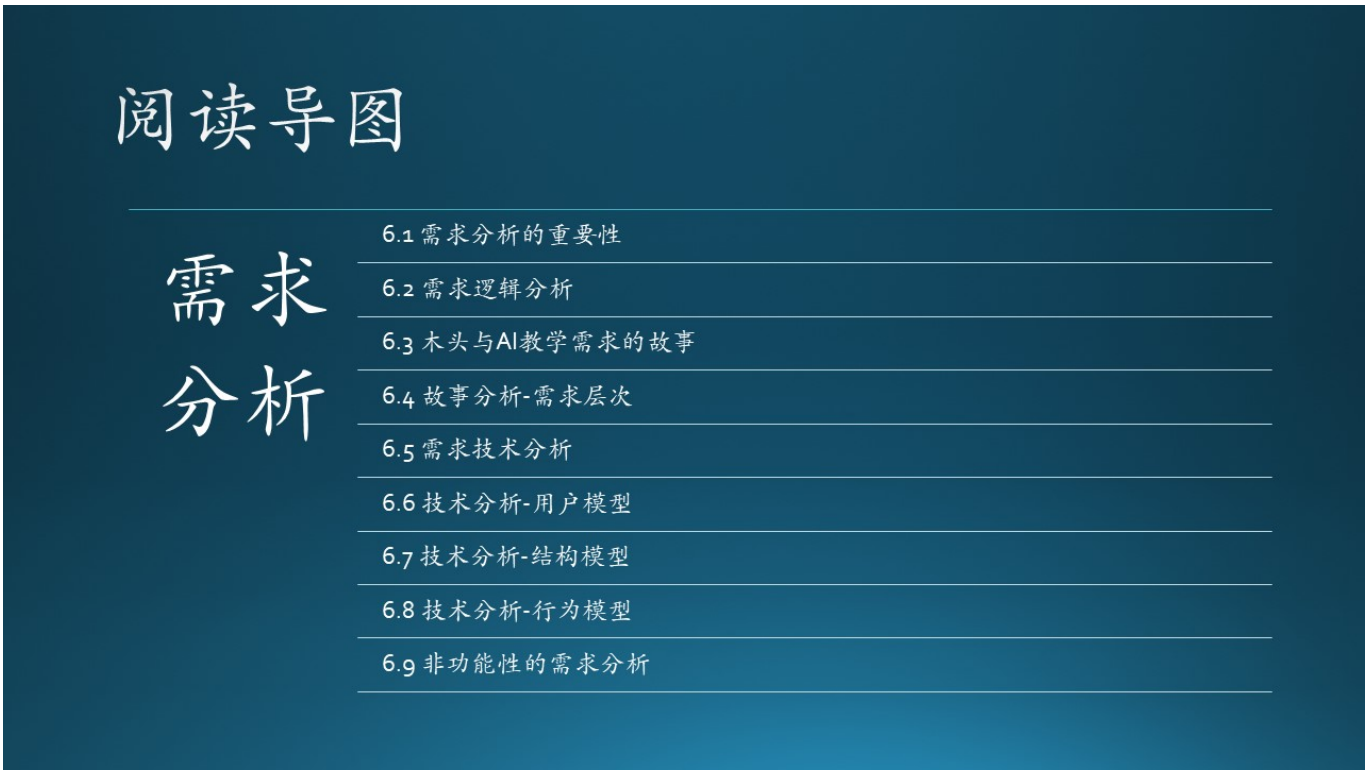
（插入一些说明性文字，重点在介绍上下文，串连）



参考资料

- [1] 《软件工程》，清华大学出版社
- [2] 周金根 《软件需求的三个层次》http://www.zhoujingen.cn/itbang/352.html
- [3] 李鸿君 大话软件工程《需求分析与软件设计》清华大学出版社
- [4] Difference Between Aggregation and Composition, https://techdifferences.com/difference-between-aggregation-and-composition.html
- [5] 杨长春 《实战需求分析》 清华大学出版社

6.1 需求分析的重要性

需求分析是软件生命周期中相当重要的一个阶段。

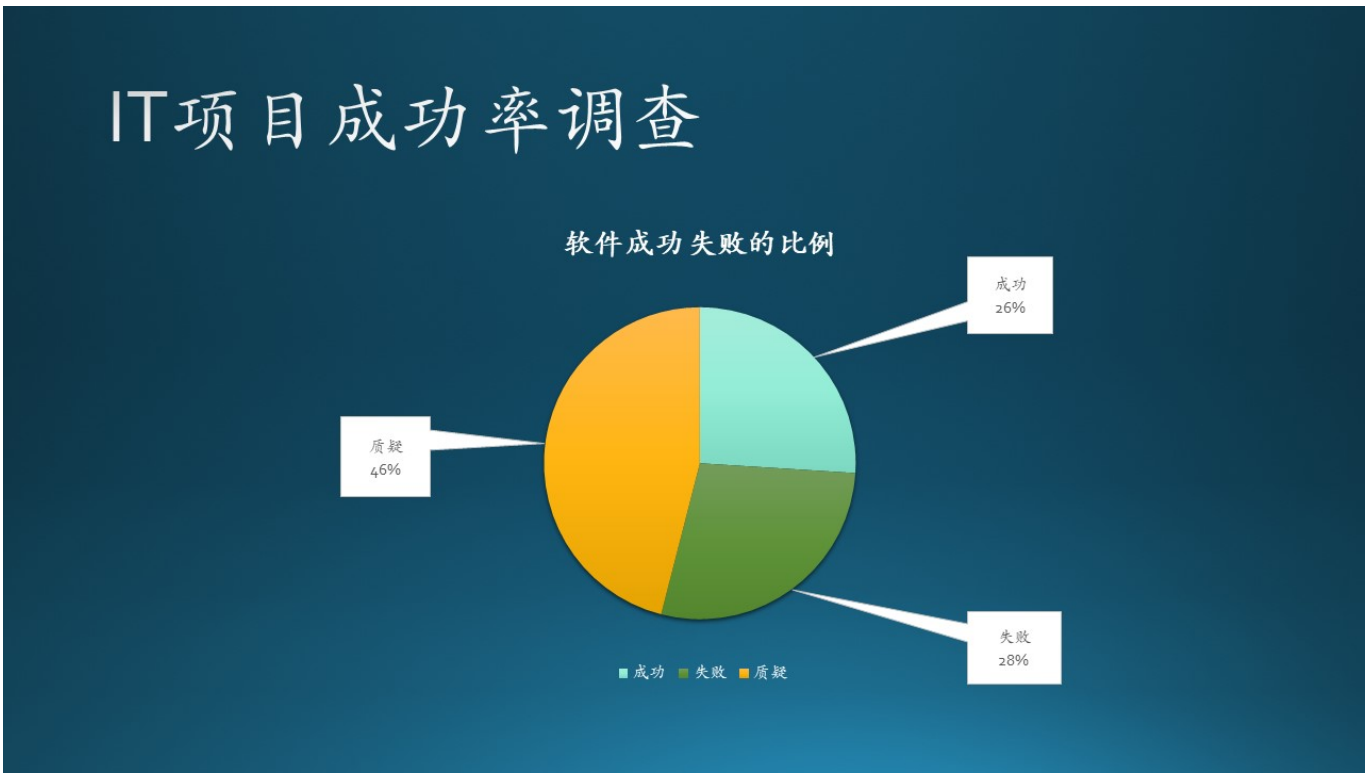


图 6.1.1 - IT 项目成功率调查

1999年，美国专门从事跟踪 IT 项目成功或失败的权威机构 Standish Group，对23000个项目进行的研究结果表明，28%的项目彻底失败，46%的项目超出经费预算或者超出工期，只有约26%的项目获得成功。

而在这些高达74%（28% + 46% ）的不算成功项目中，有约60%的失败是源于需求问题，也就是差不多有一半的项目都遇到了这个问题，这一可怕的现象引起人们对需求分析的高度重视。

需求分析阶段的主要任务是通过需求分析人员与用户之间的广泛交流，不断澄清一些模糊的概念，最终形成一个完整的、清晰的、一致的需求说明。

6.1.1 国外故事：秋千图的例子

我们先请出软件业界著名的秋千图，如图 6.1.2，表 6.1.1 中有解释。

图 6.1.2 - 秋千图

表 6.1.1 - 秋千图的解释

序号	标题	解读
1	客户的描述	挂在树上供三个孩子一起玩耍的秋千
2	项目经理的理解	两根绳子拴在树上，挂一个木板
3	分析员的设计	修改支撑系统的结构以便让秋千能晃动
4	程序员的实现	实现了静态拓扑结构设计：绳、树、木板
5	商业顾问的描述	舒适无比、光芒四射
6	项目文档	空空如也，如云如烟
7	上线的产品	缺东少西，不知所云
8	客户投资	巨大得可以修建一个游乐场了
9	技术支持	少之又少，无依无靠
10	用户的真实需求	树杈+绳子+轮胎，可以供该客户的三个孩子玩耍

需求调研与分析是一切软件的起源，差之毫厘，谬之千里。在秋千图的例子中，各个环节的误差导致了彻底的失败。

1. 首先，客户的描述是“我的三个孩子可以一起玩耍”，根据客户的描述，这个秋千必须可以同时容纳三人。显然这个描述是不准确的：“一起”的含义并不是三个孩子同时在秋千上，有可能是一个孩子在秋千上，两个孩子在地上推秋千。

2. 项目经理没见过三层的秋千，但是他觉得只要秋千的踏板足够宽，可以站好几个孩子。另外，如果想保证安全的话，两根绳子最好能拴在两个树杈上，这样可以分担承重。

3. 分析师认为分担承重的需求很重要，但是如果秋千需要前后晃动，必须有空间，那就不得不把树锯开。当然左右各加一个支撑是必须的，这样系统可以保持稳定。

4. 程序员的实现完全符合秋千的静态拓扑结构：两根绳子拴住树干上，下面挂一个踏板。至于秋千的动态模型可以后期再考虑。

5. 产品快成型了，市场人员坐不住了，开始到处宣传：我们的产品非常的安全和舒适，孩子们就好像是坐在沙发里一样，家长不必有任何担心。

6. 没有人写文档，需求、设计口口相传，导致各个环节都出现误差。

7. 工程人员自己家里也有简易秋千，一根绳子就可以搞定。所以在安装产品时忘记带踏板了，那就索性只挂一个绳索，这样孩子们依旧可以愉快地攀爬。

8. 客户抱怨，为此项目一再增加投入，最后发现累计投入可以建造一座游乐场了。

9. 后期的技术支持认为：客户的抱怨是有道理的，我们应该把树砍掉，让客户建造游乐场。

10. 其实客户的真实需求只是一根绳索下面挂一个旧轮胎。

6.1.2 国产故事：烟囱和井的例子

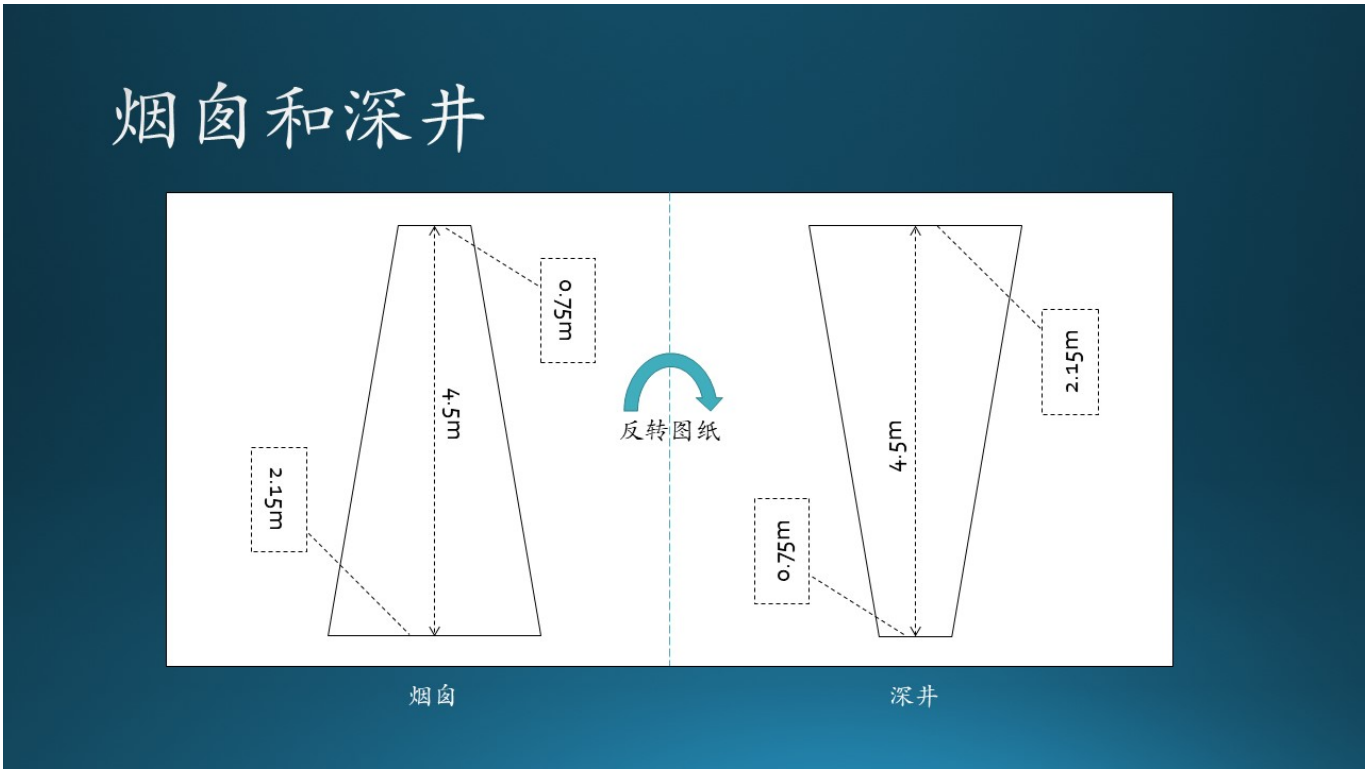


图 6.1.3 - 郭师傅家的井

在郭德纲的相声《梦中婚》里，客户郭师傅需要挖一口井，但是工程人员把图纸拿反了，最后建了一个烟囱，一边干一边还纳闷儿：这个烟囱为什么设计的这么粗？

其实，有很多机会可以避免这个错误，而且可以轻易觉察。但是如果对应到软件开发上，就没有那么容易了，这要求在软件开发中有严格的流程管理和及时（通过scrum meeting）沟通。

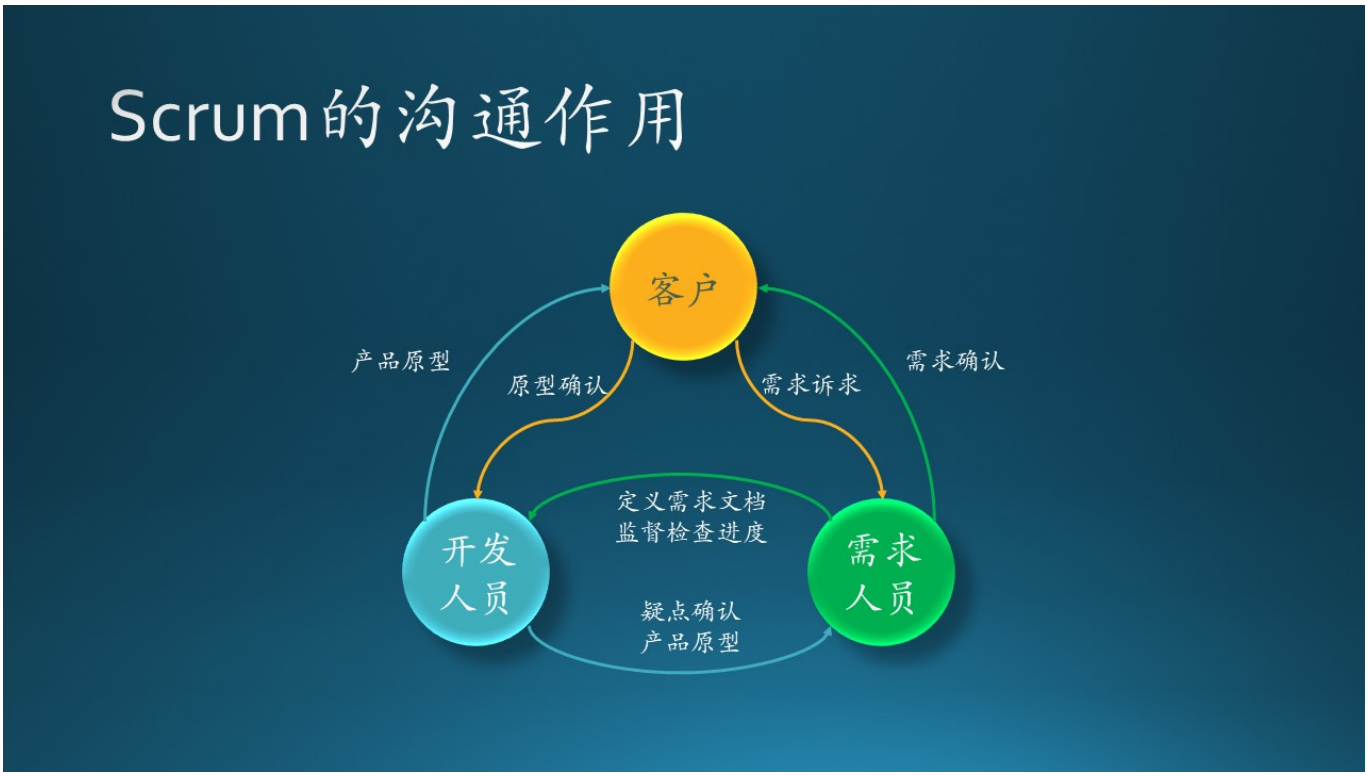


图 6.1.4 - Scrum 的沟通作用

1. 包工头作为中间环节没有给上下游交代清楚。包工头在这里可以对应到需求分析人员，他如果拿着图纸（需求规格说明书）和客户确认一下需求，或者直接告诉施工方（开发人员）客户是要挖一口井，也不会出现这样的错误。

