Software Architecture
(Forth Edition)

软件体系结构
（第4版）

覃征 李旭 王卫红 编著

清華大學出版社

软件体系结构

（第4版）

覃 征 李 旭 王卫红 编著

清華大學出版社
北京

内 容 简 介

随着软件工程的不断发展，软件体系结构逐渐成长起来，目前已独立于软件工程研究之外成为计算机科学的一个重要的独立学科分支，是软件系统开发的重要组成部分，是当今业界和学术界的热点研究领域。软件体系结构的目标是为软件开发者提供统一的、精确的、高度抽象的和易于分析的系统信息，从而使软件系统能以最快速度低成本、高质量地构建。本书详细介绍和分析了软件体系结构的理论基础、研究内容、当前的发展状况和实践应用等。通过本书，读者可以了解软件体系结构的研究背景、基本概念、描述方法、设计风格、评估方法、开发工具和柔性软件体系结构等知识。本书采用最近几年的案例、数据、图示以及其他相关材料以反映软件体系结构的最新发展。

本书可以作为计算机、软件工程以及相关专业的研究生和本科生学习软件体系结构的教材和参考书，对从事软件体系结构研究和软件开发的科研人员也有一定的理论参考价值和实用价值。

图书在版编目(CIP)数据

软件体系结构/覃征，李旭，王卫红编著. —4 版. —北京：清华大学出版社，2018
ISBN 978-7-302-51144-1

Ⅰ. ①软…　Ⅱ. ①覃… ②李… ③王…　Ⅲ. ①软件－系统结构　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2018)第 201469 号

责任编辑：张　民　战晓雷
封面设计：杨玉兰
责任校对：时翠兰
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954

印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：18.5　　**字　　数**：449 千字
版　　次：2004 年 1 月第 1 版　2018 年 10 月第 4 版　　**印　　次**：2018 年 10 月第 1 次印刷
定　　价：49.00 元

产品编号：079771-01

前言

FOREWORD

软件体系结构是计算机科学领域的重要研究分支之一，越来越多的研究人员关注如何快速、低成本地构建合理、可靠的软件系统，尤其是应用于大型、复杂场景的软件系统，如航空航天、国防等领域的智能化软件系统。而构建大型、复杂应用场景的智能化软件系统较之几十年前的软件开发过程要困难、复杂得多，特别是急剧增长的信息化、智能化社会需求以及大数据、云计算时代的来临，使得相关的软件系统的构建在迎来新的发展机遇的同时也面临着更为严峻的挑战。

在计算机软件设计的早期，软件工程师致力于如何操作计算机，并使其正常地工作，正确地解决问题。数据的组织和算法的实现是当时软件设计的核心过程。随着越来越多的底层工作，如内存管理、网络通信等被自动化或者至少可以用更小的代价来重用，程序员和设计人员通过使用高级编程语言和可以提高生产效率的开发工具，可以将更多的精力放在问题本身而不用再抱着机器代码手册埋头苦干，软件开发变得快捷、容易起来。后来，随着软件危机的出现，如何在最短的时间内低成本地构建高质量的软件系统便成为业界和学术界关注的新焦点。为了解决软件危机，两大主流思想应运而生，一个是形式化方法，另一个便是软件工程的思想。软件工程思想的引入曾经一度极大地缓解了软件危机。在软件工程学科中，软件体系结构是其重要内容之一。但随着当代软件规模、复杂度的增加，软件体系结构的重要性越来越高，其理论思想、方法、工具等有了较为巨大的发展，目前已经成为独立于软件工程之外的一个学科而受到广泛重视。例如，当来自军事、国防、企业等领域的需求问题越来越复杂、软件规模越来越大时，软件的结构也相应地变得越来越复杂，这就使得软件的质量控制也变得更为困难。软件总是有结构的，如何设计、描述、评价这些结构便成为软件开发过程中至关重要的问题，这也正是软件体系结构研究的核心内容。若将软件比作一座建筑，那么建造一个狗窝、一个简易平房以及一座摩天大厦的复杂度大相径庭，相应地，采用的建筑方法、结构、风格、设备等也迥然不同。较之狗窝、简易平房，摩天大厦结构更为复杂，若结构设计不合理，极端情况下可能会造成巨大的灾难性后果。同理，对于大型、复杂的软件系统而言，如果没有一个良好的体系结构，其可能造成的灾难性后果将会极为严重。因此，软件体系结构作为软件开发过程中的重要内容，越来越引起人们的重视，如今已成为一个独立发展的学科而备受关注。我们相信，软件体系结构是处理这些复杂软件系统构建问题的关键。

