我们应当怎样做需求分析

作者：范钢

幸福的软件项目都是一样的，不幸的软件项目却各有各的不幸；或者说，成功的软件项目都是一样的，失败的项目却各有各的问题。在众多问题中，需求分析无疑是最大的问题，因此真的有必要坐下来讨论一下，我们应该怎样做需求分析。

目　录

绪　论

又到新年了，日历又要从 2011 年翻到 2012 年了，这使我有太多的感慨，进而勾起了对太多往事的回忆。过去的 10 年，毫无疑问是中国软件业发展最快的 10 年。当我们刚刚毕业的时候，还在使用 VB、PB 开发一些简单的数据库应用，而现在却几乎看不到它们的踪影，换来的是诸如 J2EE 和.NET 这样的大型 web 应用。而这期间，RUP、XP、敏捷开发、持续集成 •••••• 一个接一个的新概念层出不穷，令人眼花缭乱。现在想来，恍如隔世。

但更令我印象深刻而难以忘怀的，是我亲自经历的、亲眼目睹的、道听途说的一个又一个的软件项目，它们有的获得了成功，但更多的是令人沮丧的失败。套用一下大文豪托尔斯泰体：幸福的家庭都是一样的，不幸的家庭却各有各的不幸；幸福的软件项目都是一样的，不幸的软件项目却各有各的不幸；或者说，成功的软件项目都是一样的，失败的项目却各有各的问题。我常常在想，我们的项目开发到底怎么了，进而把它们一个一个的剥开来深入分析，竟然触目惊心。它们有的是需求的问题，有的是客户关系的问题，还有设计的问题、技术的问题、时间管理的问题、人员培养的问题 •••••• 但归根到底更多的还是需求的问题。需求分析既是一份体力活儿，更是一份技术活儿，它既是人际交往的艺术，又是逻辑分析与严密思考的产物。正是我们在需求分析过程存在的巨大隐患，最终导致了那么多项目的失败。也许你认为我在危言耸听，好吧，我来举几个典型事例分析分析吧。

我的第一个故事来自大名鼎鼎的东软。我在 2005 年接一个项目的时候，听说这个项目之前是东软做的。当时东软在做这个项目的时候，整个过程经历了 10 多次结构性的大变更，局部性的调整更是不计其数。据说某天早上，客户对某个功能不满意，他们不得不对几百处程序进行修改。之后客户对修改的内容还是不满意，又不得不将几百处修改重新改回来。最后这个项目导致的结果是，整个这个项目组的所有成员都离开了东软，并似乎从此不愿涉足软件开发领域。多么惨痛的教训啊！我常常听到网友抱怨客户总是对需求改来改去，但客户对需求改来改去的真正原因是什么呢？当我们对客户的需求没有真正理解清楚时，我们做出来的东西客户必然不满意。客户只知道他不满意，但怎样才能使他满意呢？他不知道，于是就在一点儿一点儿试，于是这种反复变更就这样发生了。如果我们明白了这一点，深入地去理解客户的业务，进而想到客户的心坎儿上去，最后做出来的东西必然是客户满意的。记住，当客户提出业务变更的时候，我们一定不能被客户牵着走，客户说啥就是啥。我们要从业务角度深入的去分析，他为什么提出变更，提得合不合理，我有没有更合理的方案满足这个需求。当我们提出更加合理的方案时，客户是乐于接受的，变更也变得可控了。

第二个故事来自我自己的项目，一个早期的项目。在这个项目中，客户扔给了我们很多他们目前正在使用的统计报表，要我们按照报表的格式做出来。这些报表都是手工报表，许多格式既不规范，又很难于被计算机实现。这些报表令我耗费了不少脑细胞，直到最终项目失败都没法完成。这件事留给我的深刻教训是，不能客户怎么说软件就怎么做。客户提出的原始需求往往是不考虑技术实现、基于非计算机管理的操作模式提出来的。他们提出的很多需求常常比较理想而不切实际，毕竟人家是非技术的。但我们作为技术人员，需求分析必须实事求是的、基于技术可以实现的角度去考虑。那种“有条件要上，没有条件创造条件也要上”的鲁莽行事，结果必然是悲惨的。所以我们必须要基于技术实现去引导客户的需求。同时，计算机信息化管理就是一次改革，对以往手工管理模式的改革。如果我们上了信息化管理系统，采用的管理模式却依然是过去的手工模式，新系统的优势从何而来呢？因此，我们做需

求就应当首先理解现有的管理模式，然后站在信息化管理的角度去审视他们的管理模式是否合理，最后一步一步地去引导他们按照更加合理的方式去操作与管理。

2007 年，我参与了一个集团信息化建设的项目。这个项目中的客户是一个庞大的群体，他们分别扮演着各种角色。从机构层次划分，有集团领导、二级机构人员、三级机构人员；从职能角色划分，有高层领导、财务人员、生产管理员、采购人员、销售人员，等等。在这样一个复杂场景中，不同人员对这个项目的需求是各自不同的。非常遗憾的是，我们在进行需求分析的时候没有认真分析清楚所有类型人员的需求。在进行需求调研的时候，总是集团领导带领我们到基层单位，然后基层单位将各方面的人员叫来开大会。这样的大会，各类型的人员七嘴八舌各说各自的需求，还有很多基层人员在大会上因为羞涩根本就没有提出自己的需求。这样经过数次开会，需求调研就草草收场。我们拿着一个不充分的需求分析结果就开始项目开发，最终的结果可想而知。直到项目上线以后，我们才发现许多更加细节的业务需求都没能分析到，系统根本没法运行，不得不宣告失败。一个软件项目的需求调研首先必须要进行角色分析，然后对不同的角色分别进行调研。需求调研的最初需要召开项目动员大会，这是十分必要的。但真正要完成需求分析，应该是一个一个的小会，1~3 个业务专家，只讨论某个领域的业务需求，并且很多问题都不是能一蹴而就完成的，我们必须与专家建立联系，反复沟通后完成。需求分析必须遵从的是一定的科学方法，而不是盲目的大上快上。

我的最后一个故事可能典型到几乎每个人都曾经遇到过。我们的项目从需求分析到设计、开发、测试都十分顺利。但到了项目进行的后期，快到达最后期限时，我们将我们的开发成果提交给客户看，客户却对开发成果不满意，提出了一大堆修改，而且这些修改工作量还不小。怎么办呢？加班、赶工，测试时间被最大限度压缩。最后项目倒是如期上线了，但大家疲惫不堪，并且上线以后才发现许多的 BUG。需求分析不是一蹴而就的，它应当贯穿整个开发周期，不断的分析确认的过程。以上这个事例，如果我们提早将开发成果给客户看，提早解决问题，后面的情况就将不再发生。这就是敏捷开发倡导的需求反馈。敏捷开发认为，需求分析阶段不可能解决所有的需求问题，因此在设计、开发、测试，直到最终交付客户，这整个过程都应当不停地用开发的成果与客户交流，及时获得反馈。只有这样才能及时纠正需求理解的偏差，保证项目的成功。

以上的故事各有各自的不幸，各自都在不同的开发环节出现了问题。但经过深入的分析，各自的问题最终都归结为需求分析出现了问题。为了使我们今后的软件项目不会重蹈覆辙，似乎真的有必要讨论一下我们应该怎样做需求分析。

我们应当怎样做需求调研

很多需求分析的工作是从需求调研开始的，我们就从这里说起吧。需求调研是需求分析最重要的一环，也最集中地体现了需求分析的特点——既是一份体力活儿，更是一份技术活儿。它既要求我们具有一种理解能力、设计能力，更要求我们具有一种与人交往、沟通的能力。