2. 文档不严格，不完整。如果在图的上方注明《郭师傅家的深井挖建工程图》，除非工人是文盲，否则也不会拿反图纸。

3. 施工人员拿到图纸后，针对对图中的疑点，及时和包工头沟通。开发人员看到需求规格说明书后，可能有他自己的理解（和秋千图的例子一样），针对任何疑点，都要及时与需求分析人员沟通。

4. 施工人员及时与客户现场沟通。在软件开发中，客户通常是看不到软件的样子的，这和烟囱的例子不可类比。所以，更要求开发人员及时做好产品原型（prototype）交给需求分析人员（包工头）和客户（郭师傅）进行确认。

5. 包工头经常来现场检查工程进展。需求分析人员在完成文档后，并非就完成工作了，而是要阶段性地和开发人员进行交流，监督、检查进度、质量以及需求满足情况。

6.1.3 需求分析的特点及难点

需求分析的特点及难点，主要体现在图 6.1.5 所示的几个方面。$^{[1]}$

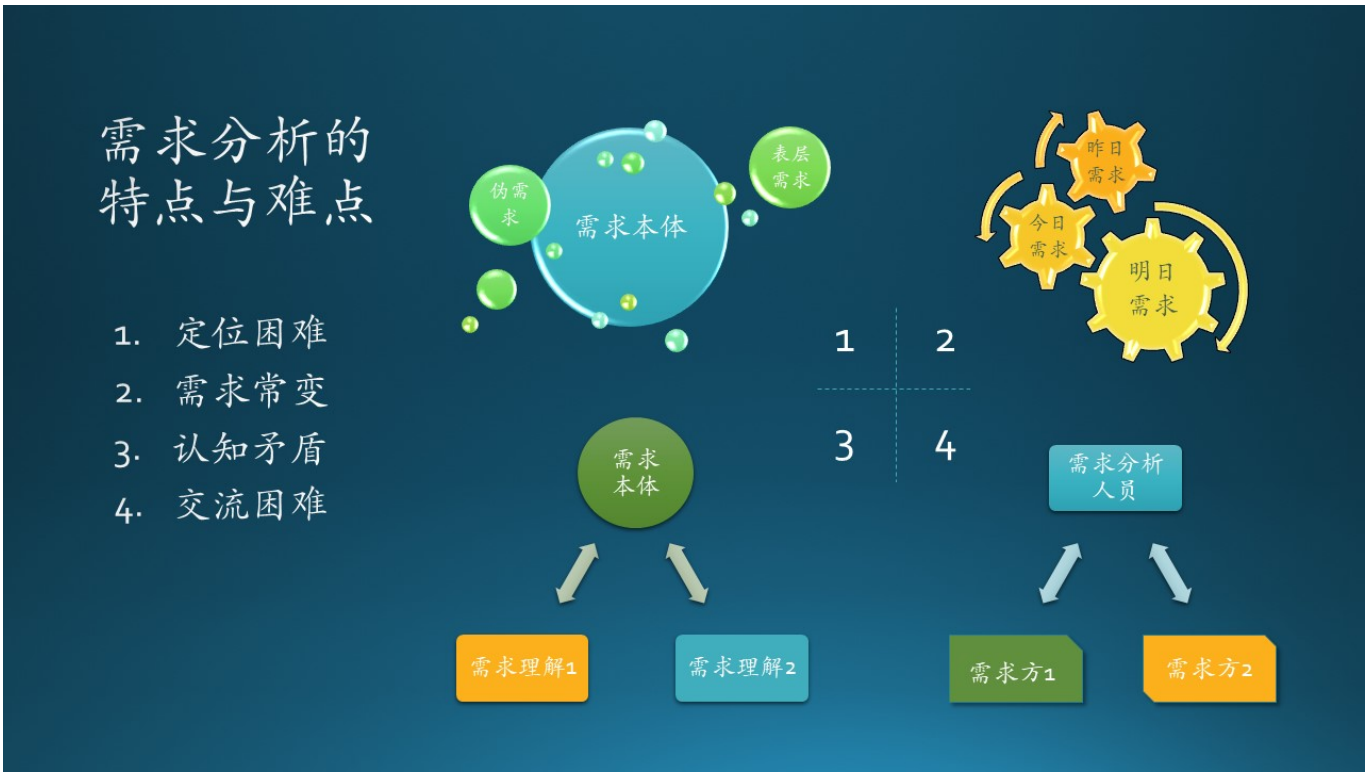


图 6.1.5 - 需求分析的特点与难点

1. 定位困难

   主要原因一是应用领域的复杂性及业务变化，难以具体确定；二是用户需求所涉及的多因素引起的，比如运行环境和系统功能、性能、可靠性和接口等。

2. 需求常变

   软件的需求在整个软件生存周期，常会随着时间和业务而有所变化，如客户环境和业务流程的改变，市场趋势的变化等，也会随着分析、设计和实现而不断深入完善，可能在最后重新修订软件需求。

3. 交流困难

   需求分析涉及的人事物及相关因素多，与用户、业务专家、需求工程师和项目管理员等进行交流时，不同的背景知识、角色和角度等，使交流共识较难。

4. 认知矛盾

   由于不同人员对系统的要求认识不尽相同，所以对问题的表述不够准确，各方面的需求还可能存在着矛盾。难以消除矛盾，形成完备和一致的定义。

6.2 需求逻辑分析

需求逻辑分析，就是通过记录用户诉求（文字的或语言的），仔细领会用户的真实意图和工作流程。具体的方法可以有以下几种：

图 6.2.1 - 需求逻辑分析

- 小组讨论，从语义上搞清需求；
- 卡片分类，整理功能需求集合；
- 单据分析，业务流程逻辑需求；
- 报表分析，数据汇总统计需求。

6.2.1 小组讨论

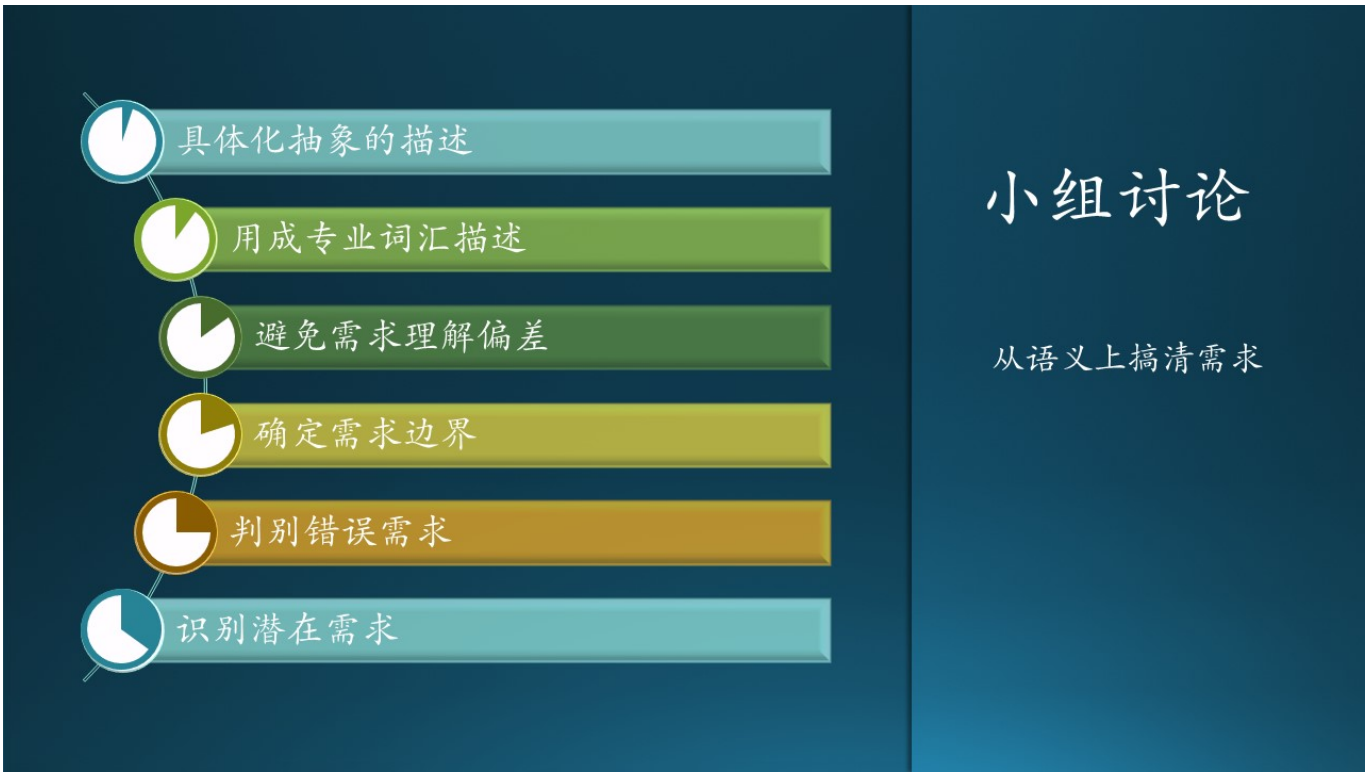


图 6.2.2 - 需求分析之小组讨论

项目小组成员通过任何形式获得用户需求，回来后自己内部开会，对需求进行讨论，目的如下：

- 把抽象的需求具体化
    - 用户：“这么多论文乱七八糟的，根本没法找！”
    - 毛毛：“我觉得用户并不喜欢读很多论文。”
    - 木头：“需要论文分类管理功能，可能是根据类别、会议或者日期。”

  解释：用户的需求有时是用情绪化的语言形式表达出来的。

- 用计算机专业词汇描述需求
    - 用户：“这个论文很有价值，以后还要再看。”
    - 毛毛：“给论文加一个类似豆瓣评分的级别。”
    - 木头：“按日期保存论文到系统指定的存储位置，并可以根据日期和名称检索。”

  解释：需求文档要明确指出存储、检索的功能。

- 避免需求理解偏差
    - 用户：“这个游戏软件主要是面对儿童的，比如我，希望家里的三个分别为7岁、10岁、12岁的孩子都可以玩。”
    - 毛毛：“用户希望如果一个家庭有三个孩子的话，可以同时玩这个游戏。”
    - 木头：“别闹，用户是希望各个年龄段的孩子都可以玩。”

  解释：用户的举例通常带有实例化性质，需求文档需要泛化这些需求。比如该用户家的三个孩子恰巧都是男孩儿，或者该用户家的孩子的身高比同龄人的平均身高要高，这些都需要广泛调查后再确定目标用户。其实在著名的秋千图的例子里，用户的原始想法是用一个橡胶轮胎挂在树上做秋千，这样他的三个孩子都可以玩儿。但是需求分析人员把"三个孩子都可以玩儿"理解为"三个孩子可以同时玩儿"，于是设计了一个三层的木板结构。

- 确定需求边界
    - 用户：“能不能加两个人手，在这个月就把那个云端同步的功能也一起做了？”
    - 毛毛：“云端同步的功能，只是把客户端的注释标记上传到云端保存而已，好像没有很大的工作量。”
    - 木头：“对不起，我们本期的需求已经确定了‘论文注释’功能只在客户端保存，开发计划已经排满了。况且云端同步功能还需要购买 Azure 资源，这部分也不在预算内。”

  解释：用户总希望少花钱多办事，而市场人员往往为了订单随口承诺一些事情，需求文档需要明确这些边界。做计划外的功能，会影响计划内的软件的质量，和一些不可预估的如预算、设备、部署等方面的麻烦。

- 判别错误需求
    - 用户：“这个论文最好还能保存成 docx 格式，便于修改。”
    - 毛毛：“保存成 docx 功能还能用 Word 打开阅读，很方便。”
    - 木头：“你……我……人家发表的论文你改什么改？”

  解释：对用户的一些需求，需要分析它的背后的含义，再确定其正确性。

- 识别潜在需求

- 用户：“我批改了这篇论文，我想把我的意见通过电子邮件转达给其它人。”
- 毛毛：“电子邮件是无纸办公的重要手段，倒是可以考虑增加这个功能。”
- 木头：“用户实际上是想通过网络的方式把自己的意见快速准确地转达给其它人，那不如我们后期考虑在 Azure 上搭建一个服务器吧，这样大家都可以分享自己的见解。”

解释：就如同亨利福特遇到的情况，用户希望有一匹快马，但其实用户需要的只是“快”，而不是“马”，“快”是真正的需求，“马”只是载体，所以福特开始造新的载体——汽车。理解用户的真实意图，用自己的专业知识提供解决方案。

6.2.2 卡片分类

用户在述说需求时，一般是杂乱无章的，整理成体系会有些困难。此时，无论用何种需求调研方法，我们可以先把从用户那里听到的每个需求，都记录在一张小卡片上面。积攒了一定程度后，把这些小卡片根据内容或者来源进行分类，就形成了一些模块或子系统的雏形。

微软内部有一种做法类似于卡片分类，叫做 Backlog，就是使用软件把一些想法记录下来，在下一阶段的需求分析开始时，把这些想法拿出来分类整理，大家评估一下需求属性，再排一下优先级，成为真正的功能写到文档中等待开发。

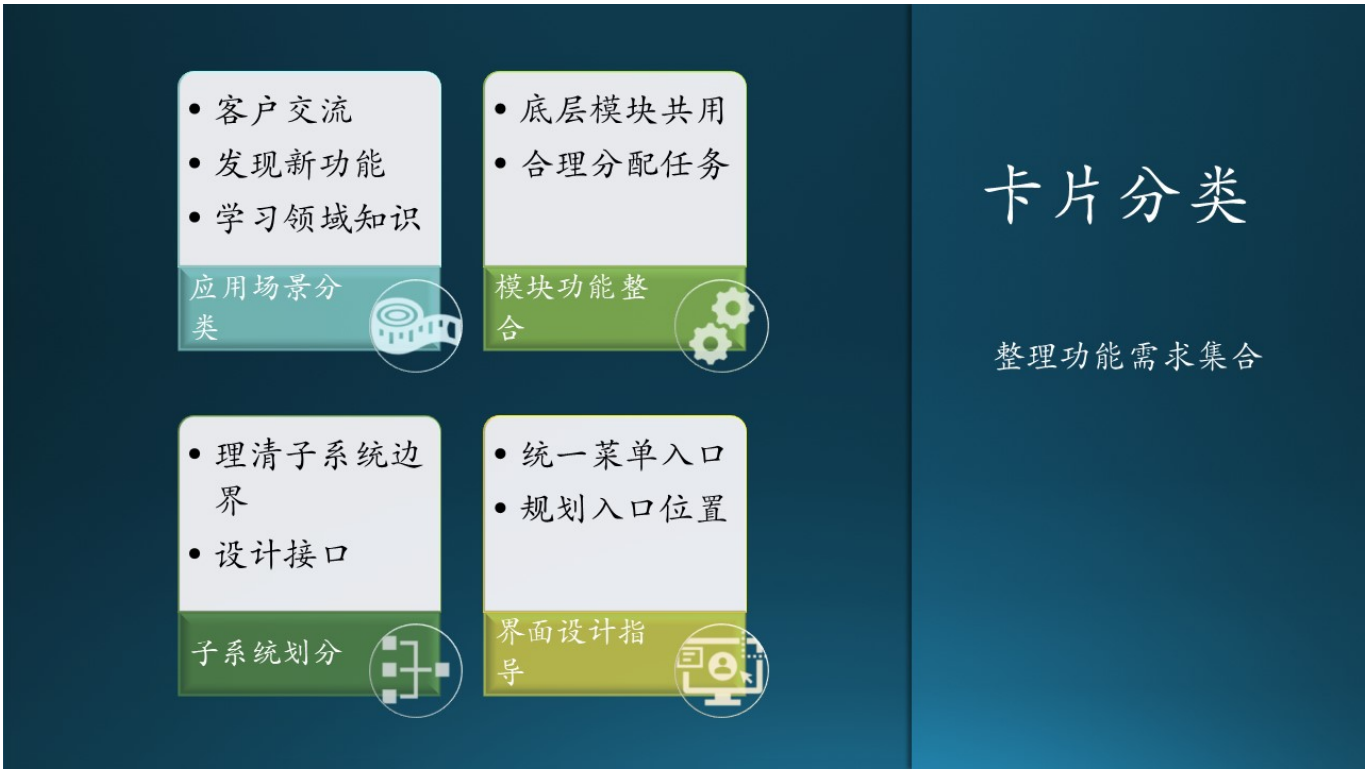


图 6.2.3 - 需求分析之卡片分类

木头在做 Edge Mobile Browser 的需求分析时，使用了卡片分类法来归纳总结一些浏览器上的功能的分类，这样做的好处是：

- 应用场景分类
    - 从客户角度可以清楚这个软件可以提供什么类型的功能，而不是看到乱糟糟堆砌的一堆功能，这可以帮助客户建立对这个软件的质量方面的信心。
    - 需求分析人员可以从已有分类中开拓思路，设计出一个惊喜型的功能。

