C++设计(1)

C++项目遇到的问题

- 功能扩展，维护困难
- 项目生命周期长
- 编译时间长
- 第三方库依赖多，更新维护困难
- 内存管理困难
- 性能瓶颈突出
- 代码调试困难

大规模C++程序设计 书

- 作者：John Lakos， 就职于Mentor Grapics ，明导， C++项目研发
- Mentor Graphics， 是电子设计自动化(EDA)技术的领导产商，它提供完整的软件和硬件设计解决方案，是全球三大EDA公司之一，2016年，西门子以45亿美元收购Mentor Graphics，并入西门子数字化工业软件部门，Mentor将于2021年1月起正式更名为Siemens EDA

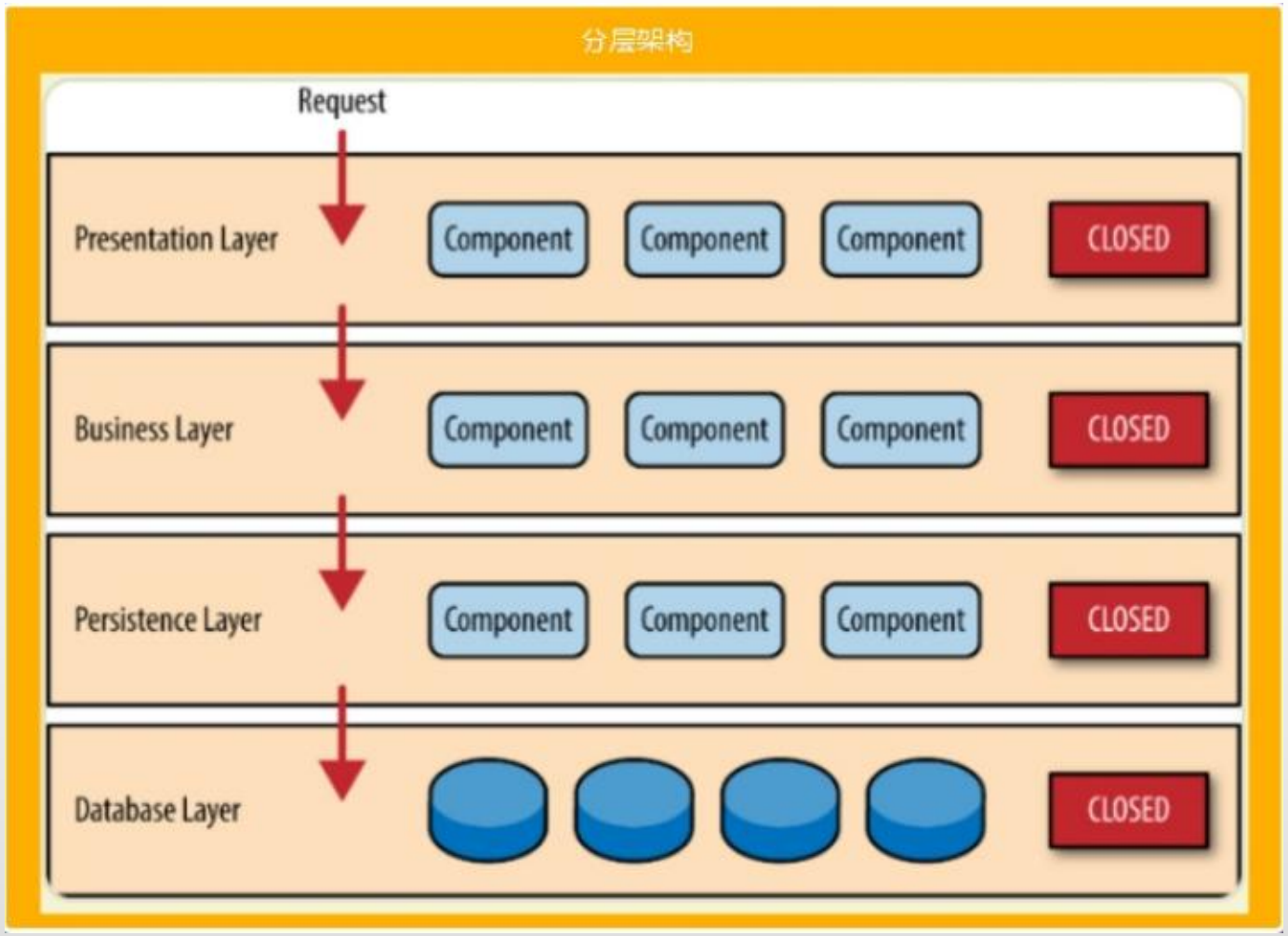
书的第二章

- 禁止使用全局变量和全局函数
- 头文件里使用前置声明代替头文件引用
- 友元破坏封装
- 使用断言assert
- 数据成员私有
- 接口记录成文档
- 规范命名
- 使用namespace

