X-Programmer 软件工程的播种机 www.umlchina.com 非程序员
总第8期 2001 12

目 录

- 本期相关的参考资料

本期审稿：Think、吕捷　　版面设计：徐远正

征 稿

关于需求，设计，构造，测试，维护，配置管理，管理，过程，工具，质量…原创或翻译均可。 (详细)

- XP On A Large Project
- Strategies for Surviving CMM

投稿： editor@umlchina.com
广告： adv@umlchina.com
反馈： think@umlchina.com

播种机
UMLChina.com

UMLChina 讨论组最新热门帖子 TOP5

对软件工程的一点看法--希望能够抛砖引玉　- <2744b> miaolin 2001/11/22 10:41 (627 次点击)

一个 java 实现上的具体问题　- <351b> joy_wind 2001/11/09 12:30 (246 次点击)

什么是 OO 方法，为什么要用 OO 方法，怎么用 OO 方法？？？　- <473b> mouri 2001/11/05 17:52 (164 次点击)

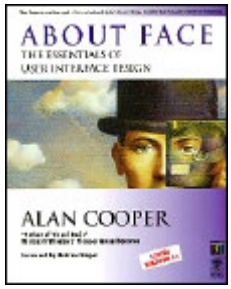
CMM vs Microsoft　- <767b> axial 2001/11/04 23:24 (122 次点击)

一个纯 UML 问题　- <636b> sealw 2001/10/31 16:49 (369 次点击)

消息：UMLChina 第十二期专家交流

北京时间 2001 年 12 月 15 日（星期六）上午 10:00-12:00

嘉宾：Alan Cooper，软件交互设计的先知和传播者，客户咨询公司 Cooper Interaction Design 的创始人和总裁。第一个可视化开发工具 Visual Basic 的发明者。交流重点：软件可用性，交互设计，其它...



要更多了解 Alan Cooper 的思想，请看以下资料：（1）《非程序员》第七期的第一篇文章“交互设计之父 Alan Cooper 访谈”；（2）去书店买他的著作：《软件创新之路--冲破高技术营造的牢笼》

详情请见>>

Ivar Jacobson 访谈

Adriano Comai 著，Tang Xiaoming 译

人物简介：Ivar Jacobson是世界著名的软件开发方法学家之一。他是面向对象软件工程(OOSE)和Use Case的发明人，与Rational公司的Grady Booch、Jim Rambaugh共同创建了一种可视化地说明、建造软件系统的工业标准语言--统一建模语言(UML)，UML在1997年被OMG（这个组织还制定了CORBA标准）正式确定为国际标准。他负责Rational公司的商业和系统工程领域的合作和产品战略。

Adriano Comai是一位意大利的方法学家。这次采访发表在http：//www.analisi-disegno.com和意大利杂志“ZeroUno”1999年的十月号上。

[Adriano Comai]（以下简称Adriano）：Jacobson先生：你是众所周知的“用例”概念的发明人，这个概念是怎么来的？是什么动力使你对这个想法做进一步的研究呢？

[Ivar Jacobson](以下简称Ivar)：它的发展已经有许多年了。首先，在我早期在电信领域工作的时候，有一种“通信案例”，通信案例就象用例，事实上是一次电话呼叫。有各种各样的电话呼叫，有各种各样的通信案例。这是我在那儿学到的某些东西。那时候除了电话呼叫以外没有其他案例。在早期，我的意思是说在1967年，我们有一个和用例相似的术语，代表和用例相同的意思，我们把它叫做“功能”。一个功能贯穿整个系统。一种电话呼叫是一种功能，但是功能更抽象一些。 术语事实上没有一个很好的定义。我们当时用的“功能驱动的开发”，现在被叫做“用例驱动的开发”。我们识别出功能，然后设计这些功能，就象如今我们使用用例一样。因此这些观念是非常古老的。我在1986年意识到用例的概念。当我发现这个概念的时候，我意识到我找到了能够解决许多问题的东西，因为我可以在任何一种系统，系统所作的任何一件事情上使用这个概念。它对我创造一种系统的方法学很有帮助。

[Adriano]：用例概念就象一个过滤器，可以区分用户相关的功能和系统内部的功能…。

[Ivar]：是的，它去掉了许多不是用例的功能。它更加具体。一个功能可以是任何东西，那是问题所在。用例不可以是任何东西。

[Adriano]：在软件工程文献中，用例概念的祖先是什么？它根源于哪种概念？

[Ivar]：我不知道。事实上，我没有那样的概念。我想最接近的东西是通信案例这个想法。但是我必须指出一点：我因为用例而出名，这的确是事实，但是1967年我们有基于组件的开发方式的时候，用例还没有诞生。因此基于组件的开发是我一生一直在努力的东西。另外一个是体系结构，我的意思是指真的先

辨别出一个体系结构---在做任何事以前。我们在1968年讨论了软件的体系结构。当我们跑到客户那里去的时候我们给他们看整个软件的体系结构。我记得他们从来没有听说过诸如此类的东西。他们教授硬件体系结构，但是他们从没听说过软件体系结构。

[Adriana]：用例概念今天看起来几乎是非常明显的，就象是常识一样…。

[Ivar]：我想是的。

[Adriano]：但是如此快速而广泛地被其他方法学家所接受，这的确是一个奇迹。为什么阻力会如此之少呢？

[Ivar]：大部分方法学家都来到了对象世界，竞争非常激烈。但是用例却没有和他们发生竞争，它解决每一个人都有的问题。甚至“场景”（scenario）这个概念也牵涉到系统的内部和内部的相互作用，而且没有被真正地详细说明。我做得太迟的事是发表。如果你去看我1987年关于OOPSLA的论文，这些东西都存在，但是在1990-1991年，我可以把我的《Objectory》卖$25,000一份。所以我为什么要跑到Addison-Wesley或其他出版商那儿，然后卖$3一份，即使能卖得更多？现在我认为我应该做得稍微有些不同，不过这也很难说，你必须在一个恰当的时间（去做）。所以我想其它书籍帮了我们，因为它们都有大问题，那就是如何从需求开始进行面向对象的分析和设计。但是我拒绝这样的观点：事实上他们出版了书，所以他们是第一个有这些思想的。这些并不是一些新思想。

[Adriano]：用例规格主要是文本方式的(虽然也可以用图来补充)。以前的方法(比如结构化分析,信息工程等)建议使用图作为"公共语言"以便在客户和开发者之间达成一致。这种文本角色的复兴背后有什么含义吗?

[Ivar]：现实中,今天的软件客户不想再看图表了。用例图表是如此的直观以致于任何人都能读懂。文本使人们不再需要学习一种特殊语言。

我们可以用活动图来表示用例,它很好,但是有两个问题。首先,它们很快变得过于详细,当细节展开时你不能确信它们是否更容易理解--即使毫无疑问在某些点上你需要更多的细节。但是我们想最好的方法是用分析模型的术语来做,其中你可以用对象之间的合作来描述每一个用例,而不是在细化用例的时候丝毫不涉及对象。你可以使用活动图,但是活动图可以是非常抽象的,因而难于理解。

你确实需要了解应该做什么,需要去了解那些活动。因此我在用例中引入形式（formalism）时是非常谨慎的。我想在你分析时你会得到好得多的形式，好得多的语言来表达细节。因为你(这时)谈到了对象。

[Adriano]：也许活动图不象文本一样能传达那么多的信息…

[Ivar]：是的,这很现实,并没有什么可奇怪的。在某些情况下使用活动图是好的。但是我想给一个警告：

当你有合适的语言来表达细节时你最好将其细节化。我想在分析时我们讨论对象,对象之间的合作,即使这些对象是概念上的,而不是实在的,可以实现的对象。它们也比活动更具体,也更容易理解。

[Adriano]：那么自然语言的歧义性呢?

[Ivar]：是的,的确有歧义性。但是我想这有一个权衡…语言容易理解，只要用英语就可以了。另一方面,当我们碰到有许多交互的、困难的用例时,你也许需要走得更远。但是把分析模型看作是需求的一部分会更好。在新的《统一过程》书中,我朝那个方向走了一小步,我把分析看作是需求的一部分。我们在分析中得到的东西之一就是我们希望在设计和实现中看到的结构。所以通过分析我们创造了一些对体系结构的需求。

[Adriano]：在你的方法中,用例扮演了双重角色。第一,它们被用于发现和确认来自用户和使用者的需求。第二,它们驱动整个系统的开发。是不是某个角色比另外一个更重要呢?

[Ivar]：不,当然不是。但许多方法学家和软件开发者在技术上是非常内向的，如果用例概念在描述交互作用,帮助定义合作方面不是那么好的话,他们不会采纳它。因此对软件开发者来说它就象一条很好的营销论题：如果它们对设计没有影响,如果它们没有驱动开发的话,它们不会象现在一样被广泛地接受。对于我,不管怎么说,两个方面是一样重要的。这是一个发现需求,在某些图表中捕捉需求的很好的方法,不用深入系统内部。它们被用于捕捉和识别场景,描述相互之间的关系。

[Adriano]：在你的书中,你谈到了把需求的特性列表作为导出用例的起点。

[Ivar]：特性列表是将被转换成用例的一些东西，而文档则描述用例。所以当你把要求的特性转换到用例时，特性列表会增长，会变短。

[Adriano]：几年以前,你把用例概念和其他对象思想应用到业务流程再工程领域。你的建议被非IT界的人接受的程度如何?用例在业务工程领域是否象在IT领域一样被频繁使用呢?

[Ivar]：不,没有。这有好几个原因。Rational公司选择了软件作为工作重点。即使我们完全意识到业务工程的重要性。然而,我们知道人们在业务工程中使用的工具可以很容易用软件工程中的工具术语来描述。如果你用ROSE来进行可视化建模，你可以扩展它来用作业务工程，而不是反过来。我们仍然需要软件可视化建模的好工具。我们已经在业务工程上工作了一段时间。但是是局部地做，在斯堪的纳维亚。我们在瑞典有一个Rational的服务包，叫做Rational业务工程，有一个ROSE的特殊化版本和详细的流程描述。软件工程的客户基础更广泛。想要进行业务建模的人通常理解软件，知道他们需要得更多。来自业务领域的人，比如Hammer，则不会对建模想得那么多，这真让人非常沮丧。所以从业务建模来的人非常少，大部分的人来自软件领域。这方面的业务量非常少，但有些客户有几百张Ｒｏｓｅ 业务工程版的许可证，比如电

信公司和瑞典的养老金系统。

［Anriano］：你有没有就你的业务建模建议联系过其他人，比如Hammer和Champy?

[Ivar]：没有，当然我读过他们的书。我们已经做了多年的业务建模，但是当我读他们的书时我说："哦，这儿有个家伙提出了一个问题，然后给出了解决方案的骨架和轮廓，但是没有对它建模，不建模你就没有理解它们。"不管怎么说，我感到Hammer的工作和我们的工作的联系是非常紧密的。另一个重要的想法是一对一的市场营销。一个在Objectory中为我工作的人现在正在这个方面做工作。她认为我们的方法是完美的。这是我将来想要做进一步工作的领域。我经常是为长远的目标比如五年后会发生的事情而工作，最近我将在两个领域工作。其中一个是业务工程，关于它是如何被新世界，互联网世界所影响。另外一件是WEB环境下的软件开发，WEB应用。即使WEB改变了一切，从根本上改变了商业上的每一件事，我们为WEB开发软件的方式并没有变得太多。在本质上是相同的，不同的只有一件事，那就是用户界面。用户界面的设计是非常重要的。

[Adriano]：我在你最近的一本书中看到你引用了Larry Constantine的一些话，你喜欢他的方法吗？

[Ivar]：非常喜欢，他的最近的一本书非常好。唯一的问题是他没有用UML而是用他自己的标记，那种标记比较薄弱，而且没有很好的定义。对什么是模型他有不同的看法。但那本书中有不少好东西。我非常喜欢，那是3年来在软件领域我看到过的最好的书。

[Adriano]：劝说IT非IT界的人士把业务建模作为新项目或改进现有系统的起点是不是更难呢？

[Ivar]：问题是我们没有时间，推向市场的时间是今天…。你尽快搞出点东西来，这比说这是一个好东西要重要得多。那意味着这些方法必须紧密集成在一起。IT界的人知道做业务建模需要6个月，12个月，而当他们开始建造系统的时候，业务已经发生了变化。我们的方法的独特之处在于业务建模是流程的一部分，所以如果你有6个月的时间来开发软件，你就有6个月的时间来业务建模。我能理解人们为什么对业务建模犹豫不决：如果我们不认为质量是非常重要而要千方百计去做到，那么无论用何种结构化的方法来开发软件都会存在问题。但是通过迭代式开发方法我们依据计划得到了一些东西，我想那会帮助人们理解持续不断，从头至尾进行业务建模的重要性。

[Adriano]：UML是集体性的创造，统一过程也是。但在后者中，你的贡献更清晰，更明显，统一化流程更根源于Objectory / OOSE而不是BOOCH方法或OMT方法。这是不是反映了朋友之间的某种"分工"？

[Ivar]：我不认为我们在目标上进行了分工。某些人是百事通，但很难有哪个领域我们三个中的任何一个认为一点经验都没有。公正地说，我认为没有分工。我们开发RUP的时候是以Objectory方法为基础的，这是事实，我们是从那儿开始的。当然，你没法仅仅从对象模型开始。并没有一条简单的途径能从OMT

或BOOCH方法过渡到我们在OBJECTORY方法中所做的东西。而反过来则要容易得多。想一想用例，然后你可以设计对象，类，子系统。

[Adriano]：Booch 和Rumbaugh来从软件出发,而你是从客户和使用者那方面出发…

[Ivar]：是的,但是事情还有另外一个方面。组件是我们不得不开始着手的东西。我实际上是从组件开始着手的。1967年我在Ericsson引进这个方法的时候,开发人员的主要反对意见是我们开发的这些组件不容易和"功能",或者"用例"相关联。如果从用例角度看,用例使用了许多组件：这就是反对意见,他们想的是"一种功能,一个模块"。

而我却说,好吧,那的确不太妙。大部分功能或用例应该用一个组件来实现;但是其他组件也会在其中扮演一个角色。那就是反对意见之一。我说,正是这个反对意见我要把它转化成积极的东西：这真正是你们想要的东西,你们需要复杂性,因为事情本来就是这样的。虽然在外部讨论用例,但在内部用例贯穿了这些组件。

拥有对象和组件的子系统,并不关心是谁使用了它们,这只是个小问题。麻烦的是实现用例,管理子系统之间的相互依赖,这要困难得多。对象的事是一个小得多的问题,是一个亚问题。

不,我不认为在分工上有任何刻意的决定。我们认为UML是一个大任务。我们必须一起工作来完成它。现在我们做不同的事,我们不认为在任何事情上都需要一起工作。

[Adriano]：你希望统一过程成功吗?你希望它对IT工业的影响能和UML类似吗?

[Ivar]：是的,绝对是的。我们有非常好的理由相信这一点。如今我们正朝着公司进军,我们的目标是到达那里。我们不认为很容易使统一过程成为一个标准,这会是一件非常困难的工作,会有许多反对意见。所以我们宁愿小步前进。与其强迫别人通过一个标准,不如让人们自己去说服自己。我想每一个看过RUP的人都会被说服,相信这就是他们想要的方法--只要他们开始去看。甚至没有任何东西能接近它。许多人说有,可是他们有的是什么?他们有可以和我的书相比的东西,但是他们没有任何能和我们的流程相比的东西。如果你从实例,深度,经验等各方面,如果你拿…方法A,或方法B有多久历史?我们知道它可以用于大型项目吗?我们知道我们的可以。这的确是非常不同的。

[Adriano]：有多少Objectory方法中的东西留在了统一过程中?