初识

在一个阳光明媚的下午，项目经理带领着项目组成员，参加了客户组织的见面会，一个新的软件研发项目就这样开始了。双方在一种友好的气氛中进行，相互寒暄，介绍与会人员，拉拉家常。逐渐地，会议开始进入了正题。初次接触客户，对于项目团队意义重大。对方对你印象的好坏，今后如何与你交往，都在这个阶段被确定下来。然而，在客户至上的今天，与客户保持适当的谦卑是有必要的，但过于的谦卑却常常给项目日后的进程带来风险。为什么这么说呢？过于的谦卑，处处都是诺诺诺，客户说什么就是什么，就会使客户变得非常强势。这样的结果就是，客户提出了许多变态的、不太现实的、不合理的需求，而我们呢却是一味地服从，客户说什么就是什么。最后我们做得很累，结果却不能让客户满意。

正确的做法是，我们对客户提出的需求进行深入理解以后，运用我们专业知识，提出比客户的原始需求更加合理、可操作的解决方案，让客户感觉你说的正是他们想要的。如果能够这样，客户不仅能够欣然接收你提出的方案，而且会感觉你非常专业，你在客户心目中的形象也会无形中提高，使你有更多的机会提出有利于开发的可行方案，降低开发的风险。这毫无疑问会形成一个良性循环，但要做到这一点并不容易，毫无疑问，在与客户接触初期的表现起到了极其关键的作用。

人与人交往，往往在接触的初期就决定了相互的行为方式，与客户交往也是一样。起初的唯唯诺诺，客户说啥就是啥，必然造成客户不再关注你的意见，对你发号施令就可以了。相反，起初展现出一位技术专家的姿态，能大方而得体地提出自己的意见，会使客户重视你的意见，甚至主动征求你的意见。这一方面要求我们对自己要有足够的自信，另一方面也要有循循善诱的表达能力。如果我们做到了这些，就会客户心目中形成一种威信，使项目向着一种良性的方向前进。

同时，这样的会议又是一个项目启动会议。客户方领导要在会议上传达给与会代表一个清晰的信号，那就是与会代表今后要积极配合我们完成今后的工作。这时候，我们要弄清，客户方有哪些角色，谁是这些角色的需求提出者与决策者。这是什么意思呢？在软件项目中，特别是管理型软件项目中，客户都代表的是一个群体，而不是个人。他们代表的可能是一个单位、一个集团，甚至是一系列组织机构。在这样一个群体中，他们按照职能被划分成了不同的角色。拿一个单位来说，横向可能划分成不同的部门，财务部、销售部、采购部、生产部 …… 不同的部门，由于业务的不同，对软件的需求自然是不同的，因此我们在进行需求调研的时候，什么部门的需求就应当跟什么部门谈。同时，纵向又可以划分为多个层次，如高层领导、中层领导与基层人员，理解这些方面格外重要：

- 高层领导关心的是宏观的目标，因此软件研发目标、宏观统计报表、决策支持功能，都

应当与高层领导谈。他们关系的都是宏观的问题，因此不要与他们谈那些细枝末节；

- 中层领导关心的是具体的效益，即软件给各个部门信息化管理方面带来的效益，因此，中层领导是各项业务流程、功能模块的需求决策者。他们关心功能的定义、业务流转的衔接、查询报表的设计，但不太关心一些具体的操作，以及一些具体业务流程的细节；
- 基层人员是每一项业务流程的操作者，也是软件今后真正的使用者。他们是真正了解你所要开发的软件的业务需求的领域专家，是你进行需求调研的重点对象。但是，基层人员往往受到自身视野的局限，可能只清楚自己工作涉及的十分狭小的一个范围，因此我们需要努力寻找那些业务涉及面广，经验丰富，又有一定大局观的真正的专家。另外，他们就是软件今后真正的使用者，让他们参加，会让他们成为今后软件推行的忠实支持者，对其他操作人员的指导者，益处多多。而他们关心的则是每项操作的细节。

划分清楚角色，弄清楚每个角色的需求提出者与决策者，就是为了在今后的需求调研中找对正确的人，使今后的调研工作事半功倍。另外，如果客户方是一个集团、一个多组织机构的政府机关、事业单位，需求的个性化问题必须引起我们的足够重视。什么是个性化问题呢？比如同样一个业务操作，在同一级别的 A 单位是这样操作的，而在 B 单位却是那样操作的。需求的个性化往往会给今后的软件开发带来巨大挑战。因此，我们要在需求调研阶段降低软件的个性化需求。要解决这样的问题，首先应当从高层领导着手，提出规范化管理的口号。同时，在进行需求调研时，尽可能地召集各个单位的代表在一起开会讨论。同时，应当有高层领导，或者指定一个负责人，在出现分歧的时候最终拍板决策。这些都需要在项目启动的时候事先规划好。

最后，与客户方领导制订出软件目标，是相当重要但常常被我们忽视的一个步骤。软件信息化管理不是包治百病的神药。很多项目的失败都归因与项目目标不明确造成的项目范围的失控。因此，这时讨论项目目标，既重要又适时。

也许在此之前我们已经做足了功课，对业务需求进行了一番详细的整理，有了一大堆疑问急需解答。但是，在这时，不是解答具体问题的地方，这是我们常常会犯的一个毛病。在这样一个会议上，我们应当询问客户方领导对这个项目的期望，渴望达到的项目预期，而我们应当描述的，是对达到这些预期的整体解决方案，如此等等。

俗话说：万事开头难。如果你在项目开始的时候总感觉千头万绪不知如何着手，在这里我给大家的三点建议：1）树立良好的职业威信；2）进行详细角色分析，将与会各方代表对号入座；3）从宏观上制订目标与方案。随后的工作，就是与各方代码建立联系，逐一拜访他们，将需求调研工作一步一步进行下去。

拜访

项目组经过一番努力，获得了一些初步的成果。首先是给客户留下了一个良好的印象，这是一个开端，但要在他们心目中树立自己的职业威信还要看你今后的表现。同时，我们与客户一起为项目制订了短期与长期目标。不要小看了这些目标，它们就是我们的尚方宝剑。正是因为有了它，今后项目中的有关各方就应当协助实现这个目标。我们应当清晰地向客户表达这样一个意思，要完成这样的目标，不是某一方的努力，而是双方共同努力的结果。这也是客户方召开这样一个项目启动会议的重要意义。最后一个成果，也是最重要的成果，就是与各种角色、各个类型的客户建立了联系。下面，我们将一个一个去拜访他们，展开我们的需求调研。

与西方人不同，中国人做事往往比较重视感情，这是与中国数千年的文化分不开的。让我们来听听一位金牌销售员是怎么做生意的：“我跟客户头几次见面，绝对不提生意的事，玩，就是玩。吃饭啦，唱卡拉 OK 啦，打球啦 •••••• 先建立关系，关系好了再慢慢提生意的事儿。”这说得比较夸张，毕竟他是在做销售，但至少传达出一个概念，那就是做事先培养感情，感情培养起来才好慢慢做事，需求调研也是一样。

需求调研不是一蹴而就的事情，是一件持续数月甚至数年的工作（假如项目还有后期维护）。在这漫长的时间里，我们需要依靠客户这个群体的帮助，一步一步掌握真实可靠的业务需求。不仅如此，技术这东西总有不如意甚至实现不了的地方，我们需要客户的理解与包容，这都需要有良好的客户关系。按照现在的软件运作理念，软件项目已经不是一锤子的买卖，而是长期的、持续不断的提供服务。按照这样的理念，软件供应商与客户建立的是长期共赢的战略协作关系，这更需要我们与客户建立长期友好的关系。

尽管如此，我们也不能总是期望客户中的所有人都能与我们合作，很多项目都不可避免地存在阻碍项目开展的人。如很多 ERP 项目会损害采购和销售人员的利益，因为信息化的管理断了他们的财路；很多企业管理软件会遭到来自基层操作人员的抵制，因为它会给基层操作人员带来更多的工作量负担。有一次，我们给一个集团开发一套软件，当我们下到基层单位时，才发现，一些基层单位已经有了相应的管理软件。我们的软件成功上线，必然就意味着这些基层单位的管理软件寿终正寝，这必然影响到基层信息化管理专员的利益和政绩。

