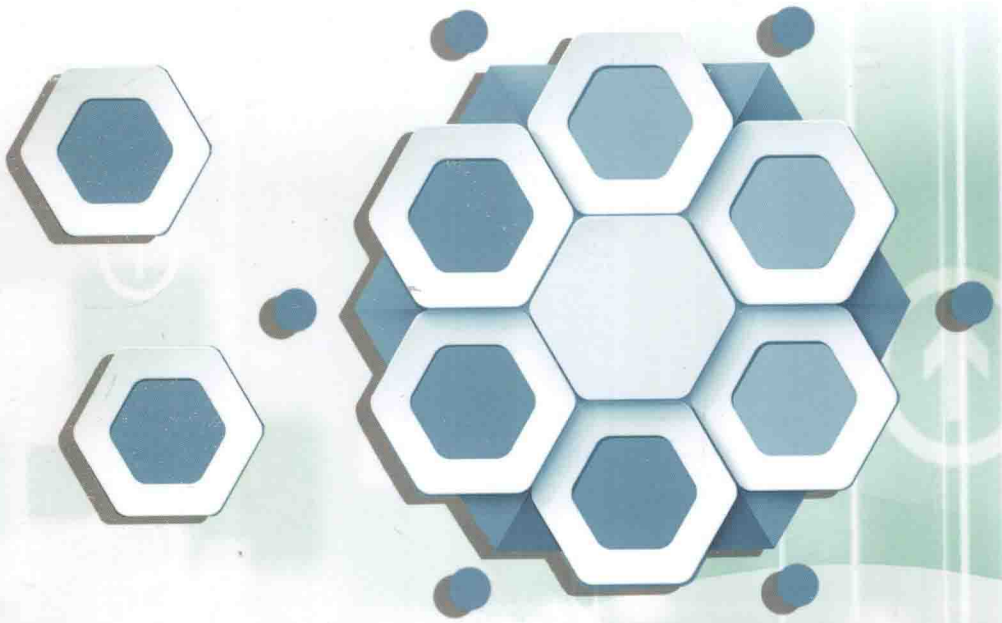
卓越工程师计划：软件工程专业系列丛书

软件体系结构与设计

实用教程

尚建嘎　张剑波　袁国斌　编著



科学出版社

卓越工程师计划：软件工程专业系列丛书

软件体系结构与设计实用教程

尚建嘎　张剑波　袁国斌　编著

科学出版社
北京

内 容 简 介

本书从实用的角度出发，围绕软件架构开发中架构需求、架构设计、架构编档、架构实现等核心过程，结合大量案例，阐述了软件体系结构与设计的有关理论方法与技术，包括软件体系结构的基本概念、软件质量属性、软件体系结构风格、软件体系结构描述与建模、软件体系结构设计与评估、软件体系结构实现与测试、软件体系结构和软件产品线。全书理论联系实际，通过引入几十个案例，包括一个实际项目软件架构设计文档的详细描述，尽可能结合模型（UML 图）、文档甚至代码等具体形式阐述软件体系结构理论知识，有助于读者学习理解有关知识点及提高动手解决实际问题的能力。

本书可作为高等院校相关专业本科生和研究生教材，也适合从事软件设计、开发、项目管理等工作的业界人士参考。

图书在版编目（CIP）数据

软件体系结构与设计实用教程/尚建嘎，张剑波，袁国斌编著. —北京：科学出版社，2016.11

（卓越工程师计划：软件工程专业系列丛书）

ISBN 978-7-03-050477-7

Ⅰ.①软… Ⅱ.①尚… ②张… ③袁… Ⅲ.①软件-结构设计-高等学校-教材 Ⅳ.①TP311

中国版本图书馆 CIP 数据核字（2016）第 265245 号

责任编辑：闫 陶/责任校对：董 丽

责任印制：彭 超/封面设计：陈明亮

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

武汉中科兴业印务有限公司印刷

科学出版社发行 各地新华书店经销

*

开本：787×1092 1/16

2016 年 11 月第 一 版 印张：18 1/4

2016 年 11 月第一次印刷 字数：429 000

定价：40.00 元

（如有印装质量问题，我社负责调换）

卓越工程师计划:软件工程专业系列丛书

编委会

前　言

当今，以互联网为基础的云计算、物联网、移动计算等技术迅速发展，软件的规模和复杂性越来越高，所运用的技术也越来越呈现集成的特点，仅从代码片段很难认识软件。正所谓“只见树木、不见森林”的感觉，人们越发认识到软件体系结构在软件开发中的作用，通过体系结构这个“森林”涉众可以很快了解软件的整体结构，便于指导后续工作及大家的交流、协作。人们认识到研究整个系统的结构和规格说明对提高软件生产率和质量越来越重要，迫切希望对软件体系结构有关理论方法和技术进行系统、深入学习并加以运用。