[Ivar]：如果你从基本概念上看,在Objectory方法中我们基本上只涵盖了需求分析,系统分析和设计。如果你从这些方面看,早先Objectory中的东西仍然有。但已经有许多新东西加了进去。我们很少是有关于实现,有关于测试的,没有配置管理,没有版本控制,没有项目管理。迭代式开发是我们首要建议的东西,但它通过流程得到了加强。我们并没有真的说出各次迭代的不同之处。所以我认为核心思想还在,但已经加进了很多其他东西。RUP是一个团队工作,有许多人参加进来。而Objectory 过程实现的首要是我的想法,和我的工作。

但基于我们的资源那么少,也应该说是很多(的工作了)。

[Adriano]：你提出的每一次迭代就象一个小的瀑布方法…

[Ivar]：是的。我们把它看作是一个小瀑布方法,但其中我们还有许多并行。在瀑布方法中开发子系统的人是在并行地开发子系统。相对独立地使用用例的开发人员互相交流来重用组件以避免重复发明。在一次迭代中它们并行开发子系统。但那仍然是瀑布模型。因为你始终是从需求分析开始,然后是系统分析,然后是设计活动。

[Adriano]：你来自瑞典,你是否感到在系统和软件工程中有欧洲的特异性?相对美国来说是否更关心,更关注于组织方面的事?

[Ivar]：我没能找到任何系统性的差别,因为美国人在开发软件时非常喜欢从业务,从理解业务开始着手。欧洲也一样。没有什么系统性的差别。更有趣的是…在欧洲,许多研发是在抽象比较多,实在的,物理的比较少的领域里。但是我必须说,无论是美国还是欧洲,许多研发已经完成,取得了有用的成果,同时也有许多工作没有任何结果。

[Adriano]：现在"统一化"努力中的最伟大部分已经完成。你是打算去休息，还是利用它,或者你是否正转向其他感兴趣的领域?下一个是什么?

[Ivar]：我身上的一部分说：我想要前进,找找看有什么要做的,要去创造一个更好的世界,我们有许多事要做。某种程度上,uml是一个标准,那很好。但这并不意味着这是些新想法。我们只不过是把它们固化下来而已。在过去几年中,我想我没有真的做了一些新东西。我推进了采纳过程而不是创造过程,现在我身上的一部分想要迈出下一步。统一过程之后是什么?uml之后是什么?我想仍然会有一次进展,但不是一次革命。但在软件领域中仍需要采取一些重要的步骤,以便赶上我们在2020年开发系统时所需要的特别高质量的水平,或其他类似的东西。这些要比我们今天能想到的系统要大得多,要复杂得多。我们需要有能力去开发这些系统。就操作系统,编程语言,和编程语言集成的UML等方面,我们需要一个更好的基础架构。也许uml的一部分会变成带有动作语言语义的一种编程语言。这是我经常会想到的。

另外一个是利用业务工程。有一些的确有趣的工作可以做。RUP已经非常适合web的开发。如今许多开发WEB站点的公司已经在使用RUP。稍微做一点特殊化,它就可以满足他们的特殊目的,但还是同一个流程。我希望看到我们在RUP所做的扩展和正确决定能够获得所需的模型改进。这些改变非常清楚地、毫无疑问地使它变成适合WEB站点应用设计的流程。

我也打算在年底以前写一本我的"The Object Advantage"（《对象的优点》）的修订版。互联网,一些其他想法,比如一对一的营销，将会对这个修订版有巨大的影响。我们需要让这本书更针对商务人士,而不只是

针对软件人员。我们将会展示在业务场景下如何使用它,而不只是在软件场景中。基本思想早就存在了,而且工作得非常好,客户很满意。但现在我们需要带它跨越过IT界的栅栏,解决人们接受技术符号时存在的问题。活动图对业务建模非常有用。

查看评论

高焕堂：恢复中华民族设计自己产品的信心

2001年6月22日晚18:30--21:30，几十位海峡两岸的软件设计人员共聚UMLCHINA讨论组的聊天室，畅谈面向对象技术和软件设计。当晚受邀嘉宾高焕堂先生，是台湾的资深面向对象专家，1995年创办“物件导向杂志”和MISOO物件教室，在台湾普及面向对象方法，育人无数。以下是交流实录。

umlsz：有没有好的办法维护已有的软件？

高焕堂：用 Object把旧的软件包装起来！

tomxh：能说说您研究OO的历程中最深刻的领悟吗？

高焕堂：艺术、文化与工程结合！

morenew：请问OO软件工程每一个阶段如何平滑过渡,是不是主要看人的经验

高焕堂：是的，最好采用iteration的方式！如果注重软体里面的设计部分，那么，这是师傅级的事，当然跟经验有关！

flyfuture：艺术、文化与工程结合的精髓是什么呢？

高焕堂：例如，铅字版的设计与物件工程的观念是一致的。

oceandeep007：请问：如何在一个管理不完善，技术人员水平参差不齐的软件企业实现OO过程？

高焕堂：应该加强软体团队的教练(mentor)

umlsz：请问高先生，一个软件项目最多可以有多少人同时参与开发？

高焕堂：如果重视软体结构，然后制订好的interface, 那么人员是多多欲善的。

windy.j：高先生，请问目前在台湾OO技术的应用状况如何？

高焕堂：还差先进国家很远！

adax：请教高先生，我的开发团队从6个人，到现在在迅速的膨胀，但是问题很多，效率下降的厉害

高焕堂：我认为是『共识』的问题！，要由mentor强势主导软体的设计风格和结构！

superyxb：请问高先生对软件开发过程中的变化怎么看？

高焕堂：过程必须随着Project的进展而持续Review, 并且调整！

czhp：请问高先生，您认为正常情况下在三个月可以完成的项目（非OO），有必要用OO来做吗？

高焕堂：如果每一个人的OO思维都足够的话，当然没问题！

wanttoknow：请问高先生对clearcase熟悉吗？您认为软件配置管理工具那个比较实用？

高焕堂对大家说：我想人的OO观点(perspective)比较重要于clearcase，因为徒法不足以自行。

superyxb：呵呵，看来大家关心软件工程问题甚于OO设计。

高焕堂：对的，所以，我们认为我们必须恢复中华民族设计自己产品的信心。

adax、tomxh：我尝试用各种软工的方法，但是好象见效不快啊？我在团队里面如何达到"共识"？是否要严格按规范来做，关键位置的人必须要扎实，最好用CMM来管理？

高焕堂：CMM 比较适合管理工程里的Coding，而不是工程里头的设计。

wanttoknow：你的意思是掌握OO的思想是最重要的？可这些必须借助于工具来实现啊！

高焕堂：答案是Yes，但是工具是手段，所以必须不断的调整，选择最佳的工具，但是，选择工具必须靠人脑。

linkhand：OO的要旨是所有事物都是对象，对象和对象之间象人一样相互作用。高先生，你说对吗？

高焕堂：不是所有的东西都是物件，那是决定在你的OO的思维。

ccanj：开发软件两个重要的概念应该是设计和管理，您认为呢？

高焕堂：答案是对的。但是，管理必须支持设计，也支持Marketing，也支持制造。

freehorses：在用OO思想来分析项目时，怎样去发现object?

高焕堂：要先分析企业里的重要Concepts，再由concepts mapping到Object。

ccanj：也就是说，管理不止包括对软件开发的管理，还包括市场的运作。我现在不清楚在软件管理中哪些是比较重要的？

高焕堂：看你软体的marketing是指整个系统的销售还是指component的销售。因为卖component跟卖整个系统的marketing是不同的。

freehorses：以前看过一些书，说先从用户需求中找名词，您认为这样有效吗？

高焕堂：用户的需求分为结构性的需求跟功能性的需求，物件大部分是由功能性需求而得出来的，例如，银行的SavingAccount(物件)跟功能性需求是无关的。

ccanj：我是指在作整个MIS系统的时候，进行软件开发的管理，并不是component。

高焕堂：我想，开发component，而且，卖component，是需要比较多OO的思维。

ccanj：OO思想主要使用在设计的阶段，在由设计到实现的映射过程中，需要管理手段。我是指该阶段的软件开发管理。

高焕堂：从设计到实现的过程，也需要Design，一般称为Detail Design。

freehorses：系统内部的子系统能不能互为actor?

高焕堂：不一定，有时候应该视为Component。

freehorses：如果视为Compenent的话，是不是要用依赖关系来联系他们？

高焕堂：yes.

freehorses：你认为用UML为系统软件建模作用明显吗？

高焕堂：答案是yes，它是很好的沟通工具。

cancan：就是我想请教您，设计模式的耦合程度由哪些方面判定的呢？

高焕堂：看一个component是不是用到另一个component的implementation，还是只用到他的interface。

ccanj：是不是说，milestone、测试、预期结果的评定起着非常重要的角色，更重要的是对各种无法预料的情况及时处理。

高焕堂：我想这些都必须配合Project管理上的iterative模式，才能解决需求变动的问题。

sslxml：关于系统设计的量化问题，如何将一个大的系统的设计分工到不同的设计人员上，这些人员之间如何合作，设计思想如何得到贯彻，如何能够保证设计思想的一致性？谢谢。

高焕堂：设计系统也有非量化的问题，也就是美感的问题，量化问题可以靠Tool，美感问题大概只能够靠Pattern.

glkk：请您谈一下怎样选择对象实例化机制好不好？

高焕堂：我想XML或是.NET的dataset是一个好的机制.

freehorses：怎样合理使用use case的include和extend关系？

高焕堂：我想更合理的是：一开始，不必使用include或extend, 因为增加了很多初学者的困扰. 初学者常常分不清Use Case与SubSystem两者之间的关系，常把include看成模组之间的呼叫关系。

fataltomato：高先生您好，我在MIS开发中曾经尝试过直接使用UML来建模，可是效果还不如使用DFD等流图来得方便而且更适用于现在的RAD开发工具，请问象MIS这类开发工作用UML来做OO开发值得吗？是不是OO在系统软件更适用？

高焕堂：如果你想开发component然后采组装成application system的话，那只能用OO与UML了，传统DFD没有足够的information来设计理想的软体结构及interface.

cber：我想问一个问题，如果开发一个没有用户界面（或者用户与系统的交互很少）的复杂系统，如何实作出UC View

高焕堂：我想，一个系统一定有User吧？

freehorses：一个Use Case是不是可以在多个SubSystem中重用？

高焕堂：Use Case通常表示一个流程，但是，流程是善变的，没有太多reuse的价值，所以才要设计component，component才是reuse的重点。

gemlei：XP是不是对QA的要求很高？

高焕堂：对于大型Project的QA, XP就不太行了。

sslxml：高焕堂先生您好，谢谢您的回答。但是如何保证合理分工合作，我的意思是一个大的系统，一方面要保证进度，另一方面大的系统往往都是复杂的系统，这些都是无法通过个人或二、三人解决的，这就需要多人合作完成大的设计，但是多个人之间的设计思路与总设计师的设计思路会产生不同程度的偏差，其它人的设计往往局限于他所设计的一部分，他们怎么能够用到其他设计人员的设计成果，并且，分散的设计如何解决设计整体性的问题？同样的问题也会出现在设计实现的阶段。谢谢。

高焕堂：所以，Booch 才说，Architecture First! and I&I.

cancan：高先生，请教一下从功能中提取物件有哪些依据呢？还是要凭感觉

高焕堂：我想物件最佳的来源不是来自功能，所以，可以先建构class diagram，然后才建构use case

diagram，两种diagram是coevolution的，才是最理想的。

freehorses：维护员启动应用算不算一个Use Case?

高焕堂：算！

freehorses：如果算的话，是不是其他Use Case都要来include这个Use Case？

高焕堂：这些Use Case本身应该都是独立的，是由Client来将他们组合成条更大的流程。

freehorses：能不能谈谈SubSystem与Component的联系？

高焕堂：Subsystem是依照功能需求的切分而得的，所以他的名称大部分是动词，而component大部分是来自对应business的concepts，所以大部分是名词，例如，客户管理是subsystem,客户是component

freehorses对高焕堂说：一个人学UML是不是效果不是很明显？

高焕堂：对，效果不会很明显，因为没有达到UML能够支持大型项目的分散式开发的合作上。

umlchina：大陆软件应走何道路

高焕堂：我想，大陆的大西部应该走印度&CMM 路线。但是，在沿海地区，应该走向软体设计，然后，双方互相合作，互相搭配。

sslxml：如果用尽量简短、精炼的语言表达UML的思想，该如何表达呢？UML思想的本质是什么呢？

高焕堂：UML就像音乐里头的五线谱。

glkk：在台湾OO思想在软件开发中实施的情况怎么样？

高焕堂：还没有进入理想的境界，还有待努力。

morene：Component在Rose里面可以看成一个类吗？

高焕堂：过去，microsoft把component定义为进出memory的单位，所以，常会含有好几个class，这是不理想的，所以，现在，microsoft已经逐渐把class视为一个component，而 class的集合应视为package或namespace比较恰当。

freehorses：UML是不是和RUP结合起来效果更佳？

高焕堂：我想，建构四合院跟建构洋房的process是不一样的，所以，RUP必须修正来配合本地文化及建构的房子格式而做新的调整才能适用。

sslxml：能不能这样理解，也就是说，会了五线谱只能有了一个理解别人音乐的能力，但是他对提升音乐本身的质量的帮助不会很大？最重要的还是对音乐本质的感觉？

高焕堂：所以，重要的是，在面对需求的时候，心中必须持有OO perspective，然后，才用UML表达出来成为可以共享的model。

gemlei：UML更像是rational公司赚钱的工具，他们所提出的理论，似乎只有在rational的集成软件中才能看到成功的希望，大家说呢？

高焕堂：我想，我们应该认为UML只是个Tool，那么，Rose更是Tool中的Tool，所以，我们的设计思维和美学，才是根本！

cancan：请问CORBA与Windows的DNA，这两种OO体系结构的最根本的不同在哪里呢？分别适合什么样的SYSTEM

高焕堂：CORBA原来只规范系统的Service，但没有规范系统的结构，也就是没有规范component的介面，而DNA就改善了这个缺点，最近，Corba的新版也改善了。

freehorses：也就是说UML是通用的，一般情况下可以不加以修改，但过程结合根据公司的具体情况？

高焕堂：我们必须依据不同的软体结构及schedule而调整开发的过程，而且，在执行的过程中，Leader还必须视情况再调整过程，不过，这点比较难。

sslxml：在面对需求的时候，我心中是有OO概念，但是问题是，如果使用UML表达出来（是的需求表达的可能更精确），这就要求其他的设计人员也要有UML的概念（否则别人怎么能够理解），也就要求其他的协作人员有更强的知识结构，这在现实中往往是不易实现的（考虑到成本、技能、知识面等方面）。这些如何解决？

高焕堂：对的，这在世界各国包括日本也都遇到这个难题，不过，尽力而为！

gemlei：软件开发的出发点是基于设计思维和美学，还是客户满意度？是基于CMM还是客户主观的认可？我指对于大多数商业企业。

高焕堂：我想，做一件旗袍，是否顾客会满意，做旗袍的师傅的经验和美感是很重要的。

cancan：我能体会设计思维和美学的重要，但以前是用传统方法设计软件的，这两年转向OO，但总觉得带有结构化的阴影，回头看自己设计的模式时总觉得不如一些例子的感觉好，您能否提供些建议使我能更好的转换习惯吗？

高焕堂：我们尽量从日常生活中去体会OO的思维，到底OO可以看成是各行各业设计思维的精华，

并不是软体才有，例如，古代老子设计苯箕，他说里头要挖空才有用，这个挖空的动作就是在建构一个interface，未来，才能放其他东西。

freehorses：是不是坚持用OO的思想去分析项目，到一定的时间就会好些

高焕堂：答案是 Yes。

cancan：CORBA通过stub and skel ,通过系列化和反系列化在网络间是传递整个对象，但DNA是产生COM的调用，网络间传递的是请求和结果，您觉得两种体系的优劣如何？

高焕堂：DNA 目前也在调整中，两种逐渐整合中。

gemlei：再有经验的做旗袍的师傅摆脱不了一个时代制约，如果他能将有限的时间投入到无限的为人民服务中去，是否他是一个高明的商人？

高焕堂：有经验的师傅可以带领一个团队让团员多提升OO的思维，但是，如何培养有经验的师傅，才是重点，就像奥运能不能得金牌，决定在教练的水平。

cancan：我曾经试着对生活中的小事也用OO的思想去解析，但用程序去实现的话，总是受开发工具的限制，如为了快速开发用VB和RDS，但RDS不支持属性，VB不支持继承，好不容易构造的模式就不得不被CANCEL，很苦恼，您有过类似的感受吗？

高焕堂：答案是有，但是，tool的发展总是要时间，VB7已经改善许多了，然后在J2EE方面，也一直在进步中，所以，苦恼在逐渐消失中。

cber：我现在倒是越来越觉得相对于Generic的思维，OO可能有点不是太好了。如DP，我们就可以把它们认为是一些Generic的解决问题的方法，而对象只是尽力去符合它们对于对象的需求

高焕堂：这就看我们的软件业是否要建构关键性的component，在国际上，component占有重要的角色，如果要的话，那么OO 思维是建构component的必要方向。

sslxml：请问您对framework在实际开发中的作用如何理解？

高焕堂：一个项目开发完成之后，抽象出可reuse的framework, 可加速下一个同domain的项目的开发。

glkk：在开发团队中，XP和RUP哪一个更容易实现?

