第 1 章 模式与反模式简介

反模式代表了计算机科学和软件工程思想中一系列革命性变化中出现的一个新概念。尽管软件领域已经走过了近50年的开发可编程数字系统的历程，但是在人们把业务概念转换成软件应用程序时，仍然有一些基础问题有待解决。许多最主要的问题来自于开发过程，这些过程需要参与者有共同的愿景和协作来实现系统。大多数已发布的有关软件科学的著作都着眼于积极的、具有建设性的解决方案。本书不同的是，我们的研究将从产生负面影响的解决方案入手。

理论研究者与实践者们设计了数千种构建软件的新方法，从令人激动的新技术到各种渐进性过程。虽然有了这些很好的理念，但实践中管理人员和开发人员取得成功的可能性仍然不容乐观。一项对数百个集体开发的软件项目的调查表明，每6个软件项目中就有5个被认为是不成功的[Johnson 1995]，而且有大约1/3的软件项目被中途取消。未被取消的项目交付时所消耗的成本和时间几乎都达到了最初计划的两倍。[①]

“系紧你的安全带，今晚的旅程将会充满颠簸。”

——Joseph L. Mankiewicz（美国著名导演）

几乎所有交付的系统都是烟囱系统，也就是无法适应变化的系统。软件最重要的一点就是它的适应性。一半以上的软件成本都是由于需求的变化或者对系统进行扩展的要求所造成的。大约30%的开发成本是由系统构建过程中的需求变化造成的。

1.1 反模式就是揭露假象

软件本应该让数字硬件具有更多的灵活性，但是软件技术一直在传播着一系列未能实现的许

① 根据Standish集团2007年发布的CHAOS报告。软件项目取消率已从1994年的31.1%下降到2006年的19%；而项目不成功（指时间和成本超过预期、客户需求未完全满足）率从1994年的52.7%下降到46%，虽有进展，但并无太大改观。——编者注

诺。可以让硬件具有更多灵活性只是这些许诺中的一个。是什么出了问题？在我们的职业生涯中有无数软件开发的时髦技术来了又去。它们在软件开发的特定领域能够取得一定的成功，但是都无法提供本来所许诺能够解决一切问题的“银弹”（见图1-1）。还记得这些主要的开发潮流吗？

