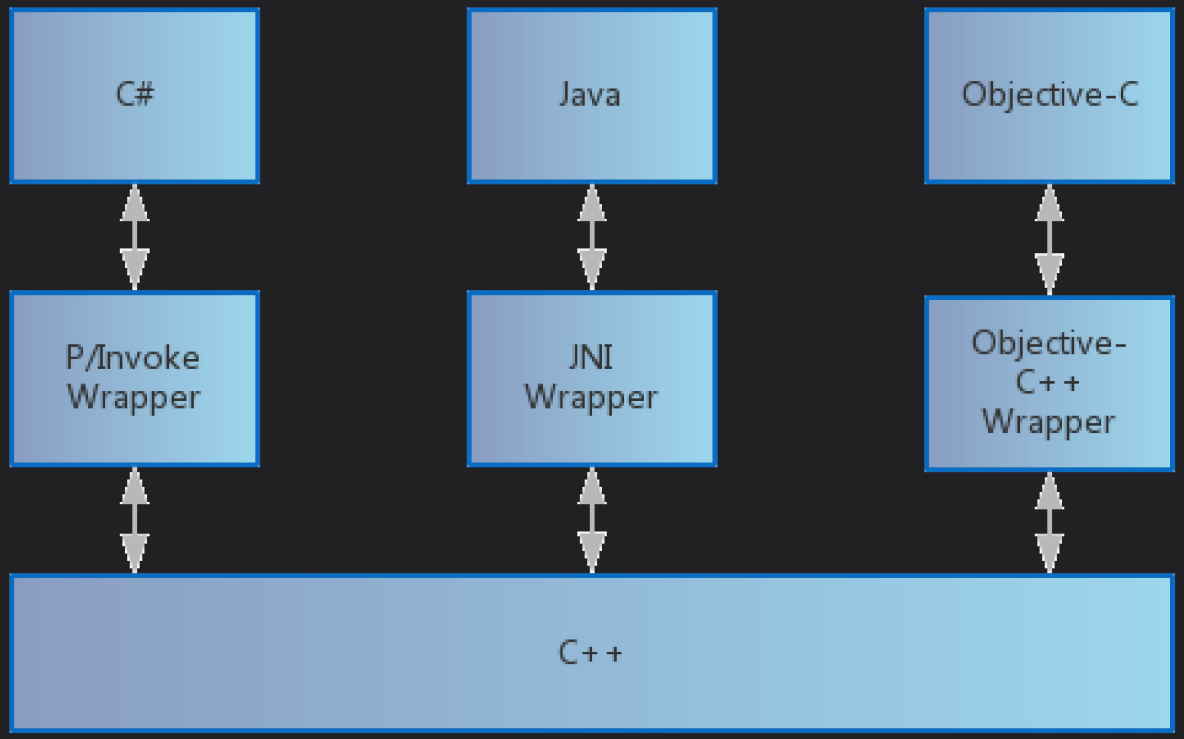
设计技术方案

- 调研是否已经有业内解决方案
  - Facebook, Google, Microsoft, Dropbox?...
- 技术方案是否适合自身情况
  - 技术方案的难度
  - 技术方案的成本
  - ...