然而，许多人最近几年才了解软件体系结构这个概念。事实上，软件体系结构有着相当长的历史，早在 C 或 C++ 语言出现之前，一些计算机科学家就已经注意到软件体系结构的概念以及它对软件开发的影响。20 世纪 90 年代，软件体系结构作为一个新的学科开始蓬勃发展，许多以软件体系结构为主题的团体建立起来，相关的研讨会和学术会议也纷纷召开。同时，有关软件体系结构的文章、书籍和工具的数量激增。今天，软件领域中

负责软件设计、分析并处理来自不同涉众的不同关注点和需求关系的职位——软件架构师，已经被普遍认为是软件开发团队的核心。但是值得关注的是，大多数软件架构师并没有专门针对这个领域进行系统的学习、研究或者接受培训。他们中有些人甚至认为软件体系结构与人工智能或者数据挖掘等领域不同，根本就不需要进行科学研究和学术探讨。这种观点出现的原因是软件体系结构还没有可以被广泛接受的定义，也没有理论和实践方法的事实标准。同时，软件体系结构的快速发展和分化也导致了其过多子领域和分支的出现，而这些分化出来的产物既不能很好地普及，相互之间也很难统一。这些都成为学习和研究软件体系结构的困难。

本书在第3版的基础上，充分借鉴了作者在研究生课程教学实践过程中的大量经验、反馈意见及最新的研究成果等，做了进一步内容修订，更为系统地阐述了软件体系结构的一些经典理论和最新进展，并试图让读者领悟到软件体系结构的本质。

1. 目标

本书是软件体系结构领域的入门书籍，将对其基础理论、子领域、当前的研究动态和实践方法进行介绍。通过本书的学习，读者可以了解软件体系结构的基本概念，例如软件体系结构的必要性，软件体系结构的形式化语言描述方法，软件体系结构风格在实践中的广泛应用和认同，软件体系结构在软件系统开发过程中的应用。一些学习案例、数据、插图和其他材料都是最近几年才被发布的，这些材料有利于了解软件体系结构的最新进展。

2. 如何阅读本书

遵循深入浅出的原则，本书分为两大部分。

(1) 基础理论，包括第1~4章。

(2) 研究部分，包括第5~8章。

每章的简要介绍如下：

第1章：软件体系结构的起源和发展。本章对软件体系结构进行基本介绍。通过本章，读者可以了解到软件体系结构的必要性、发展历史和一般性定义。希望本章的介绍可以使读者对软件体系结构的整体内容有大致的了解，对当前研究的热点和方向有清晰的认识。本章是进一步了解后面各章内容的基础。此外，一些软件体系结构的概念和应用随着当前研究的进展在不断发生变化，本章最后一节将给予说明。

第2章：软件体系结构风格和模式。本章是全书的重点，对不同的软件体系结构风格进行了划分，在每一个具体的类别中都详细地进行了讲解和介绍。为了使读者对各种风格有比较深入的理解，在介绍理论的基础上格外注重每种风格的案例剖析。希望读者在阅读本章的过程中抓住每种风格的核心问题，把握各种模式的根本要素，把案例视为对理论的实践与提高。在分门别类地介绍常见的风格和模式后，本章继续提供一些案例。每个案例中集合了多种风格和模式。在学术上，这些被称为异构风格构建。事实上，实用的软件通常都要同时采用多种风格，不管这个软件多么简单。本章的目的是将抽象的风格和模式与实际应用结合起来。

第3章：软件体系结构描述。如何描述软件体系结构是软件体系结构领域的核心问

题。它是表述软件设计、在涉众间进行有效沟通以及根据需求进行软件行为校验的基础。本章将重点放在软件体系结构基于数学的形式化描述上。对于在实践中广泛使用的UML，可以参考其他文献和学习资源。