分析一个客户人群的关系，就是在分析这个人群中，谁有意愿支持我们，而谁却在自觉不自觉地阻碍我们。那些通过这个项目可以提高政绩，提高自身价值的人，都是我们可以争取的盟友。他们是我们最可以依赖的人，我们一定要与他们站在一起，荣辱与共，建立战略合作伙伴关系。

另一种人，即使软件获得了成功，也与他没有太多关系，但你与他相处得好，却可以给予你巨大的帮助，这种人是我们需要拼命争取的人。所谓领域专家，他可以给你多讲点儿，但随便打发你，对他也没太大影响。报着谦虚谨慎、相互尊重的态度，大方地与他们交往。当他们帮助我们以后，真诚地予以感谢。这是我总结出来的，与他们交往的准则。

最后，就是那些对我们怀有敌意的人。尽管有敌意，但我们能够坦荡的，敞开心扉的与他们交往。虽然不能奢望太多，但拿出诚意去争取他们，也还是有机会化干戈为玉帛、化敌为友。

如果能够那样，那是再好不过了。

经过一番交往，我们将逐渐在客户中结识一批可以帮助我们的人。今后一段日子里，我们将依靠他们去学习和认识业务知识，收集业务需求，为日后的软件研发提供素材。

研讨会

经过一番努力，我们终于在客户中找到了一批人，可以解答困扰我们多时的业务问题了，真是不容易呀。但是，如何以合适的时间、合适的地点、通过合适的形式与客户研讨业务需求，是摆在项目经理面前的一道难题。在我所经历的项目中，业务研讨会没有一个是相同的。

我曾经做过一个政府机关的项目，在这个项目中，从总局到省、地市、区县，形成了一个多组织机构的管理系统。虽然全国管理流程大体相同，但各地因各地实际情况的不同、领导管理思路和政策理解的不同，管理模式在许多细节上存在着差异，也就是说，这个项目存在着需求个性化的问题。在项目进行之初，客户方领导提前意识到这方面的问题，因此在组织需求研讨时，分别从各个省市抽调业务人员，集中在一起进行研讨。同时，在研讨时，根据与会人员的业务特点，将他们分成若干个业务组，分别对某个相对独立的业务模块的需求进行研讨。采用这样的组织形式，各地的业务差异在会上都会被提出来。一些地区不合理的管理模式，一经提出，就会得到其它地区业务人员的纠正，进而避免了不合理需求的提出。当然业务人员之间也会出现意见分歧。在会议启动之时，高层领导就明确提出了必须形成全国统一版本。因此，一旦出现分歧时，业务人员就会通过激烈辩论、各抒己见，进而形成统一意见。如果分歧双方谁都说服不了谁，业务组指定的组长则拍板采用哪个方案。如果他不能做出决定，就立即反映到总局领导那里当场做出决定。采用这种集中式的研讨，可以使问题的处理变得高效而及时。当然，也会因地区化差异而出现多个方案，每个方案都是合理的，我们必须在软件中分别对其进行处理的情况。出现这种情况时，至少我们很容易理清楚有几种情况，有没有可以合并的地方，使得差异最小化，最终在软件维护中体现出来，让客户自己去选择自己的管理模式。

另外，将业务人员划分为多个业务组也是一项比较成功的经验。由于业务人员自身的局限，不可能对所有业务领域的细节全面掌握，往往总是有自己熟悉的部分，也有自己不熟悉的部分。划分业务组，可以让业务人员分别在自己最熟悉的业务范围内参与讨论，可以有效提高业务讨论的质量。同时，一个管理系统涉及的业务是复杂而系统的，如果划分成多个模块并行地进行业务讨论，也可以大大提高业务研讨的工作效率。这个项目采用这种方式，使这个项目在运行数年后依然能保持统一的版本，而不至于形成一个一个的地方版本。统一的版本使得软件的升级维护成本大大降低，使项目进入良性的进化、完善的循环中。

以上讲的是一种集中式的业务研讨形式。采用这是形式固然好处多多，但并非所有软件项目都能够采用这种模式。我参与过的另一个项目就没有如此幸运了。在这个项目中，虽然也是多组织机构管理系统，但总公司对各分子公司的管理是松散的，所以很难组织各地的业务代表集中在一起讨论，甚至不能要求各分子公司采用统一的管理模式。企业信息化的目的就是要建立统一的、规范化的管理形式，它本身就是一场企业管理的变革。我们的软件，如果不能规范各分支机构的管理，抑制个性化差异，而是照猫画虎地一家一家为分子公司做软件，不仅我们的成本是巨大的，客户的信息化管理效果也不能发挥出来，而且为日后的运行维护

带来巨大的隐患。毫无疑问，它是我们做管理软件的一个雷区，我们必须小心应对。

起先，总公司领导带着我们一家一家地去分子公司开需求研讨会。每个需求研讨会，我们都要着力注意各个单位管理模式的差异。当业务代表在描述自己业务流程的时候，我们常常提示业务代表，×××公司是这样管理的。这时候，业务代表会思考，采用×××公司的管理模式是否会更好，或者采用×××公司的管理模式行不行。如果他提出×××公司的管理模式可能会出现什么什么问题时，我们也会着力记录下来，下次再和×××公司讨论，他们是不是会出现这些问题。

采用这种分散式的业务研讨形式，让我们作为外人来规范客户的管理模式，常常会有这样那样的不便，但这也是我们可能面对得最多的需求研讨形式。在这样的形式中，寻找一个典型范例也许可以算是一种最佳实践。当我们面对管理松散的多组织机构时，寻找一个管理规范、对我们的支持度高的分支机构，首先将他们的信息化系统建立起来，产生预期的效益，这就树立了一个范例。它的成功就会为其它分支机构带来一种精神动力和成功案例，照着做肯定不会错。这样就可以更容易地说服其它分支机构，摒弃现有的管理模式而朝着规范化管理迈进。

业务研讨形式比较容易出现的另一个问题，就是将各个方面的业务代表拉过来开大会。在大会上，你说你的，我说我的，杂乱无章，一些重要的需求被不经意地漏掉。遇上这样的情形，项目经理应当有清醒的认识，我们需要再下来开小会。销售部门的需求跟销售部门谈，采购部门的需求跟采购部门谈 •••••• 既然是小会，每次谈的时候人不在多，在精，参会的业务人员对自己的业务了解精细而全面。这样的会议，通常有一至三个业务人员，和一个负责人（负责拍板）参加。会议之后，我们最好询问与会人员的联系方式，便于日后建立长期的联系，毕竟业务需求不是一蹴而就的事情。同时，如果我们今后采用的是迭代式开发，他们也就成为了我们业务验证的客户代表。

业务研讨会是重要的，但同时又是灵活的，没有一个定式，甚至有时都不能称之为会议。项目经理需要根据实际情况，合理地与客户组织研讨会。但不论怎样组织，必须注意两点：有效抑制个性化差异、分模块组织专项研讨会。

业务研讨

前面我们探讨了业务研讨会应当怎样组织，下面我们再具体讨论一下我们应当怎样与客户讨论业务需求。如果说组织业务研讨会是项目经理的功底，那么讨论业务需求就是需求分析人员的功底。

以往我们常常认为，需求分析是一件最简单的事情。客户说他们需要做一个什么软件，有些什么功能，我们照着做就可以了，所谓的需求分析员就是需求的记录员。我要说，这是一个极大的错误，许多失败的软件项目，或者说软件项目中的需求问题，大多都源于此。经过人们多年的研究发现，在需求分析过程中，客户存在的最大问题就是提不出正确的需求，这表现为几种形式：

1.由于对软件不了解，客户提不出需求，不知道软件最终会做成什么样子。这类客户在需求讨论过程中，往往只能描述目前自己手工管理的方式是怎样的，不知道计算机会怎样管理。

2.能提出一些业务需求，但当软件做出来摆在自己面前时，需求就变了。这类客户，他们能熟练使用电脑，对信息化管理是清楚的。他们提出的业务需求从整体上应当是八九不离十的。但是，由于没有实物，在软件中的一些具体操作并没有完全想清楚。因此，当软件真正做出来摆在自己面前时，甚至经过一系列流程操作以后，会对一些操作提出变更需求。他们正如那句经典的话说的：“I have changed when it saw it.”