高焕堂：大项目采用RUP比较恰当，XP小项目适合，但别忘了，都必须做调整，因为，任何在国外可执行的process，在中国人的手下常会变质。

gigix：请问您对《设计模式》这本书怎么看？

高焕堂：Design Pattern 最好能视为是指引我们去思考，而不是告诉我们如何去做，而且，要阅读Design Pattern这本书，我想，需要有很强的OO思维，例如，对于polymorphism如果不了解的话，就很难活用pattern了。

freehorses：能不能谈谈设计模式在软件设计中的重要性？

高焕堂：我想那是很重要的，就像孙子兵法对一位将军是很重要的。追求美感与精致的心情才是接受设计新思维的基础，如果，太过现实，追求Quick & Dirty，那么，软件工业可能永远是洋人的天下。

freehorses：你认为什么样的项目算是大型的、复杂的项目？

高焕堂：大型的一般是指整个Enterprise或跨Enterprise的系统，例如海峡两岸的系统整合。复杂的系统有时候是大型的系统，有时候是需求变化快速的系统。

cancan：所以我觉得OO，reuse并不难，难的是设计能适应需求快速变化的模式

高焕堂：所以，RUP过程的精神不是在控制，而是在Iteration，让开发团队的每一个人能够跟着User而共舞。

glkk：洋人和国人最根本的区别在哪里？

高焕堂：就软件工业来讲，洋人研究Tool，也卖Tool，但是，却不实际使用Tool，去制造产品。但是我们却用心在寻找Tool，勤练Tool，互相比较谁用得好，好象大家争着谁是比较好的司机，而不是比较谁做出更好的车子。

glkk：司机和车子的比喻很深刻，是不是说国人更喜欢研究别人生产的tool来开发，而洋人本身不用？

高焕堂：我认为，中国在秦代像兵马俑里头就有很杰出的设计，而且，能跟制造密切结合，但是，后来，国人就失去设计方面的美感追求。所以，我认为应该恢复国人设计与美感的追求和信心。才能摆脱只用Tool的陷阱。像建筑业还有土木业，设计是工程里头重要的一部份，而且是高报酬的，所以，不能把工程窄化为只是对工作的控管，而是要鼓励团队追求美好永恒的设计。我们应该朝向制造高价值的component，就像美国intel，设计CPU，而不是以组装一部PC为满足。

glkk：是啊，设计美感和money之间没有矛盾的。

高焕堂：像巴黎的服装业，因美感而创造高的利益以及高的生活水平。所以，美感的设计才能够确保工程制造之后的高附加价值。

cancan：高先生：请问您是为了学一个东西而去学东西，还是为了解决问题才去学东西的呢！我能感觉出您的实在，希望您能为大陆的同胞指明方向

高焕堂：我不敢说指明方向，但是我认为一个国家不能只有饮茶的人，而应该也有人去泡茶，去种茶，而且，每一个环节互相搭配。

stoic：高先生,请问追求完美的设计，与企业追求利润最大化是否是矛盾的，如何才能和谐统一？

高焕堂：追求完美的设计是属于虚的，但虚的能转换成为实的利润，需要一定的时间cycle, 所以追求的是长期的利益，而非短期，所以，与Qucik & Dirty的文化是违背的。

cancan：您说得很好，我觉得日本人管理的出发点是服从，美国人管理的出发点是结果，所以他们都做得好，但大陆呢，服从？没有谁真正按RUP、CMM的指导去做；结果？大家心知肚明，我希望思维有个源头。希望您能赐教

高焕堂：我想应该有人去将RUP与CMM转化为适合国人的文化和习惯。例如，上海与大西部之间如何分工合作，才能把 Design 留下来。

freehorses：你认为Component与Class有什么区别？

高焕堂：就设计角度来讲，component与class 应该是没区别，但对系统层面的人来说，component强调的是interface, 而class是implement component的手段之一，非OO的library也能support interface, 所以他implement的该component。

cancan：是的，我在公司就根据RUP总结自己应该的做法，但许多国人考虑问题的出发点并不实在，所以会碰到的困难实在是难以解决，我在台资的公司呆过，因为有方向的引导所以可以做得好，但大陆不是的，希望您能理解。

高焕堂：我指的是长期的努力方向，在目前大部分公司的环境都有些困难。我有个建议，如果想成为好的设计师，最好不要兼做 Programmer，就像建筑师不会去挑砖头、建房子，这样才能培养理想的分工与合作，如此，RUP的精神才能够发挥，但是，我知道，很多人不Coding，对Design, 就没信心。

sslxml：高先生：您认为一般程序员需要具备的素质是什么？如何设计可以让程序员能够很好的理解设计师的思想，达到优秀编码的目的？编码规则重要吗？

高焕堂：我们比较主张Programmer的Leader必须对Design Pattern以及UML有相当的了解，当一般Programmer有问题时，Leader会像一个球队的教练，去替他解释说明并提升他的水平，自然就能够逐渐顺利写出高品质的程序了。

cber：但有时Design不如Code来的直观，碰到这种情况又怎么办呢？

高焕堂：对的，Coding常常带给自己对设计的信心，因为设计是虚的，Coding是实的，设计必须与Coding的结合，才能获取利益。但是，对Design的信心的提高，并不只是依赖Coding的实证，就像抽烟可以让你比较有精神，但是，并不是唯一的途径。

tomxh：您的意思是不Coding，直接做设计师，可以吗？？？

高焕堂：我认为设计师有些Coding的经验是有益的，但是在做设计的时候，最好不要自己去 Coding自己的设计，而是，让别人去Coding，这样才能培养在施工前能做出设计的能力。

查看评论

程序调试的智力游戏

David Burns 著，史彦军 译

对于软件开发人员来说，程序调试实际上就是一种生活。而对于职业程序员来说，“捉虫”是专门描述这个智力游戏的名称。以下是关于“捉虫”和修正的10个建议。

那是一个新项目中，在我的第一个任务中，我被分配修正一个明显可见的bug。这个产品作为一个数据显示的客户端，能够根据显示区域的大小按比例调整字体的大小。但是当它在特定的情况下运行在一些机器上的时候，这个字体将很快放大，直到一个字符填满了整个显示屏幕。

该软件的原来开发者已经在国外并根据预定的日期进行计划中的海外旅行，所以我只能靠自己了。根据显示区域的范围来计算字体大小的程序代码很容易理解，明显没有错误。我猜测（确切地说，以后才看到结果）一个浮点凑整问题是导致上述bug的根本原因，我花费了一天的时间来重写代码，消除上述可能性。然而不光彩地看到我对bug的修正在项目经理的计算机上运行失败。

事实上，问题出现在代码完全不同的另一个部分。字体的运算使用了一个值，叫PixelsPerInch。这个值不象我所想象的那样，在初始化的时候被建立并且保持常量，相反，这个值在显示区域每次变化大小的时候被重新计算，而不是我已经重写的处理WM_SIZE消息的代码中。相反的这个所谓的常量在处理WM_NCCALCSIZE消息的代码中被重新计算了。这样字体大小的运算在逻辑上反复的相互作用，导致一个浮点凑整的错误，并慢慢自动放大，最终导致这个可见的bug。这样，几分钟学习这个应用程序的整个窗体尺寸逻辑运算节省了我数小时的时间。

十个技术要点

如果你是一个软件开发人员，调试就是你生活的一部分。如果你作为一个职业程序员，就像我一样，你可能参加一个项目并专门“捉虫”。我不能告诉你这些bug长得什么样，它们可能导致一个设计错误、一个逻辑错误或者一些资源的错误处理。这篇文章及其要点可能不足以说明问题，所有你能得到的将是问题的一些简要的、可能错误的描述。实际上直到你修正错误以后，才能知道错误的原因，以及如何修正它。

更糟糕的是，没有许多关于程序调试的书籍，也没有某个方法或者工具能够完全修正这些错误。程序调试就像解决一个难题或者玩一个游戏，需要机敏、智慧和创新，根据我的经历，我发现了十个必要的技术来捕捉和纠正bug。

1. 自豪感

就像奥林匹克运动员用自我激励来增强他们的勇气一样，你可以大声的谈论程序正确表现的细节，并且习惯于这些和成功相联系的思想、想法和行动，通过在公司其他人面前表现的自豪感，你也建立了一个成功的社会形象。

如果你有机会，你就应该谈论你将如何修复一个非常具有挑战性的bug。一个项目经理曾经为使我面对一个困难的问题而道歉，但我只是微微一笑说：“解决棘手的bug是我天赋的一部分，我会为你认真解决这个问题。”

同样地，你应该为自己自豪(或者，如果你更喜欢用自我激励的话)。当你面对一个新问题，想象你将如何发现问题的错误并用无可挑剔的代码消除这个错误。

有时候把你的战绩写下来会更有用。如果你修正一个特别严重的bug，写一个备忘录给其它的程序员和项目管理人员以使他们得到启发。如果你正在修正一些简单的bug，在定期的项目进展报告上跟踪和报告这些bug。使你的有效工作引起大家的注意。

2. 放松和欣赏你的工作

bug修正是一个学习的过程。如果你在滞后的时间表、忧虑的心情和巨大的压力下进行工作的话，你就不会意识到你需要学习什么东西。在这种情况下，问题本身不能得到清楚的说明。下面有两条推论：不要抱怨你不得不进行工作的代码，也不要抱怨你不得不使用的工具。抱怨代码就像抱怨一座山脉，它对抱怨者的心态的影响比山脉对大地的影响更大。成为山脉的主宰者，并把你的想法集中于如何提供对代码质量有用的建议。关于工具，如果它们很难使用，学会如何更好地使用它们。如果它们对要做的工作起不了作用，请看第六点建议。

3. 了解你的工作环境

一个写新代码的开发者可能会在一个大项目的一个小部分中花费数月甚至数年的时间，并只能学习和运用技术和相关技巧的一小部分。虽然一个开发者能够凭狭窄知识面的高深知识成功开展工作，但是一个“捉虫”者需要关于他工作环境的广泛知识。你如何获得这些知识呢？在你工作时，无论何时一旦你遇到你不熟悉的情况，要努力搞明白。要熟悉你所工作的系统的在线文档。现在的操作系统和应用程序接口有成百上千知识点。你不必全部了解他们，但是你应该能够很快找到和理解他们。

此外，确信你编程环境的帮助和文档已经完全安装了，这样你就不必总要查询你的光盘了。读取关于你的领域任何值得关注的新闻组(对于C++编程人员来说新闻组 comp.lang.c++ 是比较适合的一个)，并且像关注官方来源信息一样关注非官方的信息。阅读每一份你订阅的Microsoft Systems Journal，以及一些独

立的出版物。

4. 不要害怕开始新的工作

如果你在不熟悉的项目中开展工作，可能这个bug对你来说有点晦涩艰深，在你开始工作之前要毫不犹豫地了解整个环境。首先查看初始化和实例化代码，以及类的声明和构造。要认真回顾来了解项目的全局结构体系。跟踪开始阶段的代码，检查相关联模块和对象是如何相互依附、初始化和消毁的。你会惊喜的看到这些回顾多么地有助于看到bug的本质，即使不是这样，你也不会浪费时间。这样的回顾使你了解你正在工作的代码的风格，让你学习一些你所不熟悉的技术以及这个项目所用的专门技术技巧。

5. 熟悉你所不熟悉的知识

原有知识和经验会阻碍学习新知识。在你开始修正任何bug之前，要有意识地去掉关于bug原因的预定假设。在面对项目经理以及其他编程人员的时候（他们确信自己知道问题出在哪里），这可能很困难。不要忽略他们的意见，但是，要意识到如果他们知道这个bug出在什么地方，他们不会需要你来帮忙的。

首先要精确地列举你知道和不知道的关于代码和bug的信息。曾经有这样的情况：我被要求在我曾经遇到的最晦涩和混乱的WIN16代码修复bug。这个软件据说被冻结和中止。这个项目组的每个程序员都确信bug的原因一定是因为某个逻辑错误。首先我放弃了逻辑错误是导致bug产生原因的想法，然后放弃了冻结和中止的想法。因为我也不知道哪一个想法是对的。

一个快速的测试表明软件没有中止，它只是不能显示一些用户可接近的控制。在为期两天的艰辛的分析、测试和消息跟踪以后，我得出如下结论：Windows的消息队列溢出，导致PostMessage函数调用失败。像在他之前的许多程序员已经做的那样，这个代码的作者已经假设PostMessage函数总是调用成功。通过回顾，问题的解决方法看起来极其简单。但是我不得不熟悉我所不熟悉的知识。

6. 使用允许你了解你所需要信息的工具

许多程序员使用现成的工具并接受这些工具所提供的信息的层次。与保持消极态度相反的是判断你需要了解什么信息来分析这个bug。它可能是网络消息、DLL参数调用和硬件中断。你可能需要一个高精度的定时信息或者对象复制的日志。缺省的调试器可能是很好用，你可能仍需要一个底层的工具来调试驱动程序。如果涉及到相互作用的组件，一个调试器可能仅仅面对接口，所以，你可能通过跟踪输出和日志文件来了解你所需要的信息，或者你可能需要自己来编写这个工具。

要清楚地了解已经存在的工具，花费一个小时来学习调试器的选项或配置一个分析器的输出将有利于节省大量的时间。这样也会节省你写自己工具的时间或有助于你修正你以前没有发现的bug.

