普通高等教育
软件工程
“十二五”规划教材

12th Five-Year Plan Textbooks
of Software Engineering

工业和信息化普通高等教育
“十二五”规划教材

软件体系结构

林荣恒 吴步丹 金芝 ◎ 编著

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

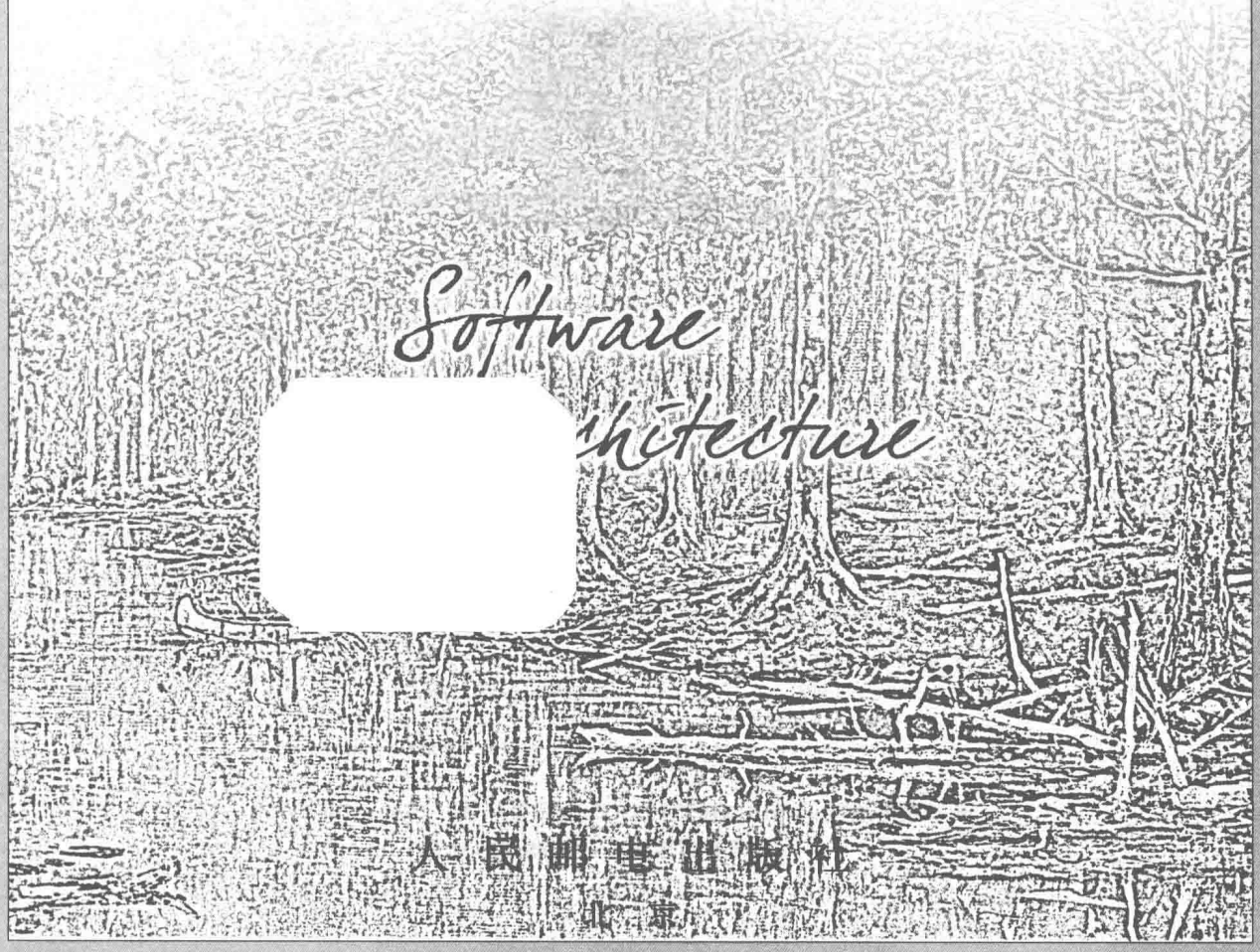
普通高等教育
软件工程“十二五”规划教材

12th Five-Year Plan Textbooks
of Software Engineering

工业和信息化普通高等教育
“十二五”规划教材

软件体系结构

林荣恒 吴步丹 金芝 ◎ 编著



人民邮电出版社
北京

图书在版编目（CIP）数据

软件体系结构 / 林荣恒，吴步丹，金芝编著. — 北京 : 人民邮电出版社，2016.4
普通高等教育软件工程“十二五”规划教材
ISBN 978-7-115-40293-6

Ⅰ. ①软… Ⅱ. ①林… ②吴… ③金… Ⅲ. ①软件－系统结构－高等学校－教材 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2015)第237656号

内 容 提 要

本书详细介绍了软件体系结构的基本概念、软件体系结构风格、质量属性及战术、软件体系结构设计方法等，希望读者对软件体系结构形成较为完整的概念，在此基础上理解软件体系结构的基本用途，从而在软件工程实践中融入相关概念。

本书的最大特点是使用了大量的例子，因此读者在阅读时需重点理解相关例子的内在含义，从而加深对软件体系结构的理解。本书可作为计算机软件专业本科生、研究生和软件工程硕士的软件体系结构教材，也可作为软件工程高级培训、系统分析员培训、系统构架设计师培训教材，以及软件开发人员的参考书。

◆ 编　　著　林荣恒　吴步丹　金　芝
责任编辑　刘　博
责任印制　沈　蓉　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：13.75　　2016 年 4 月第 1 版
字数：358 千字　　2016 年 4 月河北第 1 次印刷