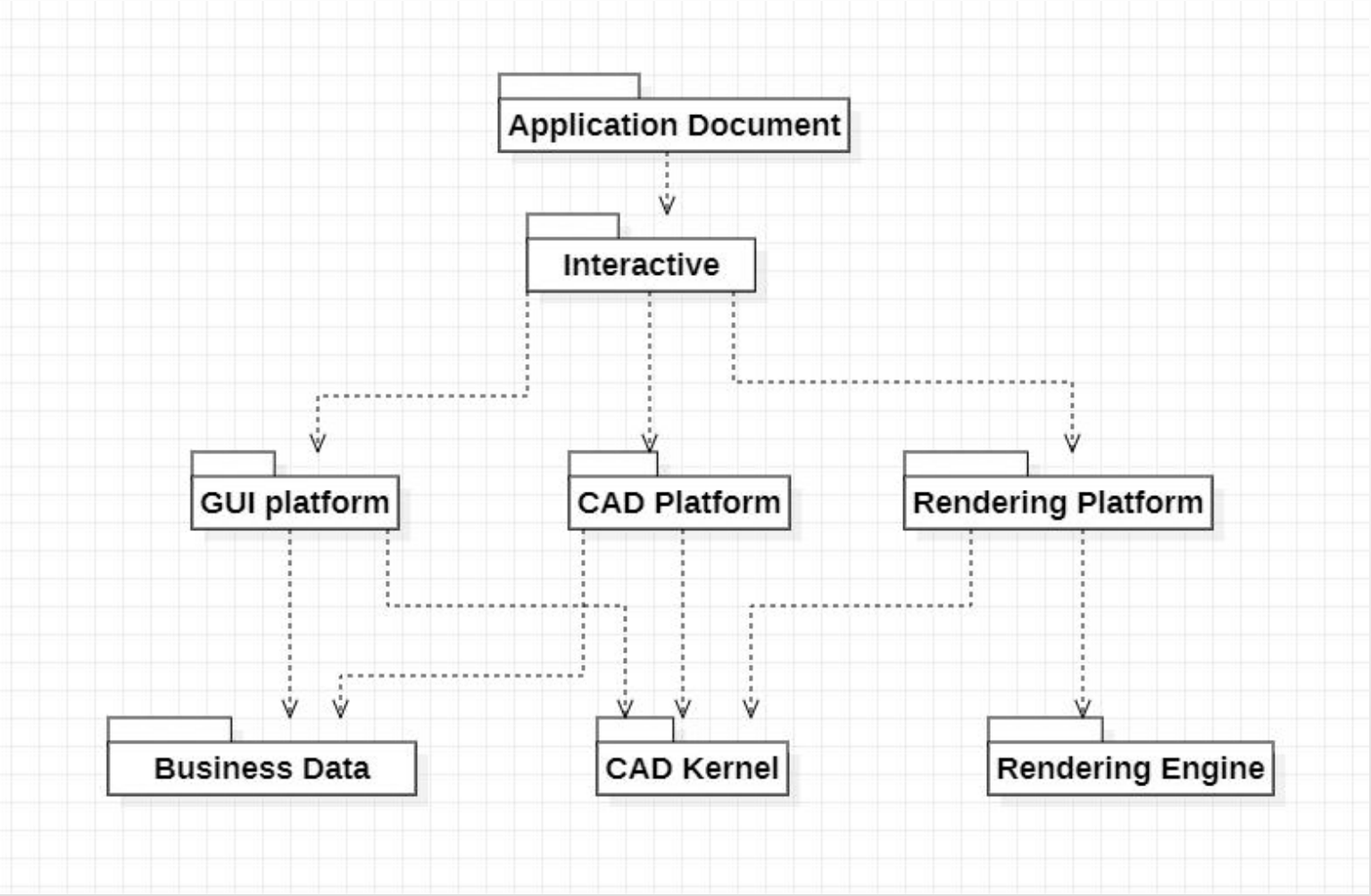
常见的分层架构


分层结构
分层架构（layered architecture）是最常见的软件架构，也是事实上的标准架构。
最常用就是四层结构：
表现层（presentation）：用户界面，负责视觉和用户互动
业务层（business）：实现业务逻辑
持久层（persistence）：提供数据，SQL 语句就放在这一层
数据库（database）：保存数据
有的软件在逻辑层和持久层之间，加了一个服务（service），提供不同业务逻辑需要的一些通用接口。
优缺点
优点：结构简单，容易分工组织，每层可独立测试，其他层的接口通过模拟解决
缺点：部署麻烦，代码调整或增加功能麻烦，软件升级时需要暂停整个服务，扩展性差