第4章：软件体系结构级别的设计策略。本章介绍基于形式化的体系结构设计。不同于实践中常用的开发过程(如RUP)中的设计，形式化的体系结构设计策略强调功能空间和结构空间之间的联系和演算关系(空间的概念是对现实过程的抽象)。为了更好地理解本章的内容，需要读者有基本的集合论和自动机理论知识。

第5章：软件体系结构集成开发环境。本章主要介绍一种软件体系结构的集成开发环境，其中详尽地阐述了该环境的使用原理、内部机制、使用过程中的注意事项以及它是如何辅助人们进行设计、开发、维护软件体系结构的。为了方便阅读和使用，本章详细地介绍了软件的安装过程、使用规范以及使用的其他细节。读者通过本章的学习，可以利用软件来辅助自己设计更为复杂、新颖的软件体系结构。

第6章：软件体系结构评估。在软件体系结构的初步设计完成之后，任何涉众都有理由搞清楚这个设计是好是坏，是否能够为项目的成功开发奠定基础，是否能够满足预期的需求而不会由于设计缺陷而失败，这就是评估的任务。本章介绍并比较目前被广泛使用的几种评估方法。大部分评估方法缺乏形式化基础，更多地要依赖参与者的经验和能力。因此，本章主要介绍实践中用到的评估方法和技术。

第7章：柔性软件体系结构。柔性软件体系结构是当前的研究热点之一，与传统软件体系结构相比，柔性软件体系结构在动态的环境中有着极其重要的优势，这也是将这一内容独立成章的原因。本章介绍什么是柔性软件体系结构，为什么使用柔性软件体系结构，如何使用柔性软件体系结构。在介绍的过程中注重理论结合实际，用浅显的例子对复杂的理论进行说明和解释。

第8章：软件体系结构的前景。本章着重介绍未来软件体系结构的发展及其对其他领域可能产生的影响。希望本章的介绍能够使有志于研究软件体系结构的读者了解软件体系结构研究的方向和目标，了解目前这一领域中主流的研究热点和方向。

考虑到每章的相对独立性，读者可以选择感兴趣的几章来阅读。此外，读者也可以通过参考文献获得关于一些问题的更详细、更深入的描述和解释。

3. 哪些人该阅读本书

软件设计和软件开发相关专业的研究生和本科生能够从本书中获得他们想要的知识。其他对软件体系结构感兴趣的人也可以将此书作为入门书籍。有经验的软件设计从业人员和项目主管也可以阅读本书，因为软件体系结构是他们每天必须接触的工作内容。

本书假设读者具有下面的基本经验(但并不是每个章节都需要)：

(1) 使用C++、Java或C#编写程序。

(2) 软件设计(即使仅仅是简单的项目也可以)。

(3) 软件项目管理。

4. 致谢

非常感谢清华大学软件体系结构小组出色的工作，特别是李旭研究员、王卫红教授。

他们对本书的专注、协作精神和勤奋是本书撰写过程中的不竭动力。

最近几年，我一直在考虑软件体系结构中的一些问题，并希望能有一个机会把它们写出来。

在本书的编写过程中，作者得到了许多人的帮助和支持。感谢第3版编写组的陈旭博士、李志鹏博士、叶文文博士，以及课题组的王斌旭、徐涛、李经纬；感谢刑建宽博士、郑翔高级工程师、董金春教授/研究员在本书第2版撰写过程中的出色研究工作；同时也感谢在第1版和第2版撰写过程中做了大量工作的王娟高级工程师、曹辉博士。由于第3版的出版在社会上产生了广泛的影响，为本书的出版打下了扎实的基础，在此向以上人士深表感谢。

感谢清华大学出版社对本书出版的大力支持。

覃　征

2018年5月

CONTENTS

目录

第1章　软件体系结构的起源和发展

1.1　软件的产生与发展

从算盘到图灵机，再经历了电子管、晶体管、中小规模集成电路，直至如今的大规模和超大规模集成电路计算机设备这4个时代，计算机的硬件按照摩尔定律以令人惊奇的速度飞速发展着。而软件则是与硬件相对的概念，甚至有人将软件比作计算机的灵魂。