定价：36.00 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前 言
Preface

软件体系结构作为软件工程学科中重要的一门分支，在这几十年中也有了长足的发展，随之，与软件体系结构相关的理论也不断更新，以适应软件开发方式的发展。

作为计算机领域重要的工程基础科目，软件体系结构是计算机专业本科及研究生重要的必修或专业选修课程。由于软件体系结构这门课程涉及的理论知识较多，并且当前软件体系结构的书籍大多以理论知识介绍为主。因此，在进行实际系统开发时读者很难将软件体系结构的知识与之对应。作者具有从事软件体系结构课程多年的教学经验，在教学过程中一直探索如何将软件体系结构的理论与实际的案例进行结合。在这个过程中，作者总结了大量来源于现实生活与实际工作的案例，这些案例既包括软件体系结构风格的探讨，又包括详细的质量属性战术，也包括典型软件体系结构架构，还包括了典型互联网事件，便于读者理解身边的软件体系结构。案例的说明方式包括文字叙述、示意图、流程图、伪代码等，尽量以最合适、最直观的方式还原实际问题的解决过程。

本书在章节编排上，力争为读者提供一个基本的软件体系结构概况，并使读者可以在日常软件需求分析及设计中关注质量属性，在考虑系统架构时融入软件体系结构思想，在软件结构分解时结合相关的质量属性场景及战术。此书主要关注软件体系结构的原理、软件体系结构风格、质量属性及其战术、软件体系结构设计等方面，对于软件体系结构的其他部分，如软件体系结构评估、软件产品线、软件体系结构描述语言等则不涉及或者较少提及。

本书前后历经了一年半最终成稿，写书是对脑力和意志力的双重考验，由于本书涉及的案例较多，作者在撰写过程中遇到了不少挑战。在这个过程中，北京大学、北京邮电大学的多名老师及研究生给予了理论上、技术上的支持和帮助，从而使本书可以顺利完成，在此向他们表示感谢。同时，本书的相关内容得到了国家重点基础研究发展计划（973）（2015CB352201）、可信软件基础研究重大研究计划（91318301）等的资助。

作 者

2016 年 1 月

目 录
Contents

第1章
软件体系结构的起源与背景
Origins and Background of Software Architecture

人们利用计算机语言等进行有意识的编写及创造形成了具备一定功能的指令序列，这些指令序列可以称为软件。软件已经深入到日常生活中的方方面面，小到各种嵌入设备，大到航天飞机等均有软件的身影。软件的设计、开发已经形成一个独立的行业，如何保障软件的质量成为软件行业的永久话题。

在软件业发展的过程中，经历了小作坊到企业化生产的多个阶段。最初软件的规模较小，一般由一个或者几个程序员进行代码编写，主要依靠程序员自身的素质及对软件的深刻理解。随着软件规模的不断发展，在软件迈向团队协作的过程中出现了软件危机。

1.1 软 件 危 机

软件危机（Software Crisis）是指落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致计算机软件的开发和维护过程中遇到的一系列严重的问题。软件危机是1968年在联邦德国召开的国际软件工程会议上被人们普遍认识到，其主要表现为软件成本日益增长、开发进度难以控制、软件质量差、软件维护困难等。

由图1-1可知，随着硬件的发展，软件占整个系统的比例呈大幅增长，由最初50年代的10%左右增长到80年代的80%以上。而且随着系统规模增大，其成本仍在增长。因此，如何控制一个系统的软件成本成为该系统重要的成本核算部分。

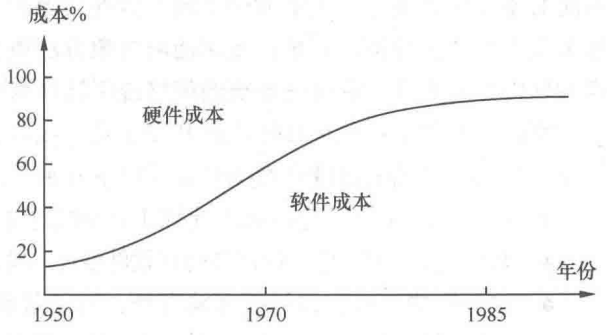


图1-1 软硬件成本的变化

一方面是软件成本的增长，另一方面则是软件开发进度的难以控制。软件的开发进度本该按照计划执行，但由于程序员素质、开发难度等问题，大多数的软件系统开发存在拖沓的现象，即系统上线的时间一拖再拖。

系统上线时间的延迟并未带来软件质量的提升，“慢工出细活”在软件行业并不成立，拖延的软件系统与软件质量无直接的促进作用。当系统上线之后，软件的维护则成为软件公司的梦魇，漫长而又烦琐。随着时间的推移，由于员工的更换等原因，老系统的维护更加困难。

软件危机一直制约着软件质量的提升，也成为软件行业变革的导火索。

为了克服软件危机，人们意识到面临的不光是软件技术问题，更重要的是管理问题，管理不善必然导致失败，人们开始探索利用现代工程的概念、原理、技术和方法进行计算机软件的开发、管理和维护。1968 年，“软件工程”的概念出现了。

1.2 软件工程的兴起

软件工程是用工程、科学和数学的原则与方法研制、维护计算机软件的相关技术及管理方法。简单地说，软件工程利用工程化的思想和管理手段对软件的过程进行管理和监控。

工程实践是利用前人的经验，采用规范、确定的方法进行实施的过程。工程实践使得平凡的操作者创造出复杂的系统。每个操作者甚至不用了解整个系统的全貌，只需完成自身的工作即可配合搭建整个系统。例如，在建筑工地中瓦工只需了解如何砌砖，油工只需了解如何涂抹墙壁，通过各个工种的贡献最终完成整个建筑的搭建。对工程规范性强的公司而言，更需要每个公司员工如同螺丝钉般完成自身本职工作，而无需具备太多的个性。