确信你理解随处可见的各种信息。比如，是否编程语言支持跟踪调试、断言和异常？是否你有权接近

介质和高精度的定时信息？是否你能够在一个变量上设置断点，确定一个循环的第28次运行或者内存使用的某个程度？这些调试器的选项对你正在运行的程序有那些影响？

虽然正确的工具和方法是重要的，你首要的注意力仍然要集中在问题本身。如果你发现你有这样令人羡慕的之处，可以选择众多诱人的调试工具，要记住，尽管有厂家的承诺，可视化的调试器、运行时的错误检测器，API跟踪器和代码分析器也不能自动进行bug清除工作。确实有这样的情况，你能运行它们并可能会发现它们提供给你的信息不是你所需要的信息。产生软件设计的接口、写代码、追捕和修正bug是一个智力劳动。一个工具有助于你了解问题，但是解决问题却只能依靠你自己。

7. 要耐心

“捉虫”需要耐心，即使它需要经常承受在最后期限的巨大压力之下。虽然急于改动一些代码来看看它是否会工作的想法特别强烈，但是不要认输。不要改动任何代码，直到你已经明白问题的原因，从来不要因为进度表而试图改动代码。直到你知道问题的原因你才能提出解决方案，而直到你获悉程序运行方式你才能知道问题的原因。没有办法知道这个过程要花费多长时间。有些管理者觉得把一个期限写到计划表就能控制解决问题的进程，然后在时间到期以后开始发牢骚。处理这些管理者的期待是很困难的，但是如果你很大度并理解她目前所受的压力，也有助于理解管理者的做法。

8. 对问题的证据保持警觉

每个编程语言和编程环境都会有一个或多个共通的错误，对这些要积极了解。在C++中，它可能是成员变量没有初始化或者不恰当地释放资源。在Visual Basic中，它可能是不恰当地使用DIM声明和不正确的ON ERROR语句处理。如果你保持警觉，你将只需等待它们自己在合适的条件出现来发现这些没有报告的bug。良好的警觉性能够提供有价值的线索，这些线索可能和bug没有直接关系。有一次，当在Lotus Agenda（一个早期的个人信息管理系统）中跟踪检索bug的时候，我注意到一个关于分解的程序没有有效的实现。我向项目经理提到这个问题，但是他不感兴趣。这个分解过程是后台空闲循环处理的一部分，它的糟糕表现从来没有被用户注意到。两个星期以后，这个分解的过程因为某个不相关的原因被移到前台来运行，用户因此大声抱怨。既然我知道问题出在哪里，我就能够迅速的重写代码，通过一个二十倍的系数提高性能。抱怨停止了，这个项目经理的感激可想而知。

9. 知道哪些是正确的

当对糟糕的规格说明进行不理智的编程实现，灾难就邻近了。一个处于创业阶段的硅谷公司雇我修改一个安装程序，因为这个程序的卸载摧毁了测试者的硬盘。卸载程序规格强调所有与该应用程序相关联的文件都被删除。因为这样做的最容易的方法是删除该应用程序的目录树，初级程序员接受了这个任务，写代码首先获得当前目录路径(C:\ApplicationName\Binary) ，分解消除 \Binary\ ，然后删除得到的路径下的

所有文件(应该是C:\ApplicationName)。不幸的是，我们这个英雄的代码有时分解的太多，导致意外删除了所有目录 C:\ 下的内容。

我花费了一些时间来说服我的客户，主要的错误是规格说明的错误。一个恰当的卸载程序不应该删除任何非安装程序创建的文件。由应用程序创建的用户文件和更新的数据库文件应该从来不会自动删除，而应该留下来让用户手工删除。

像这样的问题并不常见。迟早你会发现当你知道bug的原因和需要改变的位置时，模棱两可、错误百出和内容缺乏的规格说明让你不敢确定软件应该怎样运行才是正确的。然后确定规格说明是什么样的就成了你的工作。你可能会问软件的设计师，和用户交流，或者考虑你自己的判断。无论你做什么，要确信你安排的行为被精确描述。写下规格说明，可以作为一封电子邮件发给客户，可以是代码中的注释，或者是文档的附录。这样做可能比修正bug本身更重要，因为当规格不清楚的时候，不同的开发者（工作在不同的时间段，或者没有密切的联系），将会实现不同的语义。因为这个原因而导致的bug很难纠正。

10. 修正bug的方法

太常见的是，我不得不纠正一个bug两次或三次，因为别人的修正没有考虑预期的变更情况，或者和项目中别的改变相分离。这第二次和第三次的修正经常因为修正者试图对存在的代码打补丁而不是重写它。保持已有的设计、声明和代码路径不变，而通过调整和使用状态变量使它们继续工作，这样做看起来比较安全。但是有时候用一个比较好的设计来代替原有代码是唯一的解决方法。

我曾经接过一些到处是bug报告的代码。它的任务是在表达式被送到服务器处理之前对表达式进行检查。在不同程序员的一系列纠正之后，代码已经到处是不可读的if 判断语句。

我只有一个已知的bug要纠正，但是我看到了可能有很多个，而且在没有弄清楚错误的情况下规格的修改将不可能实现。

代替调整代码，我写了一个简单的向下递归分解程序来代替它。这花费了一天半的时间，而修正这个已知bug需要两个小时左右。但是这样做有这样做的效果，关于这部分程序的新的bug 事实上已经没有了。当需要对新的表达式类型增加支持时，它们很容易移植入已存在的框架中。的确，当代码只需要很小的调整时你不需要替换它。但是当一个bug是由于原有代码中的设计错误，你应该认真考虑如何改进和替换原有设计。

对于原来的对象模型、最初编码者编写的接口或者他们选择的设计模式和算法，都不是一成不变的。

如果你真的选择改变接口、替换原有代码的主要部分，或者进行别的大的改变，要确信你有一个很好的理由那样做。和别的投资者交流来确保你的修改将符合他们的利益，并且把你的修改完全文档化。

编写一个新软件是一个卓越的创造性劳动。通过科学知识的应用、创造性的天赋和对人类本质的深刻理解，我们从无到有构造了一些强有力的工具、使人着迷的娱乐游戏或者基础结构的一个重要部分。“捉虫”可能是很令人满意的职业，它具有吸引力、较好的报酬和成就感。所以当你有幸被分配“捉虫”时，不要感到受胁迫和厌烦。

请微笑，你是一个猎人，正在这里寻找一些最善于逃避的猎物。

查看评论

Gigix 译文专栏

Chain Constructors

Joshua Kerievsky 著，透明 译

你拥有多个构造子，其中包含了重复的代码。

将构造子串在一起，以使重复代码减到最少。

```
public class Loan {
...
public Loan(float notional, float outstanding, int rating, Date expiry) {
        this.strategy = new TermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating =rating;
        this.expiry = expiry;
}
public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
        this.strategy = new RevolvingTermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
        }
public Loan(CapitalStrategy strategy, float notional, float outstanding,
                int rating, Date expiry, Date maturity) {
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
}
}
```

```
public class Loan {
...
public Loan(float notional, float outstanding, int rating, Date expiry) {
        this(new TermROC(), notional, outstanding, rating, expiry, null);
}
public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
        this(new RevolvingTermROC(), notional, outstanding, rating, expiry, maturity);
}
public Loan(CapitalStrategy strategy, float notional, float outstanding,
                int rating, Date expiry, Date maturity) {
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
}
}
```

动机

在同一个类的两个或更多的构造子中编写重复代码，这就是在为自己埋下麻烦的种子。别人会在你的类中添加新的变量，然后更新一个构造子来对这个变量进行初始化，但是却忘了更新别的构造子。于是，“砰”的一声，向新的bug问好吧。一个类中的构造子越多，代码的重复就会伤害你越重。如果有可能，就应该尽量减少或去除代码重复，这样做的额外好处就是可以帮助你的代码系统减肥。

为了达到这个目标，我们经常会使用Constructor Chaining模式进行重构：特化的（specific）构造子调用普化的（general-purposed）构造子，重复这个过程，直到最普化的构造子也被调用到。如果你的每条调用链的末端都是同一个构造子，我就把它叫做“catch-all”构造子，因为它处理了所有构造子的调用。这个catch-all构造子通常会比其他构造子接受更多的参数，并且可能是（也可能不是）私有的或保护的。

如果你发现多个构造子降低了类的可用性，请考虑使用“用Factory Method模式替换多个构造子”的重构方法。

通信	重复	简化
如果一个类中的多个构造子在实现重复的工作，那么在特化与普化的通信上，你的代码就是失败的。要实现这种通信，就应该让特化的构造子调用普化的，并让每个构造子都做自己独一无二的工作。	构造子中的重复代码让你的类更容易出错、更难维护。寻找通用的功能，将它放到普化的构造子中，将调用转发给这些普化的构造子，并在其他的构造子中实现用途不广泛的功能。	如果超过一个的构造子包含同样的代码，要看出构造子之间的区别就很困难了。让特化的构造子调用普化的，并形成一条调用链，从而对构造子进行简化。

技巧

1. 寻找包含重复代码的两个构造子（我把它们分别叫做A和B）。确定A是否可以调用B或者B是否可以调用A，这样重复代码才可以被安全的（和比较容易的）从这某一个构造子中删掉。
2. 编译、测试。
3. 对类中的每个构造子，重复步骤1和2，以获得尽量少的重复代码。
4. 如果某个构造子不必要成为公开的，就改变它的可见性。
5. 编译、测试。

范例

1. 我们将从一段简单代码——一个Loan类——开始。这个类有三个构造子，用来表现三种不同类型的贷款业务。大量丑陋的重复代码。

```
public Loan(float notional, float outstanding, int rating, Date expiry) {
    this.strategy = new TermROC();
    this.notional = notional;
    this.outstanding = outstanding;
    this.rating = rating;
    this.expiry = expiry;
}
    public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
    this.strategy = new RevolvingTermROC();
    this.notional = notional;
    this.outstanding = outstanding;
```

```
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
    public Loan(CapitalStrategy strategy, float notional, float outstanding, int rating,
            Date expiry, Date maturity) {
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
```

我研究了前面的两个构造子。它们包含重复的代码，而第三个构造子也同样。我考虑第一个构造子应该调用哪一个，并发现它应该调用第三个构造子。因此我把第一个构造子修改成：

```
    public Loan(float notional, float outstanding, int rating, Date expiry) {
        this(new TermROC(), notional, outstanding, rating, expiry, null);
    }
```

2. 编译程序，测试这次修改是否正常工作。

3. 重复步骤1和2，尽量去除代码重复。这引出了第二个构造子，它也调用第三个构造子，如下所示：

```
    public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
        this(new RevolvingTermROC(), notional, outstanding, rating, expiry, maturity);
    }
```

在我知道第三个构造子就是我的catch-all构造子，因为它处理了所有的构造细节。

4. 检查这三个构造子所有的调用者，以确定是否可以改变某一个的可见性。在这个案例中，我不能这样做（假设是这样——在这里你无法知道其他代码的情况）。

5. 编译并测试，完成此次重构。

译者的话

在我学习设计模式的过程中，曾经有这样一种感觉：我知道相当多的设计模式，但是很多模式我不知道应该如何去使用。Christopher Alexander曾经说过：模式是在特定场景下解决特定问题的可重复的解决方案[Alex77]。可是我常常感觉不到这种场景。

在《设计模式》一书中，GoF曾经说过：设计模式的重构的目标[GoF95]。这为设计模式的学习者提供了

一种着眼点。Martin Fowler也列举出了很多重构方法，其中一些方法就是以设计模式为目标的——重构的结果就是设计模式的结构[Fowler99]。

本文的作者也是一位软件开发的专家。他在Martin Fowler的研究基础上发展出了更多的以模式为目标的重构技术，本文就是其中之一。这一系列文章，多以《设计模式》中的模式为目标，因此读来更具系统性和连续性。而他行文的风格和重构的描述格式又基本与Martin Fowler一致，也算简单易懂。我想这些文章应该对设计模式的学习者有一定帮助。

参考书目

[Alex77] Alexander, C., Ishikawa, S., Silverstein, M., A Pattern Language, New York: Oxford University Press, 1977.

[Fowler99]Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., Refactoring: Improving the Design of Existing Code, Reading Mass.: Addison-Wesley, 1999.

[GoF95] Gamma, E., Heml, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Reading Mass.: Addison-Wesley, 1995. 中译本：《设计模式：可复用面向对象软件的基础》，李英军等译，机械工业出版社，2000年9月。

查看评论

设计模式的理解

Adams Wang

面向对象指使用离散的对象来构建软件系统；设计模式利用了对象的继承、组合和代理（delegation），在较OOP高的层次上考虑问题。尤其是使用代理来对任何不稳定或不确定的方面，如状态、对象的创建、应用平台等等，进行封装，从而保证了源代码的重用和设计的稳定。实际上可以理解成为是OOP中虚函数、多态概念的延伸。即OOP中的虚函数和多态实现的是方法、对象行为上的多态，而设计模式的则对创建、结构和高层次的行为进行了多态。

Creation Pattern

当系统演化成依赖于对象的组合、聚集时，创建性模式带来了更大的灵活性。

Abstract Factory：抽象工厂是创建对象族。它能确保被创建对象家族的一致性，和对象家族发生改变的灵活性，在跨平台的设计中可以得到应用。

Prototype：一种将对象生成的责任代理给自己的模式。它与C++中的拷贝函数不同，C++不支持拷贝函数的多态，即当对象使用基类的指针进行引用时，无法按照子类进行拷贝，会造成切割。Prototype在Java中得到大量的应用。

Builder：侧重点在于对象创建的过程中，避免在code中出现大量的硬代码。当被创建对象结构发生改变时。避免了对散布在程序中大量new语句的修改。

Factory Method：实现了生成对象不确定时的解决方案。实际上在C++和Java中，对于虚函数的理解比较容易，即某个方法不确定时，使用虚函数来声明，而将具体的实现交给子类。Factory Method它实现了对象创建时的多态。

Singleton类似于一种技巧，利用OOP的技术，强制实现了有限、定量对象的产生。

对比：Factory Method实现了创建的多态；Prototype实现了拷贝的多态；Builder实现了对象创建过程的多态。

Structural Pattern

结构性模式关注于如何将类和对象组成的更大结构，它带来了对象组合时的灵活性。

Composite：适用于表达整体-部分关系，可以忽略单个对象和合成对象之间的差别。它实际采用的是一种树状结构。

Decorator：与继承不同，用代理的方式实现了多态，可以避免大量子类的派生。相应的它适用于链状和树状的结构。

Proxy：结构与Decorator非常相似，它们的侧重点不同，一种是修改对象的行为，另一种控制访问。

Bridge：实现了抽象和实现之间的永久绑定，可以理解为在基于已有的构件上设计时，而且已有的设计可能会发生变化。它往往与Abstract Factory共同用于跨平台设计的情况。

Façade：用于对子系统提供统一的接口。

Behavioral Pattern

行为模式涉及到算法和对象间的职责和分配。行为模式描述了通信方式、控制方法以及对象之间的联系方式。

Chain of responsibility：用于链状结构，将职责沿链进行传递，不显示指定责任的承担人，由对象自己实现责任的实施。

Command：将某个职责封装成对象，可以与memento结合在一起用于Undo。

Iterator：大量应用于算法库中，对遍历操作进行封装。

Mediator：通过中介对象的引入将网状的结构变成以中介者为中心的星形结构，从而保证了对象结构上的稳定，即不会因为新对象的引入造成大量类中指针的修改。

Observer：经典MVC模式的变形，与Mediator的结构类似，在对象级别是星形结构。它们均是将网状结构变成星形结构，但侧重点不同。

State：经典方法状态机的OO实现，可大量应用于控制密集型的系统中。

Strategy：将算法进行封装。适合应用于对效率要求较高的软件中，为效率的提高预留接口。

Visitor：在对象级别中实际为矩阵结构，与Abstract Factory类似。它们以采用对象的代理，为矩阵的一个维度提供灵活性和一致性。

查看评论

UMLChina 讨论组最新热门帖子 TOP5

对软件工程的一点看法--希望能够抛砖引玉　- <2744b> miaolin 2001/11/22 10:41 (627 次点击)

一个 java 实现上的具体问题　- <351b> joy_wind 2001/11/09 12:30 (246 次点击)

什么是 OO 方法，为什么要用 OO 方法，怎么用 OO 方法？？？　- <473b> mouri 2001/11/05 17:52 (164 次点击)

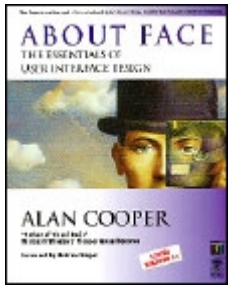
CMM vs Microsoft　- <767b> axial 2001/11/04 23:24 (122 次点击)

一个纯 UML 问题　- <636b> sealw 2001/10/31 16:49 (369 次点击)

消息：UMLChina 第十二期专家交流

北京时间 2001 年 12 月 15 日（星期六）上午 10:00-12:00

嘉宾：Alan Cooper，软件交互设计的先知和传播者，客户咨询公司 Cooper Interaction Design 的创始人和总裁。第一个可视化开发工具 Visual Basic 的发明者。交流重点：软件可用性，交互设计，其它...