分类设定，其实对人类的思维是一个双刃剑。木头的感觉是，很容易陷入已定类别中不能或者不敢突破。另一方面，也可以帮助需求分析人员仔细研究每个类别里的功能，找出缺失的部分，或者找出缺失的类别，来开发出惊喜型功能。

- 开发人员可以快速学习领域知识，理解真正的需求。

  木头自己就通过这次分类整理，真正地认识到了浏览器功能的全貌，一旦有了“上帝视角”，会不由自主地感叹到：“原来如此！那么浏览器那个组的几百号儿人，这些年都做了什么，为什么 Edge 浏览器口碑越来越差？”。话音未落，浏览器组先是换了一个大老板，紧接着决定用 Chromium 做引擎，重新写 Edge 浏览器的 Shell，而这个大老板，就是以前在 Windows Team 负责 Shell 的，这说明了一个问题 —— 上层已经决定用 Chromium 做引擎，让一个对 Shell 很有经验的人去领导 Edge 项目的开发，在 Chromium 的基础上做一个微软风格的 Shell，成为新的 Edge 浏览器。

- 模块功能整合

  - 相近的功能可以共用底层模块，减少代码重复。

    比如浏览器里常用的两个功能：网页收藏，浏览历史。从外观上看，很明显都是一个列表，里面存放一些网页记录。最起码，那个列表控件就是可以重用的部分。

  - 合理地分配开发资源，让一个人或小组负责一类功能。

    木头在 Edge Mobile Browser 项目中，担任的是质量管理任务，亲眼目睹了当时的 Dev Manager 是如何分配工作的：先给甲乙丙三个开发人员各分配一个任务模块 A、B、C，假设甲先做完了，就分配另外一个任务模块D，而 D 和 A 是两个不相关的模块。

    上面只假设了三个人，实际上当时有10几个人，每个人都是在功能类别之间跳来跳去，需要熟悉每一个功能模块的底层代码，造成资源上的浪费。

- 子系统划分

  相互之间有较强关系的模块聚合在一起，可以形成子系统。子系统划分其实不是设计阶段才开始的任务，而是在需求阶段就已经形成雏形了。

  用卡片分类法，在视觉上就可以直接形成功能聚类，理清子系统的边界，便于后面做接口设计。

- 界面设计指导

  - 同一类功能放在一个菜单入口里，用户容易找到。

    在京东、淘宝等软件里，作为软件从业人员的木头，可以明显感觉到界面菜单是经过精心设计的。但是在使用一些银行的手机软件时，就会觉得功能不容易找到，缺乏设计。

  - 不同使用率的菜单入口在屏幕上的位置不一样，体现了重要程度。

    尤其是在手机上，屏幕小，寸土寸金，所以按钮的位置需要非常精细的设计：

    - 是否需要在首页上显示
    - 如果需要显示，放在什么位置，方便用户单手操作
    - 按钮的尺寸多大，才能即明显，又不影响美观，又能很容易地点击

6.2.3 单据分析法

传统行业中，往往存在一些手工填写的单据或报表，可以给流程控制提供很好的抽象信息，比如：银行的存款单、取款单，仓库的入库单、出库单，海关的报关单、提货单，企业的采购单、验货单，财务的资产负债表、损益表等等。

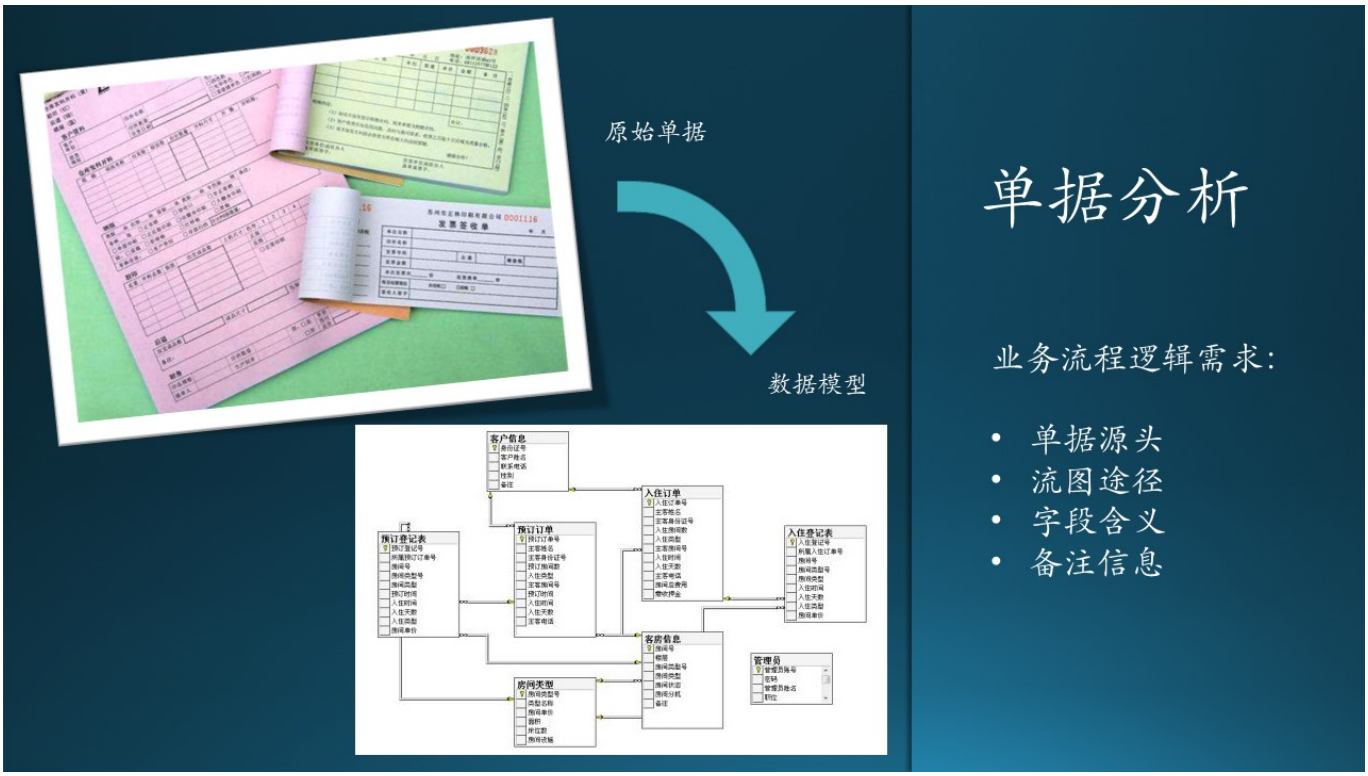


图 6.2.4 - 需求分析之单据分析

单据分析法，分析用户当前使用的纸质或电子单据，通过研究这些单据所承载的信息，分析其产生、流动的方式，从而熟悉业务，挖掘需求。一个组织，在没有信息化管理系统时，它的单据体系其实就是它的信息体系，填写单据的过程就是信息录入的过程，单据传递的过程就是信息流转的过程，最终单据进入的档案室就是数据库。因此，通过分析单据来获得关于信息管理的需求可以收到事半功倍之效。单据分析法是获取需求过程中使用得相当普遍的方法，值得仔细研究。$^{[5]}$

- 理清业务逻辑

  想读懂一张单据，需要搞清下面四个问题：

  - 单据源头
  - 流动途径
  - 字段含义
  - 备注信息

  分析好现在使用的这些报表，可以深入到管理者的管理神经，弄清楚当前公司管理者感兴趣的信息，最终给各级管理者带来真正的价值。报表是一个信息系统的集大成者，提前做好报表分析，可以加深理解管理脉络，理解信息系统的最终需求。

- 数据设计的依据

  这些单据其实就是数据流图中的数据/文件部分，也可以做为数据设计的依据，但是其中肯定有很多冗余的项，需要用范式知识来解决。范式知识我们将在后面介绍。

表 6.2.1 - 入库单示例

品名	型号	单位	数量	生产日期	批号	入库日期	备注
鼠标	LG302	个	1000	2020/09/01	12	2020/09/22	限量版
键盘	C104	个	200	2020/09/02	9	2020/09/23	京东专供

如表 6.2.1，作为一张纸质的原始表单来说，已经比较完善了。但是作为软件用的表单，有两个地方要注意：

1. 还缺一个唯一的入库单号做主键，否则无法索引。
2. 另外就是型号字段是个外键，需要设计另外一张表来描述 LG302 和 C104 的具体特性，如重量、体积、颜色、无线/有线、是否人体工程学设计等等。

6.2.4 报表分析法

报表分析法，通过分析用户使用的报表获取需求。报表跟单据是有本质区别的。单据是在业务处理过程中用户填写的纸质文件，往往是一个信息采集、传递的过程，而报表则是根据一定的规则对批量数据进行检索、统计、汇总，是一个信息加工、分析的过程。

图 6.2.5 - 需求分析之报表分析

生成报表的数据来源，是需求分析的重要手段，这些数据必须在需求链前端存在，不能到最后环节凭空产生。要做到“报表中需要展示什么，就去收集什么数据”，而不是“已经有什么数据，就在报表中展示什么”。

分析好现在使用的这些报表，可以深入到管理者的管理神经，弄清楚当前公司管理者感兴趣的信息，最终给各级管理者带来真正的价值。报表是一个信息系统的集大成者，提前做好报表分析，可以加深理解管理脉络，理解信息系统的最终需求。$^{[5]}$

对一些报表类的需求，软件除了可以做到比纸质报表更精美的格式、字体外，需求分析人员应该设计出对应的可视化数据图版（Dashboard），让数据展示更生动。

6.3 木头与 AI 教学需求的故事

这一天，木头和几个微软的同事来到一所高校调研，受到了校长、老师和同学的热情接待。在一个可以容纳 10 几个人的会议室里，大家围坐在桌前，一边品着刚刚下来的毛尖茶，一边聊着 AI 教育。

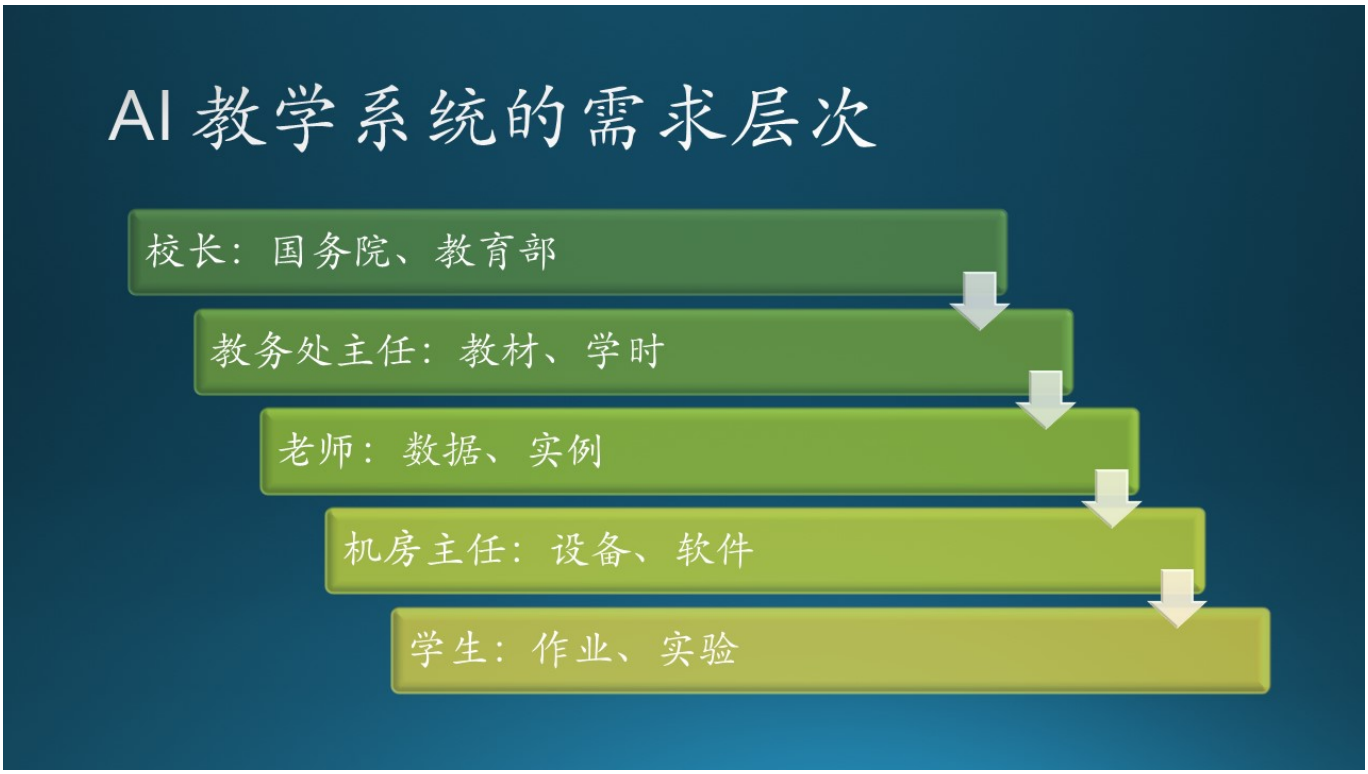


图 6.3.1 - AI 教学系统的需求层次

6.3.1 校长和教务处主任的发言

首先当然是请校长发言。

“首先感谢微软的相关同志来我校做调研，我们这边今天参会的有分管校长、教务处刘主任、机房赵主任、老师代表和学生代表。下面我先说说大形式（校长端起杯子喝了一口茶水）。

“国务院2017年印发了《新一代人工智能 发展规划》，标志着人工智能作为产业变革的核心驱动力和引领未来的战略技术，已上升为国家战略……构建包含智能学习、交互式学习的新型教育体系。

“对于AI教育企业而言，去年的政策环境足够振奋人心。教育部发布的《高等学校人工智能创新行动计划》提出……人工智能+X的复合特色专业，解决AI的人才问题。

“近日，教育部网站发布了《教育部2019年工作要点》，提出……AI+教育和互联网教育注入一剂强心针。

……（省略1000字）

“微软公司一直在软件领域执牛耳地位，而且也一直支持高校的教育，所以我们想听听微软在这方面有什么想法。”

木头：“感谢您的信任！支持高校的教育一直是我们的宗旨。刚才校长先生的讲话已经给我们指明了方向，这次来其实还是想听听各位一线老师目前在 AI 教育上有什么需求和痛点。”

教务处主任：“那我先说说吧！现在市面上的教材很多，国内的国外的都有，最知名的就是周志华老师的“西瓜书”《机器学习》，李航老师的《统计学习方法》，还有一本国外翻译过来的“花书”《深度学习》。但是我总觉

得这些书做为教材的话，难度有点儿大，老师首先要自己学习，做课件，然后再讲给学生听。但学生们往往反映听不懂。”

6.3.2 老师的发言

木头：“那为什么学生听不懂呢？”

张老师：“首先是这些书本身的内容难度偏大，教软件课的老师们也是刚刚接触这方面的知识，自己理解起来还需要时间，讲课时也做不到融会贯通。”

王老师：“学时比较紧张，一般的32课时的安排，只能紧巴巴地讲完，赶进度，所以也讲不细。”

李老师：“还有一个问题，这些教材上大多没有例子，老师们也想不出合适的例子，找不到合适的数据集，也没有合适的实验环境，这样干巴巴地讲公式，学生当然听不懂了。”

木头：“对对对，实验环境非常重要，学生动手学习的话，比了解书本知识更容易。”

教务处刘主任：“我们目前还没有可以做数据科学实验的环境，听说建设这么一套环境需要不少资金和技术支持。校长，您看看能不能批一些费用？”

校长：“这个没问题！只要大家有可行方案，我这里全力支持！”

教务处刘主任：“好好好！那么微软方面在实验环境建设方面有什么好的建议吗？”

木头：“我们有成熟的解决方案，服务器端架设在 Azure 上，但是需要校方购买一些必须的客户端设备，比如可以支持手写的电子显示屏，建立到 Azure 的互联网连接，校园网上可以部署专用的实验和作业系统，在机房的客户端机器上安装 Python 等必要的第三方环境。”

机房赵主任：“这些问题都可以解决，只是现在机房的计算机的操作系统还是 Windows XP，有些陈旧了。另外，我们也没有 GPU，听说非常昂贵？”

木头：“如果是 Windows 10 操作系统是最好的。关于 GPU，如果初期资金紧张或者不想维护这么昂贵的设备，可以使用 Azure 上的科学技术虚拟机，相当云租赁 GPU，按使用时长付费，平时不用就关掉。”

张老师：“学习 AI 看起来必须要 GPU 了？那么贵的话，普通人怎么学得起？门槛太高了！”

木头：“哦，这个忘记说了，当前流行的深度学习要想到达 SOTA（state of the art，最先进的）水平，确实需求 GPU。但是，学习传统的机器学习和入门级别的深度学习，完全不需要 GPU。”

教务处刘主任：“那太好了！有没有相关的教材或案例可以供老师参考的？”

木头：“我们最近出了一本《智能之门》的书，由高等教育出版社出版，又名‘九步学习法’，意思是一共用9步就可以入门深度学习，里面有通俗的文字、详尽的公式推导、完整的配套代码、有代表性的实验数据、可解释的实验结果，普通人用 CPU 计算机就可以学习。”

张老师：“如果有配套的代码的话，学生们可以上机动手亲自体验，这样会大大降低学习的门槛。”

王老师：“对！CPU 就能做简单实验的话，同样会降低门槛。”