软件体系结构的真正发展起步于20世纪90年代中后期，作为软件工程领域一个相对比较新的研究领域，近年来开始被人们广泛接受，社会上出现了“软件架构”“架构师”等概念及相关培训、认证，越来越多的高校在软件类专业中开设软件体系结构这门课。但在软件体系结构学科发展和教学中存在一个令人尴尬的境地，人们对“软件体系结构”表现出“既爱又恨”的心态，爱是因为需要，恨是难学、难用。

之所以会出现“难学、难用”的情况，一方面，软件体系结构具有抽象、过程模糊的特点，与人们已有知识结构脱节，因而较难以掌握，突出表现在学习了有关理论方法后仍不知道如何理论联系工程实际加以运用；另一方面，现有教材大多偏重于理论方法，即使有案例也大多可操作性不强，缺乏能让读者“看得着、摸得着”的有形的东西，如模式、风格，视点、视图，极易混淆等相关概念众多，软件体系结构的设计活动、具体步骤及与其他软件过程的融合，编档缺乏好的模板、实例等。

在软件体系结构教学中，由于授课对象多是几乎没有什么项目开发经验的大学生，在他们头脑中很难将软件体系结构的抽象理论和实际联系起来，较容易产生枯燥乏味的感觉。我们认为存在这一问题的根本原因是软件体系结构的知识体系层次超出了学生已有知识体系层次太多，出现了知识断层。在软件设计领域，存在“软件体系结构”“设计模式”“算法＋数据结构”三个层次，分别对应高层大粒度、中层中等粒度、低层细粒度软件构件。一般来讲，通过大学期间的专业学习，学生往往掌握了一定的程序设计语言及数据结构、算法方面的知识，习惯了对这类具体、规范性知识的理解和运用，当面对较为抽象，且相比较而言不够统一规范的软件体系结构方面的知识时，无法在软件体系结构和已有知识之间建立起联系，自然会感觉“难以理解”，运用更无从谈起。要解决这一问题，就要设法弥补这种知识的断层，为克服现有教材在指导工程实践方面的不足，且考虑到读者较容易接受代码、模型(UML图)、文档、工具等具象性知识，本书在编写过程中以软件体系结构理论方法的应用为导向，围绕软件架构开发中“架构需求、架构设计、架构编档、架构实现”等核心过程，通过引入大大小小几十个案例，尽可能结合UML图、文档甚至代码等具体形式阐述软件体系结构理论知识，帮助读者学习理解有关知识点及提高实际动手能力。

全书从实用角度出发，综合考虑软件体系结构知识体系要求和所涉及的相关活动划分为8章。第1章，软件体系结构的基本概念。本章主要介绍软件架构相关概念和定义，后续学习

奠定基础。第 2 章,软件质量属性。本章详细介绍了 5 类多达十几种软件质量属性,学习和掌握这些质量属性是开展后续架构活动的重要基础。第 3 章,软件体系结构风格及案例。鉴于软件体系结构风格是重要的架构设计素材,这部分内容是本书的重要章节,介绍了 10 种常用的软件体系结构风格,且每种风格都给出了 1～2 个具体案例,并给出了实现代码。第 4 章,软件体系结构描述与建模。本章介绍了几种常用的体系结构描述方法,重点介绍了 Kruchten 4+1视图模型及 UML 表达法。第 5 章,软件体系结构设计与评估。本章也是本书的重要章节,介绍了架构为中心的软件开发过程,属性驱动和基于模式两种设计方法,构件级设计与评估方法以及软件架构评估方法。第 6 章,软件体系结构编档。编档是一个操作性、综合性极强的核心软件架构开发过程,本章介绍了选择视图、视图编档、制作文档包相关内容,给出一个来自实际项目的完整软件架构编档案例,读者可以直接使用这一模板并参考该案例编写自己的架构文档。第 7 章,软件体系结构实现与测试。介绍了软件框架构造技术、常见架构级软件框架、常见架构级中间件等架构实现技术以及架构测试。第 8 章,软件体系结构与软件产品线。介绍了软件产品线及其三大基本活动、产品线实践域,给出了产品线案例分析。从软件架构需求捕获到设计、编档、实现的核心过程,读者都可以按照书中所讲述内容实施。每章后附有思考和练习题。