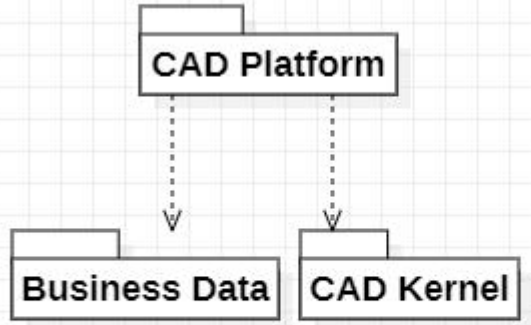
分层架构

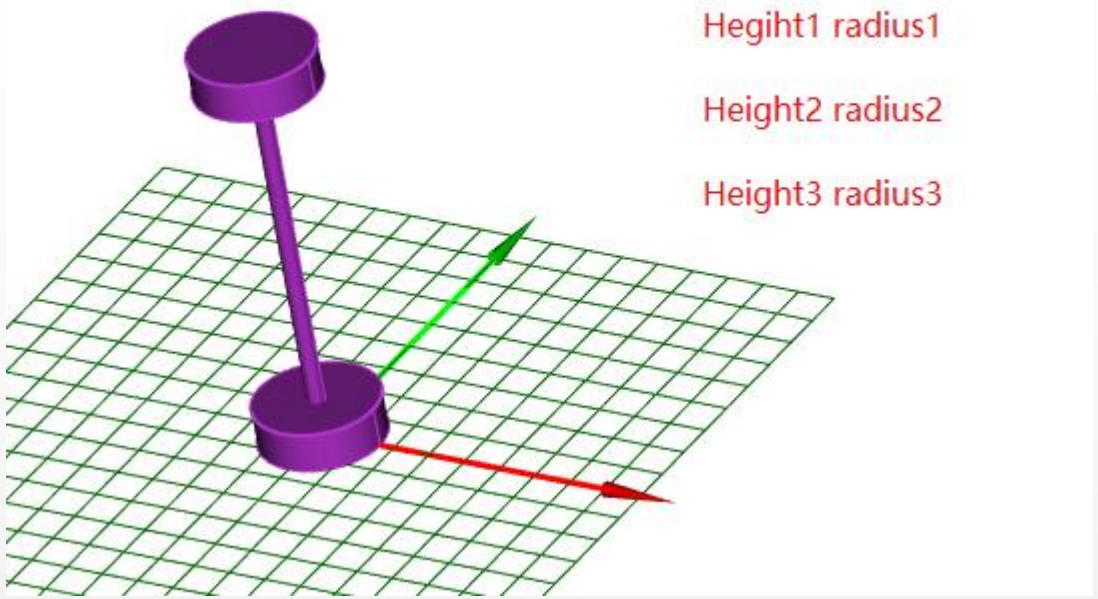

分层架构
Request
Presentation Layer
Component
Component
Component
CLOSED
Business Layer
Component
Component
Component
CLOSED
Persistence Layer
Component
Component
Component
CLOSED
Database Layer
CLOSED

CAD项目依赖图（package 包）



CAD Platform

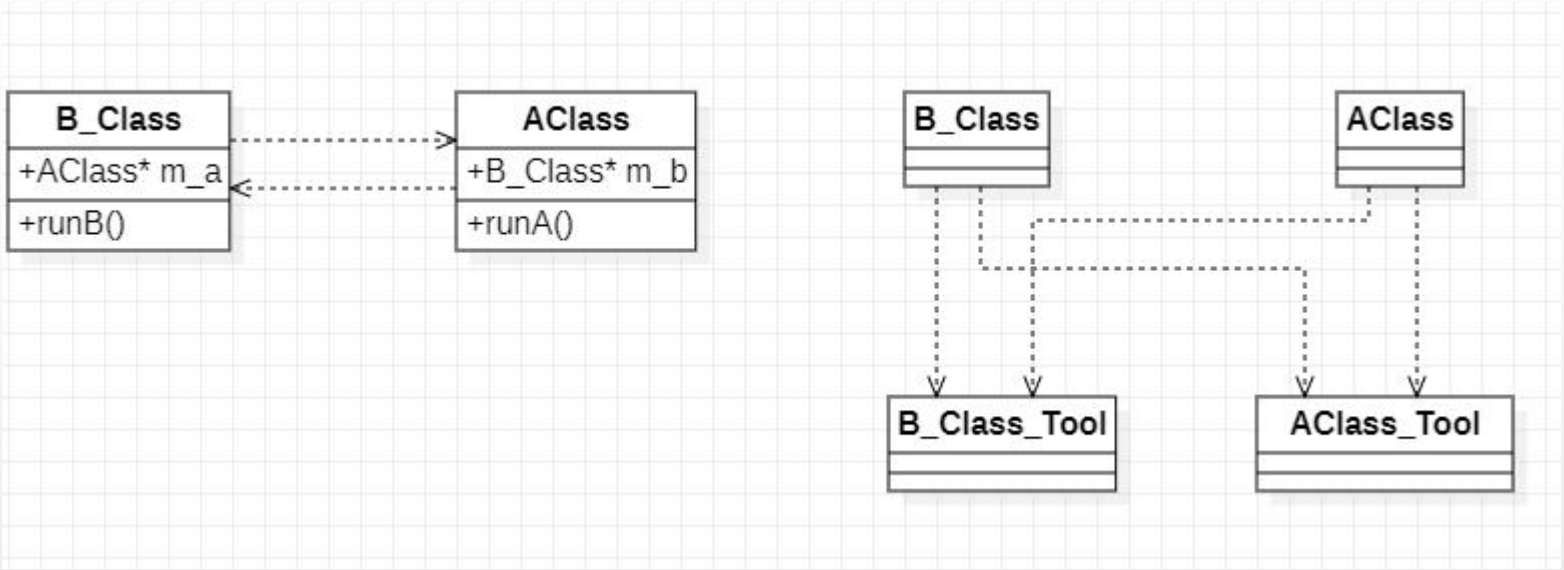





```
struct  sPadStruct
{
    double height1;
    double center1;
    double radius1;
    //...
};
```

```
Shape* ShapeCreator::CreatePad(const sPadStruct& shapeData)
{
    //1. API_Cylinder1;
    //2. API_Cylinder2;
    //3. API_Cylinder3;
    //4. API_Boolean_Unite;
}
```

谁会这样设计?