3.能非常详细地提出业务需求，甚至有时候该怎么做的提出来了。这类客户，参与过很多软件信息化建设，甚至有些还是软件开发的半专业人士。但是他们提出的业务需求过于具体，甚至怎样实现都说出来了，但这些有时候不是最佳设计方案、可能在技术上难于实现，甚至有些就是过于理想化而不可实现。

因此，我在进行需求研讨的时候，首先跟客户探讨的不是软件功能，而是客户现有的业务知识，用专业的话叫“业务领域分析”。客户现有的业务流程是什么样的，都有些什么操作？客户在业务中都有些什么事物，什么专用名词，都是怎样定义的，相互之间的关系是什么？客户在每一项操作中的目的是什么，为什么要这样做，他们制作的手工报表都说明了什么问题？后面我会更加详细地描述怎么进行业务领域分析。

在认识了客户的业务领域之后，我们才能去分析他们提出的所有原始需求。他们为什么要提出这项需求，提这项需求的目的是什么？只有经过这样的分析，我们才能深刻地理解需求，进而运用我们的专业知识，提出更加合理的技术方案。但非常遗憾，我们在需求分析中常常不是这样做的，甚至当软件都开发出来了，需求分析人员都说不出客户为什么要提出这个需求，更谈不上了解业务操作流程。一句经典的话是：“客户让我们这样做的。”

总之，我们做需求分析，眼界不能仅仅停留在软件本身，应当更开阔一些，应当扩展到跟这个业务有关的那些领域知识中。

当然，另一个极端就是为了开发软件，无限地扩大学习领域知识的范围。为了开发财务软件去考会计师，为了开发税务软件去学习税法等等。开发软件不是让我们成为这个领域的专家。

我们学习领域知识是为了更好地理解和开发软件，是学习与这个软件有关的领域知识，而不是成为一个专家。

在客户提出的所有原始需求中那些与业务实现有关的需求都是无效的需求，它们仅仅只能作为我们的一个参考。什么是与业务实现有关的需求呢？比如要求做成什么界面，数据要求怎样处理，等等。为什么是无效的呢？因为客户毕竟是非专业，我们应当有这种自信，在理解客户真实意图以后，能够提出比客户更优的解决方案。

还有一些是技术难于实现或者根本就无法实现的需求，我们应当耐心地说服和引导客户，并给他提出一个更加合理的方案。注意最后一句话：“给他提出一个更加合理的方案”。苍白的拒绝客户往往会让客户产生抵触情绪，但当我们提出一个更加合理的方案时，客户往往会欣然接受，当然这是在我们对客户提出的业务需求的真实意图进行深入分析之后。认识到这一点非常重要，为了更加清楚地说明这一点，我举一个我的例子吧。有一次我给客户做一个价格管理系统时，客户提出要做一个动态报表的需求。这个动态报表要求能让客户从无到有，完全自由的定制自己的报表。毫无疑问，这是一个典型的不切实际的业务需求。接到这个需求以后，我们将它作为一个疑问，在整个需求调研过程中着力进行了考察，明白了客户为什么提出这样的需求。当客户在向他们的客户报价时，他们的客户在各个方面都要求他们报出价格细目，而且不同的客户要求他们报的价格细目格式还不一样。但经过仔细分析，发现他们面对的客户就是固定的几家，而这几家的要求的报表虽然格式不尽相同，但其数据项大体是相同的。最后，我们给客户提出两个方案，一个是按照客户所说的动态报表，但要求客户在制作报表时必须能够详细设计报表中数据项的来源、项目的类型，以及绘制报表格式，让他们意识到，即使做出来，作为非专业的他们也是很难自己完成的。同时，我们提出另一个方案：我们为客户准备好他们需要填写的各种客户报表所需的所有数据项，让他们自由删减。同时，为他们的不同客户提供各自相应的报表模板，这些模板可以在少量的范围内进行修改，以此满足他们的客户的不同需要。当客户拿到这样的方案，既能满足他们自己的需要，还操作简便、易懂、不费事，当然就欣然接收啦。

因此，需求分析不是一种简单的你说我记的收集活动，而是在大量业务分析与技术可行性分析基础上的分析活动。只有建立在这种分析基础上的软件研发，才能保证需求的正确与变更的可控。

迭代

前面我一直在反复强调这样一个观点，需求分析不是一蹴而就的，是一个反复迭代的过程。它将从第一次需求分析开始，一直持续到整个项目生命周期。为什么这样说呢？让我们一起来分析分析。

在第一次的需求分析阶段，我们在一段时期内需要与客户进行反复地讨论，这个过程往往是这样一个反复循环的过程：需求捕获->需求整理->需求验证->再需求捕获 ••••••

需求捕获，就是我们与客户在一起开研讨会，讨论需求的活动。客户可能会描述他们的业务流程，这时我们在纸上绘制简单的流程草图，及时地记录下来；客户在描述业务的同时，可能会反复提到一些业务名词，详细询问这些名词的含义，以及它们与其它名词的关系，用类图或者对象图绘制简单的草图；客户在描述业务的同时，还会提出今后的软件希望实现的功能，如能够展示某个报表、能够导出文件，以需求列表的形式记录下来。一个功能，在需求列表中会有多个需求，而每个需求应当能够用 1、2 句话，在 20 个字以内就可以描述清楚。需求列表是客户提出的最最原始的需求，他不掺杂任何分析设计，是我们的每项功能必须实现的内容。需求列表是需求验证以及日后的用户验收测试的依据，不论我们今后如何分析和设计这些功能，都要能如实地实现这个列表中提出的需求。（需求列表应当如何编写，将在后面的章节详细描述。）

需求整理，就是在需求研讨会后，需求分析人员对研讨内容的分析和整理的过程。首先，需求分析人员应当通过用例模型，划分整个系统的功能模块，以及各个模块的业务流程。用例模型分析是一个由粗到细的过程，这样一个过程也是符合人类认识世界的思维习惯的一个过程。最先，我们应当对整个系统绘制用例图，设计用例场景，并依次对这些用例进行用例描述、流程分析、角色分析等分析过程。当然，在整体用例分析的同时，我们还应当进行一个整体的角色分析，绘制一个角色分析图，进行一个流程分析，绘制一个流程分析图（可以是传统的流程图、UML 中的行动图，甚至一个简单的示意图，等等）。

然后，我们再在整体用例图的基础上，依次对每个用例绘制用例图。每个用例图中，会更细致地划分出多个用例，并依次进行用例描述、流程分析、角色分析等分析工作。如此这般地不断细化，直到我们认为需求已经描述清楚为止。

在一个系统中，用例需要细化几次，是由这个用例的业务复杂程度决定的。对于一个简单的用例，只需要细化一次就够了；而对于比较复杂的用例，则需要细化 2~3 次，甚至更多。