软件最初是具有特定领域知识的研究人员为完成计算任务而设计的专门在特定计算设备上运行的一系列数据处理任务，早期的软件更多由工作业务或科研计算的需求而产生，直到后来慢慢开始出现一些服务于人们日常生活的小型软件。21世纪以来，计算机软件逐渐渗透到各行各业，尤其是在个人计算机普及之后，更是给人们的生活方式带来了超乎想象的变化。人们可以通过软件进行购物，可以通过软件订餐。有各种即时通信软件让沟通交流变得简单高效，有各种游戏为人们增添了新的娱乐方式，有各种办公软件帮助人们完成业务工作，也有各种学习软件可以使学习者获取免费在线课程，作曲家有编曲软件进行创作，工程师有建模软件对工件或结构进行模型评估……应该说，软件如今已是在人类社会中不可或缺的一种重要工具。

软件可以被定义为一系列按照特定顺序组织的计算机数据和指令的集合。GB/T 11457—2006《信息技术　软件工程术语》中对它的定义是“与计算机系统操作有关的计算机程序、规则，以及可能有的文件、文档及数据”。软件是建立在计算机硬件资源上的所有数据与计算机指令的集合，在一些定义中还包括与软件配套的文档。简单来说，不应认为软件仅仅是计算机程序。第一个关于软件的理论要追溯到1935年图灵的论文 *On Computable Numbers, with an Application to the Entscheidungs problem*。但直到1946年，软件的主体才形成了现在人们所公认的形态——计算机中存储的程序。而软件的发展以程序语言的发展为突出表现。

程序语言实际上就是对现实问题的数学抽象描述工具。抽象是简化真实系统、活动或其他实体的过程。在这个过程中，与本质内容无关的琐碎细节被忽略。为了得到计算机构造问题的解决方案，人们对问题进行抽象并用编程语言实现。抽象得到的目标模型很大程度上影响了程序员对问题的理解。到目前为止，编程语言的发展一直是在提高它们的抽象级别，即从对计算机的底层操作逐步到解答问题本身。

20世纪50年代，程序存储式计算机开始流行，并深深地影响了当时程序员的工作方式。程序员使用可以直接在计算机上执行的机器指令和以字节、字或双字表示的数据来描述他们的处理逻辑。指令和数据在存储器中的布局要依靠手工来组织，也就是说，程序员必须精确地记住每一个常量和变量在存储器中的起始和终结的位置。当程序需要更新时，程序员要花大量时间来检查和改变数据和执行代码的每一个引用以保证程序的一致性，因为它们的位置发生了改变。

不久,一些人认识到这些底层工作能够自动完成并得到重用。结果,符号替换和子过程技术出现了。它们的伟大之处在于它们将程序员从琐碎但是又不得不做的工作中解放出来。然而,一些常用的模式,如分支条件控制结构、循环结构、数值运算表达式的求值仍然必须被实现为机器指令才能执行。这些工作极大地干扰了程序员的注意力,使他们不能专注于问题本身。于是,高级编程语言开始发展。在20世纪60年代中期,IBM公司开发的FORTRAN语言由于其方便和高效的特点一举成为科学计算领域最重要的语言。

在20世纪60年代后期,Ole-Johan Dahl和Kristen Nygaard发明了Simula语言。它是ALGOL的一个超集,体现在引入了面向对象的思想。FORTRAN中的数据类型系统仅是简单地将FORTRAN中的类型和机器语言中的原始数据类型建立了映射关系。与此相反,面向对象思想则将数据类型看作现实问题中实体的抽象。尽管FORTRAN和C语言都有"结构"和"联合"这样的元素,但它们仅仅表示了一个类型中有哪些数据。和这些类型有关的操作与类型本身是分开的。而面向对象的规则,如封装、实现细节隐藏、访问权限控制和多态,完全没有被涉及。随着C++这种面向对象语言的发展,程序开发世界被彻底改变了。