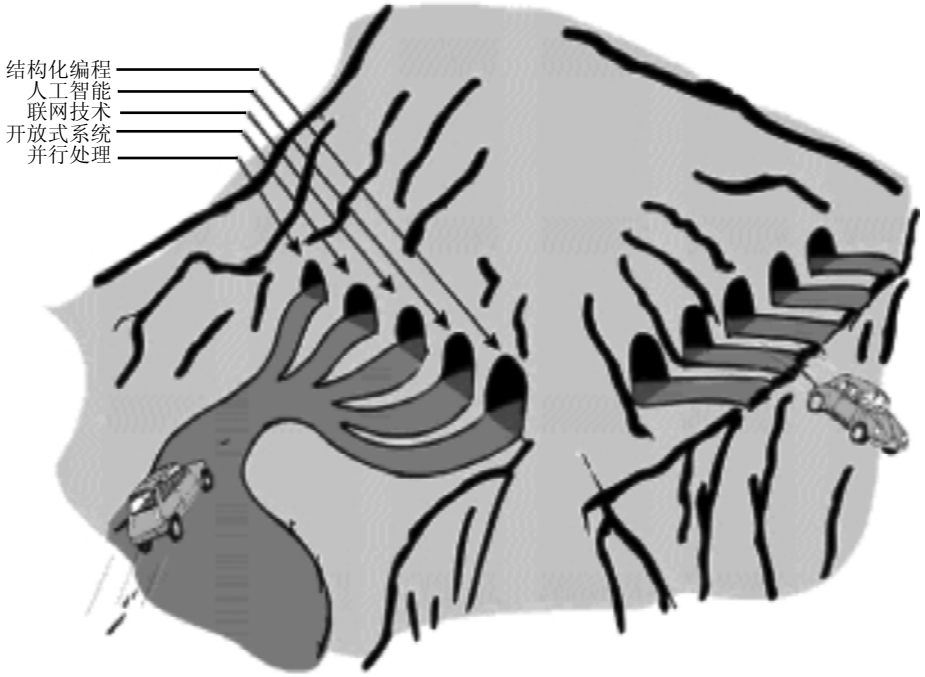

图1-1 各种途径都会走向灾难

- 结构化编程应该提高软件生产率和去除大部分软件缺陷。
- 人工智能应该让计算机具有更强的智力。
- 联网技术应该让所有的系统和软件可互用。
- 开放式系统与标准应该让应用软件可移植、可互用。
- 并行处理应该让计算机具有更强的计算能力而且可伸缩。
- 面向对象应该解决软件生产率和适应性的问题，让软件高度可复用。
- 框架应该让软件高度可复用，让软件开发具有更高的生产率。

这些听起来都像是在放一张破唱片；每次新的软件开发潮流都会做出类似的许诺。现在对许多正时髦的软件技术下定论还为时尚早，但是它们所声称的能力听起来和我们以前曾经听到的非常相似。目前的一些例子包括：

- 因特网。
- 组件软件。

- 分布式对象。
- 业务对象。
- 软件复用。
- 脚本语言。
- 软件代理。

我们要说出有关软件的真相，这个目的非常明确。欺骗与觉醒之间的不断循环似乎已经成为了软件技术的特色，而本书的目的之一就是结束这种循环。 5

尽早要说明的是，实际上并不只是软件技术供应商们有错。开发人员和管理人员可以采取一些措施来减少软件开发中的问题。无论供应商的宣传是什么，能否取得成功最终还是取决于使用这些技术的方式。

软件技术的真相

84%的软件项目是不成功的，而且几乎所有项目交付的都是烟囱系统。为什么会这样？

供应商会告诉你：

- 我们的新技术会改变整个软件开发范型。
- 业务对象可以让你的普通开发人员具有更高的生产率（可以用任何术语来替换“业务对象”）。
- 我们会在6个月内提供你需要的所有功能。
- 我们不能以明示或默示的方式提供对我们的软件的适销性或它适用于特定用途的保证。供应商几乎根本不会保证他们的软件能够做任何有用的事。如果他们的软件做了什么不好的事，那也不是他们的错。专有技术只要4~18个月就会改变。快速的技术变化会占据软件维护费用的主要部分，并对软件的开发产生冲击。在客户购买了一个软件许可证之后，供应商期待利用培训、咨询、支持和维护从同一个客户处获得4倍的收入。开发人员遇到的困难越多，供应商能赚到的钱就越多，有没有这种可能呢？

软件权威会告诉你：

- 他们的新方法改进了过去的所有问题。
- 他们的工具对包括代码生成在内的软件开发提供全面支持。
- 你需要更多的工具！
- 你需要更多的培训！
- 你需要更多的咨询！

我们这些权威彼此之间并不一致，而且我们的想法也经常会改变。权威们还会推销一些与他们过去向别人推销的想法相矛盾的新想法。权威们的方法学对任何一个实际项目来说都太一般化 6

了。这些方法中的重要细节都隐藏在他们的昂贵产品背后。例如，权威们所宣传的从建模工具生成具有产品质量的代码的能力，可能还需要好几年的时间才能实现。更常见的是，他们从来就不会说出重要的细节。

重点是，无论有多少这样的激动人心与欺骗宣传，软件技术实际上还处于石器时代（见图1-2）。开发人员、管理人员和最终用户正在为此而付出代价。

图1-2　软件技术还处于石器时代

1.2　反模式的概念

反模式是一种文字记录形式，描述了对某个问题必然产生消极后果的常见解决方案。由于管理人员或者开发人员不知道更好的解决方法，缺乏解决特定类型问题所需的知识或经验，或者是在不适当的环境中应用了一个完美的好模式，这些都会造成反模式。在采用适当文字记录下来的时候，反模式描述了一个一般的形式、其主要原因、典型症状、后果，以及一个如何把该反模式改变成更健康状态的重构方案。