李老师：“配套的 PPT 应该有的吧？赵主任，咱们机房有没有可以手写的显示大屏，这样可以一侧放 PPT 课件，另一侧写板书，一方面不用吸粉笔末了，另一方面板书可以以图片方式课后发给学生们复习，学生不用自己记笔记了。”

赵主任："手写显示屏已经下单了微软的Surface Hub 2，正在安装调试。Windows 10 操作系统，天然支持访问 Azure 以及 Office 办公软件。"

6.3.3 学生的发言

毛毛同学："那太好了！同学们听课时忙着记笔记，有时候就会漏掉一些重点讲解。我还有个问题，这个 AI 教育系统能留作业和判作业吗？"

木头："好问题！由于 AI 课程的特殊性，留作业和判作业，打分、排名，是这个系统的一大特色。"

毛毛同学："请问老师，这个系统能够平时使用吗？因为我们想课余时间训练一些模型练练手，比如练习调参、熟悉多种神经网络模型等等。"

木头："这个要看咱们学校对 GPU 的使用计划了，如果自己买一块 GPU 的话，当然使用越多越好；如果是租用 Azure 上的 GPU 的话，就是不一样的策略了，比如定时开放使用权限等等。"

6.3.4 外部系统

教务处刘主任："我们学校吧，目前已经建设了不少管理系统了，比如说这个学生管理系统，这个教师管理系统，还有这个课程管理系统。赵主任，您那里是不是还有别的？"

机房赵主任："是的，我这里还负责机房管理系统。咱们这个 AI 教学系统和这些已有的系统是什么关系呢？"

木头："现有的这些系统对 AI 教学系统来说，都非常有帮助，这样我们就不用从头儿开始建设这些基础数据库了。比如学生想使用 AI 教学系统，首先要登录，这个用户名和密码呢，就依靠学生管理系统来验证了，验证通过后，给 AI 教学系统一个 Token（令牌），学生就可以开始使用教学系统了，不需要二次登录。"

……

大家还聊了很多，木头把后来的一些细节放到了"木头与神经网络课程的故事"中，为老师和同学们描述了一下 AI 教学系统的真实体验。

6.4 故事分析-需求层次

从需求讲述人来看，不同人所处的位置不同，从而给出不同层次的需求，按照业界惯例，我们一般定义三个层次，如图 6.4.1 所示，即：

- 业务需求
- 用户需求
- 功能需求

图 6.4.1 - 需求层次分析

下面的故事虽然是面向学校教育系统软件的需求分析，而面向企业的软件需求层次也是大同小异，同样可以分为三个层次。但是面向大众的软件，比如我们常用的办公软件、手机软件，就不符合这三个层次特征，我们将在第七章讲述。

6.4.1 业务需求（Business Requirement）

我们先来看看校长和教务处主任的需求。

- 校长主要是讲政策方面的内容，以及国家对学校、企业的期望，是一个宏观的大需求；
- 教务处主任从教材入手，反映了目前高校存在的 AI 教育的普遍问题，即难度大，学生不易听懂。所以，主任的需求很简明扼要：提供一门让学生能够听懂的AI课程。

我们把这种需求叫做业务需求（因为需求来自业务管理者）。

图 6.4.2 - 第一层次：业务需求

- 来源：投资人、负责人、管理者等，非直接用户。
- 内容：说明需求的背景原因、系统目标、战略理念。
- 作用：指导系统构建的根本基础。
- 输出：通过对这一层次需求的调研，应该画出“上下文图”作为输出。

表示组织或客户高层次的目标。业务需求通常来自项目投资人、购买产品的客户、实际用户的管理者、市场营销部门或产品策划部门。业务需求描述了组织为什么要开发一个系统，即组织希望达到的目标。

业务需求一般是使用前景和范围（vision and scope）文档来记录业务需求，这份文档有时也被称作项目轮廓图或市场需求（project charter 或 market requirement）文档，但是这类文档超出了本书的讨论范围。$^{[2]}$

也有一些观点认为这一层叫做目标需求，下一层叫做业务需求$^{[3]}$。还有另外一种分类方法，把校长的宏观需求称作“组织级需求”，把教务处主任的需求称作“业务需求”。我们在这里按照大多数文献的命名方式，以便读者可以方便地相互对照。

6.4.2 用户需求（User Requirement）

我们再来看看老师们和机房主任的需求。

- 老师们对教材、时间、讲课方法等都有自己的一些看法，因为 AI 相关课程的难度是一些传统课程不能比拟的，实际上也是变相地表达了对 AI 教育系统的期望（需求）；
- 机房主任关心的机房软硬件设备对系统的支持，是否需要升级、如何维护等等问题；
- 学生们对上课的体验、作业的体验有自己的要求。

我们把这种需求叫做用户需求（因为需求来自直接用户）。

图 6.4.3 - 第二层次：用户需求

它描述的是用户的目标，或用户要求系统必须能完成的任务。用例、场景描述、事件/响应表都是表达用户需求的有效途径。也就是说用户需求描述了用户能使用系统来做些什么。

- 来源：系统的实际操作、使用者。
- 内容：从岗位自身的实际工作内容出发，讲述系统可以完成什么任务。
- 作用：指导系统设计、产品规划、功能优化。
- 输出：通过对这一层次需求的理解，应该画出“用例图”和“用例说明”作为输出。

6.4.3 功能需求（Functional/Behavior Requirement）

第三层次的需求，有些直接来自用户，更多的来自需求分析人员的需求分析结果。

图 6.4.4 - 第三层次：功能需求

这些需求分析人员被称为系统分析员，在微软没有为这个角色专门设立岗位，而是由资深的开发人员担任。系统分析员一般被要求在领域和技术两个方面都有较深的理解，才能够架起从需求到设计的桥梁。需求分析的结果就是《需求规格说明书》。

功能需求描述开发人员需要实现什么，习惯上总是用"应该"对其进行描述："系统应该发送电子邮件来通知用户已接受其预定"，有时也被称作行为需求（behavioral requirement）。用户利用这些功能来完成任务，满足业务需求。

- 来源：业务管理者、系统使用者、需求分析结果。
- 内容：描述系统提供的功能或功能组，以及对每个功能的具体描述。
- 作用：指导系统后续的设计和开发，是需求分析的完成标志。
- 输出：通过对这一层次需求的分析，应该输出《需求规格说明书》，还有其它一些重要的图和模型，将会在下几节中详细说明。

6.4.4 需求来源

从上面的故事中，虽然所有的"需求"都是从客户的语言文字描述中得到的，但其实是有背后的领域知识的。更进一步地，我们可以挖掘整理出真正的需求来源，如图 6.4.5 所示。

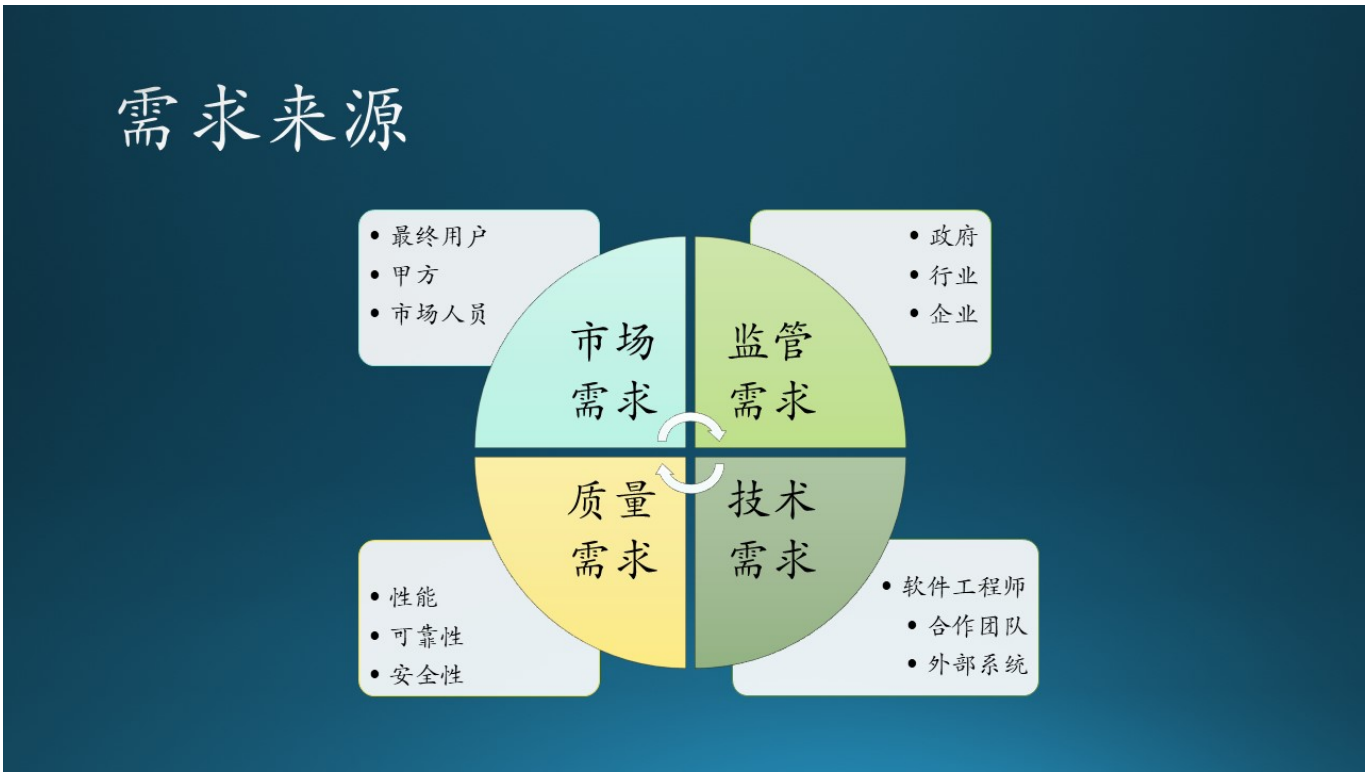


图 6.4.5 - 需求的来源

- 市场需求：最终用户、客户、产品人员
- 监管需求：政府、行业、企业
- 技术需求：软件开发团队、合作团队
- 质量需求：性能、可靠性、安全性等等

有了需求来源的明确定义，才能够有挖掘的基础和引导的目标。其中：

- 监管需求是自上而下的，软件产品要服从；
- 技术需求是有章可循的，软件产品只有服从；
- 质量需求是客观存在的，软件产品也只有服从。
- 只有市场需求是丰富多彩、富于变化的，所以需求的演进与引导都是从用户、客户、产品人员而来的。

笔者从长期的软件工程实践中发现，需求的演进动力，往往不是来自用户的，而是来自产品团队的。原因有几个方面：

- 对现有产品的附加功能
- 技术创新带来的新功能
- 多个产品的融合

所以，需求的引导也一般是由产品团队来完成的。就如同前面的一个故事中讲到的，用户需要快马，福特就制造了汽车。在福特那个年代，养马当然比买车便宜，但是现在养马反而是件奢侈的事情了，养马的需求从快速通行的需要变成了财大气粗的任性。

6.5 非功能性的需求分析

在6.3和6.4节中，我们分析了来自学校的教职员工的各个不同层次的需求，使我们清楚地认识到了业务、用户、功能三个层次的需求的区别和联系。在本节中，我们将会学习如何发现出一些隐形需求，以便能够更全面准确地把握需求。

6.5.1 需求组成

图 6.5.1 - 需求组成

我们先只看左侧分叉，可以得到四个公式：

$$ 需求 = 功能需求 + 非功能需求 \tag{6.5.1} $$

在工程实践中，功能需求占的比重很大，但是非功能需求也是非常重要的组成部分，而且这些非功能需求不能满足的话，更有可能造成整个项目的失败。所以我们有了公式 6.5.1。

$$ 功能需求 = 主要功能需求 + 辅助功能需求 \tag{6.5.2} $$

$$ 主要功能需求 = f(业务需求)+g(用户需求) \tag{6.5.3} $$

$$ 辅助功能需求 = 日志统计 + 可访问性 \tag{6.5.4} $$

公式 6.5.3 的含义是，我们用一种方法 $f()$ 来分析业务需求，用另一种方法 $g()$ 来分析用户需求，最后可以得到主要功能需求。

而公式 6.5.5 则说明了非功能需求的组成：

$$ 非功能需求 = 模糊需求 + 质量属性 + 约束条件 \tag{6.5.5} $$

接下来我们了解一下非功能需求的各个组成部分。

6.5.2 模糊需求（Overall Requirements）

模糊需求一般在业务需求层次提出，常见的就是三个字：快、好、省。

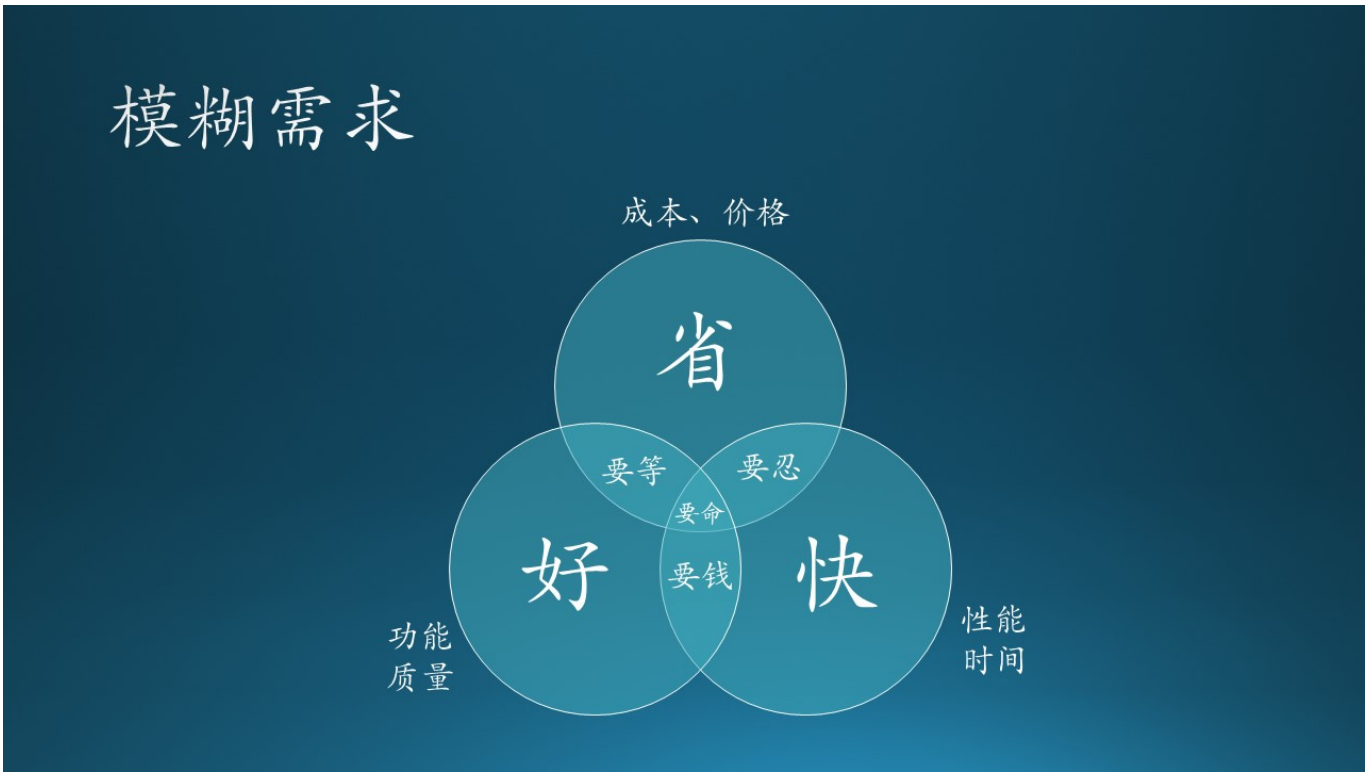


图 6.5.2 - 模糊需求

$$ 模糊需求 = 快 + 好 + 省 \tag{6.5.6} $$

对于业务级投资人、负责人等领导来说，预期的质量需求就是三个字：快、好、省。这和当年的"多快好省"经济发展总路线如出一辙，实践证明是不能兼顾的。

1. 快，可以理解为开发速度快，或者软件运行速度快。
2. 好，可以理解为开发质量好，或者软件功能好。
3. 省，可以理解为开发成本低，或者软件价格低。

这三者并存的话，会形成悖论：

1. 速度又快、功能又好的产品，价格肯定不低，做不到省。

   比如说我们训练一个深度学习模型，如果用 GPU 的话，训练速度快，效果好，但是 GPU 的价格非常的昂贵。

2. 功能又好、成本又省的产品，不是开发周期长，就是运行速度慢。

   同样是训练一个深度学习模型，用 CPU 当然也可以完成任务，但是使用 GPU 可以一小时搞定的东西，用 CPU 却需要两天。

3. 速度又快、成本又省的产品，用膝盖想一想都能知道好不到哪里去。

   还是训练深度学习模型，为了便宜就用 CPU 搞，为了速度快就少训练几轮，那么最后的模型效果肯定不好。

6.5.3 质量属性（Quality Attribute）

质量属性包含很多内容，我们将会在后面的章节中专门介绍，在本节中可以用公式 6.5.7 和图 6.5.3 来简单表述：

$$ 质量属性=性能+安全性+可靠性+可维护性+其它 \tag{6.5.7} $$

图 6.5.3 - 质量属性的组成

质量属性的项目非常多，但通常我们都用下面4个主要的属性来衡量软件的质量，而且是可以写在《需求规格说明书》中的。在公式 6.5.7 中用"其它"来代表不常用到的属性。