Design Architecture



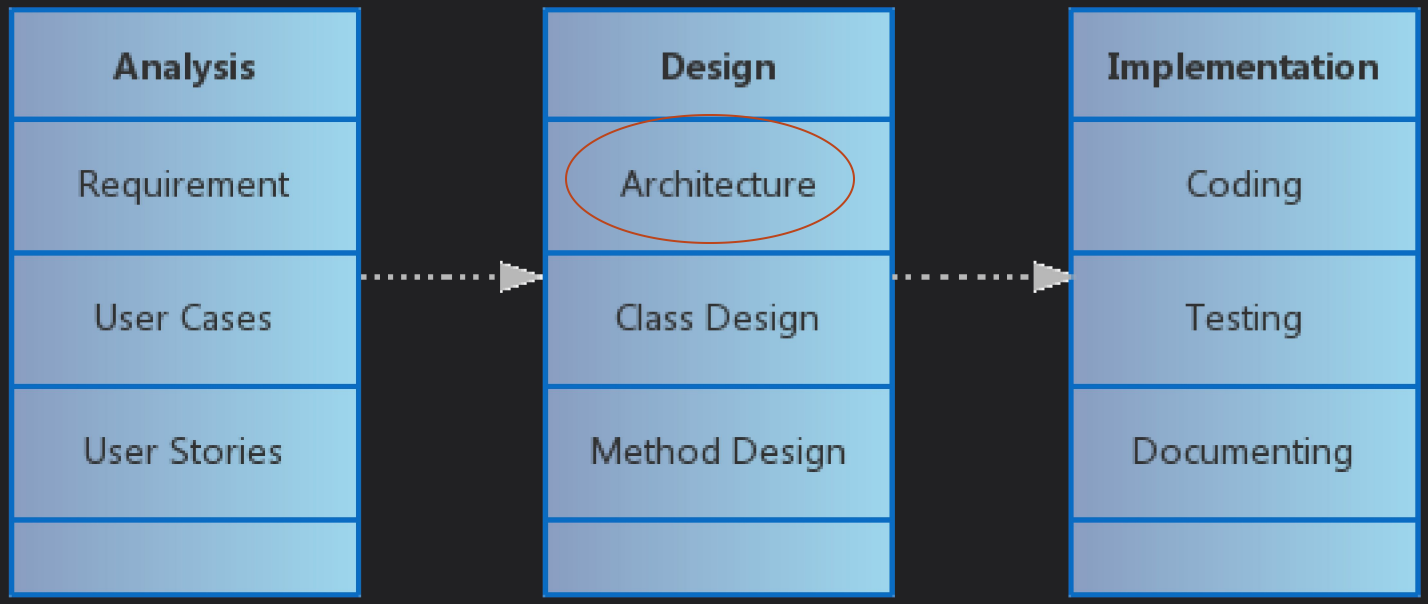
Design Architecure



讨论

是否还有更好的设计架构？

我们在哪儿？


Analysis
Requirement
User Cases
User Stories
Design
Architecture
Class Design
Method Design
Implementation
Coding
Testing
Documenting

Tips

- 代码的设计不仅仅体现在有形的Class, Function等设计，也体现在无形的需求确定，技术方案，宏观架构等方面，而后者更应该小心考量。
- 不要着急考虑代码细节方面，如我们应该用哪种设计模式，我们应该设计怎么样的Class关系链来表示等。
- 代码不是目的，代码是你的思考体现，避免返工。

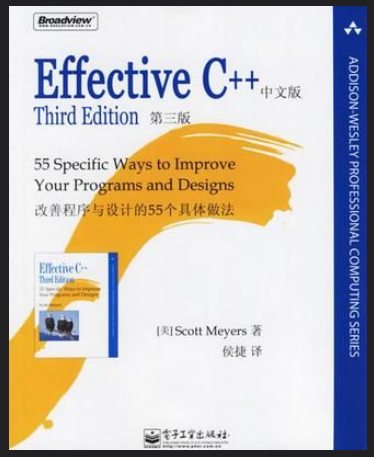
Design (Part 2)

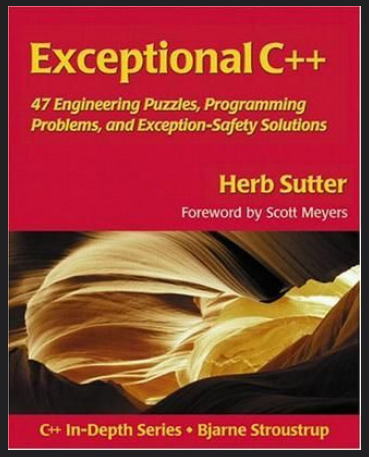
代码的功能性与非功能性

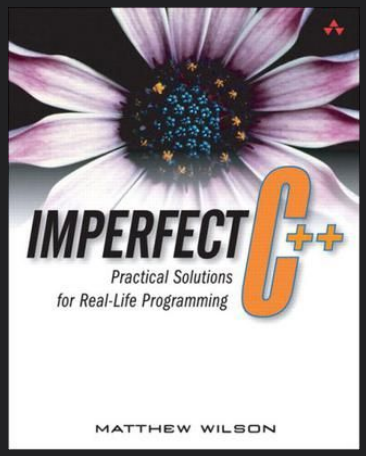
- 功能性
  - 使用代码做正确的事情，如要求对数组的元素从小到大排序，代码不能设计成从大到小排序
- 非功能性
  - 性能
  - 安全
  - 可扩展性
  - 可移植性
  - ……