软件工程的思想，即希望在软件公司中各个开发人员可以各司其职，每个人完成自身的各个模块，通过工程化的方式构架软件，并保证软件的质量。为保障软件能像建筑工程一样顺利实施，软件工程一般包含以下三要素。

- 方法：为软件开发提供了“如何做”的技术，是完成软件工程项目的技术手段。
- 工具：是人类在开发软件的活动中智力和体力的扩展和延伸，为软件工程方法提供了自动的或半自动的软件支撑环境。
- 过程：是将软件工程的方法和工具综合起来以达到合理、及时地进行软件开发的目的。

软件工程的方法，包括结构化方法、面向对象方法和形式化方法。结构化方法也称为生命周期方法学或结构化范型。将软件生命周期的全过程依次划分为若干个阶段，采用结构化技术来完成每个阶段的任务。结构化方法具有两个特点：一是强调自顶向下顺序地完成软件开发的各阶段任务；二是结构化方法要么面向行为，要么面向数据，缺乏使两者有机结合的机制。面向对象方法是将数据和对数据的操作紧密地结合起来的方法。软件开发过程是多次反复迭代的演化过程。面向对象方法在概念和表示方法上的一致性，保证了各项开发活动之间的平滑过渡。对于大型、复杂及交互性比较强的系统，使用面向对象方法更有优势。形式化方法是一种基于形式化数学变换的软件开发方法，它可将系统的规格说明转换为可执行的程序。

软件工程的工具包括各种软件开发工具、项目管理工具、项目维护工具等，一般也统称为软件开发工具。常见的软件开发工具分为以下几种。

- 软件需求工具，包括需求建模工具和需求追踪工具。
- 软件设计工具，用于创建和检查软件设计。因为软件设计方法的多样性，这类工具的种类很多。
- 软件构造工具，包括程序编辑器、编译器和代码生成器、解释器和调试器等。
- 软件测试工具，包括测试生成器、测试执行框架、测试评价工具、测试管理工具和性能分析工具。
- 软件维护工具，包括理解工具（如可视化工具）和再造工具（如重构工具）。
- 软件配置管理工具，包括追踪工具、版本管理工具和发布工具。
- 软件工程管理工具，包括项目计划与追踪工具、风险管理工具和度量工具。

- 软件工程过程工具，包括建模工具、管理工具和软件开发环境。
- 软件质量工具，包括检查工具和分析工具。

常见的软件需求工具包括 Rational Rose、Visio 等，软件构造工具包括大家熟悉的 Visual Studio、Eclipse 等，软件测试工具包括性能测试工具、单元测试工具 Junit 等，这些工具在形式上不见得一定以 GUI 呈现，也可能以软件包形式存在。有些软件开发工具如 Eclipse 也具备软件重构等功能，这种工具在意义上也称为软件维护工具。软件配置管理工具，如常见的 SVN、CVS、GIT 等。软件工程管理工具，包括微软的 Proect、在线甘特图等。软件过程管理工具包括 Maven 等。

典型的软件过程有 RUP 的开发过程、敏捷开发过程等，软件过程是用于规范和开发软件的各个过程。ISO 9000 定义软件工程过程是把输入转化为输出的一组彼此相关的资源和活动，该定义支持了软件工程过程的两方面内涵。

第一，软件工程过程是指为获得软件产品，在软件工具支持下由软件工程师完成的一系列软件工程活动。基于这个方面，软件工程过程通常包含以下 4 种基本活动。

- 软件规划、规格说明。规定软件的功能及其运行时的限制。
- 软件开发活动。用于产生满足规格说明的软件。
- 软件确认活动。确认软件能够满足客户提出的要求，此处的要求包括功能要求与质量要求。
- 软件演进。为满足客户的变更要求，软件需要在使用过程中进行演进，演进的内容包括软件的功能、非功能要求。例如，在原有软件基础上增加新功能，或者使原来的软件满足更高的性能要求。

事实上，软件工程过程是一个软件开发机构针对某类软件产品为自己规定的工作步骤，它应当是科学的、合理的，否则必将影响软件产品的质量。

第二，从软件开发的观点看，它就是使用适当的资源（包括人员、硬软件工具、时间等）为开发软件进行的一组开发活动。因此需要结合实际软件开发需求，对资源进行调度。最终，在过程结束时将输入（用户要求）转化为输出（软件产品）。软件开发过程中，如何合理地调配资源是圆满完成软件的必要条件。

综上所述，软件工程的过程是将软件工程的方法和工具综合起来，以达到合理、及时地进行计算机软件开发的目的。软件工程过程应确定方法使用的顺序、要求交付的文档资料、为保证质量和适应变化所需要的管理、软件开发各个阶段完成的任务。

以 RUP 的软件开发过程为例，RUP 是 Rational 软件公司（Rational 公司被 IBM 并购）创造的软件工程方法。RUP 描述了如何有效地利用商业的可靠的方法开发和部署软件，是一种重量级过程（也被称作厚方法学），因此特别适用于大型软件团队开发大型项目。

RUP 可为大多数程序的开发提供指导方针、模板等。RUP 把开发过程中面向过程的方面，如定义的阶段，技术、实践和其他开发的组件（如代码、文档、手册等）整合在一个统一的框架内。因此，RUP 首先明确了软件开发过程中的软件过程。