- 性能（Performance）

  比如，我们可以要求一个软件：

  - 用户登录响应时间不超过0.5秒；
  - 可以支持并发访问用户数至少10000个；
  - 系统至少可以存储100M的用户文件。

- 安全性（Security）

  其实就是正反两个方面：

  - 向合法用户提供服务；
  - 阻止非授权使用。

- 可靠性（Reliability）

  软件系统在一定的时间内无故障运行的能力，比如：

  - 72小时小负载下可稳定运行；
  - 30分钟内的超负荷运行；
  - 4小时内负载从10%增加到100%的运行情况。

- 可维护性（Maintainability）

  一般甲方会提出可维护性要求，但通常是比较虚的，比如：

- 要求乙方给甲方人员培训，熟悉系统架构和模块；
- 乙方在一年内保证软件顺利运行；
- 软件提供 API 接口，可以二次开发。

6.5.4 约束条件（Constraint）

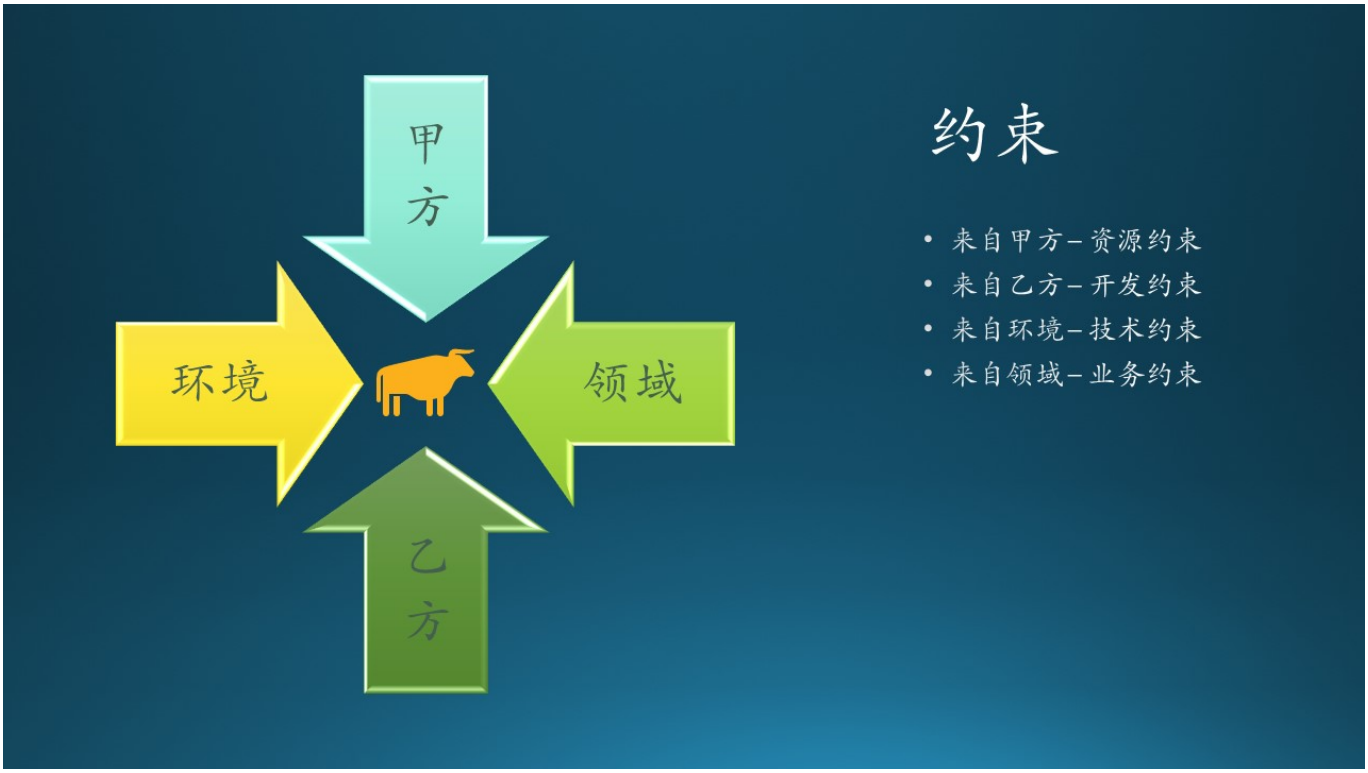


图 6.5.4 - 约束条件

$$ \begin{aligned} 约束条件=& 甲方约束（资源） \ +& 乙方约束（开发） \ +& 环境约束（技术） \ +& 领域约束（业务） \tag{6.5.8} \end{aligned} $$

来自甲方 - 资源约束

客户总是会在下面几个问题上“斤斤计较”：

- 时间
    - 即软件开发完毕交付的时间，而且如果客户不懂得软件开发的规律，往往会要求在不可能的时间内一次性交付。
    - 而对于开发者来说，一定要争取宽裕的开发时间，通常是把预估的时间乘以2.比如预估3个月完成，那么就要和客户承诺6个月完成。
    - 在交付问题上，一次性地不可能搞定所有需求，至少需要3次以上的迭代，在没有新需求出现的前提下，才有可能完成。
- 金钱

  即开发软件、部署软件、后期宣传所需要的费用。

来自乙方 - 开发约束

主要是乙方要考虑自己的开发团队的具体情况，比如：

- 领域知识

  开发人员对于目标软件所在的领域知识是否足够了解。比如要做一个银行系统的应用软件，一般银行都会找一个长期合作的软件开发商合作，避免领域知识的培训等额外工作。

- 技术水平

  - 技术岗位繁多

    如前端、后端，完全是不一样的编程思路。后端：入门难，深入更难，枯燥乏味，没有太大成就感，看一堆业务逻辑代码。前端：入门简单，先易后难，能看到自己做出来的展示界面，有成就感。

  - 开发语言繁多

    PHP 并不是最好的编程语言（这是业界流行的一个笑话），但是它和Java、C、C++、Python、C#、JavaScript等成为最流行的编程语言，一个开发人员能够熟练使用（不是精通）三种以上的编程语言，就已经很不错了。

  - 框架平台繁多

    会用 React 的就不用 Vue（二者都是JS前端框架），会用 Struts 的就不用 GWT（二者都是 Java 框架），会用 .NetFramework的就不用 .NetFramework Core（二者都是 C# 框架），等等。

- 团队情况

  一个糟糕的经验是：三个臭皮匠，顶个诸葛亮。一个合理的做法是：一个诸葛亮带三个臭皮匠。

  - 成员磨合程度，主要是在工作习惯、沟通能力方面的考量。一个团队有新人加入的话，在初期，往往是拉低团队的生产力；如果新人很多，磨合期就要预计长一些。

  - 成员地理分布，主要是指像微软这种跨国公司，经常有跨区域合作的情况。比如做手机上的 Edge 浏览器时，PM 主要在美国，开发人员主要在中国，但是分在北京和苏州两个地方，还有一些依赖模块在欧洲开发。这需要制定好接口，并协调多方的进度保持一致。

来自环境 - 技术约束

- 批准的技术清单

  许多大型组织在开发之前，都会列出一个可以使用的技术清单。如果你需要使用清单上没有的技术，就需要申请。

- 现有系统的互操作性

  在整合已有系统的时候可以使用的协议和技术。

- 目标部署平台

  目标部署平台是影响技术决策的主要因素之一。比如 Windows 和 Linux 的不同，再比如桌面软件和移动软件的区别。

  另外一个例子是目前的深度学习框架，如TensorFlow, PyTorch, MXNet等，都有自己的模型结构，无法互操作。微软开发了 ONNX（Open Neural Network Exchange，开放神经网络交换）格式的模型标准，

可使得模型在不同的框架下使用，并使用标准的方式来部署，而不需要按照框架。目前官方支持 ONNX 格式的框架有：Caffe2、PyTorch、MXNet、ML.NET、CNTK，而TensorFlow非官方（开源方式）支持。

- 使用开源软件

  GPL（General Public License）协议规定，如果使用了它人的开源软件，那么自己的软件必须也开源。

来自领域 - 业务约束

- 行业标准
  - 比如中国的财务软件就有自己的数据接口标准，否则会出现任意两个软件之间的数据交换困难。这就要求在需求中提及“输出必须符合接口标准”。
  - 在金融股票市场中，如果想提供换仓建议，那么软件必须根据历史数据（比如过去60天的），在周一上午 9:30 之前跑出结果来。
- 政策法规

  举两个例子：
  - 微软对用户的隐私问题非常的重视，所以在任何系统中，都不能保存能够反向映射到一个自然人的用户日志；
  - 中国对地图数据的安全性有要求，不能存放在中国境外的服务器上。所以当年笔者得到了一个进入微软做国内地图数据处理的工作机会，从心里感激这个政策法规。

6.6 需求技术分析

目前，软件需求的技术分析方法较多，如图 6.6.1 所示，有一些大同小异，而有的则基本思路相差很大，比如有自下而上的，也有自上而下的，这两者的思路正好相反。笔者提倡自上而下由全局出发全面规划分析，然后逐步设计实现。

图 6.6.1 - 需求技术分析的各种方法

在这一节中，我们泛泛地介绍各种需求技术分析方法，只要求大家知道有这些方法存在，不要求全部掌握，有兴趣的读者可以自己做深入研究。最后我们会详细介绍面向对象的分析方法，这是本书中要重点介绍的方法。

6.6.1 各种软件需求分析方法

从系统分析出发，可将需求分析方法大致分为功能分解方法、结构化分析方法、信息建模法和面向对象的分析方法。$^{[1]}$

- 功能分解方法

  将新系统作为多功能模块的组合，各功能也可分解为若干子功能及接口，子功能再继续分解，便可得到系统的雏形，即公式 6.6.1 所示：

$$ 功能分解 = 功能 + 子功能 + 功能接口 \tag{6.6.1} $$

- 结构化分析方法

  结构化分析方法是由数据流图和数据词典构成并表示，所以此分析法又称为数据流法。其基本策略是跟踪数据流，即研究问题域中数据流动方式及在各个环节上所进行的处理，从而发现数据流和加工方法。结构化分析可定义为数据流、数据处理或加工、数据存储、端点、处理说明和数据字典。

$$ \begin{aligned} 数据流图 =& 数据流 + 数据处理 + 数据存储 + 端点 \\ 数据说明 =& 端点 + 处理说明 + 数据字典 \\ 结构化分析 =& 数据流图 + 数据说明 \tag{6.6.2} \end{aligned} $$

- 信息建模方法

  信息建模可定义为实体或对象、属性、关系、父类型/子类型和关联对象。此方法的核心概念是实体和关系，最初由 P.P.S. Chen 在1976年提出，基本工具是 E-R 图。1981年 M.Flavin 改进后称之为信息建模法，后来又发展为语义数据建模法，并引入了面向对象的特定。其基本要素由实体、属性和联系构成。该方法的基本策略是从现实中找出实体，然后再用属性进行描述。

$$ \begin{aligned} 内部信息 =& 实体或对象 + 属性 \\ 外部信息 =& 关系 + 父子类型 + 关联对象 \\ 信息建模 =& 内部信息 + 外部信息 \tag{6.6.3} \end{aligned} $$

- 面向对象的分析方法

  面向对象的分析方法的关键是识别问题域内的对象，分析它们之间的关系，并建立三类模型，即：

  - 对象模型
  - 动态模型
  - 功能模型

  面向对象主要考虑类或对象、结构与连接、继承和封装、消息通信，只表示面向对象的分析中几项最重要特征。类或对象是对问题域中事物的完整映射，包括事物的数据特征（即属性）和行为特征（即服务）。

$$ \begin{aligned} 实体 =& 对象或类 + 结构与连接 \\ 特征 =& 继承 + 封装 + 消息通信 \\ =& 数据特征 + 行为特征 \\ 面向对象分析 =& 实体分析 + 特征分析 \tag{6.6.4} \end{aligned} $$

由于面向对象分析方法的先进性和重要性，我们在下面着重介绍一下。

6.6.2 面向对象的分析方法

OOA（Objec-Oriented Analysis），面向对象的分析方法，目前已经衍生出许多方法，每种方法都有各自的进行产品或系统分析的过程，有一组可描述过程演进的图形标识，以及能使得软件工程师以一致的方式建立模型的符号体系。

图 6.6.2 - 面向对象的需求分析方法

现在广泛使用的 OOA 方法有以下几种：

Booch 方法

Booch 方法包含“微开发过程”和“宏开发过程”。微开发过程定义了一组任务，并在宏开发过程的每一步骤中反复使用它们，以维持演进途径。Booch OOA 宏开发过程的任务包括标识类和对象、标识类和对象的语义、定义类与对象间的关系，以及进行一系列求精从而实现分析模型。

Rumbaugh 方法

Rumbaugh 和他的同事提出的对象模型化技术（OMT）用于分析、系统设计和对象级设计 。分析活动建立三个模型：对象模型（描述对象、类、层次和关系），动态模型（描述对象和系统的行为），功能模型（类似于高层的DFD，描述穿越系统的信息流）。

1. 建立对象模型

   面向对象分析首要的工作，是建立问题域的对象模型。这个模型描述了现实世界中的“类与对象”以及它们之间的关系，表示了目标系统的静态数据结构。静态数据结构对应用细节依赖较少，比较容易确定；当用户的需求变化时，静态数据结构相对来说比较稳定。因此，用面向对象方法开发绝大多数软件时，都首先建立对象模型，然后再建立另外两个子模型。

   对象模型通常有5个层次，基本的工作步骤是：

   - 确定对象和类（object and class）

     这里所说的对象是对数据及其处理方式的抽象，它反映了系统保存和处理现实世界总某些事物的信息能力。

     类是多个对象的共同属性和方法集合的描述，它包括如何在一个类中建立一个新对象的描述。

   - 确定结构（structure）

     结构是指问题域的复杂性和连接关系。类成员结构反映了泛化—特化关系，整体—部分结构反映整体和局部之间的关系。

   - 确定主题（subject）

     主题是指事物的总体概貌和总体分析模型。

   - 确定属性（attribute）

     属性就是数据元素，可用来描述对象或分类结构的实例。

   - 确定方法（method）

     方法是在收到消息后必须进行的一些处理方法。

2. 建立动态模型

   - 第一步，编写典型交互行为的脚本。虽然脚本中不可能包括每个偶然事件，但是，至少必须保证不遗漏常见的交互行为。
   - 第二步，从脚本中提取出事件，确定触发每个事件的动作对象以及接受事件的目标对象。
   - 第三步，排列事件发生的次序，确定每个对象可能有的状态及状态间的转换关系，并用状态图描绘它们。
   - 最后，比较各个对象的状态图，检查它们之间的一致性，确保事件之间的匹配。

3. 建立功能模型

功能模型表明了系统中数据之间的依赖关系，以及有关的数据处理功能，它由一组数据流图组成。其中的处理功能可以用IPO（Input/Processing/Output，输入/处理/输出）图（或表）、伪码等多种方式进一步描述。通常在建立对象模型和动态模型之后再建立功能模型。

Coad 和 Yourdon 方法

Coad 和 Yourdon 方法常常被认为是最容易学习的 OOA 方法。建模符号相当简单，而且开发分析模型的导引直接明了。其 OOA 过程概述如下：

1. 使用"要找什么"准则标识对象；
2. 定义对象之间的一般化/特殊化结构；
3. 定义对象之间的整体/部分结构；
4. 标识主题（系统构件的表示）；
5. 定义属性及对象之间的实例连接；
6. 定义服务及对象之间的消息连接。

Jacobson方法

也称为 OOSE（面向对象软件工程）。Jacobson方法与其他方法的不同之处在于他特别强调使用实例（use case）——用以描述用户与系统之间如何交互的场景。Jacobson方法概述如下：

1. 标识系统的用户和它们的整体责任；
2. 通过定义参与者及其职责、使用实例、对象和关系的初步视图，建立需求模型；
3. 通过标识界面对象、建立界面对象的结构视图、表示对象行为、分离出每个对象的子系统和模型，建立分析模型。

Wirfs-Brock 方法

Wirfs-Brock 方法不明确区分分析和设计任务。从评估客户规格说明到设计完成，是一个连续的过程。与 Wirfs-Brock 分析有关的任务概述如下：

1. 评估客户规格说明；
2. 使用语法分析从规格说明中提取候选类；
3. 将类分组以表示超类；
4. 定义每一个类的职责；
5. 将职责赋予每个类；
6. 标识类之间的关系；
7. 基于职责定义类之间的协作；
8. 建立类的层次表示；
9. 构造系统的协作图。

统一的OOA方法（UML）

统一的建模语言（UML）已经在企业中广泛使用，它把Booch、Rumbaugh和Jacobson 等各自独立的 OOA 和 OOD 方法中最优秀的特色组合成一个统一的方法。UML 允许软件工程师使用由一组语法的语义的实用的规则支配的符号来表示分析模型。

在UML中用5种不同的视图来表示一个系统，这些视图从不同的侧面描述系统。每一个视图由一组图形来定义。这些视图概述如下：

【slide】

1. 用户模型视图：这个视图从用户（ 在UML中叫做参与者）角度来表示系统。它用使用实例（use case）来建立模型，并用它来描述来自终端用户方面的可用的场景。
2. 结构模型视图：从系统内部来看 数据和功能性 。即对静态结构（类、对象和关系）模型化。
3. 行为模型视图：这种视图表示了系统动态和行为。它还描述了在用户模型视图和结构模型视图中所描述的各种结构元素之间的交互和协作。
4. 实现模型视图：将系统的结构和行为表达成为易于转换为实现的方式。
5. 环境模型视图：表示系统实现环境的结构和行为。