B_Class
+AClass* m_a
+runB()
AClass
+B_Class* m_b
+runA()
B_Class
AClass
B_Class_Tool
AClass_Tool

循环依赖

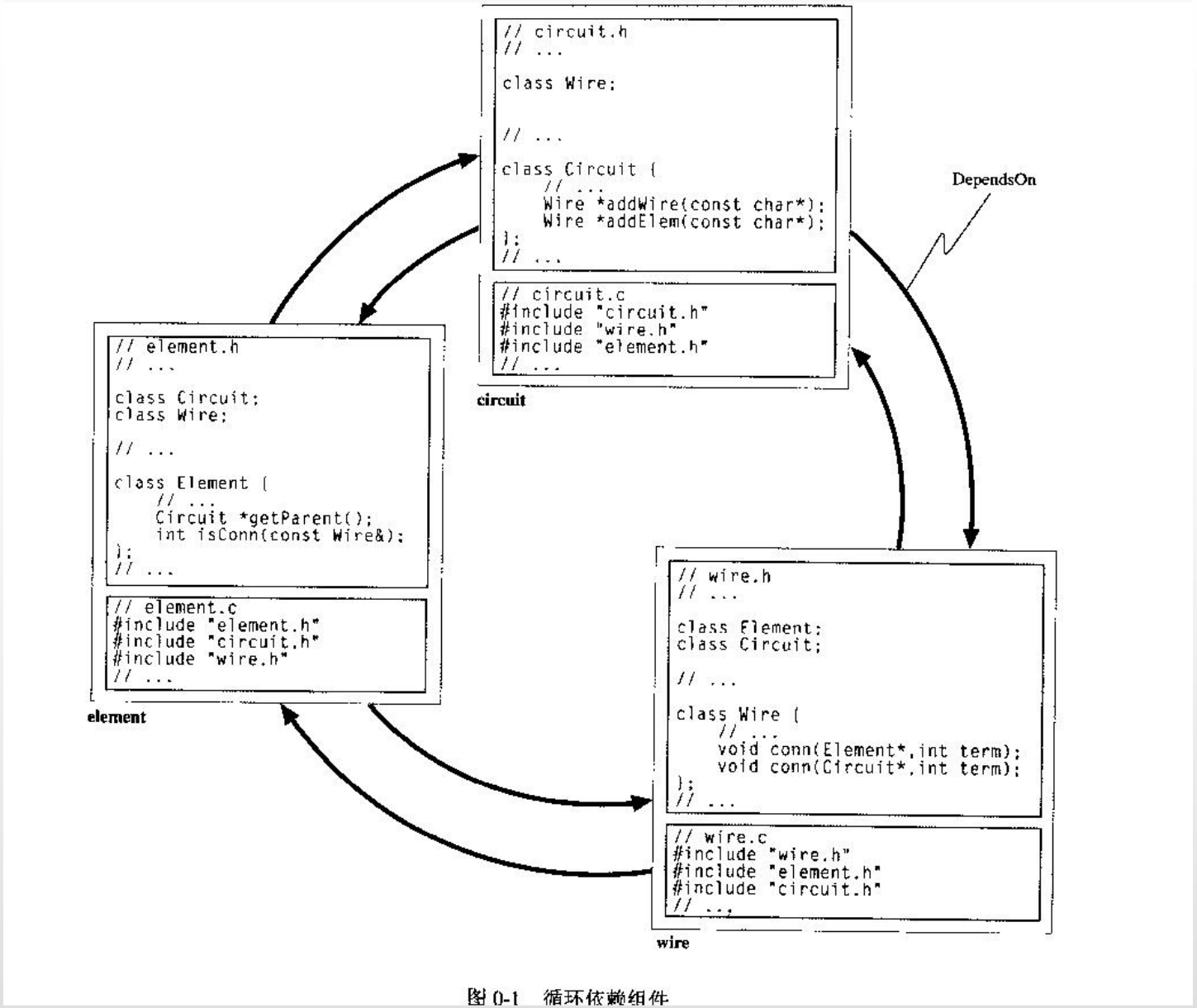
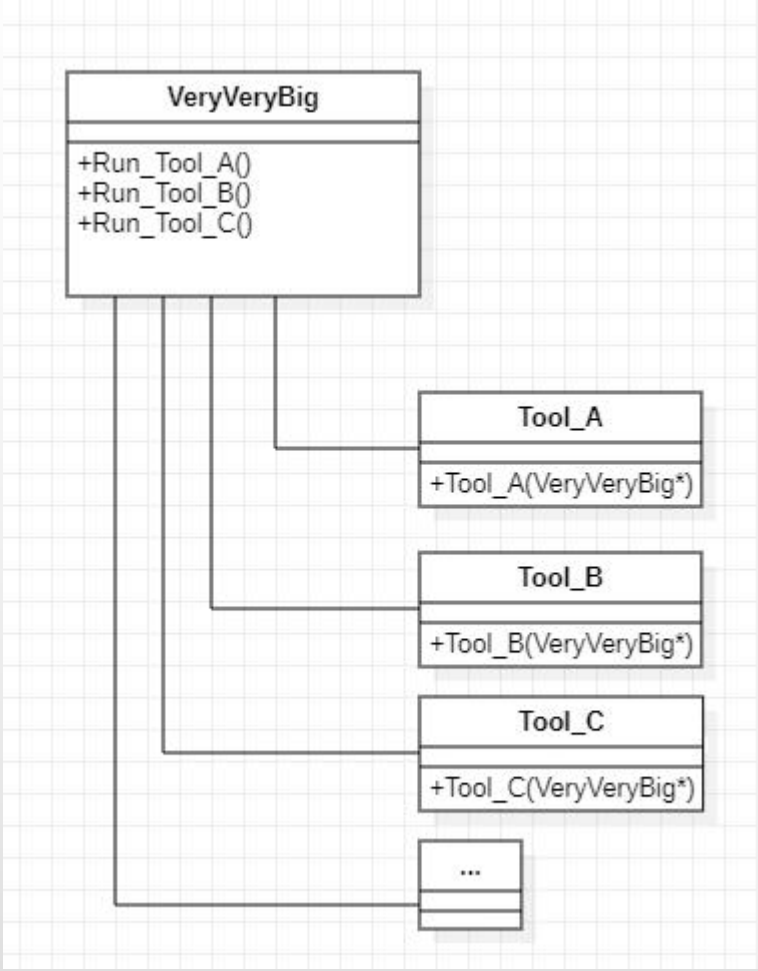