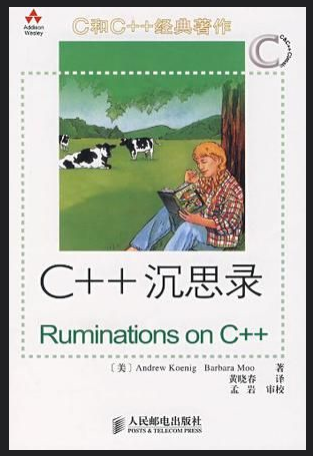
如何设计与编写C++代码达到满足非功能性的要求？

太多的书籍.....


Broadview
Effective C++ 中文版
Third Edition 第三版
55 Specific Ways to Improve Your Programs and Designs
改善程序与设计的55个具体做法
[美] Scott Meyers 著
侯捷 译
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


Exceptional C++
47 Engineering Puzzles, Programming Problems, and Exception-Safety Solutions
Herb Sutter
Foreword by Scott Meyers
C++ In-Depth Series • Bjarne Stroustrup


IMPERFECT C++
Practical Solutions for Real-Life Programming
MATTHEW WILSON


C++沉思录
Ruminations on C++
[美] Andrew Koenig Barbara Moo 著
黄晓春 译
孟岩 审校
人民邮电出版社

Big Story，但是我会根据我的经验给出一些建议

Problem #1

```
void foo () {
  Lock lock;
  lock.acquire();
  if (...)
     if (...)
        return;
  else
     throw "ERROR";

  lock.release(); // This may not be invoked... BUG!
}
```

Solution and Tips

使用RAII原则设计与内存、资源相关的class

RAII

```
class LockGuard {
public:
  LockGuard(Lock& lock) : lock_(lock){ lock_.acquire();}
  ~LockGuard() noexcept {lock_.release();}
private:
  LockGuard(const LockGuard&)=delete;
  LockGuard& operator=(const LockGuard&)=delete;
private:
  Lock& lock_;
};
```

RAII Usage

```
void foo () {
  Lock lock;
  LockGuard guard(lock);
  if (...)
     if (...)
        return;
  else
     throw "ERROR";
   // Destructor will be responsible for calling lock.release()
}
```

Problem #2

- 频繁的拷贝数据，却很少修改，然而拷贝的开销很大
  - 数据库的快照备份
  - ZFS
  - Linux fork进程创建
  - string

Solution and Tips

使用Copy-On-Write优化技术来设计处理大量拷贝、少修改的class

COW Generic Sample版本

```
template <class T>
class CowPtr {
public:
    CowPtr(T* t) : intern_holder_(t){}
    CowPtr(const std::shared_ptr<T>& refptr) : intern_holder_(refptr){}
    const T& operator*() const { return *intern_holder_; }
    T& operator*() {  change(); return *intern_holder_; }
    const T* operator->() const { return intern_holder_.operator->(); }
    T* operator->() {
        change();
        return intern_holder_.operator->();
    }
  // continue to next page
```

COW Generic Sample版本

```
  private:
   void change() {
       T* tmp = intern_holder_.get();
       if (!(tmp == nullptr || intern_holder_.unique())) {
           intern_holder_ = std::make_shared<T>(*tmp);
       }
   }
public:
   std::shared_ptr<T> intern_holder_;
};
```

使用COW

```
auto str = new string("Hello");
CowPtr<string> s1 = new string("Hello");
CowPtr<string> s2(s1);
// before COW
cout << s1.intern_holder_ << endl;
cout << s2.intern_holder_ << endl;
//after COW
s2->operator[](0) = 'h';
cout << s1.intern_holder_ << endl;
cout << s2.intern_holder_ << endl;
//Output:
012AE470 012AE470 012AE470 012AB01C
```

Some Useful Code Tips

- 多加assert，避免未期望的行为逃逸
- 避免原生的new与delete，使用智能指针
- 忘记C的编码方式，使用C++
  - 数组 -> vector
  - const char* -> string
  - .....