本书的编撰获教育部“十二五”期间“高等学校本科教学质量与教学改革工程”建设项目中“中国地质大学(武汉)专业综合改革试点——软件工程专业”子项目的资助。

软件体系结构相关知识涉及面广,本书在编撰过程中通过互联网等方式查阅了大量国内外文献资料,从中汲取知识或启发想法,由于篇幅所限,这些资料无法一一列出,在此向相关资料作者致谢。本书的编撰得到了软件工程系老师们的大力支持和帮助,研究生余芳文、郭傲、葛彬、胡旭科等同学参加了资料整理、程序编写、绘图等工作,在此一并致谢。

由于作者水平有限,加之时间仓促,书中难免存在谬误和不妥之处,敬请读者批评指正。联系方式:E-mail:jgshang@cug.edu.cn,QQ:476206397。

编　者

2016 年 8 月 8 日

目　录

第1章 软件体系结构的基本概念

1.1 软件体系结构

1.1.1 软件体系结构的概念

软件体系结构一词从字面上理解为软件的体系结构，因此要了解软件体系结构，就需要分别了解什么是“软件”，什么是“体系结构”，以及这两者之间的联系。软件在狭义上指的是计算机程序，而广义上则是指计算机程序和相关的文档（如需求规格说明书、软件设计说明书、使用手册）的集合体。传统意义上计算机程序的设计往往意味着“算法＋数据结构”的设计。随着软件系统规模和复杂性的增长，现代软件设计已经远远超出了算法和数据结构这一范畴，如何设计和说明整个系统的结构成了一个巨大的挑战，涉及总的控制结构、通信协议、约束和性能、设计元素功能分配、物理部署、设计元素的组合、设计策略选择等一系列问题。

其实“体系结构”的概念起源于建筑学，在建筑学中是指如何使用基本的建筑模块构造一座完整的建筑，其中包括两个基本元素：基本的建筑模块和建筑模块之间的黏结关系。同样，软件也总是具有体系结构的，它包括构件、连接件两个基本元素以及物理分布、约束、性能等因素，其中构件即各种基本的软件构造模块，如函数、对象、模式；连接件用于将构件组合起来形成完整的软件系统。从细节上来看，每一个程序也是有结构的，早期的结构化程序就是以语句组成模块，模块的聚集和嵌套形成层层调用的程序结构就是体系结构。因此，我们说不存在没有体系结构的软件。

在软件体系结构发展过程中，人们曾从不同角度给出了有关软件体系结构的多种不同定义。截至2005年末，有大概24个国家向卡内基梅隆大学SEI提交有关“软件体系结构”的定义多达156个。1994年，Garlan和Shaw定义：软件体系结构是设计过程的一个层次，它处理那些超越算法和数据结构的设计，研究整体结构设计和描述方法。他们认为软件体系结构由构件（component）、连接件（connector）和约束（constraint）三大要素构成。构件可以是一组代码，如程序的模块，也可以是一个独立的程序，如数据库的SQL服务器；连接件表示构件之间的相互作用，它可以是过程调用、管道和消息等；约束一般为构件连接时的条件。该模型的视角是程序设计语言，构件主要是代码模块。同样是1994年，CFRP模型定义：软件系统由一组元素（elements）构成，这组元素分成处理元素和数据元素，每个元素有一个接口（interface），一组元素的互连（connection）构成系统的拓扑，互连的语义（connection semantics）包括静态的互连语义（如数据元素的互连）和描述动态连接的信息转换的协议（如过程调用、管道等）。1995年，Boehm模型定义：软件体系结构是系统构件、连接件和约束的集合，也是反映了不同利益相关者需求的集合，同时是能够展示由构件、连接件和约束所定义的系统在实现时如何满足系统不同人员需求的原理的集合。1998年，IEEE 610.12—1990标准定义：体系结构是以构件、

构件之间的关系、构件与环境之间的关系为内容的某一系统的基本组织结构，以及指导上述内容涉及与演化的原理。近年来，SEI 从另外的角度给出了软件体系结构的定义：所谓系统的软件体系结构就是一种可用来帮助人们理解系统如何执行的系统描述。软件体系结构服务于系统本身以及开发项目，用来定义设计和实现工作组必须完成的工作任务。同时，它也是诸如性能、可修改性、安全性等系统质量的主要载体。假如没有一致的体系结构，就很难确保这些质量能被实现。软件体系结构还可看作一种早期分析的制品，它确保所使用的设计方法能创建出可接受的系统。通过构建有效的体系结构，人们可以识别出早期风险并在开发过程中加以防范。