*反模式是把一般情况映射到一类特定解决方案的有效方法。*反模式的一般形式为它所针对的那类问题提供了一个易于辨识的模板。此外，它还清楚地说明了与该问题相关联的症状以及导致这一问题的典型内在原因。这些模板元素完整地说明了特定反模式存在的情况。这个一般形式可以减少使用设计模式时最常见的问题：把特定设计模式应用于不正确的环境。

*反模式为识别软件行业反复出现的问题提供了实际经验，并为大多数常见的困境提供了详细的*补救措施。反模式突出了软件行业所面临的最常见问题，并提供了一些工具使你能够识别这些问题并确定其内在原因。此外，反模式为逆转这些内在原因的影响、实现有生产率的解决方案提供

了一个详尽的计划。反模式有效地说明了可以在不同层次上采取的措施，以便改善应用的开发过程、软件系统的设计和对软件项目的有效管理。

反模式为辨识问题和讨论解决方案提供了共同的词汇。反模式和与它们相对应的设计模式一样，为开发机构中常见的、有缺陷的过程和实现定义了行业用词汇表。一个更高层次的词汇表可以简化软件实践者之间的交流，并为更高层次的概念提供简练的描述。

反模式在可能的情况下在不同层次上利用机构的资源实现对冲突的全面解决。反模式清晰地声明要把处于不同管理和开发层次上的力量结合起来。软件开发中很多问题的根源其实在于管理层或机构层。因此，不考虑来自这些层次的力量，单纯试图讨论开发性模式和架构性模式是不够完整的。由于这个原因，我们在本书中以大量的篇幅把多个层次上的所有相关作用力集中到一起，来说明核心的问题区域。

反模式以分享软件行业最常见陷阱所带来的惨痛教训的方式提供了一种减轻压力的方法。在软件开发中，识别出有缺陷的情况往往比实现一个解决方案更为容易。在缺乏实现一个反模式解决方案的力量时，屈从于反模式力量所带来后果的个人只能通过了解到在整个行业中还有很多人也和他一样面临两难境地而感到安慰。在意识到某些反模式会导致严重后果时，反模式也可以成为唤醒受害人的警钟，提醒他开始寻找行业中的其他就业机会，写好简历准备另谋高就。

8

1.3　反模式的由来

设计模式语言迅速风靡了整个编程界，反映了软件专业人员对改善行业质量与标准的强烈愿望。由于使用和创建可复用设计模式而获得成功的项目在不断增长，设计模式所具有的内在价值是无可争辩的。但是，目前的范型无法完全表达设计模式预期的使用范围，从而导致出现了一种新的文字记录形式，它与现有的模式定义截然相反，这就是反模式。要完整把握反模式在软件开发中的重要性，就要先了解它的来源以及当前的设计模式现象如何导致了它的产生。

波特兰模式仓库（Portlan Pattern Repository）

波特兰模式仓库（http://c2.com/ppr/）公布了一个不断进化的设计模式和模式语言的集合。它目前是由Ward Cunningham和Karen Cunningham（Cunningham and Cunningham公司）主办，是一个有趣而令人激动的地方，可以在此通过因特网社区与其他的设计模式迷们进行有关设计模式的合作。在这个网站可以找到反模式的相关资料[①]。

设计模式的思想起源于Christopher Alexander。他是一个建筑师，创建了一种用于城市规划和城市建筑物构造的模式语言。他的模式语言清楚地说明了他对如何进行建筑设计的看法，并解释

9

① 网址是：http://c2.com/cgi/wiki?Antipattern。——编者注

了为什么某些城镇和建筑比别的城镇和建筑提供了更好的环境。他捕获专家意见的方法很有创新性，因为它让很多原来只有通过多年的城市规划和建筑经验才能得到的“软”属性变得清楚明确。

1987年，几个处于技术前沿的软件开发人员发现了Alexander的成果，开始使用模式来记录软件开发过程中的设计决策。特别是Ward Cunningham和Kent Beck开发了一种设计模式语言，用于在Smalltalk编程语言中开发用户界面。在接下来的几年中，他们吸引了一些具有类似思想并同样希望使用设计模式来帮助复用软件设计的人，开始在早期的设计模式运动中培养大家的初步兴趣。