要更多了解 Alan Cooper 的思想，请看以下资料：（1）《非程序员》第七期的第一篇文章“交互设计之父 Alan Cooper 访谈”；（2）去书店买他的著作：《软件创新之路--冲破高技术营造的牢笼》

详情请见>>

项目经理面试指南（下）

Patricia L. Ferdinandi 著，zhoufang 译

问题10：怎样确定人员需求？

答案10：不考虑资源限制进行计划开发。在任务旁边加上诸如数据模型制作者，业务分析员和用户等角色。再加上能将任务重叠起来的补充性的资源。在计划中要考虑开发团队包括支持团队和用户代表失去一个或多个资源的情况，要在每个任务上增加15%的余量。要使项目小组的组成容易理解，要有角色所必备的技术水平的说明。

问题11：给项目加上测量标准有什么价值？

答案11：如果使用得当，测量标准是一个有价值的工具。它们提供测定开发系统的复杂性和工作量的方法。度量结果为制定项目计划提供了信息输入资源，并且是确定发展方向的有价值的历史信息。软件测量标准将有助于开发更好的软件。不过，最好有3年的历史资料。

问题12：你怎样在计划中运用新技术？

答案12：在增加培训任务的同时要扩大工作量，缩小每个工作单元。在评价新技术在开发中的影响的过程中加上额外的原型和检查点（里程碑）。

人员管理技能

问题13：你作为项目经理要做的第一件事情是什么？

答案13：除了注意公司的发展方向并从中发现自己的发展道路外，在头脑中要建立项目经理所关注事物（商务，公司，项目，团队，个人，技术和方法论的变化）的优先顺序。因此，和部门经理开会确定优先顺序，安排用户和职员会议，得到全部成员的状态报告和评价。重要的是能尽快处理业务，项目和个人有关的事情。

问题14：当你的职员减少了30 %你将怎样着手完成公司的项目？

答案14：首先，确定和区分项目的优先次序，哪些项目是必须在今后的18个月内完成的。把绝对的最小的总人数与每个项目联系起来。向管理者和用户说明对进度表的影响。因为两者都也许不愿意接受进度表的变化，因此或许可以给你一些例外。

减掉顾问比去掉一个雇员要好。每个项目的顾问也许可以用雇员代替。坚持运用学习曲线理论并逐步减少顾问人数。可以把一些顾问的工作从一周降低到一星期中的2或3天以应付人员削减。

如果公司有提前退休的一览子法案，赶紧寻找一些有资历的、适用的雇员。牢牢记住失去“老资格的人”你也许就失去了有价值的知识。尽可能将一个快退休的人和新手组合在一起。

以满足业务目标为前提，确定剩下员工的重要性以及他们在每个项目中的重要性。使新手和经验丰富人员的比例适当。两者都是确保项目和公司不断成功的财富。

问题15：你的团队主要是由新手组成的，并且进度已经落后。你将做什么？

答案15：需要记住一个项目很少因为在截止时间内没有完成而被取消的。项目被取消，主要是诸如缺少资金，用户支持或不能满足的业务目标。

因此，要做的第一件事是培训，无论在室内还是室外，在课堂或通过录像带。另一种附加方法就是让资深的雇员或高级顾问充当教师。

举办针对个人评估和辅导的会议。帮助每个员工准确评价他们各自的优点和缺点。同时明确任务，将所有必须遵守的标准或准则阐述清楚。为每个员工提供从成功项目中得到的模板作为指南，还要允许他们发挥自己的才能。如果需要，和他们一起工作。对任何问题或完成的任务做出迅速的反馈。

对于较大的任务，看看他们的计划，有助于确定他们是否了解任务的范围和目标，以便了解他们是否能完成任务。倾听员工的观点，也许他们会有完成任务的正确的方法和途径。然而也要防止雇员陷入挫折和士气低落的困境中。

问题16：你将怎样和与你竞争相同职位的员工相处？

答案16：这是经常发生的不愉快情况。雇员总是认为他们能胜任某个职位而管理层还没有意识到这一点。因此，要进行如下调查：

- 发现员工的管理能力
- 阅读评估和状态报告
- 当雇员变得不合作时试图发现一些变通的方法并且针对这种状况进行一些个人谈话，谈话内容包括：
- 弄清楚状况
- 与员工一起分析他/她具有的能使他/她得到提升的资历

- 强调在初期协作的必要性和管理层是如何高度重视合作关系的

问题17：在决策和工作风格方面你会给你手下多大的自由？

答案17：自由的大小取决于每个人的技能和专业水平。一个好的经理是“面向结果的”并且能创造一个能使团队广泛交流的环境。无论如何，每个员工每周需提交项目和商业目标有关的状态报告并且经理要进行审查。这有利于加强组织建设并使每个员工致力于他们自己应完成的工作。

问题18：如何对待即将退休的员工？

答案18：即将退休的员工能提供大量的信息。一个人在把所有业务知识和关系网拒之门外时必须三思而后行。因此，要利用这些人的能力：他们在某些特殊技能方面可以作为新手的老师。明确主要的工作利益，要使项目能充分利用这些技能，可以利用他们从非正规途径得到的必要支持（不用通过正规的，官僚的途径完成工作）

问题19：对一个一贯迟到的员工你会怎么办？

答案19：好的经理是通过结果与所花时间来评价一个员工的。然而，还需要了解迟到会在公司和团队中造成什么影响。一个人经常迟到人们会感到领导在徇私并且会影响团队的士气。这个人也许可以按期完成自己的任务但可能会影响到别人的进度。职业特性包括可靠性。如果别人的工作进度取决于他们的工作进度，那么，他们的进度对于整个团队就很重要。

首先判断这些员工的模式。换句话说，是偶尔还是一贯如此。其次，明确公司有关考勤方面的政策，确定迟到及其相关处理方法。要了解该员工的工作是否与进度相符并了解与他一起工作的人对他迟到的反应。

最后，必须与他们进行客观的谈话。

谈话的主题包括：

- 公司的规章制度
- 对团队的影响
- 对个人评价的影响
- 强调时间进度
- 达成谅解

问题20：在费用削减的情况下，你将怎样鼓舞士气？

答案20：钱不是仅有的激励因素。人们需要了解他们是否对项目有积极的贡献。因此，要强调拥有的自豪感并且举行业务会议，在会上让用户谈谈他们对项目组的良好印象。同时，让用户对他们的功能和业务提出一个概括。培训是一个激励因素。因此，状况会议可以作为一个非正式的培训课程。不定期地举办有关新技术的内部研讨会。如果培训课程费用太昂贵，可以租赁技术录像带。订阅杂志，有许多技术杂志是免费的。必须记住的是，忽视培训将使团队的精神低落。这样会影响产品的质量和数量。

问题21：你如何雇人？

答案21：首先做一个工作所需技能的描述。如果你不了解现在的需求就很难雇到合适的人。接下来要了解团队成员的个性。列出团队现在缺乏的技能或工作风格。与人力资源部门讨论所有这些情况，包括调动现有员工。当候选人到来，针对现有工作进行面试，同时还要了解他是否具有新岗位所需的技能。

问题22：你将如何解决团队中的个人冲突？

答案22：辨别出人的不同个性。分别向员工表述每种风格的价值。当与冲突双方讨论试图分析申诉或冲突的原因时应持有客观的态度。

问题23：你将如何监控/管理顾问？

答案23：顾问也是人，也需要得到尊重。他们还需要明确的目标和任务。坚持做工作周报，将工作时间和工作完成情况联系起来。

问题24：你将如何管理外援？

答案24：和管理顾问的方法相同。不过，他们可能有一个经理来负责外包合作。首先要和这个经理一起组织日常会议。坚持做工作周报和可交付产品的拷贝。

问题25：你将如何同一个似乎总是不能按时完成工作的员工一起工作？

答案25：直到找到问题的原因时，问题才能解决。原因不一定是分析问题或解决问题的能力差。可能是一个管理方面的问题。

该员工可能没有得到适当的培训，他的工作可能超出了他的能力范围。另外一种可能是这个人有太多的事情要做而且这些事情都是最重要的或者他不清楚交付日期。

如果不是上述原因，要注意观察，找出原因所在。例如当所有人遇到问题时，都会找这个人。那么，这个人的工作经常会被无数次地打断。

沟通技巧

问题26：你将怎样使用户参与和了解项目的每个阶段？

答案26：贯穿整个项目的原型是得到用户肯定的方法。让用户对有形和无形的利益进行研究，以做出成本效益分析。和用户一起开发测试数据，测试大纲和验收标准。e-mail里程碑状态报告和更新/修改的项目计划。在项目进行阶段性检查时的同时对可交付产品进行检查。

问题27：你将如何发现和解决内部和外部问题？

答案27：从所有可能的资源获取实情并客观地记录下来。然后在相关方参与下，尽量自己解决问题。如果这种方法无效，按照组织的管理结构提出问题并参照可能的解决方法。

问题28：你将如何得到供应商的一贯支持？

答案28：虽然供应商是在管理范围之外的，但也可以将他们包含进来，如果他们：

- 得到尊重
- 了解业务目标
- 预先购买
- 将供应作为计划的输入，这样会对他们产生影响
- 参与设计

因此，在项目的早期阶段就应该考虑供应商的管理。确保他们了解业务目标和工作的利益。

问题29：如何处理“是否能破除一些规矩”现象？

答案29：单纯为了技术而采用某种技术是不能说服用户或领导的。任何人都可能抵制那些会改变现状的变化。然而，如果将技术与商业利润联系起来，用户会支持你的建议。

问题30：你如何应对不同的商业用户，如果他：

a) 拒绝确认需求

b) 经常改变主意

c) 不肯花时间

d) 坚持不现实的截止日期

答案30：无论客户有多难应付，都应该记住正因为他们我们才有工作做。他们是客户。必须以高度的职业精神，完全尊重他们。

因为他们不能了解我们的工作正如我们不能完全了解他们的那样，沟通变得比较复杂。因此，我们要花时间作规划并解释其中包含的内容。用户需要感到他们没有浪费时间，正在取得成果，并且他们的意图被很好地理解。制作原型是一个有用的工具。它提供了一幅用户能理解的、灵活的图画。

另外，对工作风格的理解也很重要。拒绝承认或不断地改变想法可能源于对问题缺乏理解，或是对未来的担心。

用户往往不愿意花时间与IT人员交谈并认为这样做是浪费时间，因为IT人员过分关注他们自己的任务。应该对过去交付产品的历史进行检查。如果用户来了多次但并未发看到有价值的输出，他们将拒绝花更多的时间。在这种情况下，你应该做你擅长的商业领域的项目以期得到用户的尊重。

召开一个历时一小时（并且要限定在该时间范围内）的需求讨论会来讨论特殊的问题。会议结束时应让用户知道下一步该怎么做（并要取得共识）。用户的观点被记录在“会谈纪要”上。这些会让用户感到他们的意见已被听取并且允许他们更改错误。

一个项目被取消往往是由于没有经济合理地达到用户的业务要求。如果在项目的整个过程中，一直保持与用户的有效沟通，他们将看到他们的要求正在逐步达到。项目很少因为延期而被取消。要注意范围变更。在原有的截止日期上增加额外的任务，将会产生不现实的截止日期。

问题31：在一个不编程，就认为你没在工作的环境中，你如何开展工作？

答案31：如果用户认为你了解了他们的业务目标，他们就希望早些开始编程。以一种他们能够理解的形式制作需求文档，提供一种开放的沟通方式，并让他们知道你了解什么，你正在做什么。通过项目计划，状态报告和原型同样能够表明项目的进展。通过让用户审查需求，原型和状态报告的形式，让用户参与项目。

方法论知识

问题32：生命周期是什么，它的作用是什么？

答案32：一个开发或维护生命周期是描述一个特定项目的开始，中间环节和完成的方法。一个生命周期包含了完成特定目标的所有步骤，任务和/或活动。每个活动可能有一种特定的方法。例如,制作数据模型可能会按照James Martins建模方法。对象建模可能会采用Ivan Jacobson方法。生命周期通过运用所有方法来完成业务目标。

问题33：描述你的项目计划中应包括的阶段、活动和可交付产品。

答案33：项目计划中应包括如下阶段（不是以瀑布/线性次序）：

1. 项目管理：

典型活动：很多人忘记加入诸如开发和维护项目计划，状态会议和报告，评估的资料收集和汇报，制作演示资料和向上级和用户进行演示等诸如此类需要花时间的，内部的项目管理活动。

典型交付：项目计划，状态报告，评估报告（例如：有多少个功能点）

2. 需求分析：

典型活动：范围定义，成本利润初步分析，建议。

典型交付：范围文档，物理和逻辑分析，实体关系图，成本利润分析，商业规则申明，任务定义和概要说明。

3. 设计：

典型活动：建立开发和测试环境，制作逻辑模型，技术系统设计，执行计划。

典型交付：逻辑数据模型，事件模型，对象模型，网络模型，物理设计，适合开发环境的规格说明，经过修改的规格说明书，测试计划，流程图。

4. 开发：

典型活动：编码，单元测试和制作用户文档。

典型交付：测试说明书，过程手册，程序。

5. 测试：

典型活动：软、硬件测试，线性测试，系统测试，集成测试，回归测试和平行测试。

典型交付：测试结果，问题报告和跟踪纪录。

6. 实施和支持：

典型活动：第一阶段成果打包；培训。

典型交付：问题报告过程。

7. 检查:

典型活动：交付后的三到六个月对目标成本，开发工作，可见/不可见收益进行检查。

典型交付：实施总结报告。

问题34：制作原型应该在项目生命周期的那个阶段？

答案34：贯穿整个项目。眼见为实。因为它是验证功能，业务规则，用户需求数据和测试的一个好工具。值得注意的是，原型不会成为粗制滥造的产品。原型需要较好地维护。原型应能在过程和数据不完全的情况下，显示各个窗口和窗口间的导航关系。

问题35：在项目生命周期中，基于客户端/服务器端开发与基于大型机开发的区别是什么？

答案35：基于客户端/服务器端开发的项目需要额外的任务编制各部分的计划。各部分计划中必须包括对事件，数据和网络位置的检查。必须根据用户的要求决定服务器/客户端的分布。在服务器/客户端环境中，要运用外观建模技术和制作图形界面的原型相结合和方法。

问题36：在一个维护项目中如何管理和保证质量？

答案36：维护本身就含有负面意义。许多公司认为维护工作是不好的，第二位的。费钱的，并且是对现有应用的不断修改。必须懂得维护也有它的生命周期。因此，应建立一个围绕维护活动的控制和质量工作的计划。新的开发计划包括交付产品和每个任务分配的时间。项目计划应考虑到需求变更的情况。这样可以使项目经理和用户看到变更对项目进度的影响。

维护阶段/活动有：

变更的确定（是否会造成产品问题，是否增加了新的功能，或技术平台的变更）

1. 正式记录变更，
2. 变更确认并初步估计变更的大小，
3. 对现有变更进行优先级排序，
4. 变更分析，
5. 对变更进行编程，
6. 对变更和变更对系统产生的影响进行系统/回归测试，
7. 用户确认变更，
8. 产品递交，
9. 生产。

问题37：面向对象的开发与传统的开发方法在管理技术上有什么不同？

答案37：面向对象的项目团队人员较少，团队成员不需要有太多创意。重要的是技术和个人的角色。每个成员需在项目的不同阶段承担不同的角色。因此，每个成员必须了解他们自己的优缺点。围绕一个或多个人员的角色有：

- 设计师（系统的整体结构）
- 抽象工程师（类和类族）
- 应用工程师（完成和组装类和类之间的消息）

由于传统的开发方法，个人角色是不能互换的。软件开发是个人的努力的结果。即使是由最优秀的，最聪明的人组成的团队，如果他们不能为共同的目标而工作，那么就是最简单的项目也不能成功完成。

问题38：你如何在处理雇员关系，项目管理，文本工作之间分配时间？

答案38：人是最宝贵的财富，因此需要花费最多的时间。然而，项目经理必须关注事物的次序应该是：

1. 商业目标，
2. 公司的目标，
3. 项目，
4. 团队，
5. 个人，
6. 技术和方法的变化

问题39：什么是PM-CMM?