尽管人们从不同角度所给出的众多软件体系结构概念不尽一致，但通过对已有软件体系结构定义和论述进行分析，不难看出有关软件体系结构的共性。

(1) 软件体系结构提供了一个结构、行为和属性的高级抽象，从一个较高的层次来考虑组成系统的构件、构件之间的连接，以及由构件与构件交互形成的拓扑结构，这些要素应该满足一定的限制，遵循一定的设计规则，能够在一定的环境下进行演化。

(2) 软件体系结构反映的是系统开发中具有重要影响的设计决策，反映多种关注，便于涉众之间的交流，据此开发的系统能完成系统既定的功能和质量需求。

(3) 软件体系结构的有关工作围绕以下几方面展开。

架构设计：怎样创建一个架构。

分析：最终产品质量属性与架构之间的关系。

实现：怎样基于架构描述建立一个实际的系统。

表达：创建怎样的架构制品来解决人与人、人与机器之间的交流问题。

经济：架构问题和商业决策怎样关联。

1.1.2 软件体系结构发展史

作为软件工程学科的一个分支，软件体系结构主要是针对大型软件系统结构的原理性研究。伴随着软件系统规模和复杂性的迅速增加，软件工程和软件开发方法经历了一系列的变革。在此过程中，软件体系结构研究也从最开始对系统的定性描述，发展成了对一些符号、工具、分析技术和构建方法的研究。也就是说，软件体系结构领域最初的研究领域是解释软件实践，而现在它能够为复杂的软件设计和开发提供具体的指导。如今，软件体系结构已经从基础的研究转变为软件系统设计和开发过程中的一个重要元素，已经进入到一个创新和概念形成的黄金时代，并开始进入一个技术更加成熟、应用更加广泛的阶段。按照 Redwine 和 Riddle 定义的技术成熟度模型，从实践的角度看可将软件体系结构的发展大致划分为如下 6 个基本阶段。

1. 萌芽阶段：1985～1994

20 世纪 80 年代中期至 90 年代中期可看作软件体系结构的基础研究阶段，这一阶段随着软件危机的出现，人们愈发认识到从整体而非单一模块考虑软件系统设计的重要性，软件体系结构的萌芽开始出现。这个时期的开发活动中，设计者往往会通过线框图和一些非正式的表述来描述复杂软件系统的结构。其中一些设计者意识到这些结构中存在一定的共性，并总结出一些风格，这些结构有时又被称为架构，但有关通用风格的知识却没有得到系统组织和研究。20 世纪 80 年代中期，伴随着面向对象开发方法的兴起与成熟，信息隐藏、抽象数据类型以及其他将软件元素看作黑盒的概念逐渐形成并得到普及，人们意识到计算机程序仅提供一

个正确的输出往往是不够的。同时认识到诸如可靠性、可维护性等其他软件质量属性也是十分重要的，并且能够通过设计良好的软件结构实现这些质量属性。20 世纪 80 年代晚期，人们又开始探索为特定问题提出的特殊的软件结构，形成了一些如航空、示波器、导弹控制等特定产品线或者应用领域的软件系统结构方案。另一些工作则主要关注软件结构的共性部分即架构风格，这些架构风格可被用于更加普遍的问题领域。通过将已有系统进行编排分类，人们定义了一些通用的架构风格，如管道、过滤器、仓库、隐式调用以及协作进程等。

2. 概念形成阶段：1992～1996

这一阶段人们开始研究和探索通过 ADL(架构描述语言)、形式化及分类来描述通用的软件结构，重点放在对实践中发现的系统结构的描述。通过 ADL 这种类编程语言来解释系统的组织思想，可以描述架构各方面的具体细节。这时期出现了 Aesop(探索风格的特殊属性)、C2(探索基于事件的风格)、Darwin(动态分布式系统的设计和规约)、Meta-H(实时航空电子设备控制)、Rapide(动态行为仿真和分析)、UniCon(连接件和风格的扩展集，与代码兼容)，以及 Wright(组件交互)等特定架构的 ADL。

早期叙事性的风格分类方法被逐渐扩展为风格分类和支持这种风格的元素分类。一些通用的体系结构形式被描述为模式。由 Buschmann、Meunier 等合著的《面向模式的软件体系结构：模式系统》一书是这方面的开篇之作。