最重要的是，RUP 有下述三大特点。

① 软件开发是一个迭代过程，正如刚才描述的四个阶段，这四个阶段可在项目开发过程中多次迭代出现。一般情况下，一个以 RUP 过程规范的软件开发过程，都将经历原型、原型迭代、最终版迭代等几次迭代过程。第 1 次迭代一般将系统的骨架及主要功能完成，用于验证系统的可行性，同时为用户提供直观的系统使用界面，便于获得相应的反馈。通过第 2 次迭代将新的需求反馈到系统之中。

② 软件开发是由 Use Case 驱动的。在 RUP 软件过程中，特别强调用例。采用用例的方式有

助于对系统进行分解，从而降低系统的难度，便于开发人员开发以及分工。例如，一个复杂的软件系统可能由上百个用例组成，每个用例涉及的功能点很小，这样有助于根据用例的功能分派给开发小组、开发人员。

③ 软件开发是以架构设计（Architectural Design）为中心的。在 RUP 的开发过程中，开发人员首先建立统一的架构，在该架构中划分模块，通过各个独立的模块来支撑各个功能。所有模块间的交互、关系都由架构规定。这样便于开发人员并行开发，同时便于第 1 次迭代中系统骨架的形成。

一个典型的 RUP 软件开发过程包括：初始阶段（Inception）、细化阶段（Elaboration）、构造阶段（Construction）和交付阶段（Transition）。同时规定了每个阶段的完成形式，一般称为里程碑，通过检查里程碑可以对每个阶段的结果进行确认。

RUP 在明确了每个阶段之后，对于每个阶段需要完成的任务也做了一定的规范，例如在细化阶段包括对相关需求的细化，而在描述需求的时候可能需要用例图的方式对每个用例进行分析，最后形成需求规格说明书。而在构造阶段，相应的设计文档、代码等需要被完成。因此，RUP 的规范包括了过程与过程中的各个组件。

RUP 的几个阶段说明如下。

- 初始阶段：初始阶段的目标是为系统建立商业案例并确定项目的边界。为了达到该目的必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。本阶段具有非常重要的意义，在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。对于建立在原有系统基础上的开发项目来讲，初始阶段可能很短。初始阶段结束时是第 1 个重要的里程碑——生命周期目标（Lifecycle Objective）里程碑。生命周期目标里程碑评价项目基本的生存能力。
- 细化阶段：细化阶段的目标是分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素。为了达到该目的，必须在理解整个系统的基础上，对体系结构作出决策，包括其范围、主要功能和诸如性能等非功能需求。同时为项目建立支持环境，包括创建开发案例，创建模板、准则和准备工具。细化阶段结束时是第 2 个重要的里程碑——生命周期结构（Lifecycle Architecture）里程碑。生命周期结构里程碑为系统的结构建立了管理基准并使项目小组能够在构建阶段中进行衡量。此刻，要检验详细的系统目标和范围、结构的选择以及主要风险的解决方案。
- 构造阶段：在构造阶段，所有剩余的构件和应用程序功能被开发并集成为产品，所有的功能被详细测试。从某种意义上说，构造阶段是一个制造过程，其重点放在管理资源及控制运作以优化成本、进度和质量。构造阶段结束时是第 3 个重要的里程碑——初始功能（Initial Operational）里程碑。初始功能里程碑决定了产品是否可以在测试环境中进行部署。此刻，要确定软件、环境、用户是否可以开始系统的运作。此时的产品版本也常被称为“beta”版。
- 交付阶段：交付阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，以及基于用户反馈的少量的调整。在生命周期的这一点上，用户反馈应主要集中在产品调整，设置、安装和可用性问题，所有主要的结构问题应该在项目生命周期的早期阶段就解决了。在交付阶段的终点是第 4 个里程碑——产品发布（Product Release）里程碑。此时，要确定目标是否实现，是否应该开始另一个开发周期。在一些情况下这个里程碑可能与下一个周期的初始阶段的结束重合。

1.3　软件体系结构层次

软件体系结构是软件工程的一个分支，主要解决如何架构软件并保障软件质量的一门学科。简单地说，软件工程解决如何工程化开发软件项目，而软件体系结构则是在工程化开发项目之初规划、设计软件的架构，从而引导软件工程朝着正确的方向发展。

软件体系结构虽脱胎于软件工程，但其形成同时借鉴了计算机体系结构和网络体系结构中很多宝贵的思想和方法，最近几年软件体系结构研究已完全独立于软件工程的研究，成为计算机科学的一个新的研究方向和独立学科分支。

正如传统软件工程中的方法，更多关注的是软件的功能如何设计、实现、测试，而软件体系结构则关注软件中的非功能属性如何实现和保障，为了更好地描述软件体系结构，一般可在设计之初将软件系统划分成多个层次。

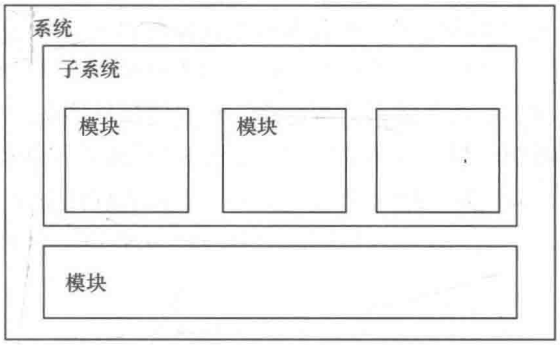


图 1-2　系统、子系统、模块层次图

如图 1-2 所示，一般软件系统可分为系统、子系统、模块、类等多个层次。为了方便每个层次的建模分析，每个层次定义类似的结构，一般包括组件、组成规则和行为规则，具体说明如下所述。