把Christopher Alexander的成果应用到软件开发中有什么好处呢？回答这个问题的时候必须考虑到当时软件开发危机的背景。即使在20世纪80年代，具有面向对象软件开发才能的架构师明显太少，无法满足整个行业的需要。而且，理论研究者未能提供解决问题所需的详细知识，也未能设计出能够应对需求变化的动态软件解决方案，这在软件行业中并不稀奇。这些知识需要多年的行业经验才能获得，而行业中的紧迫需要限制了很多架构师，使他们不能花时间来指导缺乏经验的同事。而且，行业中快速的人员流动使得大多数老板不愿意让高级员工把时间花在指导其他人上面，而是让他们去解决自己的关键业务软件问题。这种做法产生了一种迫切而强劲的需求，需要一种可复用的形式来捕获有经验的开发人员的专家知识，让这些专家知识可以被重复用于培训那些缺乏经验的人。此外，还可以在行业层次上应用设计模式，开展有关建立行业范围的设计解决方案的对话，从而允许领域框架在行业层次甚至是全球层次上进行互用。

Christopher Alexander认为，虽然设计物理建筑物所涉及的大部分过程是不同的，但总会有一个通用不变过程隐含于所有其他过程之下，它精确地定义了设计和建造这个建筑物的根本原则。这样的不变过程就是软件开发中的圣杯，因为它们提供了通用的知识框架，从而可以在当前定制的、不能互用的烟囱解决方案沼泽上构建专家软件解决方案，而不是把专家解决方案又变成烟囱解决方案。

设计模式直到1994年才进入面向对象软件开发社区的主流视线。在1994年中期，Hillside Group历史性地主办了第一次软件设计模式方面的行业会议：Pattern Languages of Program Design（PLoP），会上重点展示了一些用于开发软件应用的设计模式和模式语言。特别值得注意的是Jim Coplien题为*A Development Process Generative Pattern Language*的论文，它是首个把设计模式应用于组织结构分析的例子[Coplien 1994]。Coplien的论文立刻帮助模式运动进入了一个新阶段，不仅包括了软件设计模式，还包括了分析模式、组织模式、教育模式以及其他方面的问题。

10

接下来就是现在已经成为软件设计模式经典教材的*Design Patterns: Elements of Reusable Object-Oriented Software*[Gamma 1994]（《设计模式：可复用面向对象软件的基础》，中文版，机械工业出版社）一书的出版。它介绍了一些常见的、实用的软件设计构造，可以很容易被应用于大多数软件项目中。许多面向对象软件架构师都把这本书奉为圭臬。更值得注意的是，很多人发现它所包含的模式似曾相识，自己在先前的软件项目中应用过的构造正是这些模式所描述的。

行业中对它的最初反应是普遍的肯定，因为它把软件开发的词汇和设计的关注从数据结构和

编程惯用法的层次提升到了架构层次。而facade（外观）、adapter（适配器）和visitor（访问者）等词语成为了设计讨论中广为人知的术语。软件开发人员常常组织起草根性质的团体，在他们的软件开发项目中应用某些设计模式，并支持其他人对这些模式的使用。世界各地都组织了设计模式研究团体，讨论把软件设计模式的应用作为提高软件质量的基础。还出现了许多顾问，他们帮助各个机构挖掘机构内部存在的设计模式，以帮助缺少经验的开发人员采用更有经验的同事的技巧。设计模式带来了一个短暂的辉煌时期，似乎成为了推动整个软件行业革命化的一步，让软件行业关注设计复用，设计出能更有效应对变化的需求的软件。

如何使一个软件项目走向失败

- 两次向同样的受众展示同一个产品演示。
- 关注于技术而不是问题和场景。
- 未能让投资回报最大化；例如，开发概念验证原型比在已有原型上增加内容更有效。
- 把项目的关注点从大尺度转移到较小的尺度。
- 在多次发布之间不保持一致性。
- 把开发团队的工作和机构中的其他群组隔离开。
- 重写已有的客户端、服务器端和应用程序。
- 改变系统的目的，以致模型描述的是错误的关注点和对象。