通常，UML 分析建模的注意力放在系统的用户模型和结构模型视图，而 UML 设计建模则定位在行为模型、实现模型和环境模型。

6.6.3 划分需求分析与系统设计的边界

在 OOA 方法中，要求建立对象模型、动态模型、功能模型。其实 OOA 这个理论本身是一种系统分析的技能，对很多“需求分析”的内容并不涵盖。在很多资料上并没有介绍具体如何做，导致很多人在实践中把分析和设计阶段的任务混为一谈。

为什么这种情况能一直持续下去呢，一个重要原因是在一般的软件公司里，总会有一位“领域专家”，往往是他/她既做需求分析，又做系统设计。在微软其实也是这样，没有各自独立的角色分别做需求分析和系统设计两件事，往往是一个从底层做起的程序员，逐渐成长为领域专家后，再去负责需求分析与系统设计的工作。这样做的问题是：

1. 他/她在做需求分析的时候，脑子里会不自觉地想到如何去实现；
2. 写出来的文档没有边界，定义出来的类会让后续的开发人员感觉到限制；
3. 隐藏了一些真实的用户需求，而用自己的理解直接代替。

用一个简单的比喻帮助大家理理解它们的分界线：我们在做需求分析的时候，可以把每个角色、实体、模块等等都看作一个鸡蛋，它们在需求分析阶段都是不透明的，我们也不需要在需求分析阶段去窥探鸡蛋内部的构造，否则的话，我们会得到一堆破鸡蛋——所有的蛋液流出来并混在一起，造成无法进行下一步的分析。

这个鸡蛋壳，就处于上述的“统一的 OOA 方法”的5个视图中的第3个上。

笔者在多年的软件工程实践中也总结出了一套方法，经过与各种模型、理论的对比，发现竟然与“统一的 OOA 方法”完全一致！所以，我们会在后续的章节中通过真实的案例继续介绍这种方法。

6.7 系统边界与用户模型

这是做技术需求分析的第一步，在这一步中，我们要建立“用户模型视图”。我们使用 6.3 节中的 AI 教学需求来作为例子进行分析，步骤如下：

图 6.7.1 - 用户模型视图的建立方法

1. 这个视图从用户（在UML中叫做参与者）角度来审视系统，通过上下文图来确定系统边界，我们可以用一个"空的鸡蛋筐"来比喻这个系统。
2. 使用用例图来建立模型，每个用例好比一个鸡蛋，在这一阶段我们不需要探查鸡蛋内部的构造，只需要把所有鸡蛋都放进筐里即可。
3. 有紧密联系的用例可以组织在一起形成子系统，有可能的话，我们尽量把这些子系统也发掘出来，并规划它们的边界。这就好比一堆红皮（含有卵壳叶啉）鸡蛋要和另外一堆白皮鸡蛋分开。
4. 对于复杂的用例，可以进一步细化，定义一些子用例。这就好比鸡蛋堆里混进来一枚大鹅蛋，一口吃不掉，需要拆成几份鸡蛋大小的部分。
5. 用例说明可以帮助后续的人员深入理解，避免理解偏差。相当于给每个鸡蛋注明要做什么菜品和口味。

6.7.1 系统上下文图（Context Diagram）

上下文图是需求分析的第一站，用于确定系统的边界，这对于整个系统的目标/非目标（Go/Non Goal）是非常关键的。

正确的上下文图

我们先看第一张图：

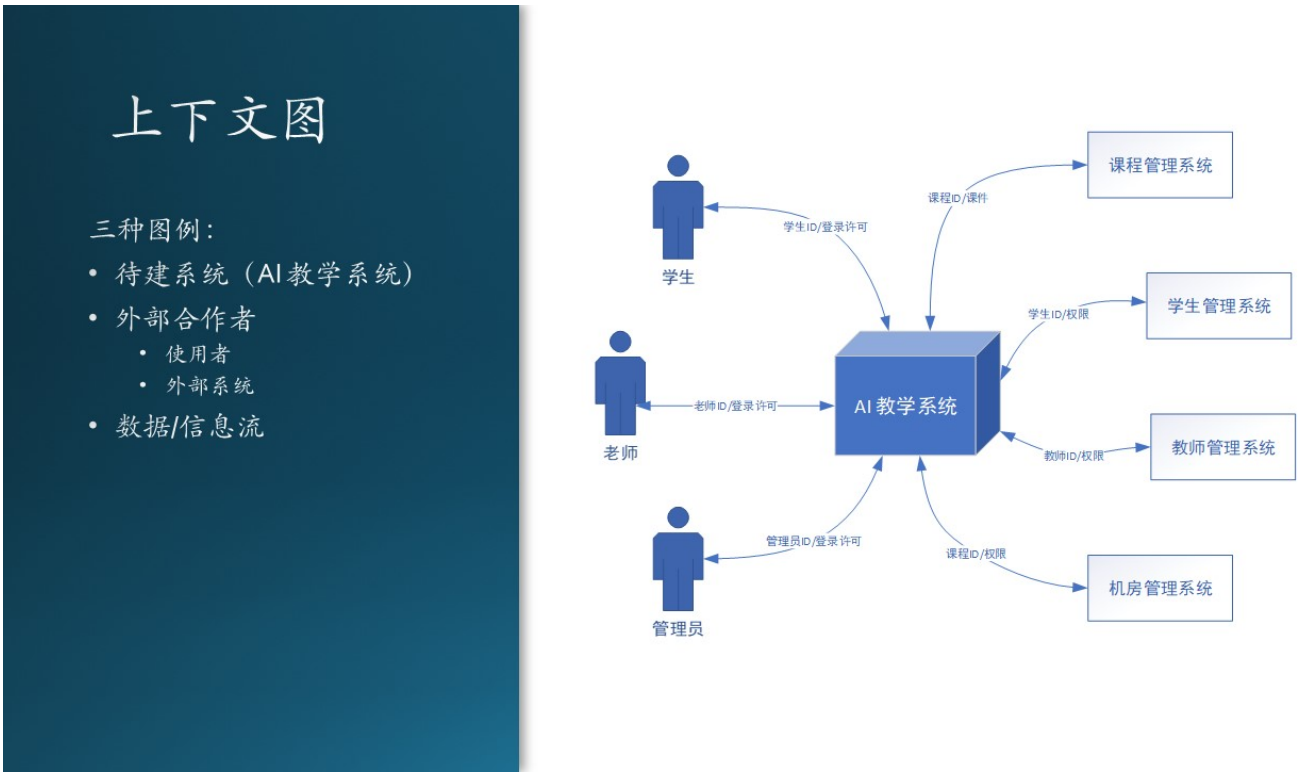


图 6.7.2 - 上下文图

图6.7.2中有三种图例：

1. 待建系统本身

   中心位置是“AI教学系统”，我们把它看成黑盒子，因为在这个阶段，我们还没有对其进行分析，所以得不到细节。

2. 外部合作者

   包括两类：

   - 用户：人机交互。
   - IT 系统：使用计算机网络协议传递消息。

3. 数据/信息流

   有以下几个特点：

   - 信息流是双向的；
   - 都是名词，表示信息或者数据；也可以使用动词短语，比如“发送登录请求”，但要求所有信息流的文字是相同的形式，全是名词或者全是动词短语；
   - 可以是协议，也可以是数据格式；
   - 特定规范，如安全性、可用性、吞吐量等。

   这些信息将成为设计外部数据接口的依据。当然，在此图中，我们并没有写出所有的信息流，比如：

   - 学生可以提交作业，然后获得作业评分；
   - 老师可以发起命令保存手写板书，然后以邮件形式发送给学生；
   - 管理员可以开启/关闭教学系统。

错误的上下文图

我们再看第二张图：

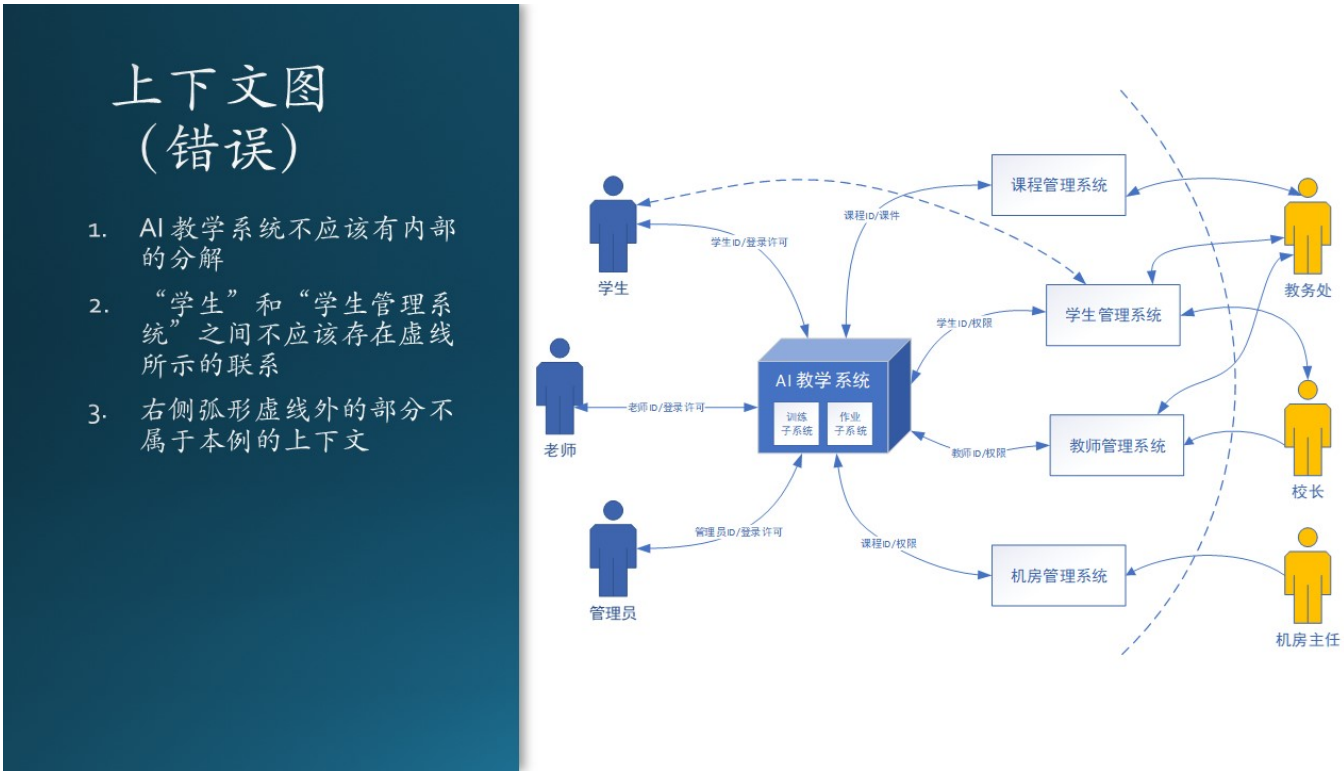


图 6.7.3 - 错误的上下文图

根据前面讲述的关于系统上下文图的绘制规则，我们可以看到这张图中存在以下错误：

1. AI教学系统不是黑盒子，多画了两个内部的子系统；
2. 在学生与学生管理系统之间，不应该有那条虚线表示的联系，因为那不属于待建系统的范畴；
3. 在右侧弧形虚线之外，还有一些其它的用例，也不属于待建系统的范畴。

小结

系统上下文图是软件需求分析中的一项很重要的内容，属于DFD（Data Flow Diagram，数据流图）中的最高层次的图，是系统功能的最高抽象。系统上下文图将整个系统看做是一个过程，这个过程实现系统的所有功能。所以上下文图中存在且仅存在一个过程，表示整个系统。

将整个系统功能抽象为单一过程之后，系统本身就变成了一个黑盒，此时只有依据系统与外界的所有交互才能准确界定系统的功能。所以，系统上下文图中需要表示出所有和系统有交互的外部实体，并描述交互的数据流，包括系统输入和输出。

6.7.2 用例图（Use Case）

获得了系统上下文后，我们应该进一步进入待建系统进行分析，这就需要用例图。用例图主要用来描述“用户、需求、系统功能单元”之间的关系。它展示了一个外部用户能够观察到的系统功能模型图。

简单的例子

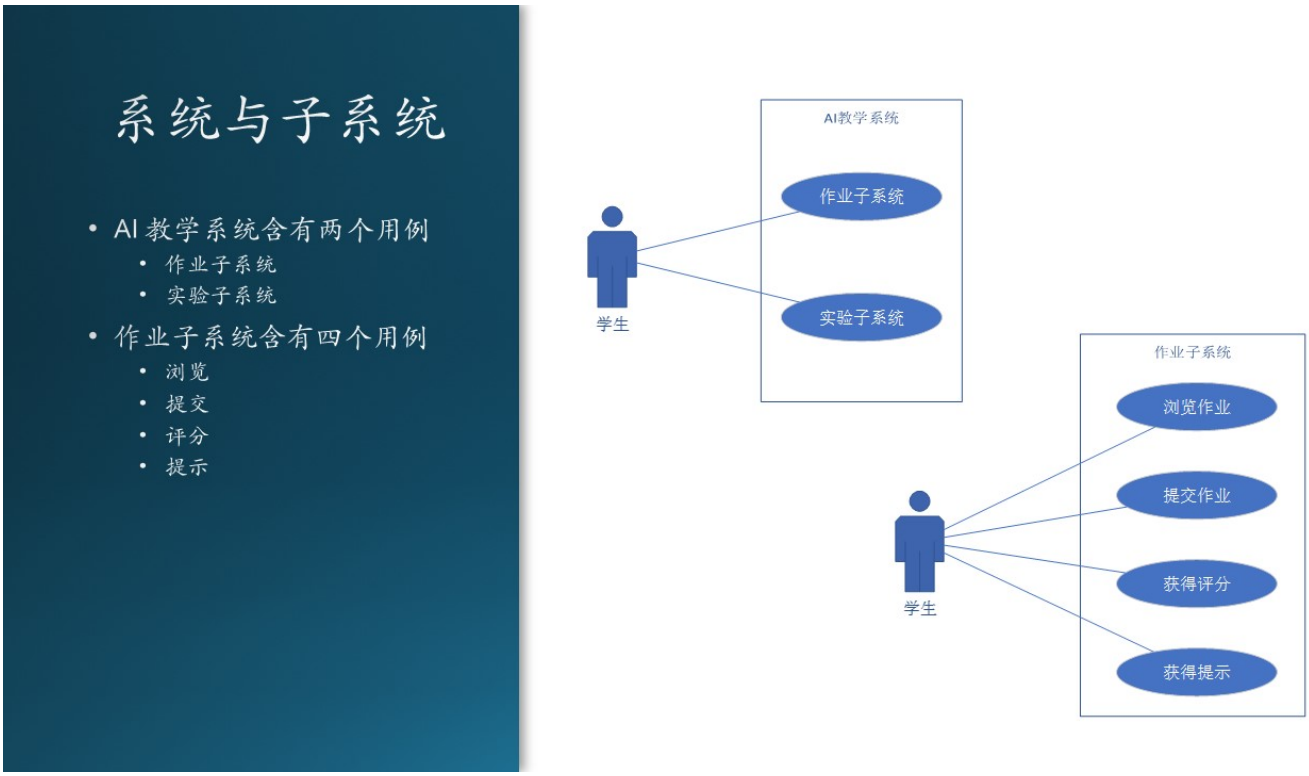


图 6.7.4 - 系统与子系统用例

上图中，左侧是一个顶层的用例，表示在 AI 教学系统包含有两个顶层用例：

- 作业子系统
- 实验子系统

当然还会包含其它子系统，我们只用这两个子系统举例说明。

在右侧的图中，深入到作业子系统中，可以看到需要至少4个用例来支持：

- 浏览作业
- 提交作业
- 获得评分
- 获得提升

在上面的图中，一共有四种图例：

1. Actor（参与者），在这里是“学生”。它描述与系统交互的人或物，代表外部实体（如用户，硬件、设备等）；
2. 系统或者子系统边界，这里是AI教学系统和作业子系统；
3. 用例，包括子系统用例（作业子系统、实验子系统）和功能用例（浏览作业、提交作业、获得评分、获得提示）。它是执行者与计算机一次典型交互，代表系统某一完整功能。
4. 关联线，表示参与者与用例的交互关系。注意这是一条直线，没有箭头，在旧的版本的UML中是一条指向用例的三角箭头线。

如何发现参与者：

- 谁使用该系统；
- 谁改变系统的数据；
- 谁从系统取信息；

- 谁需要系统的支持以完成日常任务；
- 谁负责维护管理并保持系统正常运行；
- 系统需要应付那些硬件设备；
- 系统需要和哪些外部系统交互；
- 谁对系统运行产生的结果感兴趣。