答案39：人员管理能力成熟度模型。PM-CMM和CMM都是卡内基.梅隆大学的软件工程研究所开发的概念模型。PM提供了人力资源管理的组织方法。五个层次是：

1. 随意的：人员管理没有连贯性，
2. 可重复的：组织在人员管理方面有一些政策方针，
3. 明确的：将人员管理与业务特点相结合，
4. 可度量的：对人员管理可进行目标量化，
5. 优化：有组织地致力于不断地提高人员管理水平。

小结

一个成功的团队是指由不同技能、才华、工作风格和知识的成员组成的士气高涨的团队。项目经理的职责就是将这些人组成团队并激励他们。本文通过复习一般性的概念、术语和面试中经常会问到的问题，为面试做准备。你可以根据你有关如何成为一个好的项目经理的知识和经验，对答案进行整理。不管怎么回答，尽量给你所应聘的组织留下印象。应以一种积极的态度面对。应侧重于人员管理，同时还有一个良好的技术背景。应具备应有的常识、自信、倾听和作决定的能力。

查看评论

为什么还不编码？

张俊 编译

您正从事着一项专业品质的应用开发，正处于其设计阶段——您正在和用户面谈，正在记录对象的定义，正在绘制对象的模型——这时——就在这时！——您的老板走过来问道："为什么你还不写代码？！"——好吧，也许不尽然都是这样露骨的质问——但老板对你缺乏那些在开发进程中眼见为实的东西而心怀不快，却是显而易见的。

这里提供几个有关这一问题的回答。对其任取或全用好象都恰如其分。

因为在构造之前，我需要先弄清楚我要构造的是什么。（您见过有未经细致规划就盲目垒砖砌墙而造出的好房子吗？！）——使用这样的回答需要您能解释明白设计意图。在此处引用建筑的比喻让人感到轻松自在。

因为我不想多次返工重写代码。——解释您宁愿第一次就编码过关也不想日后不得不反反复复重写代码。——如果您的老板有过因考虑不周而在预料不到的情况下必须使项目返工的痛苦经历的话，这个回答是最受用的。然后您还可以补充："难道您不记得我们在搞某某项目时曾返工重写了多少代码了吗？！"（点到他的痛处，使他唤起痛苦的记忆，用他曾付出的代价教育一下他，折磨一下他，免得日后您自己倍受折磨。）

因为我们想更快地完成这一开发任务，并且我需要将其划分成各个逻辑单元，以便于我们的团队能高效地按逻辑单元来进行工作。——这里满足了老板两方面的利益——首先，它清晰地表明您对本项目紧迫的最后期限心中有数。其次，它也明确显示了您是怎样正在帮助促使团队更有效率地开展工作。

因为我们的这个项目要求我们更紧密地跟用户协同工作。——如果您的老板有过在项目中和客户相处不愉快的经历的话，这个回答是十分奏效的。您可以接下去说道："难道您忘了当初把某某项目交付给客户使用时，他们是怎样地没给咱好脸色看的？！"您可以加上真实事例来补充这一回复，您可以找出正在同其一起工作的某一个用户——向您的老板提起——在早期就和客户搞好关系，对项目成功的影响是多么的巨大。

因为我们想打造一个艺术级的群体。——这一回答满足了老板的自负心理——在今天，艺术级就意味着面向对象——面向对象的设计要求在构造应用程序之前就识别确认出各种对象——就这么简单。

因为我们想拥有可复用的组件，以便将来更轻松地支撑起其他项目。——这一回复和上一回复同样奏效。要知道，生产力的关键点正在利用可重复使用的组件。它要求预先规划好要开发的组件，以便其他应用程序也能使用它们。

我正在编码啊。——如果您的老板对有关设计步骤的解释不是很感兴趣的话，这一回答是很奏效的；为了使您的台词更恰如其分，您可以在专心于设计工作的同时把您的编程工具——诸如 VB 的 IDE（集成开发环境）——始终打开着并使其最小化。当瞄到老板走过来的时候，迅速恢复起 IDE 窗口。并确信代码窗口有许多热情的代码正等在上面，欢迎着老板的光临。为此，只需运行一下 Application Wizard（例程向导）就能生成您想要的一些漂亮的代码了。

编译者续貂：

——假如俺那老板——正向俺走过来了——不是假如——此刻，就在此刻！——要知道俺这么个老实透顶的大好人，可不会耍什么把 IDE 窗口缩成小条然后再迅速提起变大的把戏呀！——老板已经拍着俺的肩膀了，不怀好意地亲切问道："为什么还没编写代码啊？！"——天啊，俺这可怜的家伙——该怎么办啊？！

——俺会朗声作答："老大，据我所知——早产的婴儿往往最容易夭折。即便活下来的，也常常会带有残疾或弱智。我可不想让我的孩子非傻即呆或瘸着腿走路——'九月怀胎，一朝分娩'——您看，我现在正在精心孕育着我的孩子——并且，我太想生出一个又正常又健康的孩子了。"

——然后俺会用充满威慑的眼光盯住老板（延时几秒钟），同时用饱含母性柔情的语调坚定地说道："请相信，我会是个好产妇！"

——哈——哈——嗬！

:-D

查看评论

UMLChina 讨论组最新热门帖子 TOP5

对软件工程的一点看法--希望能够抛砖引玉　- <2744b> miaolin 2001/11/22 10:41 (627 次点击)

一个 java 实现上的具体问题　- <351b> joy_wind 2001/11/09 12:30 (246 次点击)

什么是 OO 方法，为什么要用 OO 方法，怎么用 OO 方法？？？　- <473b> mouri 2001/11/05 17:52 (164 次点击)

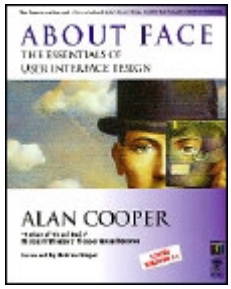
CMM vs Microsoft　- <767b> axial 2001/11/04 23:24 (122 次点击)

一个纯 UML 问题　- <636b> sealw 2001/10/31 16:49 (369 次点击)

消息：UMLChina 第十二期专家交流

北京时间 2001 年 12 月 15 日（星期六）上午 10:00-12:00

嘉宾：Alan Cooper，软件交互设计的先知和传播者，客户咨询公司 Cooper Interaction Design 的创始人和总裁。第一个可视化开发工具 Visual Basic 的发明者。交流重点：软件可用性，交互设计，其它...



要更多了解 Alan Cooper 的思想，请看以下资料：（1）《非程序员》第七期的第一篇文章“交互设计之父 Alan Cooper 访谈”；（2）去书店买他的著作：《软件创新之路--冲破高技术营造的牢笼》

详情请见>>

《面向对象项目成功之道》（草稿）节选

Alistair Cockburn 著，乐林峰 译

“Surviving Object-Oriented Projects: A Manager's Guide”是Alistair Cockburn所著的经典著作之一。它的中文译本《面向对象项目成功之道》即将发行，译者为UMLChina翻译组的乐林峰，这是翻译草稿的片段。

第 2 章：项目预期

尽早计划

你是不是因为维护成本太高才去研究对象技术？还是你的竞争对手声称他们的产品已经采用了对象技术并获得了成功，而你的产品还以传统方式的开发？你手里是不是常有积压的定单？你的老板是否命令你采用对象技术？不管是什么原因使你转向对象技术，你应该知道的一点是，转向OO技术是需要付出一定代价的。

你愿意付出什么代价？你希望得什么样的回报呢？你希望在什么时候获得回报？本章将讨论一些你可能会关心的内容，比如，你预期的收益是什么，需要付出什么代价以及一些可能被你忽略的问题。

项目历史

本节中将对11个项目进行总结。我和每个项目的负责人进行了交谈，收集了大量的相关数据，并据此对项目中的得失进行了反思。这些总结揭示出一个项目成功或失败的奥秘。同样的情形我在其他项目中也看到过。出于信息隐私等方面的考虑，只在描述以下项目时采用真实名称“国际对象技术”、“ 布鲁克林联合汽油”和“门特图形”，其它项目则使用绰号，以便在书中其它地方引用。

对每个项目的总结都以一个简介卡开始，包含如下信息：项目名称（或绰号），项目主题、项目规模尺寸、开发人员的经验、项目演化历史描述以及一些附加的评估内容等信息。

阿尔佛雷德项目：成功的需求变更

项目名称	阿尔佛雷德
主题	项目设置
开发人员规模	少（5-10 人）
项目类型	研究
员工开发经验	没有 OO 经验
开发周期	6 个月

Alfred项目的目的是为了研究对象技术。项目开发人员包括一个高级程序员，两个普通程序员，一个用户界面设计师和一个没有任何编程经验的业务方面的专家。项目经理熟悉增量式和迭代式开发。整个项目组中唯一具有OO经验的人是一个兼职的OO专家，他负责对项目开发人员进行培训和设计指导。

第一个开发阶段为6个月。该阶段前期工作是建立业务模型；用户界面在后期完成。但是在项目第一阶段将要结束的时候，项目主管告诉设计小组，只有对象技术能够通过如下测试才能够被正式接受：

很长时间以来，我们一直希望增加一个新需求，以前之所以没有提出来，是因为实现这一新需求可能需要10个人4个月的工作量，而且还要改动数据库设计。但是现在，你们只能在本周五早晨到下周一早晨这段时间内来完成它。

周五,6个小组成员象往常一样工作了一天，然后有四个人加班到了周六的下午。该需求以及相应的数据库变更就完成了。在周一的早晨，设计组为项目主管演示了工作成果，并以此证明了他们应变的能力。

该项目的成功之处在于，项目主管为了检验对象技术的可行性而建立了一个测试环境，通过精心的安排，对对象技术进行了成功的测试。这是一个研究性项目的很好的例子（详细内容参见第三章）。

布鲁克林联合汽油项目：成功

项目名称	布鲁克林联合汽油
主题	风险管理
开发人员规模	多（100 人以上）
项目类型	改写现有系统
员工开发经验	没有 OO 经验
开发周期	几年

1986年以前，布鲁克林联合汽油公司（Brooklyn Union Gas简称B.U.G）一直采用PL/1语言为大型机、IBM 3270终端开发基于关系型数据库的应用程序。公司决定改写定单处理系统。在对常用的结构化开发技术进行了一番评估之后，管理层发现，如果采用这些技术完成改写工作将耗费太多的时间。因此，他们认为有必要冒些风险，采用下一代开发技术而不采用已经开始落伍的现有开发技术。

没有人确切知道这个决策将带来什么样的结果。他们能够肯定的一点是，确定系统继承结构将是一项风险较大的工作，因为一旦在原有数据库基础上建立起继承结构，将很难再做任何变动。因此，他们决定尽量 设计一个简单而稳定的继承结构，而不是一味的追求复用最大化。他们选择并遵循了简洁但相对保守的设计标准。在某些关键而重要的地方，比如在实现OO分配机制的时候，允许存在一些不同的程序版本。通过上述以及一些其它的策略，他们使用PL/1语言改写了系统。以后的运行表明该系统的改造工作是非常成功的。

他们的成功表明，对象技术可以在任何平台，通过任何编程语言实施。我能够感受到的另外一点是，他们的管理层和开发小组的水平确实很高。这个项目的成功说明，如果想保持一个项目充满活力，必须做到以下几点：

- 周密的计划（见第四章）。
- 采用增量式和迭代式开发（见第五章）
- 充分利用以前的经验（见本章后续内容）
- 建立并遵循一套简洁实用的标准（见第四章）。

- 吸取早期开发方法的精华（见第四章）。

在本章的最后将介绍“Morgan’s Eyewitness Account”项目，这是一个比较详细的关于B.U.G项目的报告，发表于1993年。

英格里德项目：成功地向C++移植

项目名称	英格里德
主题	错误恢复
开发人员规模	中（20-40 人）
项目类型	产品
员工开发经验	OO 初学者
开发周期	2 年

出于很明显的错误动机，项目经理希望开发人员通过项目的实施学习对象技术并掌握C语言，决定使用C++进行开发。25个没有任何OO编程经验的程序员被委派开发该项目。公司没有雇佣任何这方面的顾问，也没有为员工提供任何培训。他们相信优秀的程序员应该能够通过自学掌握C++.

在认为已经讨论清楚所有风险之后，项目经理领导了一场背离与正规公司工作方式的“战役”，而且采用了基于瀑布开发周期模型的进度安排和阶段划分的策略。他被获准使用增量式开发方法（不包括迭代式开发方法），所有需求必须在（统一？）时间完成，即使是那些在下一开发阶段才会用到的需求。

在第一个增量开发阶段结束的时候，开发小组面临了一场灾难。进度落后，设计存在缺陷，士气低落。因此，项目经理替换了25个程序员中的24个，调整了管理结构，对开发人员进行培训，并在第二个阶段将前一阶段的工作重新做了一遍。虽然开发进度仍然落后于新的日程表，但开发小组还是将第一阶段和第二阶段的工作一起完成了。

在采取了一些措施和管理层的变更后，小组建立起一套内部自我培训制度。这时，他们才开始慢慢适应并掌握对象技术。第三和第四阶段的开发工作顺利完成。项目经理说越来越得益于具有内部相似性和可复用性的软件，这使得开发工作能够按时完成。

我认为这是一个很完美的项目，因为它恰当的说明了本书的主题。Ingrid项目在开始的时候是失误的，但开发小组及时改正了错误。这个项目给我们的主要经验是：为了获得成功，开发小组应该积极主动地改变所有需要改变的东西。同时，在增量式开发的阶段划分时，要留给开发人员适应改变的余地。

Manfred项目：失败的原型设计

项目名称	Manfred
主题	错误恢复
开发人员规模	中（10-20 人）
项目类型	产品
员工开发经验	没有 OO 经验
开发周期	1 年

Manfred项目是计划用Smalltalk语言开发一个新的并且比较严格的软件产品。项目由一个没有OO编程经验的C程序高手领导。与Ingrid项目类似，他们也没有雇佣顾问和对员工进行培训，认为程序员能够通过自学掌握一门新的开发语言。开发小组的领导认为Smalltalk拥有一个优秀的开发环境，应该建立一个原型。在原型被最终接受后，再按照生产产品的质量重新编写全部代码。

在第一个原型被建立之后，他们被告知原型的执行速度太慢。“没关系。”开发人员说，“这只是一个原型而已。告诉我你想要什么。”他们得到的答复是“我们希望它运行的快一点。” 然而，同样的情况在第二和第三个原型上也发生了。这时候，公司的高层开始感到不安了，因为种种迹象似乎表明要么是开发小组没有十分重视大家的反馈意见，要么是Smalltalk语言开发的软件执行速度太慢。最后，开发小组接到通知，如果不能建立一个和真正产品运行速度一样的原型，项目将被取消。开发人员花了很大力气优化原型性能，但还是没能在规定的时间内达到要求。在付出了巨大的成本代价后，项目还是被取消了。