- 组件：组成的构件，指相关构成的实体。
- 组成规则：组件或系统的构造规则。多个组件以何种方式、关系组成当前层次的系统，组成规则一般静态地描述一个系统的构成。
- 行为规则：系统的语义，组件间交互的规则。行为规则动态地表明了一个系统中各个组件之间如何动态地交互。

针对每层的层次不同，相关的组件、规则具有自身不同的含义。每层可以有自身不同的符号系统、解决方法。因此每个层次可以进行独立设计，层次间可通过预先设计的接口进行传递，下一层是上一层的服务提供，从而简化层次上的设计。

一般来说，系统可以分为多个层次，如表 1-1 所示。典型的层次包括系统—子系统—模块—类（进程、线程）—数据结构—二进制代码。其中，（系统—子系统—模块—类）层面一般称为体系结构层次。数据结构等则称为编码层次，二进制代码称为可执行代码层次。在各个层面中，均利用组件及组件之间的协作完成相关功能。例如，在体系结构中，系统层面的组件包括各个子系统，子系统之间可以通过调用、管道等进行交互；在模块层面的组件包括各个类、接口，通过类之间的调用等方式进行交互；而在编码层面，各个对象通过方法调用、消息传递等方面进行交互。

表 1-1　　系统层次对比

系 统 层 次	结 构 归 类
系统、子系统、模块、类	体系结构层次
数据结构	编码层次
二进制代码	可执行代码层次

- 体系结构层次：通过组件与组件间的协作达到系统功能。本层的组件包含多种类型（如模块、进程、线程等），组件之间的交互通过各种方式完成（调用、管道、共享、同步等）。
- 编码层次：包含算法与数据结构的设计，组件是编程语言的原子如数、字符、指针、线程等，连接的操作符是该语言的算术或数据操作符，组合机制包括记录、数组、过程等。
- 可执行代码层次：解决内存映像、数据组织、调用栈、寄存器分配等问题。组件为硬件所支持的位模式，操作符及组件的组合分布在机器代码中。

作为软件体系结构这个学科，主要关注体系结构层次的相关内容，解决如何构建体系结构、如何分析体系结构、如何评估体系结构等问题。随着软件业的发展，软件规模越来越大，对软件的架构也提出了更多的要求，要求软件构架满足修改性、易用性、性能等多种非功能属性。软件体系结构的发展，产生了一些经典的软件体系结构以及现代软件体系结构，这些软件体系结构是人们在软件实践中取得的经验和教训。

经典的软件体系结构，主要是对传统软件常见的架构进行总结，包含了管道过滤器、调用返回、层次等体系结构形态。经典的软件体系结构产生于软件规模较小的时代，因此诸如管道过滤器等所描述的体系结构的层次较低。随着软件规模的扩大，现代软件体系结构应运而生。诸如 P2P、C/S、基于 SOA 架构等成为了现代软件体系结构的代表。这些体系结构满足了现代软件设计的需求，对当前的软件架构具有一定的现实参考价值。在之后的章节里，本书将对上述软件体系结构进行详述。

1.4 软件体系结构的理想与现实

1.4.1 软件体系结构的理想效果

软件体系结构的理想效果是可完整、高效地重用整个软件体系结构，将现有软件体系结构应用于新的项目中。

用户只需将需求明确、选择适合的软件体系结构，就可完成整个项目的功能、非功能的需求分析，并且确定好相关软件模块的划分、采用的相关技术，以及技术对应的相关软件战术，这些战术可以提升用户开发的软件模块质量，同时结合软件的非功能需求，快速满足非功能属性的要求。简单地说，如果一个体系结构得以理想化的复用，那么软件的需求、设计乃至相关关键技术均可快速被确定及复用。下面举一个例子说明软件体系结构的理想复用。

某一用户公司完成了多个软件项目，这些软件项目均属一个软件序列。公司即将开发一个新的应用 newAPP。新应用基于软件体系结构复用的开发过程，与传统过程的对比如表 1-2 所示。

表 1-2　　体系结构复用与传统开发过程对比

体系结构复用	传 统 开 发
体系结构分析	需求分析
体系结构的选择	总体概要设计
体系结构的分解	模块概要设计
体系结构的责任划分	概要设计
体系结构填充、定制	总体框架设计与实现

表格左列为以体系结构复用的相关开发过程，右侧为传统的开发模式。左列由于可利用现有的体系结构，在体系结构分析上只需对与原有体系结构有区别的需求进行分析，即可尽快地完成体系结构的选择，以及新需求的分解，同时责任划分也相对容易，可快速地利用原有体系结构进行填充、定制。与体系结构所对应的原有代码结构可保持不变，只需在修改部分的结构上进行代码重构即可完成新的应用开发。

例如笔者所在实验室具有一套电信业务开发平台，基于该平台可快速地开发各种功能的通信业务。基于平台的开发正是对基于体系结构复用的开发过程，用户可借鉴该通信平台的已有架构，在开发时，将重点转变为如何利用和定制框架以完成新的通信业务需求，从而大大减少了开发的工作量。架构也是类似的，也有利于小组模式的开发和维护。同时，同样的架构大大提高了代码的复用率，减少了额外的代码测试，提高了代码的成熟度。对程序员新手而言，在一个框架中进行开发，更像是在一个半成品的积木模型上进行再创造。

1.4.2　现存软件复用的层次

软件互用层次按照复用的内涵可以分为如下几种。

① API 级别的复用，软件库的复用。API 的复用是最基础的复用形态，用户通过调用某个或几个 API 而形成对遗留代码的复用。

② 软件构件的复用。它是将多个 API 有机地组成一套具备通用功能，可在相关中间件上使用的组件。软件构件的复用比纯粹 API 级别的复用多了一些环境的限制，相关上下文信息较为明确。