用例分析的过程，之所以称之为分析，它掺入了很多需求分析人员对业务的理解与设计：模块如何划分、流程如何设计、业务如何转换，等等。用例分析，还需要让需求分析员与架构师、设计师等技术人员共同协作来完成，因为用例分析还包含对业务需求的技术可行性分析。只有一份可行的需求分析，才能为后续的设计开发扫清障碍，有效降低项目风险。最后，需求分析员应当将需求列表中的内容，逐一地与用例进行核对，以避免分析人员忽略用户的某项业务需求。（后面将详细描述用例模型的搭建过程。）

在用例分析的同时，需求分析人员还需要对业务中的相关事物，制作领域模型。领域模型，

是对用户业务领域中相关事物、相互关系、相互行为操作的描述，它是以对象图和类图的形式表达的。需求人员对领域模型的分析，对业务理解的深度，对日后软件的设计，以及软件的功能扩展、升级演化，都起到了至关重要的作用。（后面将更加详细地讲述领域模型。）

最后，当我们完成了一系列的分析整理并形成文档以后，应当对及时地与客户进行反馈，确认我们的理解是否正确，也就是需求验证工作。需求验证工作应当贯穿整个研发周期，并且在不同时期表现出不同的形式。首先，在需求分析阶段，需求验证工作表现为对需求理解是否正确的信息反馈。需求分析人员与客户再次坐在一起，一项一项描述我们对需求的整理和理解，客户则时不时地对一些问题进行纠正，或者更加深入地加以描述。我们则认真地记录，回来整理，并等待下一次的验证。在需求分析后期，我们还可以制作一些简单的原型，更加形象地描述我们对需求的理解，会使我们与客户的沟通更加顺畅。随后的设计开发阶段，我们则应当以迭代开发的形式进行。每开发完一个迭代周期，将开发的成果与客户反馈。这样做的结果是，客户可以及时地提出我们对需求理解的偏差，或者及时提出对我们设计不满意的地方，使我们存在的问题得到及时地发现与解决。问题及时的解决，使我们修复问题的代价得以降至最小。之后，当开发进入到验收测试阶段，我们则是与客户一道，一项一项地验证我们的软件是否满足需求列表中要求的业务需求。最后，当软件迎来下一次升级开发时，我们将开启另一次轮回。

因此，需求分析就是按照这样的过程，每次多理解一些，再多理解一些，更多理解一些，逐渐深入的过程。每深入一步，我们的软件就更接近客户的满意。

需求捕获

前面我们讨论了，需求分析工作是一个迭代的过程：需求捕获->需求整理->需求验证->再需求捕获 •••••• 需求捕获是这个迭代过程的开始，也是整个需求分析工作中最重要的部分。没有捕获哪来后面的整理与验证工作？但是，非常遗憾，按照我以往的经验，需求捕获是我们最薄弱的环节。前面我提到的许许多多项目开发的问题都可以归结为需求分析的问题，而许许多多需求分析的问题又都可以归结为需求捕获不完整的问题。需求捕获是整个需求分析工作中最难把握的一个部分，它不仅仅是一个技术的问题，还涉及到人际交往、沟通、知识理解，以及心理学等一系列问题。但更让我感到遗憾的是，在我读过的许许多多关于需求分析的书籍中，讨论需求分析与建模的书很多，但讨论需求捕获的书籍却寥寥无几。确实，要讨论这部分内容，真的已经远远超出了软件开发这个知识领域。

那么，在软件需求捕获过程中，最根本、最容易犯错的问题是什么呢？我认为是一个态度的问题，是采用主动态度去捕获需求，还是采用被动的态度去捕获需求。如果需求分析人员总是诺诺诺，客户说什么，我们就记什么。客户处于非常强势的地位，给我们提出了非常多变态、技术难于实现的需求，而我们的需求分析人员却成为记录员，埋头记录客户说的每一句话，不加分析地就直接扔给了开发人员。这就是采用被动的态度去捕获业务需求的方式。毫无疑问，这样的需求分析必然将给项目开发的后期带来巨大的风险。

为什么会出现这样的情况呢？经过深入分析我们会发现，从客户嘴中说出来的需求，只是整个软件需求中的冰山一角，还有两类需求需要我们自己去挖掘：客户嘴中没有说出来的需求，和客户压根儿就没有想到的需求。

什么是客户嘴中没有说出来的需求，并不是客户故意卖弄官子不愿说出来，而是在客户所在业务领域已经约定俗称，在他们看来已经是天经地义，根本就不用说出来的业务规则。然而，作为刚刚涉足该领域的需求人员，他们是不了解这些规则的。如果采用被动的方式去仅仅记录客户说出来的需求，毫无疑问会遗失这部分需求，这就是为什么直到项目后期，软件被研发出来即将交付使用，客户才提出说这不是我想要的软件，并提出大量变更需求的原因。这时，我们常常问客户，你们为什么不早说呢？而客户却十分委屈，这么简单的道理还需要我说出来吗？

举例说明吧：在我从事的税务行业中，对纳税人征收的税种包括增值税、企业所得税。增值税通常是按月征收的，而企业所得税是按季或者按年征收的。就拿增值税来说吧，税款所属期是开票日期的上个月，为什么呢？纳税人往往是在上个月产生销售收入，然后在下个月完成申报和缴纳税款。这些知识对于税务人员来说是太基本的常识了，所以在他们看来就是天经地义而不需要说出来的业务规则。但作为软件开发人员的我们却常常因为不知道而将业务弄错。

如何破解这样的问题呢？那就是要求我们在需求分析的整个过程，不断进行业务领域知识的学习。在我做需求访谈的初期，我往往不是跟客户谈需求，而是先跟客户谈业务。你们是怎样操作的？都经过些什么流程？谁来完成这些操作的？为什么这样操作？注意，在所有这些问题中，最后一个问题是最重要的。客户业务领域中的所有操作、所有流程都是有它存在的意义的，它体现了其内部的原因与作用。多问为什么，可以让我们深入地理解这些领域知识，

站在客户的视角去思考问题，进而深入地理解客户为什么要提出他们的那些业务需求。当一个需求分析员能达到这样的水平，客户嘴中没有说出来的需求就会被源源不断地被发掘出来，最终做出来的需求分析才是完整的、准确的。

另一种就是客户压根儿没有想到的需求。也许你会提出这样的疑问，客户压根儿没有想到的需求我们还提出来做什么？这种压根儿没有想到的，实际是在业务需求阶段压根儿没有想到的，并不代表最终都没有想到。很多开发人员总在埋怨，说客户需求总是在软件项目的后期改来改去，为什么？客户并不是软件研发领域的专业人员。在业务需求阶段，由于没有可以展示和操作的实物，客户总是在空对空的凭空想象今后的软件应当做成什么样子。这就注定了客户会有很多自己压根儿没有想到的需求。那么为什么他们会在软件研发的后期提出来呢？因为软件研发的后期，客户能拿到那些研发成果的实物，去操作，可以看到。这时候，很多他们起初没有想到的需求就会源源不断地被提出来。但这时候，我们作为研发人员会很伤，我们付出的代价会很大。所以，以被动的态度去完成需求分析工作，必然会给项目研发带来巨大的风险。

如何解决这样的问题呢？首先，在需求分析阶段，虽然客户压根儿没有想到，但需求分析人员是软件研发领域的专业人员，他们应当在深入理解业务领域与需求的基础上，通过分析提前发现这些需求。作为需求分析人员，他们应当站在客户的角度去思考，我们的软件应当设计成什么样子，每个需求的真实意图是什么。站在这个基础上，再运用专业知识去整理、分析与设计。我前面谈到，客户描述的最原始的需求是编写在需求列表中的，而经过需求分析人员的整理、分析与设计，经过用例分析、领域建模，最终形成产品需求说明书（或称为产品规格说明书）。从需求列表到产品需求说明书，这之间要经过一段长长的路，这段路就是我们的分析与设计，而不是简单的记录与编写文档。只有经过这样的过程，最后得到的才是高质量的需求分析，才能有效地指导软件研发，避免项目的风险。所以说，好的需求分析人员就是软件项目的司命，掌握着项目的生死。

