面向对象程序设计

——一种循序渐进的系统构造方法

杨芙清　陈钟　章远阳　编译

北京大学出版社

新登字(京)159号

面向对象程序设计
——一种循序渐进的系统构造方法

杨芙清　陈钟　章远阳　编译
责任编辑　李采华
*
北京大学出版社出版发行
(北京大学校内)
北京印刷三厂印刷
新华书店经售
*
850×1168 毫米　32 开本　10.875 印张　280 千字
1992 年 3 月第一版　1992 年 8 月第一次印刷
印数:0001—3000 册
ISBN 7-301-02012-0/TP・165
定价:9.80 元

内 容 提 要

本书从面向对象程序设计的角度论述了如何利用可重用部件——软件IC构造大型软件系统的原则、方法和技术。作者深入浅出地介绍了面向对象程序设计中对象、消息、类、类簇、封装与继承等基本概念，详细描述了如何将这些概念与传统风格的程序设计语言(如C语言)相结合，形成具有面向对象特性的混合式程序设计语言(如Objective-C)，从而可以像硬件集成电路那样，建立商品化的可重用性软件IC库以及相应配套设施，支持大型复杂软件系统的构造，提高软件质量，降低软件成本。作者还通过采用软件IC技术和传统技术分别构造同一个应用实例的方法比较了两种技术之间成本和效益的差别。最后作者提供了一个基于Objective-C语言的实用软件IC库，并讨论了更深入的课题。

本书可以作为广大软件工作者研究、学习和实践面向对象程序设计的入门书，也可以作为从事实际工作的软件开发人员运用面向对象程序设计技术进行软件设计与实现的实用参考书，还适于作为大专院校计算机软件专业本科生、研究生的教材或教学参考书。

第一章 系统构造

本世纪40年代世界上第一台电子计算机的诞生，无疑是人类文明史上一次新的技术革命。半个世纪以来，计算机技术已经渗透到人类生活的各个角落，成为人们用来解决客观世界问题的有力工具。但随着计算机应用广度和深度的不断扩大，人们逐渐感到计算机应用的"灵魂"——软件，却愈来愈难以驾驭。这一矛盾的加剧，终于在60年代末爆发了所谓的"软件危机"。软件危机的挑战，促使人们认识到：软件作为一种智力产品，必须用工业化的方式开发和生产。这一认识在随后的二十多年中成为人们进行大规模软件开发活动时考虑问题的出发点，并由此产生了诸多的概念、模型和技术，形成了以开发大型软件为目标的"软件工程学"。

既然软件这一智力产品的开发和生产，必须以工业化的方式进行，那末首先就让我们追溯一下历史，考察一下工业革命对(硬件)产品的生产(如火器制作)带来的变革，然后再看看能给我们今天的软件生产带来哪些启迪。

在工业革命以前，火器制作只是工匠们独自进行的一种手工作坊式生产，远远谈不上是一项工业。工匠们"各行其是"，独立完成火器制作的所有工序。这样制作出来的火器成品，不仅价格昂贵，而且规格、功能和质量各不相同，完全取决于工匠个人的手艺和经验。

二百年前，美国的康涅狄格州以火器制作和纺织业而闻名。许多火器制造公司今天依然耸立在那里，但纺织业在很久以前就已移居它处，昔日的厂房已被众多的现代化企业大楼(如PPI软件公司)取代。1798年，艾利·惠特尼(Eli Whitney)采取了一项引起制造业革命的措施，而今在软件业也正进行着一场类似的变革。

1798 年，当轧棉机的发明者——艾利·惠特尼在接到政府的一大宗步枪生产合同后，他决定对陈旧的火器制作方式进行一场变革。惠特尼先生采用的办法是将整个制作过程分为若干道工序，每道工序遵循一定的标准生产某一部件，并由专门人员来完成。于是工匠们就可以在其专门从事的工序内各尽其职，各显其能，以提高相应部件的质量。这种将手工作坊式的制作改造成批量生产的方式，不仅降低了产品的制造成本，更重要的是促使政府部们很快地认识到：制作标准应该统一，使相同部件之间可以互换，这样就可大大简化武器的维修问题。