Evolve

从C++98/03到C++11/14/17

- 编译器版本的选择
  - GCC 4.8, GCC 4.9, GCC 5.X, Clang 3.X, Visual C++ 201X?
  - 兼容性：第三方库、Core Language / API / ABI
- 项目代码的升级
  - 重构
    - 逐步升级?
    - 推倒重来?

了解C++11/14/17

- 现代C++与C++旧标准的不同点
  - 语言的升级
  - 库的升级

- 语言的改变
  - 语言的便利性: auto, decltype, initializer_list, lambda...
  - 性能提升: constexpr, move & rignt value references...
  - 安全性: nullptr, =delete, final...
  - ...

了解C++11/14/17

- 库的改变
  - 便利性与安全性: Smart Pointers, type_traits.....
  - 性能: move, STL算法并行 (C++17), SIMD (C++17).....
  - 标准化: thread, atomic.....
  - .....

Tips

- 在代码、API 、ABI等兼容的前提下，升级支持C++11/14/17的合适版本编译器
- 逐步升级为新标准的代码，如新标准的代码便利性部分，提升性能部分等
- Test-Driven替换

Optimize

"On the other hand, we cannot ignore efficiency."

- Jon Bentley

Simple Case

```
#include <cmath>
int b = 0;
int a = rand() % 2;
void foo() {
  for (; a < 50; a++) {
          b++;
   }
}
int main() {
    foo();
}
```

Time Data

rand	1	0.28	0.28	0.00
std::_Generic_error_category::_	3	0.36	0.27	0.00
check_managed_app	1	0.25	0.25	0.00
__set_app_type	1	0.24	0.24	0.00
pre_c_init	1	14.13	0.21	0.02
__security_init_cookie	1	0.19	0.19	0.00
_CrtSetCheckCount	1	0.17	0.17	0.00
`dynamic initializer for 'std::_E	1	1.83	0.17	0.00
_onexit	4	7.52	0.12	0.00
`dynamic atexit destructor for	1	4.91	0.12	0.01
_onexit	4	0.11	0.11	0.00
std::_System_error_category::~	1	4.79	0.11	0.01
std::_System_error_category::_	1	0.16	0.09	0.00
std::error_category::error_cate	3	0.09	0.09	0.00
foo	1	0.08	0.08	0.00

优化的第一件、最重要的事情：
Profile! Profile! Profile!

Optimization

- Step 1: Profile
- Step 2: 热点函数
- Step 3: 思考与选择优化技术
  - 通用优化（语言、库等）
  - 编译器优化级别（O2-> O3?）
  - 使用优化技术（COW，模板元编程等）
  - 并行化 / GPU（OpenMP, CUDA等）
  - .....
- Step 4: 再次Profile

Some C++ Code Tips

- 按引用传递，而非按值
  - void foo(const vector<int>&) v.s. void foo(vector<int>)
- 为你的类加入move构造函数与move赋值操作符
- 明白异常的开销，若不会发生异常，加上noexcept
- 尽量避免转换（转换的代价很昂贵）

Some C++ Code Tips

- 指针是编译器优化的“万恶之源”，使用restrict考虑帮助编译器做Pointer Alias优化

Restrict Case

```
void f(int *a, int *b, int *c)
{
  *a += *c;
  *b += *c;
}
```

```
void f(int * __restrict__ a,
int* __restrict__ b, int*
__restrict__ c)
{
  *a += *c;
  *b += *c;
}
```

GCC O3 Assembly Output

```
f(int*, int*, int*):
    movl    (%rdx), %eax
    addl    %eax, (%rdi)
    movl    (%rdx), %eax
    addl    %eax, (%rsi)
    ret
```

```
f(int*, int*, int*):
    movl    (%rdx), %eax
    addl    %eax, (%rdi)
    addl    %eax, (%rsi)
    ret
```