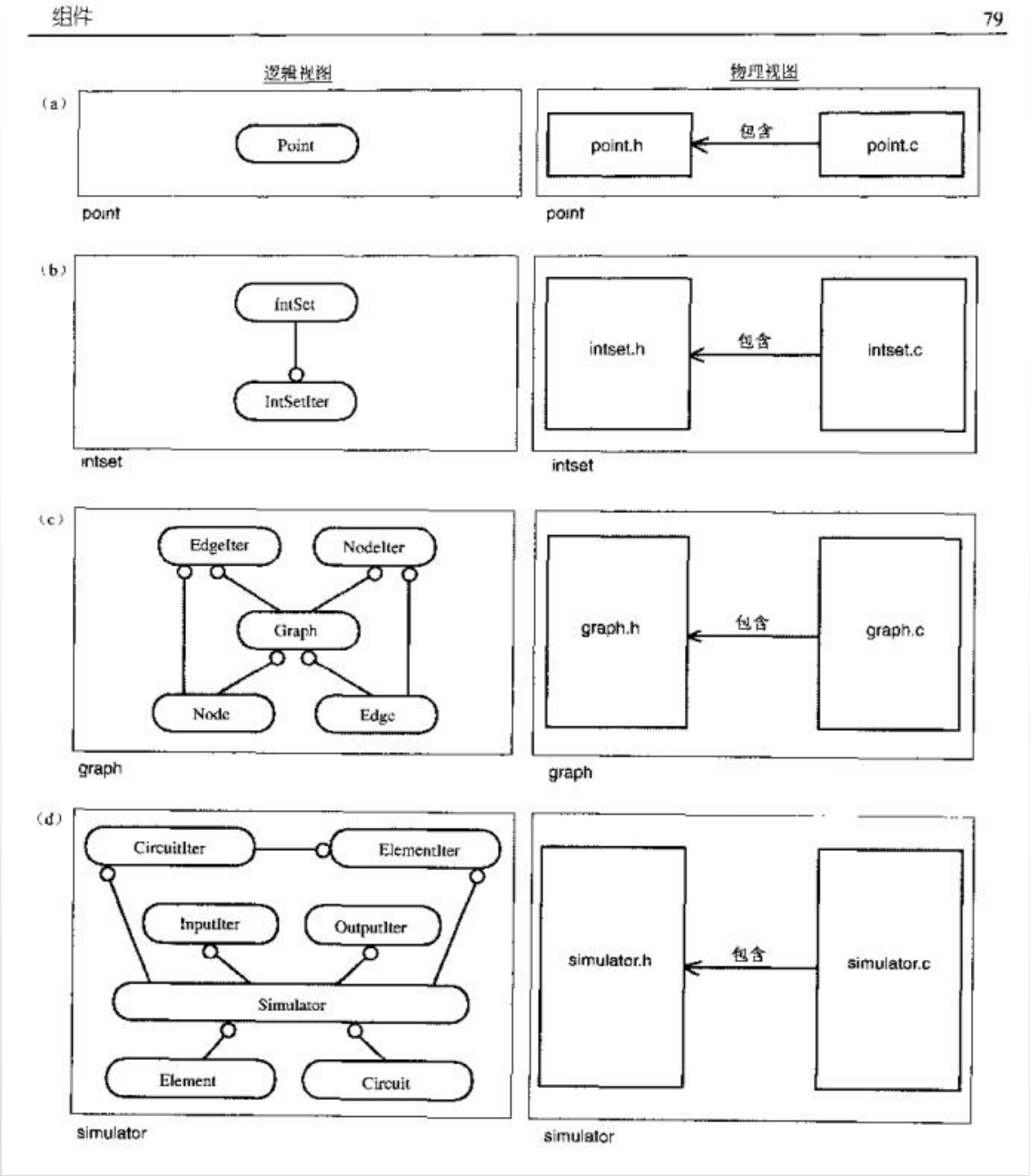
图 0-1　循环依赖组件

拆分Class

VeryVeryBig 是一个
近7万行代码的C++ Class



组件


组件
79
逻辑视图
物理视图
(a)
Point
point
point.h
包含
point.c
point
(b)
IntSet
IntSetIter
intset
intset.h
包含
intset.c
intset
(c)
EdgeIter
NodeIter
Graph
Node
Edge
graph
graph.h
包含
graph.c
graph
(d)
CircuitIter
ElementIter
InputIter
OutputIter
Simulator
Element
Circuit
simulator
simulator.h
包含
simulator.c
simulator