我们再换一个角度来分析，客户之所以提不出需求，关键就在于他们没有可以展示和操作的实物，总是在空对空的凭空想象今后的软件应当做成什么样子。我们能否改变这样一种现状呢？于是，迭代式的需求分析与开发就出现了。我们先用最短的时间先做一个可以展示和操作的原型给客户看，让客户提一些意见。然后我们再在这个原型的基础上再多做一些，再给客户看。我们就这样一步一步推进，直到最终项目研发结束。采用这样的方式，最适合那些客户在项目初期提不出什么需求，也没用合适的参照物来进行需求分析的软件项目，特别是那些数据分析与决策类的软件项目。

接下来，我们再回到那些从客户嘴里说出的需求。在需求分析人员中，比较普遍的一个看法就是，只要是从客户嘴里说出来的，就一定是对的，我们必须照着做的，这种看法是不正确的。因为客户在软件开发方面是非专业的，所以他们在提出需求的时候往往会考虑不够周全。有一次，客户在提出来一系列业务操作以后，最后提出了一个统计报表的功能。这个统计报表是从前面这一系统操作数据中统计出来的，因此我们就对这些业务操作及其结果数据进行了一个详细的分析，最后发现根据这些数据统计出来的数据存在很多的问题，甚至可能出现相互矛盾的地方。随后我们与客户就这些问题进行了深入地探讨，最终客户不得不承认，他当初在设计这个报表的时候考虑不周全。在提出问题的同时，我们又提出了我们的解决方案，这是非常关键的。当我们提出我们的合理化建议以后，客户欣然接受了。同时，客户对我们这种非常专业的分析与处理过程大加赞赏，无形中也提高了我们在客户心目中的威望。

不仅如此，客户作为一个群体，客户与客户之前对同一问题也可能存在不同的看法，这特别突出地体现在那些多组织机构的管理系统中。因此，对于一些客户非正式的场合提出的需求我们要仔细甄别。一个比较可行的方法就是，先在一些非正式的场合单独跟客户聊，产生第一手资料，最后将这些需求在比较正式的场合，如各部门参加的业务讨论会、有用户代表参加的需求评审会、需求定稿签字确认会等等，以比较正式的形式讨论和确定下来。

最后，我不得不说，企业信息化管理实质就是一次改革，是企业摒弃手工操作，向信息化建设迈进的一次改革。既然是改革，就必须要改变过去不合理的管理流程，向更加合理和高效的管理流程迈进。因此，我们的需求捕获最初是源于企业现有的操作流程，但当我们深入理解了客户现有的操作流程以后，应当有意识地发现那些不合理的部分，并最终提出更加合理、更适于信息化管理的流程。如果需求人员能上到这样一个高度，我们的需求分析就进入了一个更加崭新的层面（关于需求分析中的流程分析，我们还会在后面详细探讨）。

我们应当怎样做需求分析

在我们进行一系列需求调研工作的同时，我们的需求分析工作也开始启动了。需求调研与需求分析工作应当是相辅相伴共同进行的。每次参加完需求调研回到公司，我们就应当对需求调研的成果进行一次需求分析。当下一次开始进行需求调研时，我们应当首先将上次需求分析的结果与客户进行确认，同时对需求分析中提出的疑问交给客户予以解答。这就是一个需求捕获->需求整理->需求验证->再需求捕获的过程。

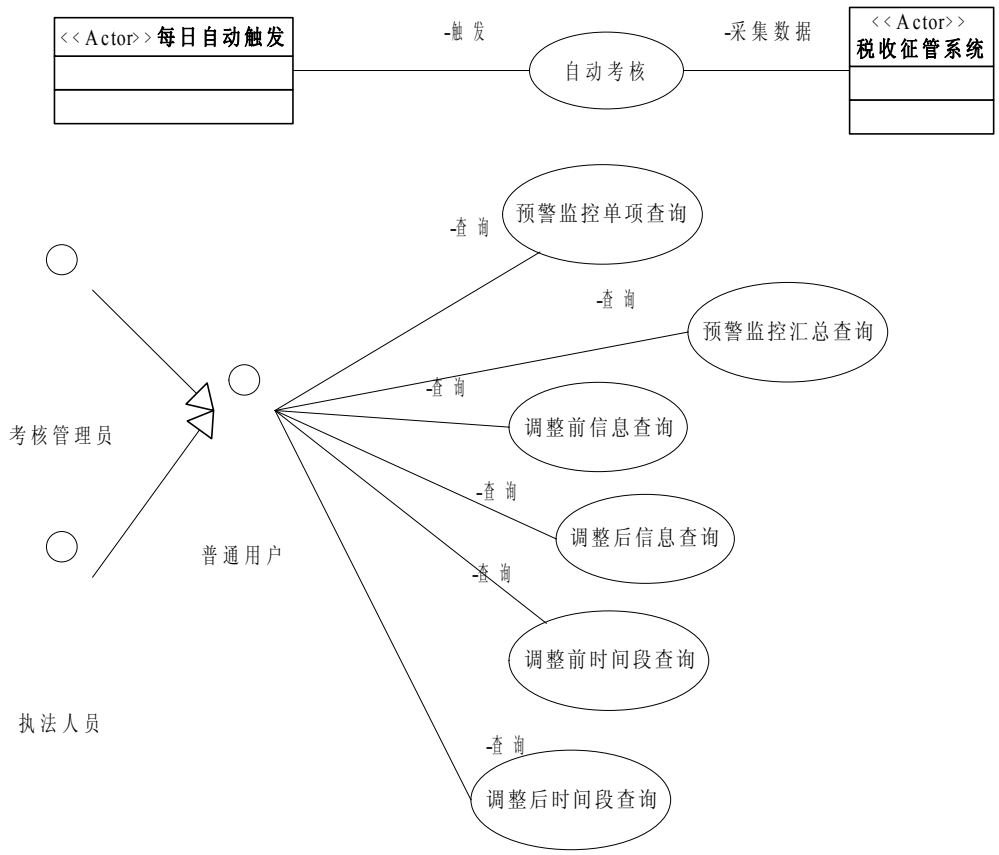
但是，当我们经过一番忙碌，将需求中的第一手资料从调研现场捕获回来以后，我们应当怎样进行分析呢？不少团队对此都比较迷茫，没有一个统一和有效的方法，往往采用想到哪里做到哪里的方式。一些问题想到了就做了，没有想到则忽略掉了。实际上，需求分析不应当是太公钓鱼，而应当是拉网排查。任何一个疏忽都可能对项目研发带来风险。因此，我们应当采用一套成熟而完整的分析方法，稳步而有序地完成这部分工作。不同类型的软件项目其分析方法可能存在差异，但一般来说，信息化管理类软件项目通常从这几个方面着手分析：功能角色分析、业务流程分析与业务领域分析。

功能角色分析与用例图

需求分析不是一项一蹴而就就可以完成的工作，它需要一个长期的过程，而这个过程是一个由粗到细的过程，它体现了人类认识事物的客观规律。在需求分析的初期，我们对需求的认识往往是整体的、宏观的，随着分析工作的逐渐深入，一步步细化。按照这个思路，我们对需求的分析，首先应当从功能角色分析开始。所谓功能角色分析，就是从一个外部用户的视角分析整个软件系统能够提供的功能，以及这些功能到底是提供给哪些角色使用。

对一个系统进行功能和角色方面的梳理和分析，可以采用的比较主流的方法之一就是绘制用例图。用例图是 UML 的 4+1 视图中的一种，准确地说就是那个“+1”。用例图是贯穿整个面向对象分析/设计（OOA/D）的核心视图，它描述的是系统到底为用户提供了哪些功能，以及到底是哪些用户在使用这些功能，是沟通用户与技术人员的桥梁。运用用例视图对业务需求进行分析、抽象、整理、提炼，进而形成抽象模型的过程称之为用例建模，而这个模型就是用例模型。