这一阶段，人们开始系统研究架构决策和系统质量属性之间的关系，并将软件架构验证作为一种有效的减少风险策略加以研究。软件互连的度量、架构师检查清单，以及特定属性的架构分析技术促进了架构评估方法的发展，形成了诸如 SAAM 的评估方法。

在这一阶段最重要的当属架构视图概念的出现。早在 1974 年，Parnas 就基于其观察给出了这样的结论：一个软件系统有着多种不同的结构，这些不同的结构着重于不同的工程目标，并且单独描述任何一个视图意义不大。Kruchten 1995 年提出的“4＋1”视图模型，引起了业界的极大关注，并最终被 RUP 采纳，现在已经成为事实上的架构设计结构标准。

3. 发展和探索阶段：1996～2003

在这一阶段，人们开始全面探索软件体系结构理论方法的实践问题，架构模式作为一种非正式的设计指导受到普遍关注并被广泛采用。

有关实时系统设计的形式化分析方法也已经出现。例如，分布式仿真的高层次架构的架构说明书能够在实现之前识别不一致性，因而减少了重新设计的开销。

架构分析和评估方法也逐渐发展。在 SEI，软件架构分析方法(SAAM)被架构权衡分析方法(ATAM)代替，该方法不仅支持质量属性的分析还支持质量属性之间的交互。有关应用研究到实践的书籍为外部探索打下了基础。这一时期包括架构评估以及架构编档等有关软件架构实践的书籍开始出现，表明整个领域逐渐成熟。

另一方面，人们也展开了对架构设计策略的进一步探索，这些设计策略作为一些细粒度的架构设计决策，能够形成特定的架构模式。在这个阶段，质量属性的重要性逐渐增大，同样也伴随着达到这些质量属性的架构角色。2000 年初有许多将质量属性和架构设计决策相关联的工作，使得研发自动化的架构设计工具成为可能。

面向对象软件框架的发展为面向对象风格架构提供了一套丰富的开发环境，同时也增强

了公众对面向对象开发的热情。面向对象软件框架具有良好的内在组织结构及良好的可用性、可交互性，这些优点使得人们很乐意接受这样的架构，这些都满足了架构的需求。因此，人们不再一味追求通用架构描述语言，转而考虑如何更好地解决特定架构的问题。与此同时，这一架构为基于构件的软件工程提供了一个足够稳固的基础。

从 1997 年发布 UML 1.1 起，在 Rational 的推动下，UML 已经集成了一系列的设计符号并开发了一套系统应用它们的方法，包括用来进行分析、一致性检查，或者将 UML 表达信息与系统代码相互转换的方法。UML 与面向对象技术天然融合并支持模型驱动开发。与 UML 紧密相关的是 Rational 的统一过程(RUP)，这是将 Kruchten 的“4＋1”视图模型进程产业化的工具。UML 与面向对象技术紧密融合、支持架构设计等特性，使得 UML 在支持实践方面具备了其他 ADL 无法比拟的优势，已经成为事实上的产业标准 ADL。

4. 普及阶段：2003～现在

在实用化架构描述语言方面，随着 Rational 公司推出 Rose 软件以及 IBM 并购 Rational 公司后推出 Rational Software Architect(RSA)这一产品，UML 作为产业标准 ADL 的地位得到了进一步增强，也有力地推动了软件体系结构技术的普及。RSA 是一个基于 UML 2.1 的可视化建模和架构设计工具，允许架构设计师和分析师创建系统的不同视图。RSA 构建在 Eclipse 开源框架之上，兼具优秀的可视化建模和模型驱动开发能力。无论是普通的分布式应用还是 Web Services，这个工具都是适用的。需要指出的是作为产业标准 ADL 的 UML，尽管最新版本 UML 2.1 和之前版本相比有所改善，但它仍然缺乏基本的软件体系结构概念，如“层”或“连接器”的准确概念；它也缺乏分析视图之间交互作用的能力。这些都需要借助其他方法和工具加以弥补。这也说明，即便今天，在构建功能足够强大且实用化的 ADL 及其配套方法、工具方面还有很长的路要走。