C++和其他同时代的面向对象语言最初的目标是将"类"作为基本的重用单元。然而,这些语言的设计和实现本身就注定了这个目标无法达成。一方面,类的元数据的缺失造成类不能像预期那样可以在不修改类外引用代码的前提下完成类内部实现的修改;另一方面,类和类实现之间的通信协议没有很好地分离,限制了其复用的能力。人们看到,大部分的C++程序的复用都发生在源代码级别,而二进制级别的复用通常会造成更多麻烦。你能从文献(Joyner,1996)中获得这方面更详细的信息。当人们发现软件能够被若干独立的零件组装起来,并且这样做可以让大型系统的构建耗费的时间和成本大为降低时,他们开始明确这个观点:找到一种合适的复用单元并确立这种单元的使用原则是多么重要。文献(Ning,1996)给出了第一个完整的基于构件的软件开发模型。

构件通过增加软件组成构件块这样的概念来提升软件设计级别。这样做的好处是它能让设计人员在定义和遵守严格的通信协议的前提下,通过使用独立的构件来组装系统。面向对象是构件开发的良好基础,但并不是所有构件都必须用对象才能实现。在20世纪90年代中期,COM和COBRA开始流行起来,因为它们扩展了C++以及其他类似的语言,满足了构件模型的需求和原则。Java和.NET平台从一开始就支持构件级别的开发和部署。这是因为它们明确地使用了"接口"和"元数据"这样的语言元素。更重要的是,使用UML建立的设计模型可以被轻而易举地转换为这两种平台上的代码。UML包含了来自无数设计人员、软件工程师、方法学专家和领域专家的概念、建议和经验,提供了一整套基本符号来让人们将注意力放到构件、构件之间关系和它们的约束等方面。换句话讲,UML对软件的描述达到了目前软件抽象的顶峰。

1.2 软件危机的出现与软件工程的兴起

在20世纪60年代,随着硬件电子技术的发展,计算机的性能越来越强大,软件的应用领域越来越宽广,工程量也越来越庞大。但是由于软件开发并未成为一个系统的工程学分支,几乎没有人关注软件项目的管理工作。软件复杂度逐渐增高,可靠性问题逐渐凸显出

来，软件维护变成了一件难以进行的操作，生产率急速降低。典型的软件危机事件就是IBM 公司的 OS/360 项目，在操作系统软件开发的过程中，需求没有得到有效的约束与清晰的边界划分，缺乏正确的理论指导，软件的规模庞大臃肿，变得难以控制，复杂度极高，导致程序调试困难，软件维护难以进行。这一项目也催生了《人月神话》这一软件工程著作，其中说：缺乏良好管理的软件开发过程就如陷入一个焦油坑，而解决软件工程成本与质量的矛盾问题实际上也并不存在“银弹”。与 IBM OS/360 相似的爆发软件开发问题的项目还有很多，如美国银行信托软件系统开发案等。软件规模大小不一，业务应用服务于各行各业，全球性的普遍问题爆发，软件行业面临前所未有的困难与挑战。1968 年，北大西洋公约组织(NATO)正式将这一时期的问题称为“软件危机”。

软件危机的爆发不仅严重影响了计算机行业的理论科研与工程实践的进展，而且影响了涉及计算机软件的各行各业的发展。也正是在这样的背景下，NATO 提出了软件工程的概念。

软件工程概念的提出标志着软件开发正式成为有工程学理论知识指导的工程项目。它涉及软件开发过程的控制，通过不断总结前人软件开发经验所形成的一套软件开发方法论，以及相应出现的专门用于软件开发工作或管理组织的工具，能够合理调控人力、时间、资金等软件开发成本，使得软件开发与维护更加顺利。

软件工程的关注点在不断变化，软件危机过后，人们开始关注如何组织代码和数据以增强程序可读性、可追踪性、可调试性和可维护性的问题。这次改变现在被称为“结构化”。非结构化的程序可以看作一条连续的指令序列集。在这个指令集中，可以通过跳转语句跳到任何位置。汇编语言是典型的用于编写非结构化程序的语言。然而，无限制地使用跳转指令会导致严重的后果。读者可以从文献 *Go To Statement Considered Harmful*(Dijkstra，1986a)中找到对 GOTO 指令的一段经典的批评。在编写结构化程序时，整个程序被拆分为若干子过程。它们的执行需要依靠调用才能进行。通过使用结构化的组织策略，软件设计人员开始采用自顶向下的设计理念，即先将大规模的软件拆解成若干小的模块，然后分别对这些小的模块进行详细的设计。这些子过程之间的关系仅仅是简单的调用，一个子过程调用一系列其他的子过程。这个关系不断递归，直到原子操作为止。最顶层的过程可以看作系统的一个组成部分。于是当时的设计通常使用控制流程图来描述任务是如何被一步一步执行的，这将指导程序在运行时的执行方式。