层次化

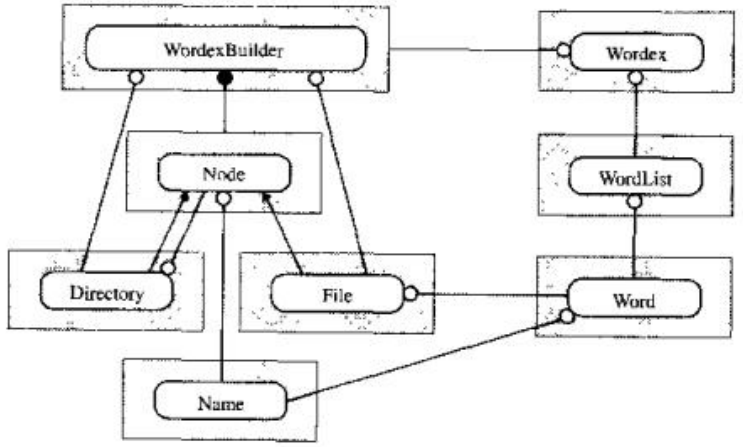


图 4-13　这个设计是可层次化的吗

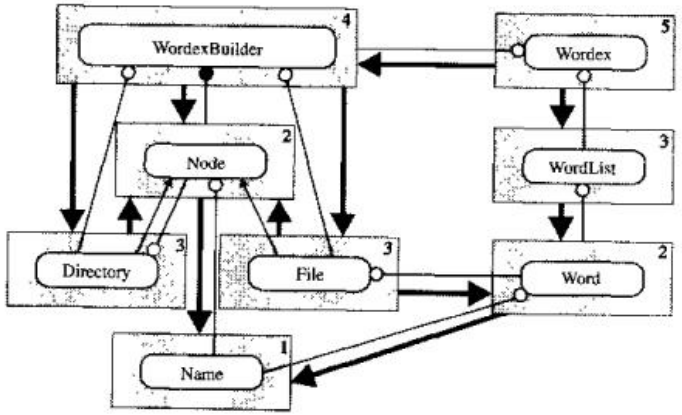