架构模式方面，随着 WWW 和基于 Web 的电子商务等应用的爆炸性增长，且正在引领商业化的浪潮。多层客户-服务器架构、基于代理的架构、面向服务的架构，以及与之相关的接口、描述语言、工具和开发环境及整体实现的组件、层或子系统等架构模式概念得到了广泛应用，并被公众所熟悉。Microsoft 声称其.NET 平台“包括了用来开发并部署一个 Web Service 连接的 IT 架构的所有事物：部署 Web Service 的服务器、创建 Web Service 的开发工具、使用 Web Service 的应用程序，以及一个世界级的包括超过 35000 个微软认证的合作伙伴机构的网络来提供任何需要的帮助”。互连的服务、工具、应用程序、平台以及厂商的团队，都是围绕着架构进行构建的。

目前，大学里，软件体系结构这门课已成为软件工程类专业从研究生到本科生的必修课，诸如“系统分析与设计”这样的课程也都会有单独章节讲解软件体系结构设计；社会上有关“系统架构师”、“软件架构师”的称呼、岗位、培训、认证考试等已非常普遍；在 ACM/IEEE 的本科生软件工程课程中，有 20%的软件设计单元是关于软件体系结构的；SEBOK 在软件设计章节中将软件架构定义为最主要的部分。

作为领航领域发展的 SEI 自 2000 年以来出版了一系列针对软件体系结构的图书、报告、白皮书，内容涉及软件体系结构设计与开发、质量属性、评估与分析、表达与编档及应用案例等方面。此外，SEI 还结合产业界的需求，推出了许多针对软件体系结构的商业培训课程，大大促进了研究到产业应用的转化。

针对软件体系结构的组织和会议也在持续发展壮大，不仅针对研究团体，还包括用户网络。SEI 架构技术用户网络(SATURN)就是这样一个面向全球企业界、学术界和政府机构人员有关软件、系统和企业架构的组织，每年举办一次年会。

归根结底，软件体系结构作为软件工程学科的一个分支，如何使软件体系结构的研究成果和理念更加具有可实践性是其追求的最终目标。本书以此为目标，以期能为读者进入该领域提供系统知识和设计指导。

1.2　软件架构结构

在医学领域，神经科、血液科以及皮肤科医师对人体的结构有着不同的观察视角。眼科专家、心脏病专家和足病医生研究治疗的是身体的不同部位。运动学家和精神病学家关注的是整个人体行为的不同方面。尽管看待人体的角度不同，但它们都具有内在的相关性，即描述的都是人体的结构。再如，在建筑领域，建造一栋建筑一般会涉及建筑、结构、施工三大类图纸，其中建筑图表述建筑物功能房间布置、平面及竖向交通组织、外观造型、内外装修等，又可细分为建筑平面图、立面图、剖面图等；结构图用来表述建筑物中结构构件的布置、构件材料的选用、构件选型及构件做法等，又可细分为基础图、结构平面布置图、结构构件配筋图等；施工图是施工阶段完成的图纸，主要用途是指导施工。结构图与建筑图两者表达的内容、表达的角度虽各有侧重，但属于一个有机整体，缺一不可。

其实，软件也是如此。由于现代软件系统往往非常复杂，难以一次性对其进行全面的描述。因此，我们在某一时刻往往会将注意力放在系统结构的某一种结构上。为了更有意义地阐述软件架构，往往需要先明确讨论的是哪种或哪些结构，即采用的是架构的哪个视图。

软件结构大体上可分为 3 类，包括模块结构、构件和连接件结构及分配结构，这取决于它们所展示的元素的主要特性。

1.2.1　模块结构

模块结构所体现的是系统如何被构造为一组代码或数据单元的决策。在任何模块结构中，元素都是某种类型的模块(类、层，或仅仅是功能的划分，所有的实现单元)。模块表现了系统的某种静态结构，通常不关注软件在运行时所表现的行为结构。模块结构常关注如下问题：

- 分配给每个模块的主要功能职责是什么？
- 允许模块使用的其他软件元素是什么？
- 它实际使用的其他元素是什么？
- 什么模块通过泛化或继承关系与其他模块相连？

模块结构能够直接传达这些信息。当分配给模块的功能职责发生改变时，也能够通过扩展模块结构来反映其对系统的影响。也就是检测一个系统的模块结构。常用的一些模块结构如下。

(1) 分解结构。这些单元是通过“子模块”关系将彼此关联起来的模块，展示了如何将较大的模块递归地分解为较小的模块，直到它们足够小且很容易理解为止。该结构中的模块常被看作设计起点，因为设计师列举了软件单元必须做什么工作，并把项目划分成模块以进行更详细的设计和最后的实现。与模块相关的通常还有针对模块的接口规范、代码和测试计划等。