Some C++ Code Tips

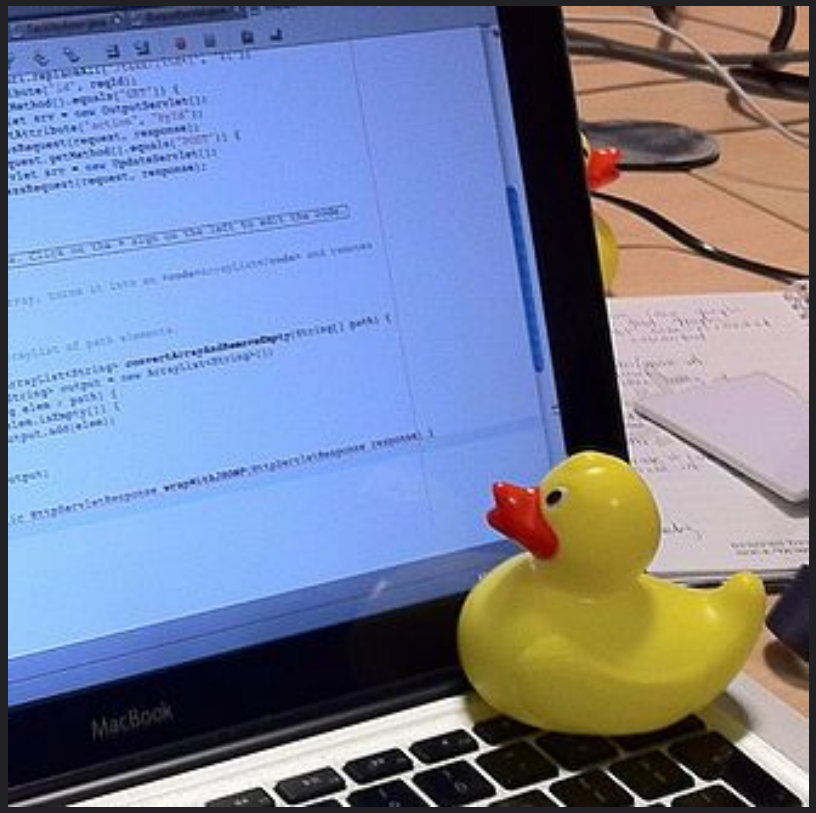
- 并行化不一定是万能药，若使用的不好，可能比串行还差，如真选用并行模型，优先考虑OpenMP
  - 若利用GPU Offloading，考虑OpenMP + CUDA
- 明白RVO、std::move，COW的适用条件
  - https://isocpp.org/blog/2015/07/rvo-v.s.-stdmove-zhao-wu
- 正确性是第一，性能第二。若发生性能瓶颈，则不惜代价，哪怕使用移植性很差的inline asm

One ASM Cookie: Android Kenerl

```
INT32 InterlockedIncrement(INT32* lpAddend){
  INT32 i = 1;
  __asm__ __volatile__(
  "xaddl %0,%1"
  :"+r" (i),"+m" (*lpAddend)
   ::"memory"
);
  return *lpAddend;
}
```