这个项目所犯的第一个错误是混淆了原型与最终产品的界限。这是常见的原型技术并发症（原型看起来很好，我们是不是可以交货了？），实施除了建立一个非产品化的原型之外，我没见过其它成功的例子。我建议将原型称做“需求模型”，一旦产品正式开发，即将之尽快丢弃。

第二个错误是设计者认为原型能够代替设计（见第四章和K.L’s Eyewitness Account）。我曾经看到同样

的错误发生在不同的项目中，有200人的项目也有10个人甚至1个人的小项目。

第三个错误我称之为“原型的遗憾”，即没有采用迭代式方法控制原型的规模和深度。这点将在第五章中讨论。

门特图形项目：向C++移植过程中遇到的问题

项目名称	门特图形
主题	公司转换项目
开发人员规模	大（越 600 人）
项目类型	产品
员工开发经验	没有 OO 经验
开发周期	几年

门特图形公司曾一度在计算机辅助设计（CAD）软件市场占据主导地位。是一个C语言开发经验很丰富且极具竞争力的公司。在20世纪90年代初期，公司决定使用C++开发将要发布的8.0版本软件，以使整个公司转向C++语言。

这个决定差点使公司关门大吉。新产品并没有获得预期的市场回报，很快，一切问题就清楚了，门特图形公司的原有客户必须升级他们的硬件，才能使用新版本。（门特图形前任副总裁格伦.豪司参加了8.0版的开发工作。他为我们提供了第7章中的”目击者帐户”的例子）。

在很短的时间内，公司迅速失去了大部分市场份额。虽然通过努力，公司恢复了元气并逐渐掌握了C++语言。但是，公司大部分的主管人员仍然对C++语言心有余悸。

这个案例很明确的告诉我们两点，第一，优秀C程序员在使用 C++进行开发的时候不一定会获得成功（见第3章）。第二，将整个公司完全转向OO技术是件很复杂的事情，并非只是制定一套规章制度和选择一个新编译器(见第7章)。

国际对象技术：成功于生产率和生产速度

项目名称	国际对象技术
主题	生产率
开发人员规模	小
项目类型	产品
员工开发经验	专家
开发周期	几个月

在20世纪80年代末，国际对象技术公司(Object Technology International –OTI) 希望以外包的方式开发一套实时系统，开发语言为Smalltalk。他们的提出要求是“按时交货否则一分钱也别想得到。”

公司总裁戴夫托马斯说，就启动的时间来看，没有人会愿意承接这个项目，除非允许项目被延迟或者是那个公司已经陷入绝境。

由于雇佣了有经验的开发人员和使用优秀的开发工具。OTI公司在很短的时间内完成了一般需要10（？）才能完成的工程。

托马斯说，他对每个新雇佣的员工进行了3个月的培训，允许他（她）在一年只工作9个月的情况下领取全额工资。他认为，对于一个小公司来说，控制成本和培训员工同样重要。

除了遵循的一些科学方法之外。托马斯认为OTI公司能够成功的另外一些因素是他们雇佣了很出色的员工并提供给员工优秀的开发工具。在大约1993年的时候，他们就拥有了象代码自动生成工具、开发测量工具，但那时侯还没有图形化的建模工具。最后，他们的一位设计人员甚至为Smalltalk开发了一套版本控制和配置的管理软件，并作为公司的一款新产品推向了市场。

在OTI公司的例子中，他们的研发人员开发了一套自己的组织管理方法（见第4章），这套方法涵盖了公司项目的全部开发过程以及使用的开发工具。他们发现并接受了自己的不足以及当前所使用的开发方法的局限性（见第3章）。最后，他们证明了Smalltalk也可以成功的开发性能要求十分严格的硬件实时系统，甚至包括示波器的波形生成功能。

雷金纳德项目：失败的变更规则

项目名称	雷金纳德
主题	项目设置
开发人员规模	小
项目类型	产品
员工开发经验	没有 OO 经验
开发周期	一年

最初，雷金纳德项目由两个程序员负责，他们被授权研究对象技术。并要求他们在6个月后提交一份该技术的可行性报告。这两个人都是经验丰富的C语言高手，公司为他们提供了C++语言方面的培训，在整个研究期间，还有行业顾问对遇到的问题提供咨询。项目的主要任务是学习开发通信和协议程序。

由于项目对部门越来越重要，公司又增派了一名程序员到开发小组。小组成员分布在2个不同的国家，他们通过卫星进行联系。在一年的时间，项目的范围不断变化，最后，项目经理被调走，该项目在没有任何结果的情况下，彻底失败了。

在本报告中，我们试图找到到底是哪里出现了问题。看起来在项目中使用的C++语言没有招致太多的怨言，

In the debriefing, we tried to determine just what went wrong. It appeared that the language was not to blame, but rather the deliverables-heavy methodology used, the distances separating the groups, and the changes in the project’s scope .

当项目的基本规则改变时，没有重新启动项目，也没有重新定位项目的目标。最初的两个设计师本来负责的是一个不太重要的研究性质的小项目，目的是为了研究C++语言能够做什么。突然间项目变成了一个重要的产品项目，他们也成了开发组的领导者。但是他们并没有做好准备（见第3章）。

斯坦利项目：过于前卫的技术所导致的失败

项目名称	斯坦利

主题	前卫的技术
开发人员规模	大（100-200）
项目类型	产品
员工开发经验	没有 OO 经验
开发周期	几年

负责斯坦利项目的公司不是一个计算机行业的公司，也没有任何有经验的软件开发队伍。该项目的发起人与对象技术一见钟情，并且声称公司将要进行一个庞大而现代的项目，是另外一个“布鲁克林联合汽油”项目。需求文档里全是一些最新的想法和最近几个月才有的新概念。

随着时间的推移，项目的逐步发展，需要多个分散在对等网中的服务器，运行分布式的Smalltalk程序。那时侯，还没有现成的Smalltalk分布式框架，CORBA标准也还没有被公布。在几年里，该项目做做停停的反复启动了几次，到写这篇文章的时候，还没有真正完成。

很明显的教训：一是不要频繁的变动需求，对于这点我很难再做任何评论。二是不要计划在一个时间要求很严格的项目中采用任何新技术，特别是当你的开发人员还没有该技术经验的时候（见第3章）。虽然在这个故事中体现的还不是特别明显，但确实有很多项目由于计划野心过大而失败。

特蕾西项目：幼稚所导致的失败

项目名称	特雷西
主题	用户参与，方法学
开发人员规模	中（10-20）
项目类型	产品
员工开发经验	没有 OO 经验
开发周期	1 年

特蕾西项目由一个经验丰富的小组负责开发，成员都是开发方面的好手，并且正在学习对象技术。他们阅读了一些相关文档后得出两点结论：让用户参与到项目中来以及为要解决的问题建模。

整个项目是建立在这样一个假设的基础上的“对象技术只不过是另外一种编程语言，程序员将很容易学会它”。这意味着将不会有针对开发人员的对象技术方面的特别培训。

开发小组试图使用户参与系统规格定义和设计评审工作，但得到的却是不太热心的支持，因为在进行上述工作的时候，他们只能要求有空闲时间的用户参加。因此，与他们合作的用户组成员经常变动，客户的意见、观点和技术水平也就常常变化。所以，根本不可能与用户进行连贯的交流，很多有经验的用户也没有时间参加开发研讨会。

他们一直坚信,只要建立了“现实世界的模型“，就能够从这些模型直接得到软件需要的类。但在项目总结的时候，他们才发现，他们建造的模型是十分幼稚和不完善的，况且，即使是拥有一个很合理的模型，也不一定能就能得到很好的设计方案。

主机（mainframe）开发组和工作站(workstation)开发组被分隔在两地，其中前者的人数最多。两个小组没有很好的沟通，因此很难对系统整体结构进行调整。尽管两个小组都声称他们在工作上没有任何错误，但是整个系统却很不稳定。最后，他们改变了开发小组的结构。

这个例子所告诉我们的是：让用户真正参与到项目中来（见第5章周.马歇尔见证人帐户项目），尽量建立一个成熟的业务模型（见第5章），为每个可发布的表单、用例和类指定合适的负责人。

优达项目：小规模的开发组

项目名称	优达
主题	用户参与，方法学
开发人员规模	中
项目类型	产品
员工开发经验	不均衡
开发周期	2 年

优达项目是一个中等规模的软件项目，在启动后不久，开发人员就遇到了麻烦，原因是他们缺少一个内聚性较高的系统结构。部分开发人员向管理层建议，将项目中的问题留给一个很小的开发组去解决。他们的建议被采纳了，公司安排大约12个人在项目真正需要他们之前去做其它工作。剩下来的小部分人通过几个月的工作建立了一个新的系统结构，定义了各子系统，为将来的开发工作做好准备。然后，公司委派有能力的组长领导一个开发小组负责一个子系统，小组成员由组长负责召集。组长只召集他认为得力的组员。最后，项目被成功发布了。

这个项目的成功之处是充分考虑开发人员的意见，在发现开发工作发生偏差的时候，果断推翻了已有的设计方案，并且允许出现开发人员暂时过剩的情况。非常幸运的是，在最初的项目计划中，成功发布产品比设计人员的自尊心和节省员工工资更重要（当你的公司有人手闲置现象时不妨参考一下）。我们学到的另外一个方法是，先以一小部分人设计系统结构，然后分派小组开发各子系统。

威尼弗雷德项目：疏忽大意自以为是

项目名称	威尼弗雷德
主题	增量式开发
开发人员规模	中（20-40）
项目类型	改写原有项目
员工开发经验	没有 OO 经验，有顾问
开发周期	2 年

威尼弗雷德项目的任务是将一个原有大型机应用系统改写成采用3层结构的客户端/服务器应用系统。工作站端软件采用Smalltalk开发，使用C开发关系型数据库端的应用，主机应用程序的开发则使用COBOL。项目主管、工作站应用部分主设计师和主机应用部分的设计人员都具有丰富的OO技术经验，项目由一个资深项目经理管理。另外，公司还为项目提供了OO技术的培训、先进的OO开发工具以及OO技术专家顾问；项目开发资金充足，得到了管理层的全力支持，并且采用了增量式的开发策略。项目要达到两个目标：一是发布新产品，二是通过项目开发工作培训新来的员工掌握对象技术。

项目启动不久就在员工士气方面遇到了问题。开发小组内部和小组之间不能进行很好的沟通，OO顾问只是机械的授课而没有了解具体情况。需求阶段在经历了大约4个月的时间终于完成了，但大部分开发

人员已经对无休止的等待感到厌倦。任务分派不是很明确，技术领导经常被更换。士气和沟通的问题在项目进行了将近一半的时候也没有得到很好的解决。

大部分编程工作是由新手完成的，开发组中的高手则负责指导工作，分派开发任务和培训他们。最初，开发人员是按照发布功能分组的。所有类都不是由专人设计的，因此，这些类的设计集中了不同人的思想，过于复杂的设计导致没有人愿意使用这些类。

第一个增量阶段几乎没有什么可提交的产品，主要是因为系统内部结构很糟糕且实现的功能很少。在第二个增量阶段，专家级的雇员都被更换了，并成立了新的开发小组，同时加强了内部系统结构的设计。第二个阶段成功完成了，并第一次拥有了一个内聚性很高的系统结构。在随后的增量阶段中，系统结构被进一步完善，开发小组能够按时发布功能完善的系统，开发人员也越来越有信心。

为了适应经常变化的需求，在每个增量阶段又引入了一个迭代过程，每个过程实现的功能需要用户小组进行两次评审以确定最终的需求。在整个开发过程中，管理层为项目提供了很好的支持，用于购买设备的经费很充足，并且，客户小组很公正且稳定，并能随时进行交流。

当项目经理被问及她认为对象技术在维护和生产效率方面存在的问题的时候，她说Smalltalk小组完成变更所需要的时间比她以前的经验要快得多，但处在同一迭代周期中的的主机COBOL小组或者短期雇佣的数据库小组却要落后于进度。作为一项研究结果，他们改变了Smalltalk小组的迭代规则，每个增量阶段由三个迭代周期组成，其他小组在获得Smalltalk小组最终的规格说明之前将获得一份草稿。

威尼弗雷德项目带给我们的第一个经验是不要忘记以前的经验教训：明确的工作分工、良好的沟通渠道以及产品开发负责制是十分必要的（见第3章）。第二是在士气低下的时候，高层管理人员应该为他们提供的坚定的鼓励和灵活的管理。第三点是采用增量式开发方法（见第5章），这会使开发小组有机会发现他们的薄弱之处，并能够重新组织开发人员和设计出更好的系统结构。还会逐渐增加开发人员的信心和士气。通过三次迭代，他们知道有能力并且知道怎样在每三个月完成一个子系统。这使得即使在时间压力很大的情况下，他们也能够灵活运用自己的开发经验。

第四点是一定要让用户始终参与开发工作（见第5章）。在第5章中将对威尼弗雷德项目进行更详细的分析。

有多少关于对象技术的成功或失败的例子？常规软件工程开发方法有什么不足？本书第一个目标是提醒你不要忘记你在非OO项目中所熟悉的一些规则。对某些读者来说，可能在转向对象技术的时候，他

们才第一次使用这些规则。

学会如何扫除障碍才能提高开发效率

对象技术可能带来的收益

你可能听说过一些对象技术可能带来的收益。下面11节的内容是我和很多项目经理、行业顾问以及专家进行讨论，并且阅读了大量的项目报告之后总结的对象技术的收益。我将按照可能的影响程度、可行性和价值排列它们。

产品演化能力

当你的下一个产品和前一个产品相比变化不大的时候，面向对象软件的优势就会体现出来。OO设计技术就是为处理这种情况而产生的。你可以为一个设计良好的类或框架增加新的功能，用于实现新系统与旧系统的不同之处。

如果你的公司因为不同的需要（比如为了促销、财务方面的原因以及保障策略等），想尽快发布同一个软件的不同应用版本，那么对象技术就很适合你们了。

软件变更的能力

由于对象技术的封装特性（见第1章相关内容），大多数变更能够很容易实现。封装设计使软件的变更局限在很小的范围内。部分OO设计的任务是标识出可能的变更，并将之局部化。本章后面提到的汤姆摩根的 “见证人帐户”项目是一个很好的关于封装的例子。

坚持一贯的封装设计思想，当你遇到预料之外的或重大的需求变更的时候，一切都会轻松解决。这些变更可能是更换硬件平台、更换服务器或服务器的某项功能，或者是用户界面的风格。我之所以将这些需求方面的变更与维护阶段的变更和为了完善软件而发生的变更区分开来，原因是它们发生的频率、实现它们所需要付出的努力以及成本是有很大不同的。

抢占市场

如果某个原因促使你转向对象技术，那很可能就是它能使你的产品迅速推向市场。大多数系统都会有一些内部相似点 -- 是指同一个软件系统内部存在的相似部分，而不是与其系统相比较。采用增量式开发

方法，开发小组能够很容易发挥对象技术的两大优势（封装和产品演化能力），利用软件的内部相似性快速完成开发工作。

在第一个增量开发阶段发布的初级产品中，基类和框架已经被建立。利用内部相似点，在后续的开发阶段，你只需要在已有类和框架的基础上做些完善扩展工作。快速地将软件产品推向市场，取决于你发现的软件内部和外部相似点的数量。

开发人员、用户和管理人员之间的沟通

对象技术促使开发人员在工作的时候能够使用用户和管理人员的术语。一般情况，在一套软件的开发工作中存在两个转换过程：一个转换是开发人员将用户或管理人员的要求转换成能够用编程语言实现的设计方案。另一个转换是将程序内部功能转换成用户界面上可使用的东西，供人使用。这两个过程容易产生错误，并且也增加了开发成本。如果有一个简单的转换过程，将大大节约开发成本并提高开发效率。