工业革命导致了产品的大机器批量生产，其中关键的前提是：产品部件的可组装性、产品部件的标准化及由此而具备的可互换性。目前，软件的生产方式也正在发生一场类似的变革，这就是面向对象程序设计(Object-Oriented Programming，OOP)方法。OOP 方法和上述惠特尼先生在火器制造方式变革中所采用的思想是一致的：产品的部件应具有可互换性。基于此，应在软件产品的开发中，将软件分割成若干模块单位，各开发者分别为相应模块编制子程序，而不必"统包统揽"。这些作为软件产品部件的子程序遵循一定的接口标准，在不同的软件产品间可以互换。应用程序员不必再按照将程序设计语言"逐句拼装"的方式来构造整个软件，只需组合、重用由系统程序员开发、可供他人用来装配的可重用软件部件即可。

借助于硬件领域中的概念，我们将可重用的软件部件称为软件集成块(Software IC)。计算机硬件工业由于采用了与上述类似的革新方式——IC 技术，在过去的二十多年中获得了长足的发展。在软件生产发展相对落后的今天，采用软件 IC 技术，无疑同样会使软件生产发生一场变革。本书正是试图采用循序渐进的方式来描述软件生产中的这场变革。

1.1 系统构造(System building)与软件危机

系统构造的目的是要开发出一个符合用户要求，并在用户要求及系统环境发生变化时能予以扩展(升级)的系统。但现实中的情况如何呢？Objective Solutions International(OSI)公司是一家资金雄厚、雄心勃勃的软件公司，其主要开发的领域是办公室自动化(OA)，以期一改传统办公室，如政府、银行、保险公司等办公室中纸张充斥的景象。OSI 公司接到的一个典型的 OA 项目是要求实现一个反映当前技术水平(state-of-the-art)的系统。该系统要求采用分布的工作站和高分辨图像(high-resolution raster graphics)技术来处理任意一种办公用的表格，如留言条(WhileYouWereOutNotice)、备忘录(OfficeMemo)、价目表(PriceList)、保险申请表(InsuranceClaimForm)、租赁申请表(LoanRequestForm)、驾驶执照申请表(DriversLicenseApplication)、旅费收据单(TravelExpenseVoucher)等等。这些表格的花样层出不穷，因为办公室处理的数据类型集合在不断地变化。

该项目的客户决定花大价钱来解决其办公室自动化问题，但考虑到目前用户计算机技术的水准较低，因此除要求该系统在响应速度及可靠性等方面达到要求外，还需尽可能地保持目前人工操作的特点，希望电子报表和真实报表一样具有预定义格式和填写规则。相比之下，传统的电子邮件式的标准正文编辑器就难以满足用户的这一要求了。此外，某些客户还期望系统能支持集正文、声音、图像于一体的多媒体管理。

OSI 公司计划采用快速原型(rapid prototyping)技术实现该系统。首先抽取一些办公室具备的公共概念，并以此实现一个原型，然后根据用户的具体要求对原型再作进一步扩展，并保证按用户要求的变化，使系统升级。起初该系统的原型只支持一些由 OSI 公司选择的常用报表，而大部分特殊报表处理功能的开发是随时追加的。

该系统的软件结构应保证易于构造和扩展，以使系统升级后，原有的工作不至于白费。这样的系统不仅减少了纸张处理费用和操作人员的培训难度，而且促进了办公过程的自动化，当然会受到客户的青睐，同时对 OSI 公司本身也有利。原型可以吸引新的客户，因为客户们希望在原有系统的基础上，通过不断追加投资使其升级换代。OSI 公司认为：用户的要求是无止境的，只有采纳快速原型技术，才能持久地占领高利润的市场。