Extra One Tips

小黄鸭调试法



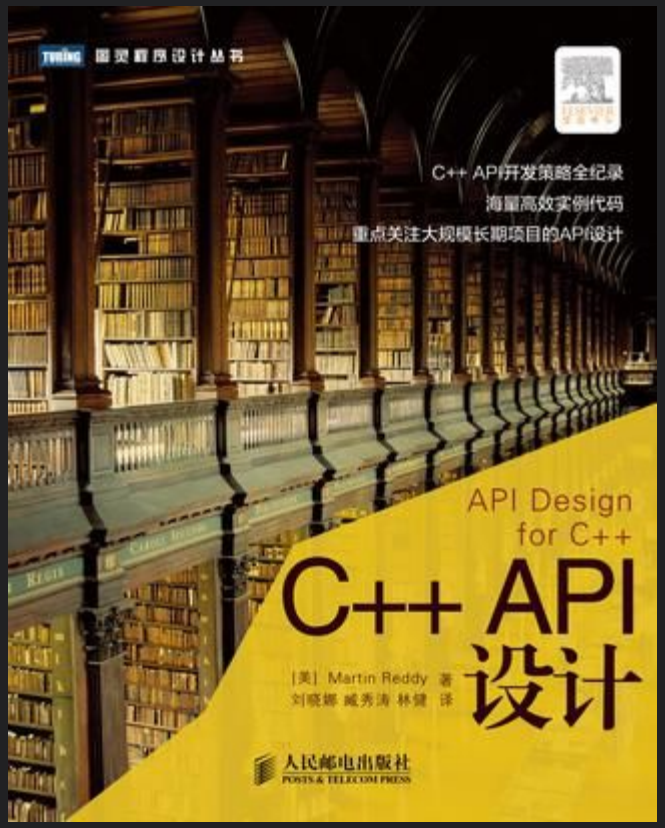
在你走向任何人、告诉他们为何某事做不到、为何担搁、为何出问题之前，先停下来，听一听你心里的声音，与你的显示器的橡皮鸭交谈.....

----《程序员修炼之道》

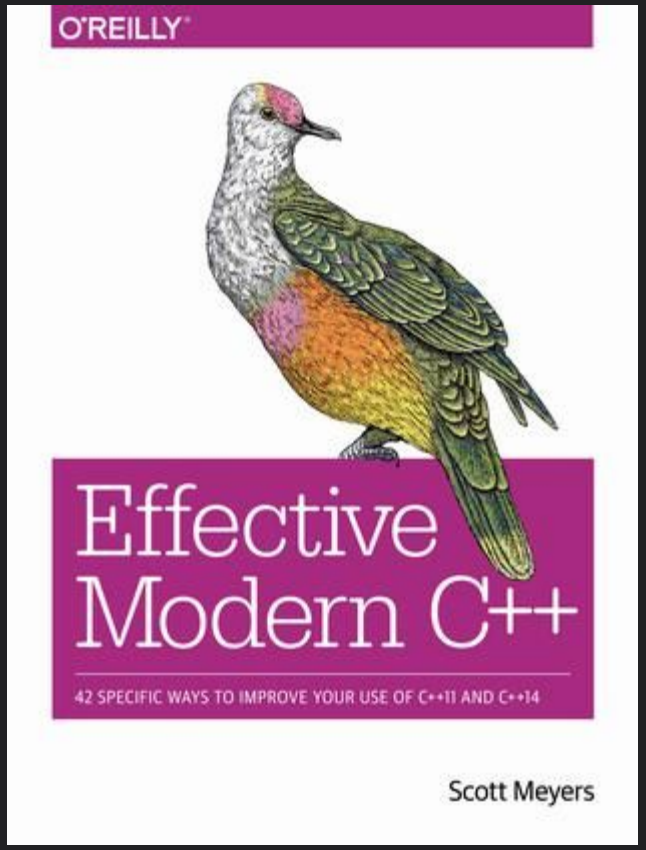
NEXT


PEARSON
大规模C++程序设计
[美] John Lakos 著 刘冰 张林 译 周尚贤 审校
Large-Scale C++ Software Design
机械工业出版社
China Machine Press