然而结构化并不是现实世界的良好映像，因此它也很容易带来一些问题和风险。设计人员依旧需要将问题模型转换到结构化模型，然后再将其拆分成模块。这个过程并不是那么自然。此外，代码重用也不能轻而易举地进行，原因是：如果想重用一个过程，那么与它相关的一系列数据结构也必须被引用，但是这些数据结构并不一定被实现在一个制品[①]当中。因此，以数据为中心的组织方式开始逐渐流行起来。在以数据为中心的组织方式中，数据实体的行为属于这个实体(而不是像结构化方式那样将二者分开)。越来越多的设计人员逐渐开始热衷于将包含相关操作的数据类型打包，并以此作为复用单元。面向对象明确了对这种组织方式的支持，并借助继承、多态的特性对其进一步扩展。从 20 世纪 80 年代中期开始，对问题中的实体及其关系的建模开始逐步转到新的设计方式上来。软件设计师可以

① 制品(artifact)是指代码实现或信息所在的实体，如一个可执行文件、一个库或者一个数据库表等。

直接用问题中的概念和词汇来思考他们的系统的结构。

然而不幸的是,面向对象不是万能药。举个例子,纯粹的面向对象不能很好地解决概念交叉的问题。例如在商业系统中,类 Customer 的实例和 Transaction 的实例的紧密耦合会导致对其中一方的修改迫使另一方也必须修改。再如使用 Log 类(日志)是典型的多个类跨越一个类的例子。这时,原本认为不会再出现的恼人的更新工作又回来了。最近,面向方面程序设计试图修正这个问题。在面向方面程序设计中,设计者将实体划分为两类:独立的(如 Customer)和交叉的(如 Transaction 或 Log)。通过明确地指出交叉点和交叉样式,面向方面的编译器可以自动帮助人们处理这些交叉的工作。面向方面是面向对象的有益补充,但仍然与其处于同一级别。

站在更高的级别上,面向对象本身并不能解决对象之间复杂的交互问题。不像多年前那样,现在的软件的复杂度急剧增加,主要是由于软件的执行方式已经从单独运行转变为协同工作。结果,交互和数据交换的方法和技术在目前得到了相当的关注。一些交互方式,如调用、点到点的信息传送、发布-订阅,被广泛地应用于各种已经实现了的通信协议中。从大多数的大规模系统可以看出,软件的行为可以被分为两类:一是计算行为,主要处理业务运算;二是体系结构行为,即系统的交互行为。不论是结构化还是面向对象都不能明确地将这二者分开。尽管面向对象给人们提供了良好的设计单元,但它却无法清晰地表现软件的运行时结构(例如,C++ 编写的程序在运行时的结构和 C 程序几乎一样;而 Java 和. NET 平台也仅仅是在执行时存储元数据而已)。此外,面向对象中的"接口"相当原始。它只是规定了方法的签名,却忽略了大量其他和交互协议相关的内容,如对方法的性能要求或对内存占用的要求等。设计中的"接口"包含更广泛的含义,用以处理对服务的语义理解和操作等。无论如何,为了获得更加明晰的交互机制,人们不得不自己去构造它们。为此,人们需要某种东西来描述它。

在这个级别上的另一个关注点是如何评价系统结构对软件质量的影响。功能性来自那些实现了的计算性模块。而其他的质量,如可用性、易用性和可测试性,则和系统的运行时结构紧密相关。人们可以为关键数据做冗余备份来提高性能,也可以为计算性构件附上加密功能来提高安全性。简单来讲,功能性主要是由客户的需求确定的,而其他非功能质量则是软件在运行时如何组织的结果。更重要的是,如果能够获得一套在某个领域有相当优势的结构,人们如何记录、调整和重用它呢?这就是领域体系结构,它对任何软件厂商的生存都至关重要,因为它是应用软件生产线的基础。在生产线上,可以根据需求来对领域体系结构进行微调,并主要通过组装的方式来实现软件。这从根本上降低了软件上市需要的时间和成本。