通过确认将可能的改变限定在局部范围内，分解结构可在很大程度上决定系统的可修改性。分解结构常被用来作为软件开发项目的组织基础，体现在文档结构、集成和测试计划中。

(2) 使用结构。这是一类重要但却经常被忽略的结构，这一结构中的模块可能是诸如类这样的单元。单元之间通过“使用”关系这种特殊的依赖关系实现彼此关联。如果一个单元的正确性必须以另一个单元正确执行(不是桩)为前提，那么称第一个单元使用第二个单元。使用结构常用于设计需要通过扩展添加功能，或需要方便提取有用功能子集的系统。这种提取系统功能子集的能力便于进行增量式开发。

(3) 分层结构。当以一种特定的方式小心地控制该结构中的使用关系时，就出现了由层组成的系统，在该系统中，一个层就是相关功能的一个一致的集合。在一个严格分层的结构中，第 n 层可能仅使用第 $n-1$ 层提供的服务。然而，在实际情况中这可能会出现多种变体。通常把层设计为将下层的实现细节对上层隐藏起来的抽象(虚拟机)，从而增强了系统的可移植性。

(4) 类或泛化结构。这种结构中的模块称为类，模块之间的关系为继承或是实例化。可以根据该视图推断出类似行为或能力(也就是其他类所继承的类)的集合，以及通过划分子类所引起的参数化的差别。类结构能使我们对重用以及功能的增量式增加进行推断。如果一个项目采用了面向对象分析和设计过程，其文档中给出的结构就是类或泛化结构。

(5) 数据模型。数据模型描述了数据实体的静态信息结构以及它们之间的关系。例如，在银行系统中，所涉及的实体包括账户、用户以及贷款等。账户有一些属性，如账户号码、账户类型(存款或支票)、状态以及当前收支情况。实体间的关系可以表示为：一个用户可能会包含一个或多个账户，而一个账户可能是属于一个或两个用户的。

1.2.2 构件和连接件结构

构件和连接件结构体现了系统如何被设计为一组具有运行时行为(构件)和交互(连接件)的元素。在这些结构中，元素包括运行时构件(一般来说是计算的主要单元，如服务、客户端、服务器、过滤器以及其他运行时元素)和连接件(组件间通信的工具，如调用返回、处理同步操作、管道等)。构件和连接件视图能够回答如下问题：

- 主要执行构件是什么，它们是如何交互的？
- 主要的共享数据存储是什么？
- 系统的哪些部分是复制的？
- 整个系统中数据是如何处理的？
- 系统的哪些部分可以并行运行？
- 在系统执行时，其结构能够发生怎样的变化？

通过扩展，构件和连接件视图能够反映系统的运行时属性，如性能、安全性、可用性等。在构件连接件结构中，上述模块都已经被编译为可执行形式。所有构件和连接件结构与基于模型的结构之间的关系是正交的，它处理的是系统运行时的动态因素。在所有构件和连接件结构中的关系是“连接”，展示了构件和连接件是如何结合在一起的。一些常用的构件和连接件结构如下。

(1) 服务结构。此处的结构单元是服务，服务之间通过一定的协作机制进行交互(如SOAP)。服务结构是一个十分重要的结构，对于一个由独立开发的组件构成的系统有重要意义。

(2) 客户机/服务器结构。构件是客户机和服务器,连接件是协议和消息。

(3) 共享数据或存储结构。构件是作用在数据上的计算单元和数据存储单元。连接件则是提供的数据存取机制。

(4) 并发结构。这种结构使得架构师能够决定系统并行的时机以及可能会发生资源争用的位置。此处的结构单元是组件,连接件是它们之间的通信机制。组件被划分成逻辑线程,一个逻辑线程指的是一个计算序列,它能够在之后的设计过程中被分配到独立的物理线程当中。并发结构早期用在设计过程中,用来定义与并行相关问题的需求。

1.2.3 分配结构

分配结构体现了系统将如何在其环境中与非软件结构关联起来(如CPU、文件系统、网络、部署小组等)。该结构展示了软件元素和创建并执行软件的一个或多个外部环境中的元素之间的关系。分配型结构将回答如下问题:

- 每个软件在什么处理器上执行?
- 在开发、测试和系统构建期间,每个元素都存储在什么文件中?
- 分配给开发小组的软件元素是什么?