11

从1994年以来，有关设计模式的文献出版量呈指数形式增长。这种增长既有利亦有弊。对有经验的面向对象架构师来说，现在有许多的还在不断增长的可复用设计，可以对它们进行评估并应用到软件开发中去。而且，有大量的论文和研讨会可以帮助架构师把他自己的领域知识记录成设计模式，让行业中的其他专业人员可以更容易使用它们。弊端就是，许多使用设计模式的人未能正确评估特定设计模式或设计语言针对他们的特定上下文环境的适用性。此外，部分开发人员对一些知识一知半解，就急切地想在完成领域分析之前就把所有的事情都归并到某个设计模式，或者用一组特定的设计模式来解决所有的问题。

Michael Akroyd在此期间为1996年的Object World西部会议准备了一篇题为*AntiPatterns: Vaccinations against Object Misuse*的报告[Akroyd 1996]。他的报告基于对行业中面向对象文献的详细分析，试图定义一个有关面向对象的综合观点。报告集中于识别反复出现于多个软件项目中的有害软件构造。这是GoF（Gang of Four，指*Design Patterns: Elements of Reusable Object-Oriented Software*一书的四位作者）模式的直接对照，GoF模式强调的是在构建新软件时使用经过验证的良好设计。

在Akroyd之前，其他一些人也曾提到过反模式的概念，但他是第一个建立起反模式的正式模型的人。对反模式的作用的讨论几乎是和模式的引入同步开始的。Fred Brooks[Brooks 1979]、Bruce Webster[Webster 1995]、James Coplien[Coplien 1995]和Andrew Koenig都曾记录了类似的工作，在确定机能不良行为和重构解决方案的基础上为软件开发提供指引。

由于有如此多的人为反模式做出了贡献，因此把反模式的原创性归功于任何个人都是有失公允的。还不如说反模式是实现设计模式和扩展设计模式模型的工作中会自然发生的一步。本书试图在GoF设计模式学院式的形式主义和那些缺乏经验的软件开发人员之间架起一座桥梁，因为他
12 们需要更多的相关背景信息来评估和判定特定的方法是否适合他们的特定情况。

对反模式的研究

> “对反模式的研究是一项重要的研究活动。仅仅在一个成功的系统中展现出‘好’的模式是不够的；你还必须说明这些模式不会出现在不成功的系统中。与之相似，揭示出特定的模式（反模式）出现在不成功的系统中，而不会出现在成功的系统中同样有益。”
>
> ——Jim Coplien（模式先驱，C++大师）

反模式的概念是首个关注于负面解决方案的软件研究方向。考虑到软件缺陷和项目失败出现的频率，负面解决方案也许是具有更丰富研究内容的领域（也就是所谓的目标丰富的环境）。在反模式研究中，我们试图分类、标记和说明反复出现的负面解决方案。我们并不止步于此。对每一个反模式，我们都会给它附加一个或多个设计模式，它们为解决导致问题出现的根源提供了建设性的替代方法。

我们从三个角度来介绍这些反模式：开发人员、架构师和管理人员。

- **开发性反模式**包括程序员遇到的技术问题和解决方案。
- **架构性反模式**确定和解决规划系统结构时常见的问题。
- **管理性反模式**针对软件开发过程和开发机构中的常见问题。

这三个视角是软件开发的基本方面，每一个角度都存在许多问题。

1.4 本书组织结构

本章描绘了反模式的发展历程以及它们和改善软件行业状况的关系。反模式提供了一种有效的方法，可以说明在软件开发中发现的存在问题的开发行为和机构组织方法，并详述相应的解决
13 方法。为此，本书如下组织后面各章：

- 第2章介绍反模式参考模型。该参考模型定义了在第二部分对反模式的定义说明中用到的所有常见概念。
- 第3章展示了反模式的两种形式：小型反模式模板和完整反模式模板。
- 第4章解释了反模式在软件开发机构中出现时意味着什么。该章说明了使用反模式时基本

的指导原则和行为方式。我们还介绍了自行编写反模式的步骤。

- 第二部分提供了反模式和小型反模式的详细说明。
- 第5章定义软件开发性反模式。
- 第6章定义软件架构性反模式。
- 第7章定义软件项目管理性反模式。
- 第三部分是各种附录，提供了相关的参考资源，包括缩略词表、术语表和反模式摘要。