图 4-14　组件/类图

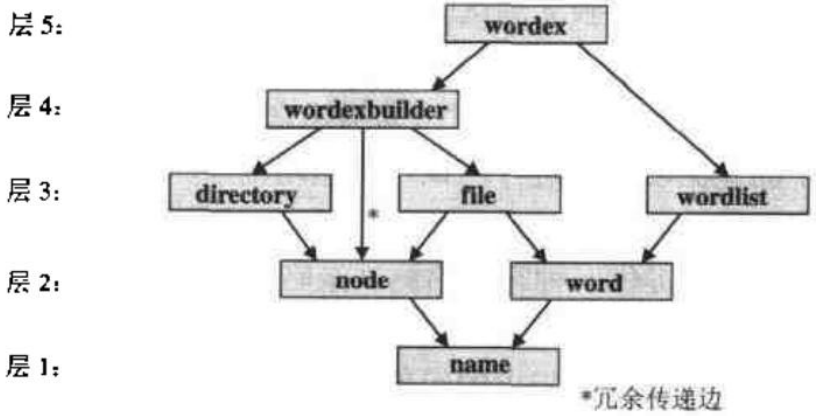
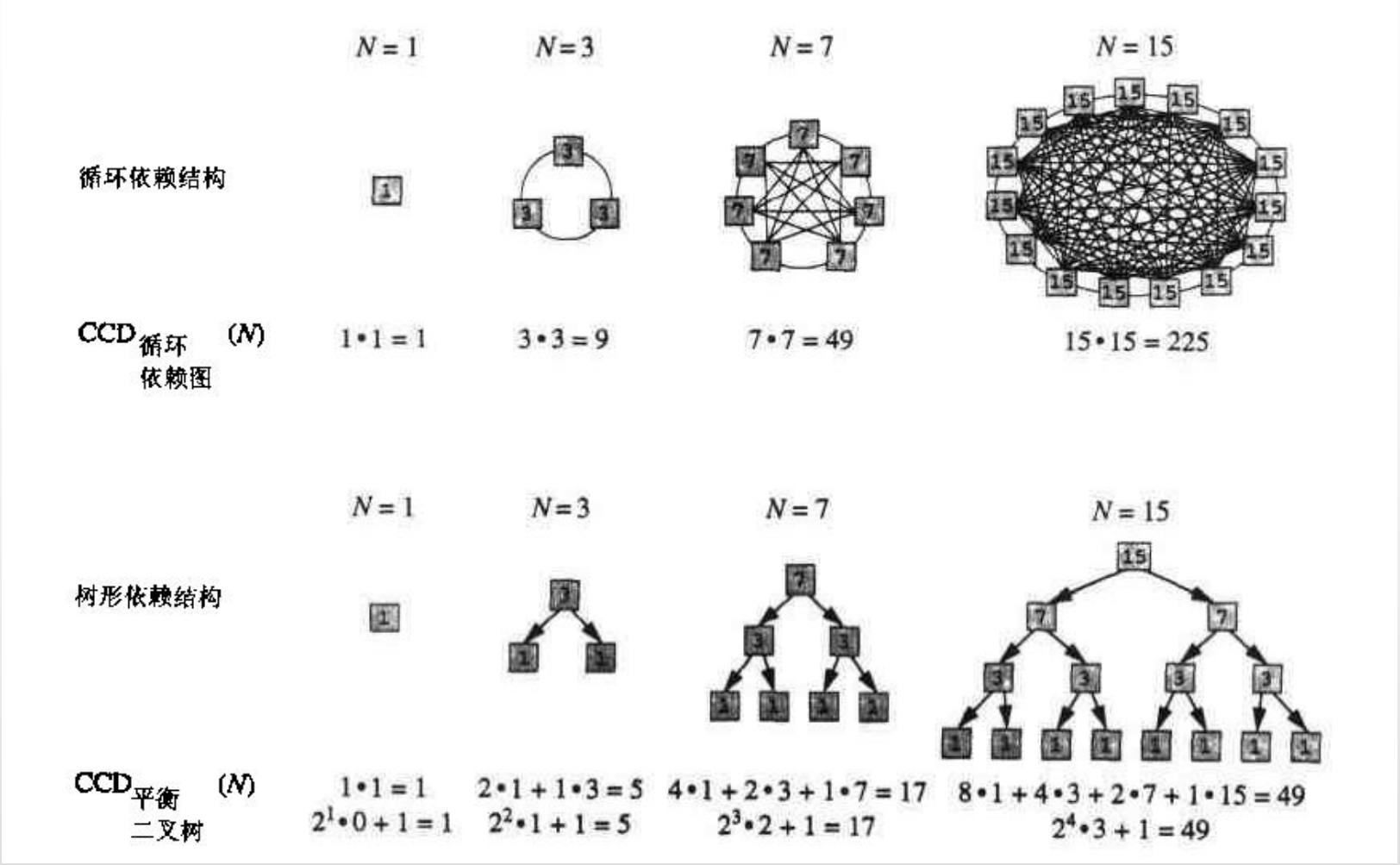