对象技术就是一项允许开发人员直接使用用户术语的打包技术。因而，开发人员可以讨论“到底是汽油表发生了故障还是油路发生故障”，而不是枯燥的讨论磁盘上的数据结构。尽管将这点作为对象技术的一个优点有些令人觉得奇怪，但很多已经成功转向对象技术的组织不断的证实，对沟通方式的改进是对象技术带给我们的重大好处之一。

可维护性(Maintainability)

你可能会遇到这种情况，现有的程序代码陈旧，结构不合理，很难修改。你希望新的软件具有很好的结构并易于维护。在“软件变更的能力”一节讨论的封装设计对你来说会是一个好消息，它能增强你的系统可维护性。

一个结构很差的软件本身就是一个新的风险。如果需要建立很多类来实现系统的某个功能，你将很难理解它是如何工作的。每个新加入的人都需要花费大量的时间来了解系统工作的机理，由于理解不同，在修改类的功能的时候，大多数人都会犯一些错误。尽管OO设计不会自动增强程序的可维护性，但它提供了增强程序可维护性的能力。

复用（Reuse）

你也许希望在后续的项目中，通过软件复用而取得巨大的收益。在对象技术中，复用是指使你原有的软件能够“照常营业”，这倒是个好的消息。对象技术提供了复用机制：类、继承、多态和构架。理论上，

允许复用软件中的所有部分，在开发一个新软件时，只需要开发与原有软件不同的部分即可。

不好的消息是，软件的复用仍然受人为因素的制约。首先，要求开发人员能够压抑强烈的自我意识，能够发现并使用别人的开发成果；其次，要求经理们允许花费时间和金钱去建立可复用的软件构件。复用在一个项目内是很容易的事情，跨项目要难一些，如果跨组织就是非常难实现的了。对象技术本身并不能减少人为因素的影响。

生产效率

也许你正在面临一场因效率低下而造成的生产危机 ，希望通过OO技术提高每个人的生产效率，并最终按时交付产品。我可以信心十足地将我的钱投向任何一个经验丰富的OO程序员。但其它的程序员则不行，我的这种信心取决于两点：丰富的OO经验和很高的编程水平。

请记住一点，编程所占用的时间只是整个开发周期中很少的一部分，收集/分析需求，系统测试，导入数据，安装系统以及培训所占时间比例与编程所需的时间大致相当。如果想减少编程所占用的空间，你需要雇佣有至少12个月OO编程工作经验的程序员。不过从另一个角度来看，如果你的员工没什么OO经验，正好可以在项目开发过程中获得。但是，就别指望生产效率会有多高。

基于 Windows 的用户界面

面向对象技术在提高用户界面美观方面并没有太大的帮助。同样的界面，可以使很多种语言实现，用户并不关心这点。事实上在20世纪90年代中期，很多OO开发工具的界面都是由非OO语言实现的！我的观点是不要为了获得一个“OO风格”的界面而转向面向对象技术。

这么说，如果使用对象技术编写用户界面就没什么好处啦？不是，在20世纪90年代，开发基于Windows的软件是非常耗费人力的事情，OO技术是少数能够有效解决这一问题的技术之一。用户界面生成器使建立界面变得容易，但降低人们对学习OO编程的兴趣。所谓的拖放式（dray-and-drop）界面很难被重建，但却非常适合以对象划分代码的编程方式。

这些矛盾使得编写用户界面变得困难，因此，出现一些工具来简化这一工作。但这些工具并不十分完善，还有待进一步发展。采用对象技术的理由之一是简化用户界面的开发工作，但这必须依赖一些现成的工具。

士气

如果你仅是为了增加开发人员的士气而引进对象技术，好象不太现实。但它确实会带给人们快乐，这一点确实也值得考虑。大多数程序员会发现OO开发环境比他们以往所有的开发环境都显得高级，开发任务也因此变得更有意思。他们会对未来充满信心，因为他们正在使用最新的技术。这对于一个工作在对技术十分敏感的行业的人来说是非常重要的。

正如我们将在第三章中讨论的关于员工的话题，很少一部分员工对OO技术产生不满。有两类人可能会在转向这一新技术时遇到麻烦：一类是某种语言的专家，他们不能承受因为重新成为初学者而受到的压力。另一类是对瀑布（Waferfall）开发模型产生依赖性的开发人员，他们已经习惯于“给我需求，走开，我来编程”的工作方式。大多数员工会很乐意接受这种转变，但有些却不是。

代码自动生成

在广告中一些工具提供商宣称使用他们的工具你的程序将由模型直接自动产生。因为分析和编程的共同媒介都是对象。事实上，在我们最近的一个项目中，经过评估，那些自动生成的代码有5%-15%不太令人满意。这些代码主要来自业务模型生成的结构类和用户界面类。这些类的行为部分仍然很难用非手工方式编写，其难度和用户界面与基本功能之间的连接程序差不多。软件开发过程中可能会遇到的问题，有半数以上在编写这些代码时都遇到了。这到底在说明什么？我要告诉你的是，别指望按下某个按钮就会拥有一个自动从分析模型创建的系统。

软件过程

在很多软件组织中，不同的开发小组进行合作会遇到很多问题，因为他们的合作是以开发人员的个人能力为基础的。对象技术并不能改变这一点，所以，如果你的项目已经延期，并且没有一个很好的软件开发过程，那么我还是建议你不要采用对象技术。

原因是，对象开发需要进行更多的沟通，一个项目小组对他们这方面经验的描述是“靠别人的钱谋生”。对象技术强调变更的影响最小化，即使这意味着增加人员之间和项目之间的交流。因此，别指望对象技术会改进你的软件开发过程。当然，通过它可能会促使你建立一个全新的或者更易于控制的软件开发过程。

使你的软件具有按照某一风格演化的能力

以下是“见证人帐户”项目的描述，该项目采用封装技术实现产品的演化，以降低开发成本。该项目的负责人是汤姆·摩根，他自20世纪80年代以来任B.U.G公司的OO项目结构设计师。

OO 设计。封装和系统演化

汤姆·摩根，布鲁克林联合汽油公司

在编写本方案之前的几周，我们组织了一大规模的共有 50 人参加的测试。整个测试覆盖了软件投入使用之后将会用到的大部分具有代表性的情况为了谨慎起见我们引入一个中立的观察小组以确保测试结果的真实性。观察小组中有一个 CICS 方面的专家。

在测试中我们发现进程工作管理模块存在大量内存泄露;这可能引起 CICS 模块在一到 2 个小时内耗尽所有内存资源。

在一个标准的 CICS 事务处理系统中，进程工作管理模块所需要的代码几乎分布在所有程序代码中，那个 CICS 的专家很负责地向项目经理报告说，修复上述内存泄露问题可能需要几个月。

但是我们第二天上午就已经解决了这一问题。因为我们合理地采用了一项 OO 实现技术（见第一章封装]。进程工作管理正如程序内的其它部分一样是作为一个对象的行为被实现的，这一实现被该行为的所有使用者所共享。只需要改动引起泄露函数并重新编译一下就可以解决这个影响到整个系统的错误。

案例文档，由汤姆·摩根提供并获许在本书中使用。

这个项目很具代表性。发现的问题之所以能够被很快解决，关键就在于它只存在于一个软件模块内，由此可见封装设计能够减少变更所需的时间。

成本

既然看到了上述的收益，现在我们该谈谈你需要付出什么了。也许你愿意付出1到2周的时间花在一门新编程语言的培训上，也可能你打算花钱购买一套新的CASE工具。并且正在寻找降低这两项花费的途径。

你是否低估了些什么？

请牢记这点，肯定会有些东西被你低估。后面的7个小节，我们将讨论这些容易被低估的东西。

使开发新手熟练掌握 OO 技术所需的时间。

这是指一个新开发人员掌握OO技术并能熟练编程所需时间的长度。如果你原来认为需要2周，请改为8到10个月。

OO 工业的不成熟度

尽管在几十年前就有了对象的概念，但它进入商业信息科学（IS）市场的速度很慢。有些已经被视为标准的东西，在IS界仍然不完善；典型的例子是一些金融和事务处理的类，建模和编程的版本控制工具等。

C++的风险

C++是一种编程语言，它们处于联接对象和C语言的战略要道。其它类似的语言还有objective-c, som, c++以及java。C++是编程语言竞争的胜利者，也许是因为它的兼容性和性能，但它确实是一门很难掌握而且复杂的语言，因为某些原因，一些大规模的IS项目仍然采用C++语言，即使有些项目的开发人员还不能完全掌握它。

一个有经验的开发工作领导者曾经在几年中使用C++语言开发过一些中等规模的项目，最后，还是放弃了C++，他对我说；“我认为使用C++语言什么象样的项目都完成不了。你应当写本书警告人们这点… 但他们不会相信你，就象我当初一样。”

我不同意他说的C++完成不了什么象样的项目这一观点。我见过很多成功的例子，也有很多地方在使用它。 但C++所带来的挑战还是十分严峻的（见第3章）。与之比JAVA语言因其容易掌握和可移植等特性，越来越多被人们所重视。

软件复用的难度

在开发人员中实现软件复用是件很难的事情，你需要减少期望值，增大对开发人员的培训力度，并加入适当的检查。“复用”作为一个概念再简单不过了，但它比听起来要难得多。我这么说的目的是希望你建立起一个“软件复用并不是很容易实现”的观念。

建立一个软件开发过程

使开发人员遵循一个开发过程很不容易。建立一个过程失败的原因有两个：一是该过程本身可能就是错误的，二是开发人员没有能力（或者不愿意）遵循它。希望你的人少走些弯路。

业务建模与软件设计

业务建模和软件设计之间有着很大的区别。尽管有许多美好的愿望，但一个精确的业务模型不一定能

产生一个很好的软件设计方案。将一个清晰的业务模型转化成一个好的设计方案需要深思熟虑及创造性。软件设计时必须考虑可修改性和系统性能方面的限制。

一个最初的业务模型被清晰的建立之后，你仍然需要很多努力才能获得一个很好的软件设计方案。如果你的设计人员表现不错的话，最终的设计不但能够完全实现业务模型，而且能够以足够快的速度运行，并能够做到变更局部化。当然软件的设计与最初的业务模型大相径庭也完全有可能。

CASE 建模工具的成本

建模在整个开发工作中只占很小的一部分（见收益相关内容）。CASE工具提供商和研究人员对他们所“宠爱”的工具的能力自然是十分乐观的，但历史是不一定站在他们的那一边的。你需要做的是计划如何减少那些由对象建模工具自动生成的代码。

可能的成本

以下是一些比较真实的在时间和金钱方面的成本：

- 培训：为每一个开发人员提供3-6周的课堂培训。
- 经验：在每个开发人员学会并掌握以OO思维方式思考和工作之前，你可能需要等待6-9个月。
- 工具：忘记你对现存CASE工具的投资。如果一个工具不是为做某件事情而设计的，你强迫它去做只能使它沦为一个昂贵的绘图工具。购买正确的CASE工具。
- 顾问：优秀的顾问要价昂贵并且很难请到。出于这些原因，那些有着6到9个月工作经验的所谓专家还是有一定的市场的。可是，你需要经验。你可能会犯错误，并从中吸取教训，或者雇用一些有经验的人来掌握项目的方向，使你避免犯错或在犯了错之后改正它们。这两种情况都需要你会出代价，但后者所需的时间更少。

这里，所有的坏消息都摆到桌面上了，你可以随时看到它们。你该怎么办？为了使你的组织能在未来的几年里稳定增长，你做好准备了吗？

非对象解决方案清单

事实上，项目管理很大程度上并不依赖对象技术。收集和理解需求不依赖这项技术。没有它，测试、

编写、文档和开发工作也一样要完成。20或更多的人一起工作需要一个良好沟通机制的项目框架。

以你现有的工作方式为基准。下面的清单列出了与对象技术无关的所有内容。第一个清单是项目中的常见活动。第二个清单列出了对象技术需要的工具和开发过程。

标准项目活动

- 与用户和项目负责人交流，决定项目需求
- 研究和归档现有系统，列出所有新系统必须匹配的接口。
- 培训开发人员和项目经理。
- 分析用户的愿望和需求。
- 就不同的主张和误解进行讨论。
- 建立一个技术结构。
- 开发一个原型，并在获得用户反馈后重新评估需求。
- 建立测试条例
- 协调各开发小组之间的工作
- 编写文档
- 在产品最终发布之前，对系统做最后的调整
- 发布产品的alpha和beta版并改正这两个版本中发现的错误。
- 准备培训材料。

工具和过程

- 技术需要—版本控制，配置管理工具LAN链接，e-mail以及类似的东西。

- 技术人员—COBOL程序员C程序员，网络专家和数据库设计人员。
- 过程的需要—产品发布的形式；开发小组之间的交互行为。
- 人事需要—小组组长，文档编写人员，测试人员以及小组内部的沟通机制。

以上两个清单可以使你有信心（即使你没有面向对象技术的背景）管理好一个面向对象的项目，但有些地方还需要重视，特别是在第5章中我们将要讨论的小组结构。

然而，确实有很多因素造成OO技术很难应用，比如：复用、增量式和迭代式开发，以及许许多多关于面向对象技术的神话故事。

对象系统仍然需要与历史遗留下来的非OO主机应用系统进行“对话”。但这并不会破坏你现有的项目；这只是接口设计的问题。请确信你没有这样的设计师或者有好几位这样的人手，但他们之间存在分歧。那么，你需要尽快摆平这一切。

可以考虑将你原有的COBOL系统重新构造成一个新的OO系统。现在有很多功能强大的COBOL分析和构造工具以及相关的服务也使重构工作得以顺利实施。作为系统重构的成果，你会发现COBOL系统接口和归档以及模块化分工工作将会变得非常简单，同时面向对象系统的版本的尺寸会大大的缩小，这点会为你节省很多钱。

你的结构设计师可能会建议你以完全不同的方式重新开发原有主机系统上的OO代码。一些主机系统专家曾经跟我谈过，以“对象为中心”的设计才是真正有效率的，意思是服务器上的程序应该以工作站的数据访问请求为中心的。数据访问请求是类结构的镜像。因为这些请求是由类产生的。很有趣的是，这一设计方式的建议并不是由OO 开发人员而是由主机系统专家最先提出来的。

不幸的是，主机系统和工作站开发人员并不是使用同一种工作术语。并且常常在协商方面出现问题。这就要求你的结构设计师和技术主管们能够善于处理任务分配工作。

通过你以前的经验避免大多数常见的陷阱。

查看评论

UMLChina 讨论组最新热门帖子 TOP5

对软件工程的一点看法--希望能够抛砖引玉　- <2744b> miaolin 2001/11/22 10:41 (627 次点击)

一个 java 实现上的具体问题　- <351b> joy_wind 2001/11/09 12:30 (246 次点击)

什么是 OO 方法，为什么要用 OO 方法，怎么用 OO 方法？？？　- <473b> mouri 2001/11/05 17:52 (164 次点击)

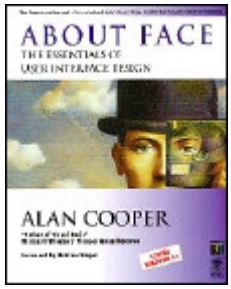
CMM vs Microsoft　- <767b> axial 2001/11/04 23:24 (122 次点击)

一个纯 UML 问题　- <636b> sealw 2001/10/31 16:49 (369 次点击)

消息：UMLChina 第十二期专家交流

北京时间 2001 年 12 月 15 日（星期六）上午 10:00-12:00

嘉宾：Alan Cooper，软件交互设计的先知和传播者，客户咨询公司 Cooper Interaction Design 的创始人和总裁。第一个可视化开发工具 Visual Basic 的发明者。交流重点：软件可用性，交互设计，其它...



要更多了解 Alan Cooper 的思想，请看以下资料：（1）《非程序员》第七期的第一篇文章“交互设计之父 Alan Cooper 访谈”；（2）去书店买他的著作：《软件创新之路--冲破高技术营造的牢笼》

详情请见>>