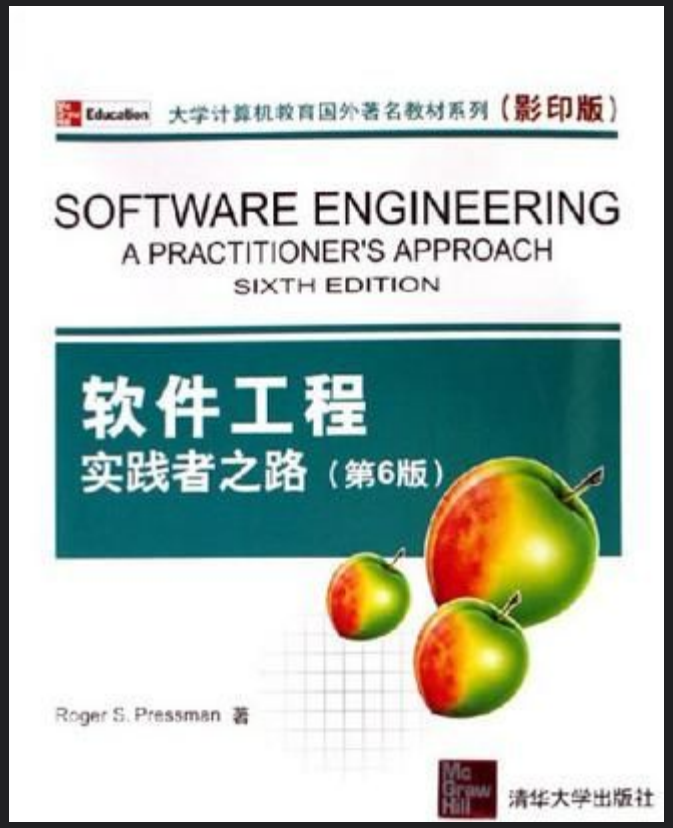
NEXT


C++ API开发策略全纪录
海量高效实例代码
重点关注大规模长期项目的API设计
API Design
for C++
C++ API
设计
[美] Martin Reddy 著
刘晓娜 臧秀涛 林健 译
人民邮电出版社
POSTS & TELECOM PRESS

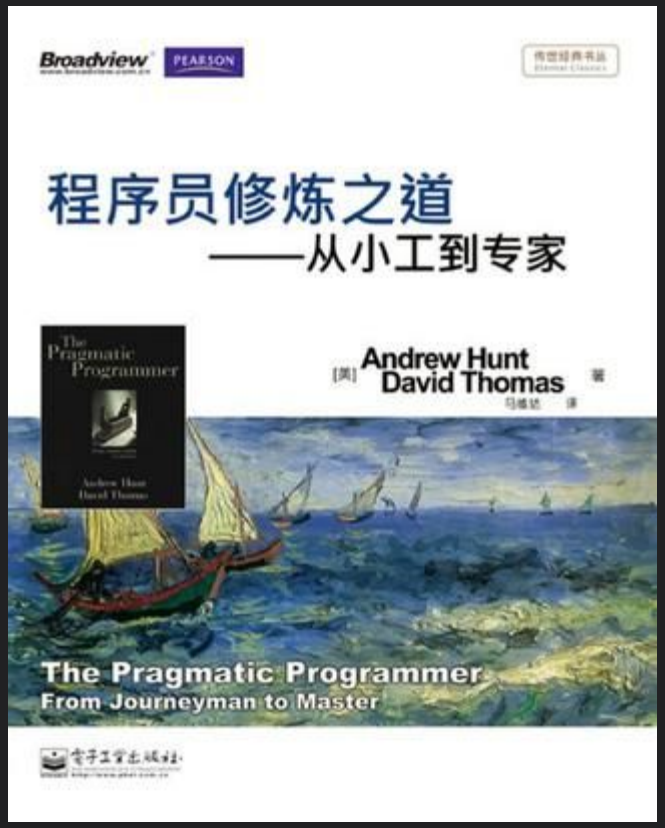
NEXT


O'REILLY
Effective Modern C++
42 SPECIFIC WAYS TO IMPROVE YOUR USE OF C++11 AND C++14
Scott Meyers

NEXT


EducaBon
大学计算机教育国外著名教材系列（影印版）
SOFTWARE ENGINEERING
A PRACTITIONER'S APPROACH
SIXTH EDITION
软件工程
实践者之路（第6版）
Roger S. Pressman 著
Mc Graw Hill
清华大学出版社

NEXT


Broadview
PEARSON
程序员修炼之道
——从小工到专家
The Pragmatic Programmer
[美] Andrew Hunt
David Thomas
著
The Pragmatic Programmer
From Journeyman to Master

NEXT

- 非常好的有关C++优化的资料：
  - http://www.agner.org/optimize/optimizing_cpp.pdf
- My book...

THANKS!