这个庞大的系统对程序员们来说真是极其诱人：分布的高分辨率图形工作站、图符式(iconic)用户接口等等都是颇具吸引力的新技术。为此，他们满怀信心地制订该系统的实现计划：采用专用的报表生成语言生成报表；采用基于规则的人工智能技术实现办公过程的自动处理；采用适于快速原型的开发方法，并附以结构化方法学的支持等等。这些梦想能实现吗？

以上这一切似乎都是自然而然的。但图 1.1 中的信息却向我们表明：目前软件工业界能交付用户使用的系统只占整个开发系统中的很少一部分，而且效果并不佳。上述项目的实现计划只是程序员们一厢情愿的梦想，最后的结果甚至会是：无法交付给用户一行正确的源代码，项目计划只得取消，使客户失望；OSI 公司的效益受损，使程序员们感到有失体面。这一切也正是软件危机所具有的类似症状：预算超支，进度延期而无法控制，质量低劣，最终导致取消项目计划，信誉受损。

原因在哪里？硬件当然不成问题，因为当今的硬件技术足以支持象 OSI 公司上述 OA 项目的这类系统甚至更复杂的系统；同样，问题也不在于分布式系统技术，面向报表的用户界面是否友好以及电子邮件技术、存储系统(archival system)和工作站等单项技术本身。通常，我们总是单独应用以上领域的技术，但要将它们综合应用，在现有的软件开发水平上是会引起一些问题的。

问题的本质在于我们现有的程序设计工具、方法及概念等并不支持如何在外界条件发生变化的条件下进行软件设计。OSI 公

司的 OA 项目在某些方面的要求超出了当今软件技术的发展水平，因为软件工业界还未探索出一套能适应外界因素变化的软件系统方法。

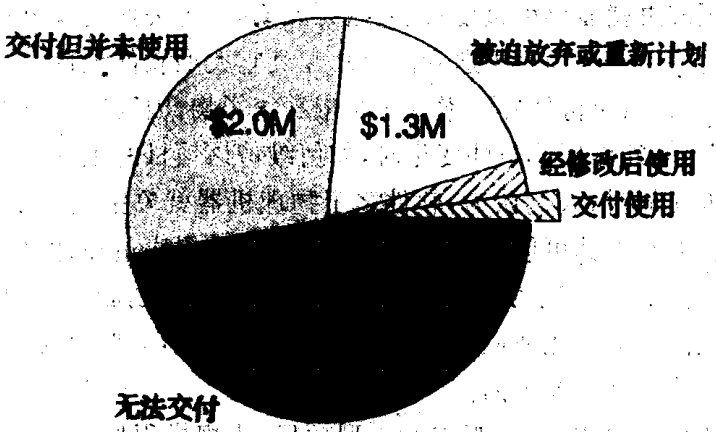

图 1.1　软件的开发效益

（美国国家统计局 1979 年度报告(FGMSD-80-4)表明，软件开发现状不容乐观：在总投资为 680 万美元的 9 个联邦政府软件项目中，有 47%(320 万元)投入后连产品都无法交付，有 29%(200 万美元)交付给用户但未使用，有 19%(130 万美元)的投资计划被迫放弃或重新计划，3%(20 万美元)的投入所得到的软件经修改后可以使用，只有 2%(10 万美元)的投入才产生真正能交付使用的软件(ACM Sigsoft Software Engineering Notes，第 10 卷，第 5 期，1985 年 10 月))

系统分析员(system architect)不应涉及具体编码，对于 OSI 公司的上述 OA 项目，许多系统分析员会认为，该项目的负责人应该从一开始就进行系统的软、硬配置及其界面分割工作：即确定系统的硬件配置(如工作站、微机和主机(mainframe)等)，各类硬件上必须完成的软件功能(如报表编辑、文件的存储及后援服务等)，再