③ 服务的复用。将某些软件的功能开放出来以服务的形式对外提供，一般采用 SOAP 等标准协议进行开放。服务的复用在某种层次上比软件构件复用多了一些通用性，同时一般可支持远程调用，有利于系统的解耦合。

④ 框架的复用。框架的复用比上述复用更进一步，框架一般规定了框架自身提供的功能，也规范了框架中应用的行为准则。框架一般还提供相关通用服务，便于用户快速地开发相关应用。

⑤ 体系结构的复用。对体系结构的复用包括体系结构的静态与动态结构，可完整刻画软件系统的功能与非功能属性，便于快速地搭建软件结构。

目前的软件复用层次，仍停留在前四个阶段，很多开发人员甚至还停留在第 1 阶段左右。例如一个本科生在学习编程语言的过程中，一般先学习语言自身的关键词、各种语句，之后学习的库函数就属于第 1 阶段的复用。在第 1 层次的复用中，开发人员可基于标准库或者已知第三方库函数从最底层开发相关应用。显然，这种开发过程相当烦琐。若开发一个大型系统，需要做很多工作才能完成。

为此针对第 1 阶段的复用，大多数软件开发企业会过渡到第 2、3 阶段的复用层次。通过选择合理的软件构件或者服务，简化系统某些部分的开发难度及工作量。例如，一个软件中需要使用一个天气预报的功能，若软件开发人员还自己去编写天气预报的功能，不仅费时费力而且也不见得准确。在这种情况下，调用一个第三方的天气预报构件或者服务，可快速地完成该功能的搭建。

在进行第 2、3 阶段的复用之后，框架的复用也就顺理成章了。若一个系统的若干个基本功能可由一套软件框架统一提供，选择相应的软件框架可为软件的快速开发、定制提供良好的支持。例如，开发一个基于 JAVA 的 Web 应用，采用 Spring+Structs+Hibernate 的框架可为用户的数据存储、前后端分离等提供良好的支持，也便于用户快速地扩展相关功能。

现存的这几个复用方式，虽然可以在一定程度上加快软件开发的效率，提升软件质量，但是仍存在着诸多不足之处。例如，基于 API 的复用层次较低，用户使用 API 开发应用，着眼点过于

细节化，很难对整体软件的架构提出建设性意见。如表 1-3 所示，举例说明了不同的复用层次。

基于构件或者服务的软件复用，虽然层次相对 API 有了一定的提升，但是仍对软件架构有直接的帮助。同时，由于构件、服务很难直接与新应用的需求相匹配，复用的场景受到限制，具体体现为构件、服务功能粒度大的难以满足应用细节要求，功能粒度小的则复用层次较低。

基于框架的复用，虽然可提供相应的通用服务，然而应用自身对通用服务的个性化需求，使得通用服务仍需定制。同时，由于基于框架开发，应用的相关特殊流程、交互方式受到框架的约束，应用开发变得束手束脚。

表 1-3　　不同复用层次的说明

复用层次	代码示例
API 级别复用	API 级别是最细粒度，也是最常见、通用的方式。事实上构件、服务从调用形式上也属于 API 调用 System.out.println(); Thread thr=new Thread(){new …};
软件构件复用	此组件是一些基础功能 API 的集合，该组件具备一定领域的内涵，在一定领域内通用。但单独的构件并不能提供完整的功能，仅是功能链条的一部分。 Comp comp=getComp() comp.dosomthing1() comp.dosomthing2() 软件构件可以体现为可视化形态，可在软件中拖曳使用
服务的复用	服务可以理解为具有一定完整软件功能的构件，例如天气预报服务。一般该服务是远程服务，典型的可利用 SOAP / RESTful 进行调用 Service s = GetService ( ) Service.function()
框架的复用	框架为应用开发提供了强制的规范，并在该规范下保障应用正常运行。典型框架如 J2EE 框架、Spring 框架、复合的 SSH 框架等。 以 SSH 框架为例，用户在使用框架进行复用时，框架规范了存储使用 hibernate，在表示层可以使用 Structs，并区分了 Model、Controller、View 的实现。用户只需按照相关框架进行开发即可达到快速开发的效果
体系结构复用	体系结构的复用则比框架的复用更为上层及抽象，一般体系结构的复用考虑了整个系统的设计包含功能、非功能因素，试图借鉴历史的架构，从架构层面进行复用，从而达到系统层面的设计理念复用，以及代码层面的高质量复用。 从典型的例子上来看，软件产品线的诞生是体系结构复用的一个很好的例子

综上所述，现存的软件复用层次与软件体系结构的理想仍有较大的差距，如何缩小差距是当前软件体系结构的重要研究问题及目标，也是读者学习软件体系结构的目标之一。

1.5　相关软件的失败案例

在详细介绍软件体系结构之前，让我们一起先了解一些软件开发过程的失败案例。这些失败的案例都是没有遵循软件工程或软件体系结构而造成的。

1.5.1　瑞典船的故事

本故事跟软件并没有直接关系，然而它给人们带来的启示，可以帮助软件系统的成功编制。在 17 世纪上半叶，作为北欧新教势力的代表，瑞典的军事力量达到鼎盛时期。1625 年，号称“北

方飓风”的瑞典国王古斯塔夫·阿道夫斯二世（GustavsII Adolphus）决心建造一艘史无前例的巨型新战舰——“瓦萨”（VASA）号战舰。“瓦萨”号战舰在当时确实是一艘令人望而生畏的战舰，其舰长 70 米，载员 300 人，拥有三层甲板并装有 64 门重炮，火力强大。然而，当时的设计师并没有设计如此大战舰的经验。设计师不敢违抗国王的命令，只能借鉴原有中型战舰的设计原则设计瓦萨战舰。