如何发现用例：

- 参与者需要获取何种功能，需要做什么；
- 参与者需要读取产生、删除、修改或存储；
- 系统发生时间和执行者间是否要通信。

复杂的例子

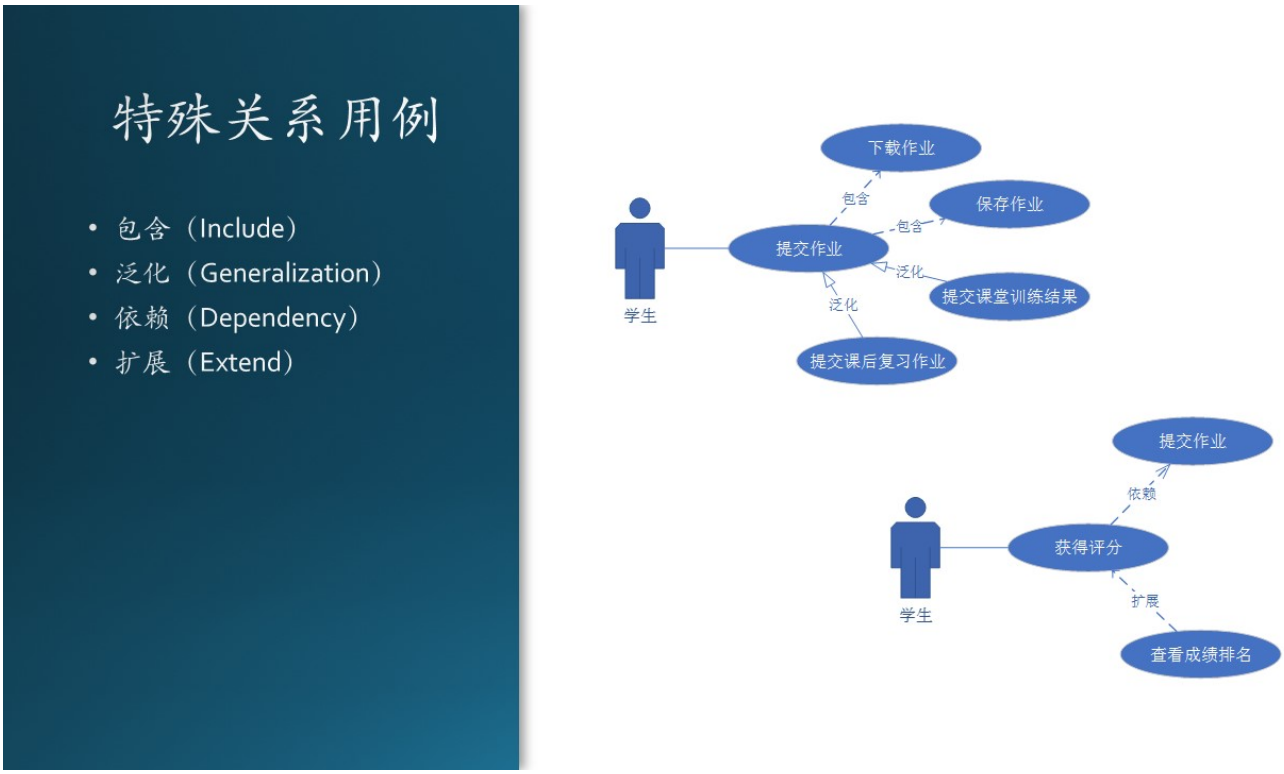


图 6.7.5 - 特殊关系的用例

1. 包含（include）关系

左图中，用“提交作业”作为基用例，首先，它“包含”了两个底层用例：

- 下载作业：做作业之前，先要把作业下载下来，包括描述、数据、模型等等；
- 保存作业：作业做了一半，还没到提交的地步，可以先保存起来，留着明天做。

这里用的是“包含”的关系，从基用例指向被包含用例，虚线短箭头，即一个基本用例包含另一个用例行为（要实现基本用例必须满足另一个用例行为）。

2. 泛化（Generalization）关系

其次，提交作业还有可能只是一个抽象的概念，它实际上可以“泛化”为两个实际的用例：

- 提交课堂训练结果作业
- 提交课后复习作业

这里用的是“泛化”的关系，从子用例指向基用例，实线三角箭头，是“一般情况”与“特殊情况”的关系（特殊者指向一般执行者）。

3. 依赖（Dependency）关系

右图中，如果想“获得评分”，必须先“提交作业”，所以这是一个依赖的关系，用虚线长箭头，从向基用例指向被依赖用例，表示没有后者的话，前者无法执行。

4. 扩展（Extend）关系

右图中，在“获得评分”之后，还可以看看整体得分情况，以得到自己的排名，所以“查看成绩排名”是“获得评分”的一个扩展。

这种情况允许一个用例扩展另一个用例提供的功能，与泛化类似，但有更多限制：基本用例必须声明“扩展点”，扩展用例只能在扩展点上增加新行为。

在参与者之间，也可能存在泛化关系，比如“学生”可能会泛化为“本地学生”和“留学生”，“老师”可能会泛化为“本校老师”和“访问学者”。

6.7.3 用例说明

用例说明通常与用例图配合使用，以进一步描绘用例功能，见表 6.7.1。

表6.7.1 - 用例说明表

项目	内容
用例编号	如：ASE-01
用例名称	即用例图中的描述，如“浏览作业”
参与者	在本例中为“学生”
优先级	高、中、低（三者选一）
用例说明	如：学生可以浏览老师布置的所有作业历史及最新作业。
前置条件	执行本用例前的系统状态，如：学生必须先输入学生ID和课程ID。
基本事件流	说明在正常情况下，最常用的流程。如：学生点击“浏览作业”按钮，系统根据课程ID查询作业历史，然后以列表形式在界面上显示。
异常事件流	说明在出现错误和其它异常时，和基本流程的不同之处。如：登录失败，课程已到达上限，该课程尚无作业，等等。
后置条件	数据库的变化、系统界面变化等，如：学生可以看到作业列表。

6.8 数据流图与结构模型

这是需求分析的第二步，在这一步中，我们要建立“结构模型视图”。步骤如图6.8.1：

图 6.8.1 - 结构模型视图

1. 首先通过数据流图把上一阶段的每个用例都串联起来；
2. 发现用例之间的数据交换关系；
3. 根据这些数据定义数据结构模型；
4. 确定各个模型之间的关系；
5. 发现隐藏用例，以建立功能模块的集合，形成子系统。

6.8.1 数据流图

每个系统或功能都会涉及到不同角色、业务概念和物品等，这些事物之间会有很多关系，发生很多事情。特别是当我们刚接触新的业务时，最急迫需要解决的问题就是理清楚这些业务概念以及他们之间的关系。数据流图可以帮助我们做到这一点。

数据流图（DFD，Data Flow Diagram），它从数据传递和加工角度，以图形方式来表达系统的逻辑功能、数据在系统内部的逻辑流向和逻辑变换过程，是结构化系统分析方法的主要表达工具及用于表示软件模型的一种图示方法。

作业自动评分的数据流图

我们看看 AI 教育系统的例子中，关于学生作业上传后自动评分的功能是如何实现的，请看图 6.8.2 中的上半部分。

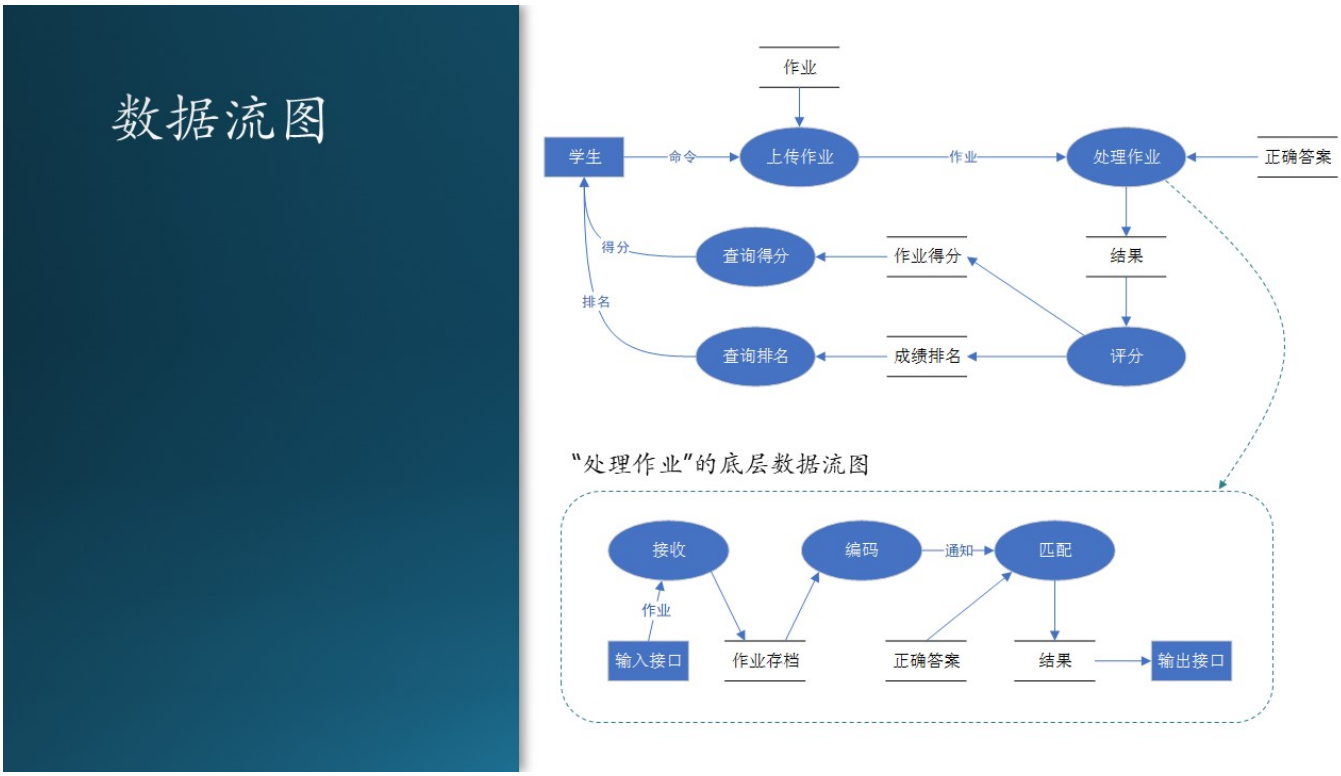


图 6.8.2 - 数据流图

图 6.8.2 中一共有4种图例：

1. 矩形框图，外部交互方，数据的起点或者终点；
2. 椭圆，数据加工逻辑，是对数据进行处理的单元，接收数据、处理、输出，在目前阶段，不需要考虑具体用什么语言、算法、代码来实现；
3. 双边线，数据存储，可以是文件或数据库，在目前阶段，不需要考虑具体用什么设备来存储。
4. 连线，数据流，如果是处于交互方与加工逻辑之间，或者处于两个加工逻辑之间，则在线上有说明，表明是一种内存数据流。箭头表示流动方向。

图中描绘了系统逻辑模型，没有具体的物理元素，只描绘数据信息在系统中流动处理情况。是非常好的软件设计出发点，因为它反应系统的业务逻辑，确定了模块/函数边界。

我们把"上传作业"、"查询得分"、"查询排名"叫做功能（Function），把这三个的组合叫做"自动判作业"的特性（Feature）。

底层数据流图

在上图中，名称为"处理作业"的加工逻辑可能很复杂，那就需要进一步细化，绘制下一层的数据流图，如图 6.8.2 的下半部分。

大家可以不必纠结图中的加工逻辑的具体细节，比如为什么要有"编码"这一步骤，只需要理解这种分解形式即可。

再比如，我们确实需要"作业存档"这一存储吗？这也属于细节实现，在具体设计时，再根据"需求"进一步讨论。这里说的"需求"并不是用户需求，而是技术需求，如：系统要求每个上传的作业都需要存档，便于后续做分析模型。

训练、推理的数据流图

图6.8.3这个例子，场景是考察同学们对预处理特征值的理解：

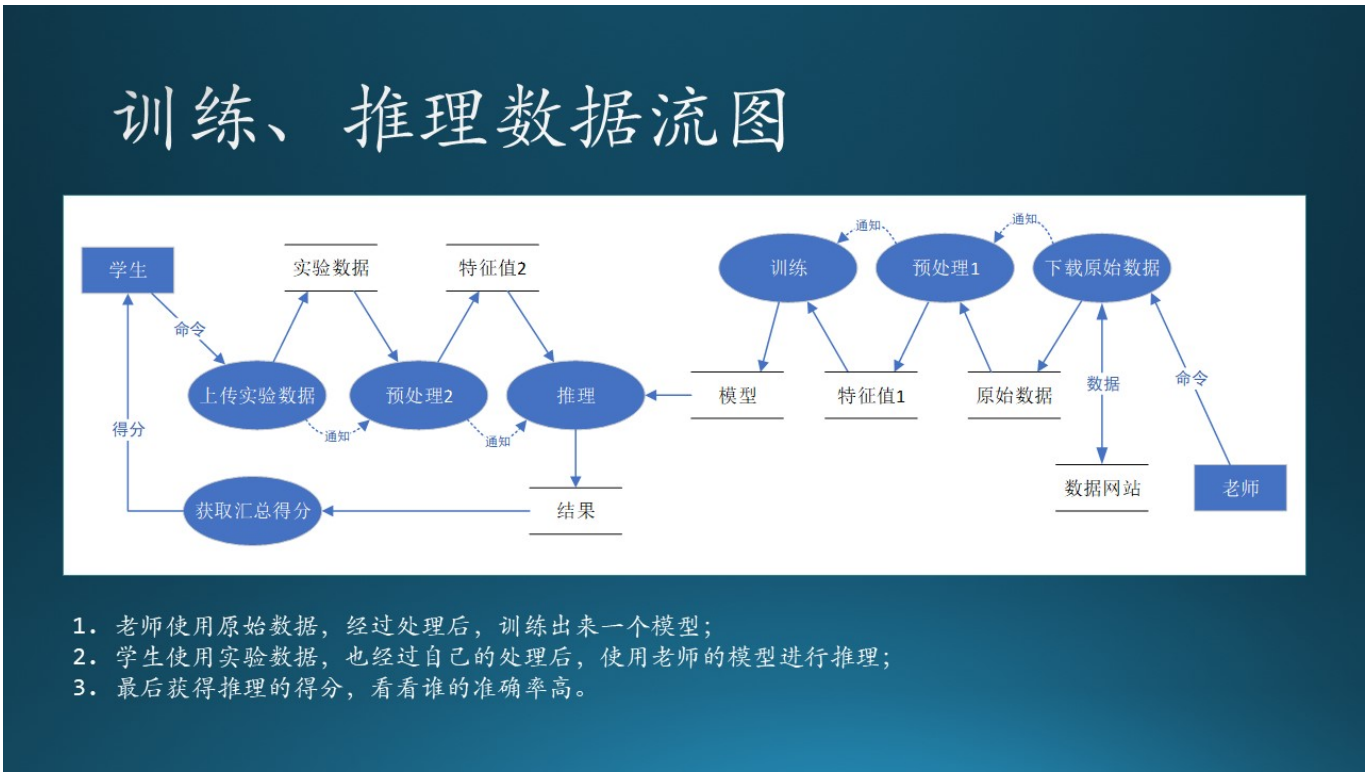


图 6.8.3 - 数据流图

1. 老师使用原始数据，经过处理后，训练出来一个模型；
2. 学生使用实验数据，也经过类似的处理后，使用老师的模型进行推理；
3. 最后获得推理的得分，看看谁的准确率高。

“模型”作为数据存储，是训练和推理的结合部，它是个静态存储，不会主动跑到“推理”模块中，所以训练和推理两个子过程是异步的：训练子过程先发生，然后推理子过程可以再发生无数次，再然后训练子过程可以再次发生，修改模型。

这个例子和大多数 Pipeline（数据处理流程）类型的软件一样，每一步都要求有数据输出，保存在文件或数据库里，便于解耦各个加工逻辑，设计、实现、调试都很方便。

小结

画数据流图需要注意的要点：

1. 一个加工的数据不应与输入数据流同名，即使他们的组成成分相同。
2. 保持数据守恒。简单点说，加工处理后的输出数据必须是从加工的输入数据流获得，或者是其产生的数据。
3. 加工必须有输入输出数据流，因为系统不会凭空出现不明数据，这个还是比较好理解的。
4. 所有数据必须由一个外部实体开始，也要从一个外部实体结束。比如图xx中，
5. 外部实体之间不允许存在数据流。
6. 提高数据流图的易懂性。注意合理分解，要把一个加工分解成几个功能相对独立的子加工，这样可以减少加工之间输入、输出数据流的数目，增加数据流图的可理解性。

第5点很有趣，因为会有这样的场景发生：老师训练完一个模型后，用微信通知学生们可以开始推理了。这就是两个外部实体之间存在了数据流，但是，这个数据流（微信）不属于这个系统。我们要想克服这个缺点（即，

不使用微信通知），必须给系统增加一个“自动通知”机制：当新模型生成后，系统会自动通过电子邮件、短信等可控方式通知学生们。

我们回忆一下6.6节中的用例图，里面有很多椭圆形的用例，通过名称的对比，我们可以在数据流图中看到同样的椭圆形的“用例”，只不过在这里叫做“加工”。用例图只与外部系统和参与者有关，而数据流图更进一步发掘用例背后的故事，包括逻辑和数据，这样才能形成完整的功能。

数据流图的作用：

1. 数据流图中的数据起点终点，实际上是系统的用户。
2. 数据流图中的加工，就是下一步要设计和编码的模块。
3. 数据流图中的数据，就是下一步要设计的数据库、文件。
4. 数据流中数据源（矩形）与加工（椭圆）的交互，就是下一步要设计的用户界面或者接口。

6.8.2 定义数据模型

“数据流图”的名称本身就非常明确地告诉我们它是着眼于数据的，在画数据流图的时候，一定要仔细考虑实体之间存在的是内存数据流还是外部数据存储，如果是后者，那么就要做进一步的数据模型定义了。