用合适的网络将它们连成一体。

但作为系统分析员，首要的是分析该项目的技术可行性，当然还要采取一些传统措施，如保持文档整洁、结构说明清楚、软硬功能分解等，但这些措施只治表不治本。例如在打猎时，如发生疟疾，则消灭蚊虫就显得很重要；但在遇见大象时，再去捕捉蚊虫就显然不合适了。这个项目中遇到的挑战和其它大型软件项目中所面临的一样：如何应付外界因素变化所带来的影响。因此应该将主要精力放在：在外界环境条件变化时，如何维持该项目的生存和升级。

曾几何时，人们认为：软件之于物理机器就象铅笔之于纸，软件的魅力在于其可创造性、可延展性。软件是程序员在计算机这种物理媒介上，相应于实际需求的设计思想进行的创造性活动，这种创造性活动应是能适应实际需要变化的。因此，只要可能，软件总是应能响应外界要求的。

从目前看来，上述观点对小型项目（小规模程序设计，即程序构造）来说，在某种程度上是正确的；但对大型项目（大规模程序设计，即系统构造）来说，已证明是错误的。事实上，在系统诸元素之中，软件部分适应外界因素变化的能力是最差的。程序构造和系统构造之间的主要区别在于两者对外界变化的适应性不同。因此，如果说小型项目（程序构造）对外界变化的适应性尚可的话，那末大型项目（系统构造）则不行，因为前者程序量小，目标单一，易于适应条件变化，显得灵巧；而后者程序量大、复杂、脆弱，对外界的干扰抵抗能力差，故对外界变化的适应性也差。针对这种情况，出现了若干学派，企图解决这类问题，其中最主要的是当今流行的结构化软件设计思想。

1.2 对策

既然对变化的适应性问题是解决大型软件开发的关键，为此我们考虑以下三种不同的策略：封闭式策略、开放式策略和综合策略。这同时也反映了人们对软件生产过程认识的不断深入。

1.2.1 封闭式策略

应付变化的传统方式是建立复杂的保护机构，以避免外界变化的发生。正如软件管理者熟知的：软件开发，尤其是大型软件的开发过程，首先是确定系统需求及其文档，将文档变换成说明并交用户，用户改进后认可签字；一旦确认了需求，就进行一系列的设计、复审过程，并产生正式的设计说明；经上述一系列有效的设计过程后，最后将设计转换成代码，经测试后提交给用户。一切工作按计划仔细进行，用户理应对此感到满意，因为所交付的代码完全达到了当初协议的要求。但在用户和开发者之间经常会发生下列情景："什么，要改变需求？天啊！请看看文档，还有您的签字？这将需要重新设计和重新安排计划，真要如此，请再付钱！"变化摧垮了一切，前功尽弃！

复杂的保护机构最终还是无法抵挡外界因素的变化。通常，因外界需求变化而要求对系统的修改会落在维护人员身上。维护工作量虽然极大，但能干的程序员总认为这是一种"二线工作"，缺乏吸引力。

软件不像物理产品，它不会生锈或腐蚀，也勿需打蜡或清洗。软件中会出现一些需要加以注意并修正的错误，但这种排错(debug)并非是维护阶段的任务，而是属于维修(repair)的范畴。排错就是发现错误并进行修正的过程；相反，一旦软件的环境发生变化，就必须努力使软件适应这种变化。这种努力才是维护，它的目的是使软件的适应能力保持稳定，相应地使软件系统不断地改进、完善。

OSI 公司的 OA 项目开发人员也许会忽略项目的维护工作，而重视软件的维修，但这样显然不会使系统升级。维护的实质在于不断地改进、完善产品，使其适应用户需求的变化。诚然，在项目的设计阶段，也会采取一系列方法来适应用户可能变化的要求，如源代码注释，利用可读性良好的语言，废除 goto 语句，保持文档的及时更新等等。但这些方法都没有真正解决该系统的技术可行性问题。