分配型结构定义了如何将元素从构件和连接件结构或模块结构映射到非软件的事物上:一般包括硬件、小组以及文件系统等。一些常用的分配型结构如下。

(1) 部署结构。部署结构显示了软件是如何分配到硬件处理以及通信元素上的。元素是软件(从构件-连接件的观点看通常是进程)、硬件实体(处理器)和通信路径。它们的关系是"分配给"和"移植到",前者展示了软件元素所驻留的物理单元,后者的条件是分配是动态的。该视图能够使工程设计人员对性能、数据完整性、可用性和安全性进行推断。在分布式或并行系统中,部署结构显得非常重要。

(2) 实现结构。该结构展示了软件元素(通常是模块)如何映射到系统开发、集成或配置控制环境中的文件结构上。这对于开发活动和构件过程的管理非常关键。

(3) 工作分配结构。该结构将实现和集成模块的责任分配给适当的开发小组。拥有作为架构一部分的工作分配结构,使得由谁来完成这些工作在架构和管理意义上变得很清晰。设计师应该知道对每个小组的技术要求。此外,在大规模多源分布式开发项目上,工作分配结构也是协调处理共性功能并把它们分配给某个小组的手段。

1.3 软件架构视图模型

1.3.1 视图

视图原意是指视觉范围内(看到的)某物的一幅图景,由于对待同一物体观察角度不同看到的内容不同,引申后指从一个特定位置或者角度看事物的方式,为了较完整地描述事物,往往需要给出从不同视角观察得到的多个视图。视图在不同领域具有不同的含义。在软件体系结构领域,一个架构视图是对于从某一视角或某一点上看到的系统所做的简化描述,描述中涵盖了系统的某一特定方面,而省略了与此方面无关的元素。例如,一个安全视图往往关注于安全问题,一个安全视图的模型则仅包含那些与安全相关的视点模型。

1.3.2 视点

视点是系统工程中的一个概念，它用来描述系统中关注点的划分问题。采用视点这一概念，有助于分别表达人们所关注的某些方面的问题。一个好的视点选择会将系统设计划分为不同的技术领域。

视点提供了构建、表达和分析视图的约定、规则和语言。ISO/IEC 42010：2007（IEEE-Std-1471—2000）中规定：视点是一个有关单个视图的规格说明。视图是基于某一视点对整个系统的一种表达。一个视图可由一个或多个架构模型组成。

1.3.3 视点建模

给定一个视点，就可以构建一个系统模型，使它仅包含从这一视点看来可见的对象，但同时也可包含系统中与这一视点相关的所有对象、关系和约束。这样的模型被看作视点模型或从这一视点看来系统的视图。

一个给定的视图可看作基于特定视点有关系统一个特定抽象层次的表达。不同抽象层次包含不同细节层次。高层视图使得工程技术人员可在一个较大的层面理解系统设计，识别和解决问题。低层视图则允许工程技术人员集中于设计和开发的细节层面。

就系统本身而言，出现在不同视点模型中的说明最终都要在实现构件给出实现细节。关于某些构件的说明有可能来自不同视点。另外，由于特定构件的功能分布和构件之间交互而引起的说明，又会折射出不同于原视点的关注点划分。因此，创建新的视点，弄清楚单个构件的关注点并进行系统自底向上综合或许都是必要的。

1.3.4 架构模型

无论是房屋、飞机、汽车等物理模型，还是数学模型、软件模型，其实质都是一种抽象，即被构建的真实事物的近似代表。一个软件架构模型就是一张基于某一标准创建含义丰富且严谨的图及其文字描述，其主要用来说明一个系统或生态系统结构和设计方面的折中方案。但如果一个图不是架构意义上的图则不应被看作架构模型。软件架构师使用架构模型来和其他涉众交流。一个架构模型可看作软件架构中视点的一种表达。例如，在软件设计当中所使用的UML图，如果其是架构意义上的图（如构件图、部署图），就可称为架构模型。

1.3.5 视图模型

在系统工程和软件工程领域，一个视图是关于整个系统某一方面的表达，一个视图模型则是指一组用来构建系统或软件架构的相关视图的集合，这样一组从不同视角表达系统的视图组合在一起构成对系统比较完整的表达。

Kruchten于1995年提出的“4+1”视图模型是最典型的多视图模型，包括逻辑视图、开发视图、进程视图、物理视图和场景，用来从上述5个视点描述软件密集型系统的架构。IBM公司Zachman于1987年提出的Zachman框架的实质是一种企业架构的视图模型，该模型关注开发过程中“什么”“怎样”“谁”“哪里”“什么时间”等问题，相应的框架中每一类视图表达某一