图 4-15　组件（直接）依赖图

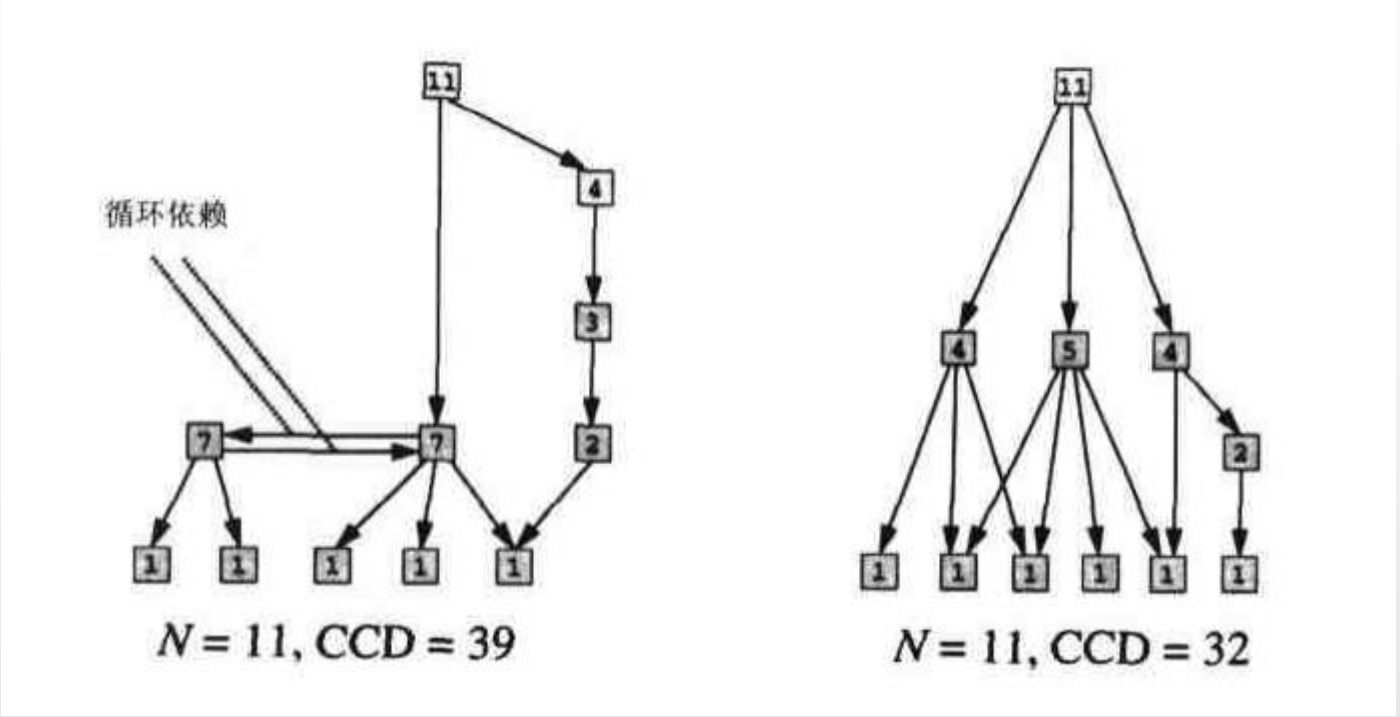
CCD(Cumulative Component Dependency)

- 可测试性
- 累计组件依赖：一个子系统内所有组件进行增量测试时，测试每个组件所需要的组件数量总和

CCD计算


N = 1
N = 3
N = 7
N = 15
循环依赖结构
CCD循环依赖图 (N)
1•1 = 1
3•3 = 9
7•7 = 49
15•15 = 225
N = 1
N = 3
N = 7
N = 15
树形依赖结构
CCD平衡二叉树 (N)
1•1 = 1
2•1 + 1•3 = 5
4•1 + 2•3 + 1•7 = 17
8•1 + 4•3 + 2•7 + 1•15 = 49
2^1•0 + 1 = 1
2^2•1 + 1 = 5
2^3•2 + 1 = 17
2^4•3 + 1 = 49

CCD计算

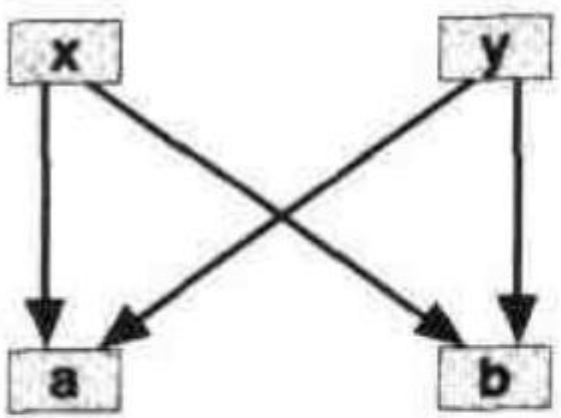


NCCD (Norm CCD)

定义：**标准累积组件依赖**（NCCD）是指包含 N 个组件的子系统的 CCD 值与相同大小的树型系统的 CCD 值的比值。

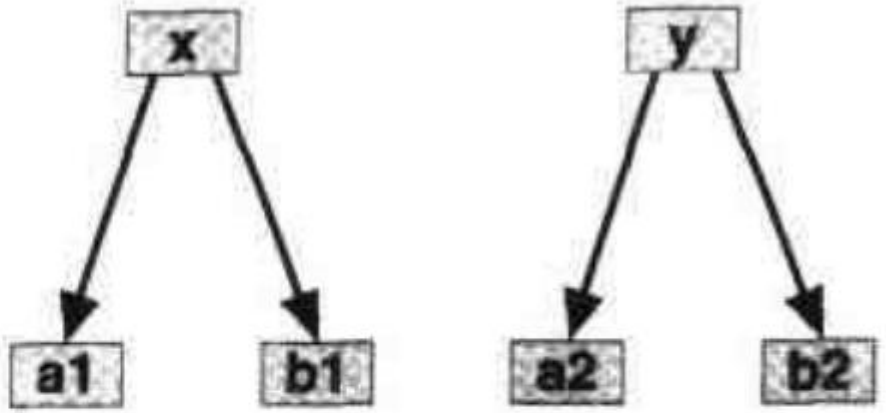
$$\mathrm{NCCD}(\text{子系统}) = \frac{\mathrm{CCD}(\text{子系统})}{\mathrm{CCD}_{\text{平衡二叉树}}(N_{\text{子系统}})}$$

NCCD计算实例



SIZE = 4
CCD = 8
NCCD = 1.05

（a）设计 A



SIZE = 6
CCD = 10
NCCD = 0.73

（b）设计 B

Q&A