一般地，在一个用例图中通常有三种元素：参与者（Actor）、用例（Use Case）与系统边界（Boundary）。用例描述的是系统为用户提供的功能，也就是系统能为用户做什么，通常被绘制成一个椭圆；参与者，我认为称为角色更加合适，也就是系统为哪些类型的用户提供服务，他们都各自承担哪些不同的职责，通常被绘制成一个小人儿；最后是系统边界，也就是系统是对现实世界哪个范围的内容进行的模拟，它涉及到软件设计的工作范围与工作量，通常被绘制成一个方框。但是，通常情况下系统边界只是一个概念而不用真正绘制出来，因为被绘制成用例的必然是系统内部的功能，被绘制成参与者的必然是系统外部事物。从这个意义上讲，用例图中的参与者不仅包括人，还包括那些外部系统和自动触发器。根据这样一个思路，我以往常常将外部系统和自动触发器绘制成一个小人，这常常令客户感到困惑。随后我改变了思路，将外部系统和自动触发器绘制成另一种表达形式——类元符号表示法，并在构造型上标注为 Actor。



上图是一个考核系统中一个子模块的用例图。图中的用例就是这个系统提供给用户的各项功能。注意，这里仅仅是在罗列功能而不表示它们之间诸如流程调用等相互关系，这是一些初学者常常犯的毛病。参与者与用例通过实线关联起来，代表的是一种使用关系。箭头代表的是一种导航，即动作施与的方向。在这个用例图中，普通用户执行查询操作，查询系统提供的“预警监控单项查询”、“预警监控汇总查询”等查询报表；每日自动触发器触发自动考核功能，自动考核功能从“税收征管系统”这样一个外部系统中采集数据。

图中考核管理员和执法人员代表的是两个完全不同的角色，但他们在这个图中体现的是一些共有的特性，即对这堆报表的查询，因此被绘制成继承自普通用户。继承是参与者间唯一的关系，代表继承者拥有被继承者所有的功能与权限。除了参与者以外，用例与用例直接也存在着一些类型的关系，这我们在后面详细讲述。

在绘制用例图时一个值得思考的细节是，用例是怎样通过分析获得的。这个问题，在一些客户对信息化管理比较有经验的项目中不存在问题，因为在客户提供给我们的需求文档中就清晰地划分出了一项一项的功能。这些功能可能会在日后的需求分析工作中有所调整，但它从整体上形成了一个雏形，成为我们进行用例分析进而形成用例的依据。

但当我们面对的是一些对信息化管理没有经验的客户，情况就有些不妙了。在这种情况下，通常客户只能给我们一些管理目标、基本想法，其它的调研工作就需要我们自己去做了。这时，我给大家的建议是，首先从组织机构上划分清楚系统涉及哪些部门、哪些科室，然后在这个基础上划分出来这些部门这各个科室的人员都扮演哪些不同职能的角色，以及完成哪些业务操作。系统中的一个功能，在一般情况下是组织机构中某个（或多个）角色，为该机构

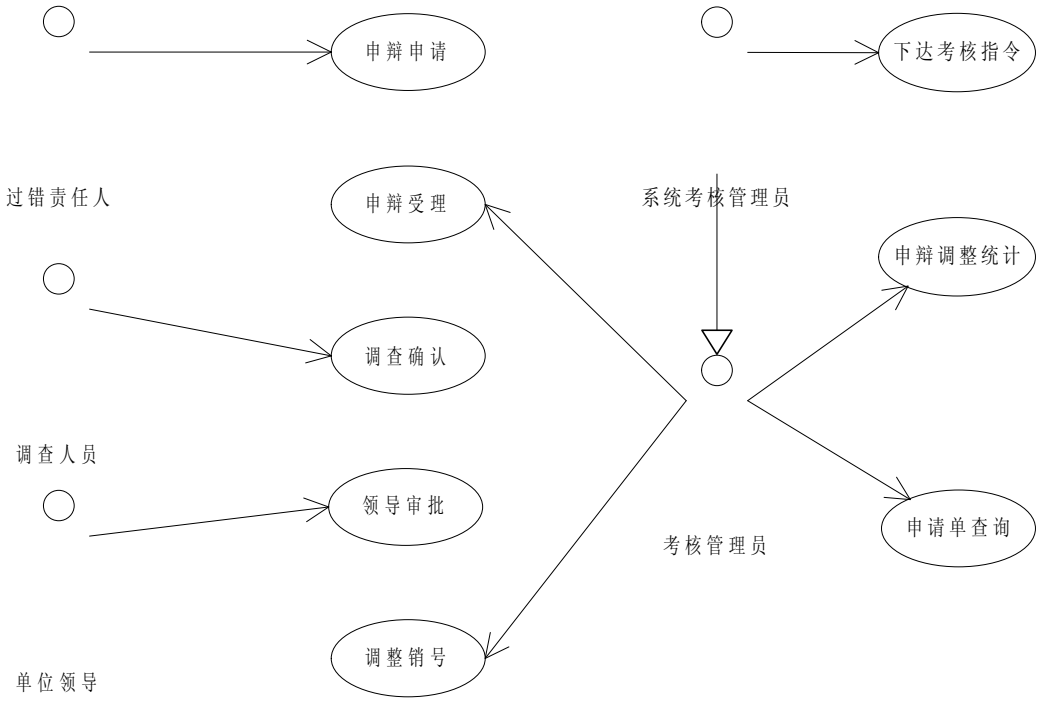
某项业务流程完成的某个操作，并且这个操作应当有某个确定的结果（即产出物）。而这个功能就是我们需要提取出来的用例。虽然功能角色分析在整个需求分析过程中可能会随着认识的深入而不断调整，但分析过程大体是这样进行的。

有人说，我们绘制的用例图拿给客户看不懂。这样一个清晰明了的用例图，辅之以我们对图形的描述，客户怎么会看不懂呢？关键问题在于，我们没有将用例图的精髓弄明白，再加上出现一些常见问题，使得用例图画得不伦不类，客户当然就看不明白了。现在我们看看用例绘制都有些什么常见问题。

1. 没有正确理解用例图的视角。前面我反复强调了，用例图的视角是用户，也就是说，站在用户的角度来观察的我们需要设计的系统。从这个视角，用户看到的系统是什么呢？当然是一项一项的功能，这些功能是客户能够理解的、具体的、对客户存在价值的功能。从这个意义上说，那些技术性的功能不应当出现在这里，或者应当描述为用户可以理解的文字，比如“自动考核”。而那些应当绘制的用例，在取名时也应当站在用户角度去取名。举个简单的例子，一个员工档案信息系统，以往我们总爱将用例取名为“添加员工信息”、“更新员工信息”、“删除员工信息”，这就是典型的技术人员编写的用例。“添加员工信息”对于用户来讲应当是做什么呢——填写新员工资料；“更新员工信息”对于用户来讲又是做什么呢——更改员工资料；“删除员工信息”又是什么呢——员工注销。不论是“填写新员工资料”、“更改员工资料”，还是“员工注销”，对于客户都是日常工作中需要完成的操作，将用例命名为这些名字必然为用户所理解。同时，每一个用例对于用户来说应当是有价值的，也就是说，用户使用这个功能是要完成一项操作，或获得什么信息。比如上图的“自动考核”会产生一批考核结果，执行“预警监控单项查询”可以获得预警监控结果数据。