经过几年，这艘巨大的战舰终于完工。在斯德哥尔摩的王宫前，“瓦萨”号战舰举行了盛大的下水典礼。一声令下，战舰扬帆起航，乘风前进。然而，巨大的战舰并没有带来胜利的凯旋。船刚驶出船坞不久，摇晃了一下，便向左舷倾斜，海水从炮孔处涌入船舱，战舰迅速翻入水中，几分钟后，这艘雄伟战舰的处女航，也是唯一的一次航行结束了。

国王如何处置船的相关设计师，已经无从考证。现今“瓦萨”号战舰被打捞上来，静静地停放在博物馆，成为人们的一个警示。这次事故的发生并非偶然，究其原因包括设计师经验、国王的催促、船体的比例等因素。由于设计师自身的知识水平有限，采用的技术手段建造如此大船存在一定的困难。国王对该船的急迫，使得该船的工期缩短，没有太多时间进行验证、实验。传统的小船与大船之间的比例关系并非直接成比例放大，采用借鉴设计原则没有问题，但是不能直接照搬。

同样，在软件设计开发过程中，如何合理利用设计师经验、技术是一个重要问题。另外，合理地安排项目进度、进行软件验证等，也是软件开发中的重要组成部分。在软件设计中，需要有良好的原则、模式对软件整体进行规划。软件体系结构可以在理论和实践中对相关的风险进行一行一定的预测，从而降低项目返工的可能性。

1.5.2　集团通信业务系统项目

某公司应约开发一套集团通信业务项目，该集团通信项目要求实现集团内部的短号呼叫、语音信箱、群呼、短消息群发等功能。开发小组包括了多个事业部的人员，担任该项目的项目经理低估了该项目的协调难度以及该项目的时间成本。在项目开发过程中，该项目经理未能建立统一的项目跟踪管理系统，在项目功能的划分上存在着责任不清、接口过多等问题。从而导致了该项目沟通成本过大，各个部门的人员对自身需要完成的任务互相推诿，从而使得开发过程变得极为漫长。在临近验收时，各个子系统的功能仍未开发完成。最终，项目延期了多次，且项目代码的质量较差。可以说本项目的失败是因为没有按照软件工程的要求进行管理造成的。

软件工程与软件体系结构的课程将为项目开发和质量保障提供良好的支持。一个良好的项目经理必然需要了解软件开发规律，对项目过程中可能遇到的风险、项目中成员的知识储备等问题负责，从而降低整个项目的执行风险，最终保障项目的质量并按时交付。时间计划是软件开发中重要的组成部分，如何在项目执行之前对项目所需的时间进行规划，是需要长期软件开发的经验。软件质量是直接导致项目成功与否的关键，软件质量的影响因素包括软件的分析、设计、开发、集成等多个环节。如何从软件设计之初就保障软件的质量正是软件体系结构课程需完成的任务。

1.5.3　邮政信息管理系统的开发

某公司开发一套邮政信息管理系统，该项目需建立一个用于邮政业务的监督和管理的系统，从而提高邮政的服务效率。由于需求方自身对相关监督、管理等功能的要求并不明确，使得该公司在未明确详细需求的情况下，便开始对项目进行开发。实现方由于对行业知识的缺乏及设计人员水平的低下，不能完全理解客户的需求说明，而又没有加以严格的确认，以想当然的方法进行

系统设计，导致结的果是推倒重来。另外，该项目的项目经理对整个项目的工作量估算有误，他未综合考虑开发的阶段、人员的生产率、工作的复杂程度、历史经验等多方面因素，而是简单地基于过去的某次成功经验，直接对工时进行了估算。再者，开发计划并不充分，开发计划没有指定里程碑或检查点，也没有规定设计评审期，最终导致该项目无法正常按时交付。

针对上述软件出现的问题，一定程度上可以通过学习软件体系结构进行规避或者克服。例如，需求的变更是每个项目都会出现的现象，如何在软件设计之初为变更的需求提供预留或预期，是软件体系结构中可修改性的要求。

针对工数估算过少的问题，可通过软件体系结构及软件工程的知识，对项目功能自顶向下的分解，以及对关键质量属性的代价分析得出具体的软件代价。同样，开发计划的分析也可通过对各个模块质量属性、功能的完成难度等得出。

1.6 软件体系结构的发展历程

软件体系结构的兴起最早可以追溯到 20 世纪 60 年代。随着 20 世纪 60 年代软件危机程度的日益加剧，使得人们开始认识到软件体系结构的重要性，尤其对于大规模复杂软件系统，软件体系结构的设计成为提高软件生产效率的有效途径。软件工程先驱 Edsger Dijkstra 在 1968 年首次提出“体系结构”概念，认为软件应注意分解和结构化，并提出层次结构的概念：一层中的程序只能与相邻层程序通讯。1969 软件工程巨匠 Fred Brooks 年认为体系结构是：“用户接口完整详细的说明，对于计算机，是程序设计手册；对于编译器，是语言手册；对于整个系统，是用户完成整个工作所用到的所有手册”。

20 世纪 70 年代软件体系结构概念进一步明确，并且出现了体系结构描述语言。1972 年 David Parnas 提出了信息隐藏模块的概念，于 1974 年提出了软件结构的概念，于 1975 年提出了程序家族概念：将一些程序划分成一组，即一个程序家族。1976 年 F.Deremer 和 H.Kron 设计了模块互连语言(Module Interconnection Language, MIL)用于描述结构化的基于模块的程序。 1978 年 Tony Hoare 提出 CSP 语言（Communicating Sequential Processes）描述并发系统各部分交互，CSP 是并发数学理论，比如进程代数、进程演算等知名理论的家族的一员。1983 年 Butler Lampson 在“*Hints for Computer System Desigin*”一文中总结了许多有关计算机系统设计的一些通用的注意事项。