当人们将关注点放在上文提到的那些方面时就会发现,找到一个设计、记录、评估和复用的基础是非常有必要的。人们相信软件体系结构可以满足这些需要。

软件开发范式自从 20 世纪 40 年代"软件"这个词刚刚出现时(也是最原始的程序存储型计算机诞生的时候)开始至今经历了多次的革命性变迁。每次开发方法、模式和工具的改变都是为了适应新的环境和新的需求。人们相信软件体系结构就是下一次革命。许多人已经开始顺应这个趋势;当然,也有很多人根本不理会软件体系结构,就像若干年前人们不愿改变他们的开发习惯,采用新的开发技术一样。历史总是有相似性的,从历史的角度,人们能够更清楚地看到软件体系结构如何逐渐成为当今软件工业的重要组成部分,以及为什么

人们应该改变习惯去尝试它的原因。

随着编程语言的进化，软件开发的重点也一直在改变。众所周知，软件工业若想获得成功，具有竞争力的开发速度、产品质量和产品满足客户需求的程度是必须被保证的。因此，在软件开发过程中，主要的关注点被放到了如何发现和解决那些不利于上述因素的瓶颈上。当然，这要依靠一些工具和方法才能做到。

实际上软件危机的问题并未完全得到解决，软件工程只是提出了理论指导与方法论以缓解这一矛盾。不恰当的软件开发过程与管理仍然会使软件开发困难重重。因此软件工程这一学科也在不断发展与完善，还要克服更多困难，迎接在新时代新技术背景下出现的新的挑战。而与此同时，有理由相信软件体系结构将会再一次改变软件开发模式。但就像高级编程语言和 UML 的关系一样，软件体系结构不能完全取代现有的方法和工具，它只是一个有力的补充，帮助人们处理大型软件系统的快速开发和升级问题。

1.3　软件体系结构的诞生与发展

1.3.1　软件体系结构诞生的背景及意义

1. 诞生背景

软件体系结构的起源可以追溯到 20 世纪 60 年代后期。那时软件危机正在肆意横行。当时软件的成功对于整个系统的成功具有决定性作用，这是因为相对于硬件，软件设计师在选择和组织软件结构时有更自由的空间。但是软件开发过程与其他工业产品的生产（如汽车、机械等）有很大不同，即无法明确地区分软件开发的阶段。这就是说，不能用常规的经验来规划它。1968 年，NATO 软件工程大会在德国召开，会上确立了软件工程的科学地位，就是为了解决上面的问题。

随着软件工程学科的发展，人们逐渐意识到，越来越庞大的软件程序更加需要一个清晰的结构来支撑运行与指导维护，这就像建筑学里巨大的摩天楼也需要前期完善的楼体结构设计一样，只有恰当的楼体结构才能保证高楼不倒，也只有恰当的软件体系结构才能保证软件的鲁棒性。

与几十年前那种着重于机器指令或者倾心于数据结构和算法的集合的软件相比，现在的软件更加复杂，更加难以控制和维护。一般来讲，软件系统是通过构件装配而成的，不管这些构件是为了满足需求而开发的还是堆在复用库中的。在这种环境下，一个团队需要面对系统的不同侧面。他们有的要处理必须实现的功能模块，有的则要让不同构件正确通信，从而良好协作。同时，在这个过程中，一些质量因素也必须得到保证，目的是确保项目的最终成功。

简单地增加程序员的数量不一定能提高生产力，相反，却容易引起项目的失败（Brooks，1975）。软件开发不是简单地组装零件；相反，在这个过程的背后隐藏着相当复杂的关系，有些至今还没有被归纳出来。大多数人认为，“软件体系结构”这个概念在 20 世纪 90 年代开始有了较严格的定义。

首次提到软件开发中的“体系结构”这个概念的文字记录可以在 Edsger Dijkstra 的论文 *The Structure of the “THE” Multiprogramming System*（1968b）中找到。Parnas 在论