我们仍然用作业自动评分的功能能来举例，分析一下老师、学生、作业、题目、答案、排行榜之间的关系，见图 6.8.4。

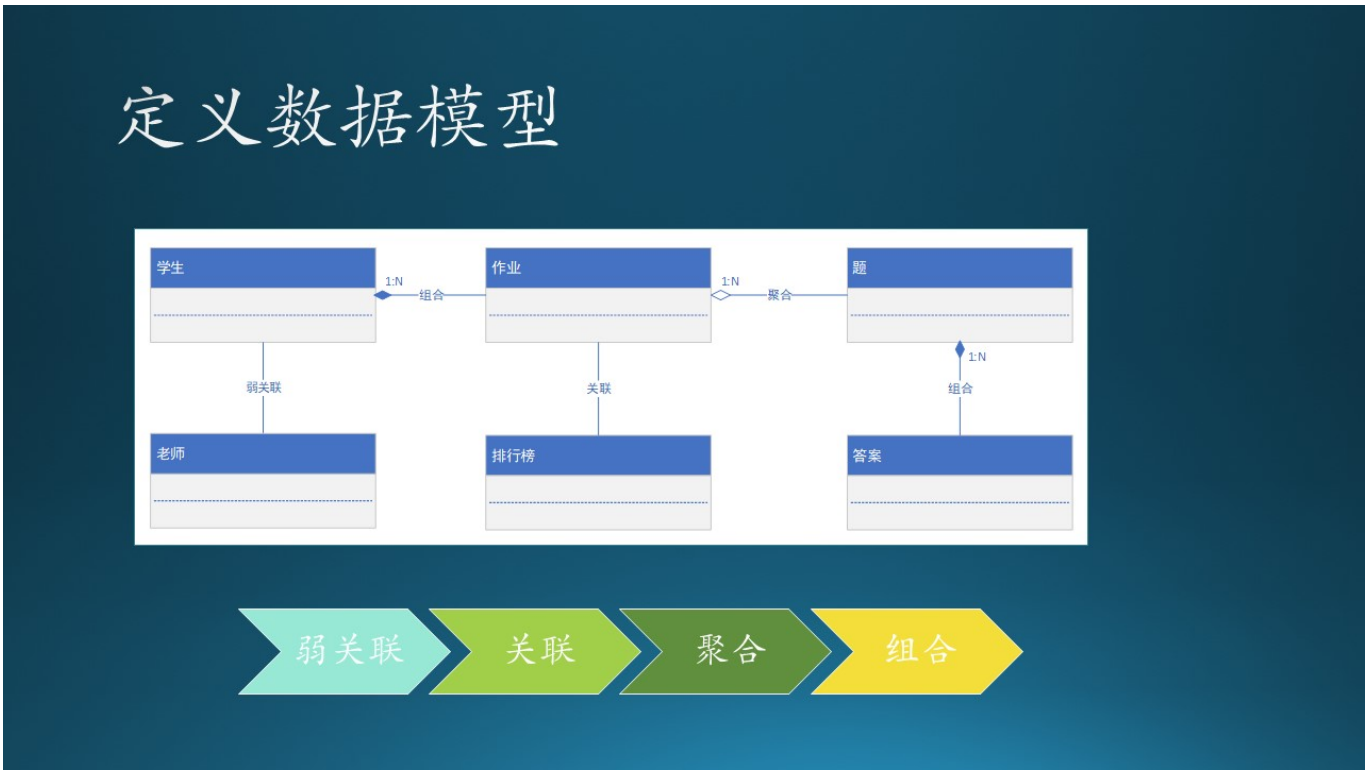


图 6.8.4 - 定义数据模型

笔者强烈反对在目前的需求分析阶段就给每个数据实体填上属性（字段），那应该是在设计阶段才完成的事情。所以在图6.6.4中，大家可以看到每个数据实体的属性字段都是空着的。

“老师”与“学生” - 关联（Association）

- 老师和学生没有强相关关系；
- 一个老师可以有多个学生；

- 一个学生可以有多个老师。

从严格的数据建模问题上看，老师和学生甚至都没有关系。人们通常所说的“师生关系”，实际上是一种社会关系，而我们这个系统不是为“社会学”而开发的。在教育系统的软件中，除了“班主任”这一特殊职务外，老师和学生实际上是通过课程联系在一起的，即：王老师是学生毛毛的数学课老师。

“学生”与“作业” - 组合（Composition）

- 一个学生可以写很多份作业；
- 一份作业只属于一个学生；
- 没有学生，就没有作业。

以上观点是从实例角度来看的，即，有一个叫毛毛的学生，他写了很多份作业，他和作业之间符合有上述三个描述。

“作业”与“排行榜” - 关联（Association）

- 作业会得到一个分数，所有的作业的分数按大小排序后，会得到排行榜。所以作业和排行榜是一个关联关系；
- 缺失一个学生的作业，其它学生的作业依然可以形成排行榜；
- 没有排行榜的话，作业依然有它存在的意义。

“作业”与“题” - 聚合（Aggregation）

- 作业由很多不同的题聚合而成；
- 题可以脱离作业而存在。

“题”与“答案” - 组合（Composition）

- 一个题目可以有多个答案（当然通常只有一个答案）；
- 答案脱离了题目就没有意义了，必须说“这个答案是那个题目的答案之一，而不是别的题目的答案”。

下面我们专门说说组合和聚合的区别，见表6.8.1：

表6.8.1 - 组合和聚合的区别$^{[4]}$

比较项目	聚合 Aggregation	组合 Composition
存在	从类独立地存在于主类之外	从类不能独立存在
关系	has-a，主类中有一个从类	part-of, contains-a，从类是主类的一部分
程度	弱	强
图例	空心菱形	实心菱形
功能	删除主类不会影响从类	删除主类，从类将无从依靠
举例	汽车 vs 轮子和发动机、雁群 vs 大雁	人 vs 大脑和身体，树 vs 树叶

6.9 状态转换与行为模型

在这一步中，我们会建立“行为模型视图”，这种视图表示了：

- 系统动态和行为，或者是对象的状态和行为；
- 还描述了在用户模型视图和结构模型视图中所描述的各种结构元素之间的交互和协作。

请注意：到了这里已经快接近设计范畴了，也就是说，我们已经快破开鸡蛋壳了，一旦破壳，就会进入设计的大门。所以到了这一步，是需求分析的最后一个环节了。说到底，需求分析和系统设计也不是完全隔离的，所以在这一步里有些交融也是合理的。

6.9.1 状态转换图（State Machine）

状态转换（迁移）图是描述对象的状态在响应外部的信号后进行转换的一种图形表示。

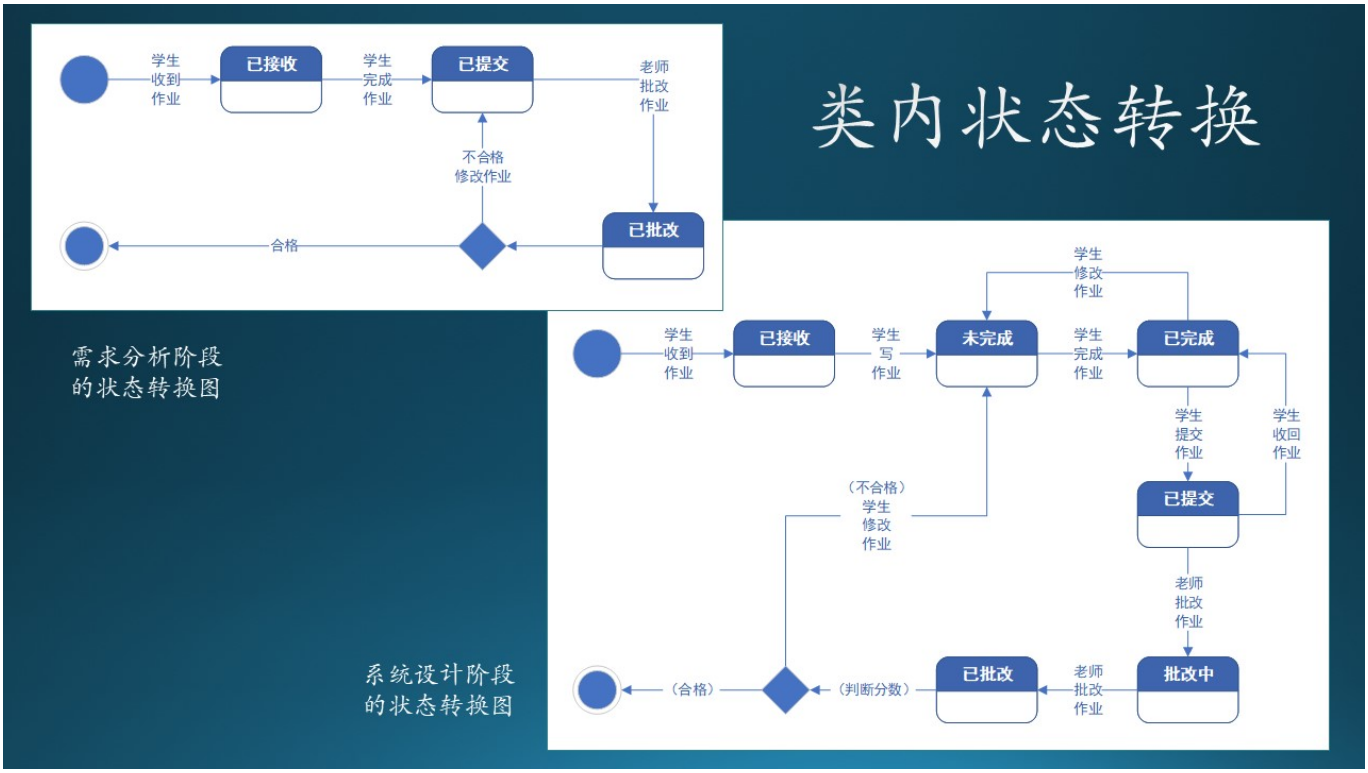


图 6.9.1 - 类内状态转换图

对现实世界的状态转换分析

我们以作业对象为例，绘制出状态转换图，如图 6.9.1 的左上角的子图。我们可以看到四种图例：

1. 起始和终止状态，圆形；
2. 状态，被观察到的对象行为模式，用圆角矩形表示；
3. 事件或行为，引起状态转换的外界事件，用有方向的连接线表示；
4. 菱形，判断条件。

也可以用表格来表示这种转换关系，但是不如状态转换图方便理解，如表 6.9.1。

表 6.9.1 - 状态转换表

源状态\目标状态	到起始状态	到已接收状态	到已提交到状态	到已批改状态	到终止状态
从起始状态	N/A	学生收到作业			

源状态\目标状态	到起始状态	到已接收状态	到已提交到状态	到已批改状态	到终止状态
从已接收状态		N/A	学生完成作业		
从已提交状态			N/A	老师批改作业	
从已批改状态			不合格，学生修改作业	N/A	作业合格

对软件系统的状态转换设计

在需求分析阶段，很容易犯的错误是绘制如图 6.9.1 右下角所示的状态转换图。比较左上角和右下角子图，我们可以看到后者多出来几个状态：

- 未完成
- 已完成
- 批改中

这是为什么呢？

因为前者是现实世界中的状态转换图，是一种（客观）需求分析的结果；而后者是软件世界的状态转换图，是（主观）系统设计的结果。

我们用“未完成”状态为例说明：

- 在现实世界中，未完成的状态确实存在，但是它对环境来说毫无意义，老师最后只看你是不是及时完成了作业，而家长会一直督促你，直到你完成作业，所以，“未完成”状态在现实世界中是一个不可接受的状态；

- 在软件世界中，如果处于“未完成”状态，学生是可以临时保存作业，留着以后再完成的，所以“未完成”状态对于软件系统（环境）来说，是有意义的：可以保存入磁盘，可以再次从磁盘加载到内存。

而“已完成”状态的意义是：到达这个状态后，软件系统才会允许提交作业进入下一个状态，相当于软件代替了老师和家长的监督作用：学生没有完成作业就必须完成，不能提交。

“批改中”的作用和“未完成”类似，也是可以保存入磁盘，以后再次加载的。

依赖于软件系统的支持，学生还可以决定：

- 是否在完成作业后，在回过头来修改；
- 是否在提交作业后，在老师还没有批改的前提下，收回作业，做进一步的修改。

系统的状态转换

前面我们介绍的是某个对象（类）的微观状态转换，对于一个大系统来说，也是有状态转换的，比如AI教育系统中的训练子系统，如图 6.9.2 所示：

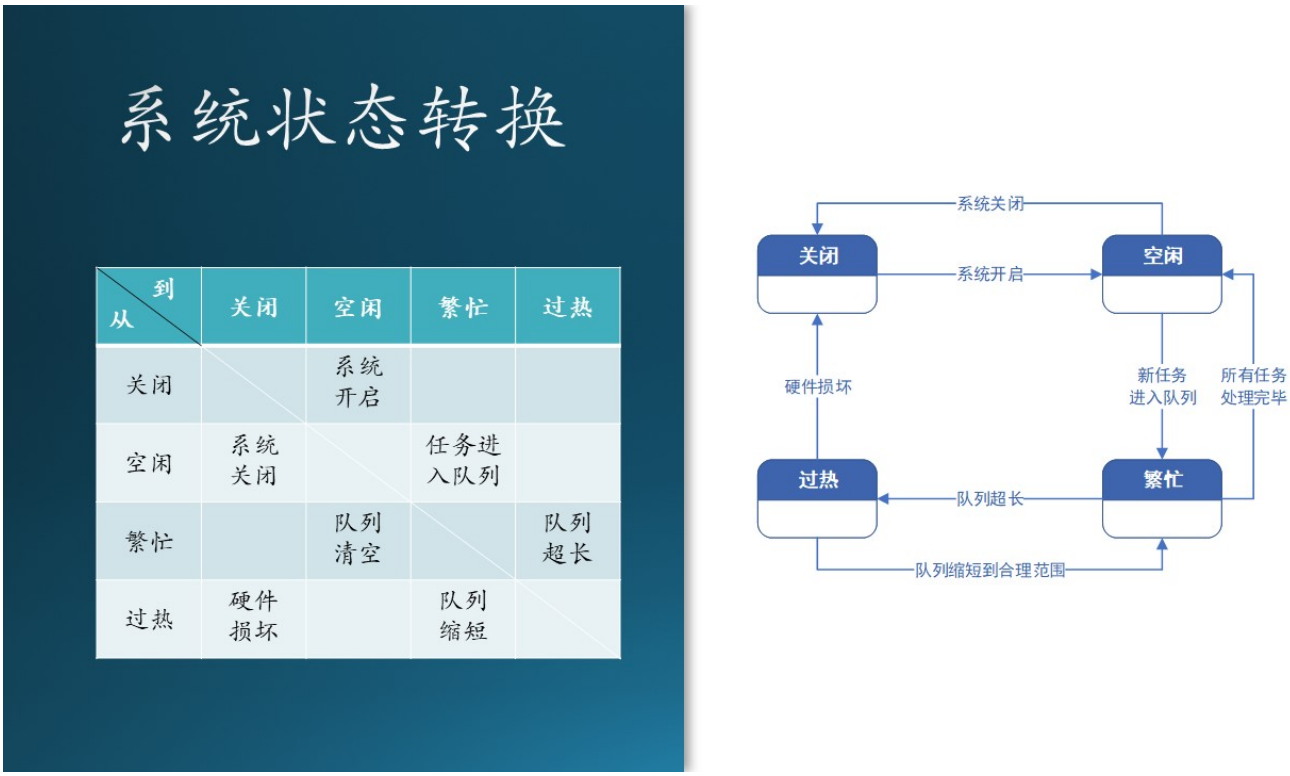


图 6.9.2 - 系统状态转换图

表 6.9.2 列出了状态转换表。

表 6.9.2 - 系统状态转换图表

	关闭	空闲	繁忙	过热
关闭	N/A	系统开启		
空闲	系统关闭	N/A	新任务进入队列	
繁忙		所有任务处理完毕	N/A	队列超长
过热	硬件损坏		队列缩短	N/A

转换过程解释：

1. 平时，训练子系统处于关闭状态，因为计划租用 Azure GPU 硬件，所以关闭时并不付费；
2. 管理员可以开启系统，达到空闲状态；也可以关闭系统，从空闲状态返回关闭状态；
3. 有新的任务进入队列后，系统进入繁忙状态；所有任务处理完毕后，系统从繁忙状态返回空闲状态；
4. 如果队列超长，系统会报警而进入“过热”状态，其实并非硬件温度过热，而是指系统的使用程度“过热”。这种报警的目的是让系统管理员知道情况，便于后期调整资源安排，比如忙时多加几台机器。当队列缩短后，系统返回“繁忙”状态。
5. 如果有硬件损坏或者掉电等情况，或者是管理员强制关闭系统，会返回到“关闭”状态，但是此事件极少发生。

这种系统级别的状态转换，是属于需求分析阶段还是属于系统设计阶段呢？有两种看法：

- 可以算作需求的一部分：需求方根据以往的经验，或者借鉴它人的经验，提出这个需求。当然，用户的需求可能是很简单的一句话：“当系统太忙的时候，我们需要得到通知。”

- 也可以算作设计的一部分：需求方没有想到这一点，而是需求分析人员或者系统设计人员从技术角度给与的建议，需要征求用户的同意后，成为技术需求的一部分。

笔者倾向于后一种看法，即它是系统设计的一部分，所以在需求分析阶段，可以不绘制这张图，而是留在系统设计阶段再拿出来。