进入 20 世纪 90 年代，软件体系结构进入迅速发展阶段，体系结构方法和语言研究大量涌现。1991 年 Winston W.Royce 与 Walker Royce 首次把软件体系结构定位在技术和实现之间。1991 年 Philippe Kruchten 在文章《*An Iterative Software Development Process Centerer on Architecture*》中把迭代开发与体系结构相结合并定义了使用大型命令与控制系统的多种观点。1992 年 D.E.Perry 和 A.L.Wolf 在 ACM SIGSOFT Software Engineering Notes 表“*Foundations for the Study of Software Architecture*”，建立了软件结构的根基，并提出一种软件工程的模型，该模型由三部分组成：elements, forms, rationle。

20 世纪 90 年代中期卡耐基梅隆大学在软件体系结构方法上做了很多工作。1995 年卡耐基梅隆大学软件工程研究所提出 SAAM（Scenarios-based Architecture Analysis Method）方法，是一种非功能质量属性的体系结构分析方法，用于评估体系结构对于特定系统需求的使用能力，也能用来比较不同的体系结构。同年，卡耐基梅隆大学的 Robert J. Allen 开发出 Wright 体系结构描述语言，该语言从组件，连接器，角色和端口等概念入手界定了一种软件架构；其主要特点是将 CSP 用

于软件体系结构的描述，从而完成对体系结构描述的某些形式化推理(包括相容性检查和死锁检查等)。但它仅仅是一个设计规约语言，只能用于描述，无法支持系统生成。同年，David Garlan 和 Mary Shaw 认为软件构架是设计过程的一个层次，应处理算法和数据结构之上关于整体系统结构设计和描述方面的一些问题，加大体组织结构和全局控制结构。卡耐基梅隆大学的 David Garlan 同年提出 ACME 语言，旨在各种 ADL 语言之间的转换:支持 ADL 之间的映射及工具集成的体系结构互交换语言。其目标是作为体系结构设计的一个共同的互交换格式，以便将现有的各种 ADL 在这个框架下统一起来;而它本身也可以看作是一种 ADL。1995 年，由伦敦帝国学院开发 Darwin 描述语言，这是一种面向对象或者面向组件的语言，Darwin 语言开发的程序的一般形式是一颗树，树的根节点和中间节点是复杂组件，叶子节点是封装行为的原始组件。同年，由斯坦福大学的 David Luckham 开发出 Rapide 语言:一种事件驱动的 ADL，它以体系结构定义作为开发框架，支持基于构件的开发。该语言提供了建模、分析、仿真和代码生成的能力，但是没有将连接子显式地表示为一阶实体。

1996 年 Rational 软件公司创造的软件工程方法 RUP，是一个面向对象且基于网络的程序开发方法论，描述了如何有效地利用商业的可靠的方法开发和部署软件，是一种重量级过程，因此特别适用于大型软件团队开发大型项目。它有三大特点：（1）软件开发是一个迭代过程；（2）软件开发是由 Use Case 驱动的；（3）软件开发是以架构设计（Architectural Design）为中心的。1996 年，加利福尼亚大学欧文分校 UCI 开发语言 C2，一种基于消息传递的体系结构描述语言，主要是应用于带有图形用户接口(GUI)的应用系统。C2 风格的核心在于构件之间的“有限可见性”，即处于系统中某个层次的构件只能“看到”上层的构件，而不清楚下层到底是什么构件在与之进行通信。

1997 年，OMG 组织（Object Management Group，对象管理组织）发布了统一建模语言（Unified Modeling Language，UML）。UML 的目标之一就是为开发团队提供标准通用的设计语言来开发和构建计算机应用。UML 提出了一套 IT 专业人员期待多年的统一的标准建模符号。UML 的主要创始人是 Jim Rumbaugh、Ivar Jacobson 和 Grady Booch.其中，Grady Booch 提出了面向对象软件工程的概念;James Rumbaugh 提出了面向对象的建模技术并引入各种独立于语言的表示符;Jacobson 于 1994 年提出了 OOSE 方法，其最大特点是面向用例(Use-Case)，并在用例的描述中引入了外部角色的概念。UML 随后不断演化产生了 UML1.2、1.3 和 1.4 版本，其中 UML1.3 是较为重要的修订版本。并于 2003 年推出 UML2.0。

20 世纪 90 年代后期之后，软件体系结构进入高级发展阶段，重视体系结构在软件开发实践中的风格、质量属性。1998 年卡内基梅隆大学提出了 ATAM（architecture tradeoff analysis method），该方法旨在为软件系统通过探索权衡取舍与灵敏点，选择一个合适的体系结构。该方法优点：促进精确质量要求的收集，在早期开始创建架构文档，促进在生命周期的早期风险识别等。

1999 年，一种软件产品线工程方法-BAPO 出现，即 Business/Architecture/Process/Organisation，该方法覆盖了软件工程的四个评估维度（商业、架构、流程和组织），其中对架构维度从三个方面考虑：重用资产，参考架构，可变性管理。

2000 年 MCC（Micro-electronics and Computer technology Consortium）开发了 ADML（The Architecture Description Markup Language），这是一种基于 ACME 的语言，ADML 主要应用在企业体系结构一层；ADML 是一个标记符号，提供体系结构描述文本符号。同年，飞利浦研究实验室为软件密集型的电子产品族开发了面向构件的平台架构 COPA（Component-Oriented Platform Architecting）方法。COPA 方法的目标就是在业务，架构，过程和组织中达到最佳的适应性。COPA