2. 图形绘制杂乱无章。一个系统，特别是一个大型系统，提供给用户的功能是繁杂的。如果你想将所有的功能，不管粗的细的，都试图绘制在一个用例图中，几乎没人看得懂。我们之所以将分析设计图形化，是因为图形能给人形象立体的感官，使人立即就明白了其中的意思，但前提是，这个图形是主题清晰的、形象生动的。因此，我们绘制用例图要学会拆分，由粗到细地一个一个绘制。先整体的绘制，再划分成各个模块一个一个详细绘制，再进一步细化。所以，描述一个系统应当有许许多多的用例图。

3. 用例是一个场景。在现实世界中，我们常常面对的是一个个长而复杂的操作流程，但在软件世界里，我们要将它们拆分成一个个的用例，怎样拆分？一个用例必须有一个场景，也就是时间相近、地点单一的一系列操作，并且这些操作最终应当有一个明确的结果。



如上所示这个用例图，“申辩申请”就是过错责任人填写了一张申辩申请单，最终的结果是将申辩申请单提交给考核管理员；“申辩受理”就是考核管理员接收了过错责任人的申辩申请单并予以受理，当然另一个结果是对其不予受理，该申请单被退回给过错责任人。每个用例都有确定的场景，明确的目的和结果。

功能角色分析是对系统宏观的、整体的需求分析，它用简短的图形绘制出了一个系统的整体轮廓。但仅仅进行功能角色分析是远远不够的，我们还需要在它的基础上做更加详尽的分析。

业务流程分析

我们将从客户调研现场拿回来的需求，经过一番功能角色分析，整个系统的整体脉络与轮廓已经被勾画出来。在这个过程中，我们首先将系统划分成了几个功能模块（如果系统规模较大，还应先划分为几个子系统，然后再划分出各个功能模块）。然后，我们为每个功能模块绘制用例图。用例图是站在用户角度去观察的系统，即系统为用户提供了哪些功能，这就是功能分析。同时，这些功能是为哪些用户服务的，这就是角色分析。我们绘制的用例图应当能够为用户所理解，这也是 UML 其中的一项核心思想——与客户形成统一的、能够相互理解的语言，这对于需求分析过程中与客户的沟通是大有好处的。

但形成对系统的整体轮廓，对于软件的需求分析来说是远远不够的。许多软件最终失败的非常重要的原因就是对需求分析过于草率、浮于表面，而没有深入细致地去分析，往往到了项目后期才把需求搞懂，才发现真正的需求与起初的认识大相径庭，才恍然大悟需求原来是这样，而往往那时已经追悔莫及了。这样的经历相信你也有过吧。所以，我们一定要沉下气来认真仔细地做需求分析，一定要做到位。

同样，细化需求也需要一定的方法与思路。一般来说，我们可以有两个方向细化需求：业务流程分析与业务领域分析。这里，我们先谈谈业务流程分析吧。

如果我们现在做的需求分析是一个企业信息化管理系统，毫不疑问，我们的软件系统就是在模拟企业已有的那些业务流程。在现实世界中，企业是按照怎样的流程来管理，我们的软件就应当去模拟这样的流程。但是，我们的软件不可能也不必要完全去模拟这样的流程，在这个流程中的有些环节是应当由软件去模拟的，但有些环节则是应当在系统之外，由人工去完成的。我们进行流程分析，就是要求分析哪些是系统之内的，哪些是系统之外的。

我曾经做过一个疑点信息库系统。该系统模拟的原有业务流程是这样的：高层纪检方面的领导通过信访、举报、数据查询分析等方式发现了一批问题，然后将这批问题制作成一套调查清册，亲自或者交由下级相关单位，下到基层去调查问题。直到调查工作完成以后，才从基层回到自己单位，填写调查工作底稿，详细描述调查情况，并结束调查工作。

首先，我们应当抛开软件实现，对这样一个流程进行梳理，形成这样一个步骤：
1. 高层领导通过信访、举报、数据查询分析等方式发现一批问题；
2. 将这批问题制作成一个调查清册；
3. 自查或将清册下派给下级去调查；
4. 下到基层执行调查；
5. 从基层回到自己的单位，填写调查工作底稿，详细描述调查情况，并结束调查工作。

然后，在对原始需求分析的基础上，分析我们的软件能做什么事：
第一步：信访和举报虽然有自己的操作流程，但那些都在这个系统之外，在这个系统中仅仅只需录入最后的结果。数据查询分析过去只是业务人员在相关业务系统中根据自己的经验执行各种查询，现在则可以上一套数据采集和分析系统，提高数据分析的质量。
第二步：形成调查清册，可以在系统中设计一个功能实现。
第三步：自查或下派，可以在系统中设计一个流程实现。

第四步：下到基层执行调查，由于网络条件等因素的限制，业务人员不可能也不必要在系统中去完成调查，只需要执行一个标志调查工作开始的操作，并打印或导出调查清册，然后去基层调查。最终，这部分被设计成一个“开始实地核查”的操作，并提供打印导出功能。
第五步：调查人员从基层回到自己的单位都是系统外的事情，而填写调查工作底稿，详细描述调查情况，并结束调查工作，则是系统内的功能。最终，这部分被设计成一个“调查完结”功能，标志调查工作结束，并提供工作底稿的填写功能。

计算机信息化管理并不是万能的，它并不能代替现实世界中的所有工作。因此，我们进行业务流程分析，就是要分析业务流程中哪些是需要信息化管理的，而哪些则不需要。信息化管理过细，无疑会加重基层业务人员的负担（这也正是为什么许多基层业务人员会排斥信息化系统的原因），而适当的信息化管理则可以提高工作效率。试想一下，如果你工作中的每一个步骤都必须在计算机中操作一下，怎么不让人烦呢？而如果在工作中一旦需要先查一个什么信息，或者需要计算一下，系统立即可以替你完成这些工作，或者那些过去基本靠吼的操作，现在立马通过信息化就传递过去了，怎么不让人舒心呢？我们做信息化管理，不是要加重人的负担，而应是降低人的负担。以这样的思路去进行流程分析才能设计出优秀的、人见人爱的管理系统出来。因此，我做需求分析，最喜欢下到基层去了解基层业务人员的需求，去分析怎样设计流程才能提高他们的工作效率，而避免加重他们的负担。“水能载舟，也能覆舟。”一套系统是否能顺利推行下去，基层人员是否支持往往起到十分重要的作用。

另外，业务流程分析的另一个重要的分析内容就是流程差异化分析。不同的领导有不同的思路，不同的单位有不同的情况。因此，我们在进行流程分析的时候，常常面临流程差异化的问题。我们说企业信息化就是一次改革，这首先体现在业务流程的规范化操作，也就是消除这种流程差异。但不同的单位有不同的情况，这特别体现在不同地域和文化的不同，又常常造成这种流程差异不可避免。分与合，分治与一统，常常是一个都要兼顾的问题，非常微妙，我们要小心处理。在这个问题上你也许会问，使用工作流引擎就可以了嘛。工作流引擎不是万能的，它只能解决一部分问题，更多的问题还需要我们的分析人员去分析与处理。

最后，企业信息化就是一次改革，这特别集中地体现在了业务流程分析这一部分。当我们详细分析了客户现有的业务流程以后，应当进一步思考这样的流程是否合理，是否值得改进。信息化对于企业流程管理的冲击是巨大的，最典型的实例就是 ERP。ERP 的前身是 MRP（Material Requirement Planning 物料需求计划）。起初，企业也就是希望有一套软件系统来管理它们的仓库。后来，企业领导希望他们在进货的时候能有一定的采购计划，避免出现仓库中的物资挤压，MRP 就出现了。然后呢，企业开始思考整个生产制造的链条管理，MRPII 的概念出现了。再然后呢，物料需求的动因是生产的需求，生产需求的动因是销售的需求。企业要真正做到零库存，就必须切切实实地把从销售到采购的每一个环节都管理好，ERP 的概念就出现了。一个典型的信息化流程改进的例子。

ERP 对企业流程改进的思路是宏大的，但我们在分析每一个系统的时候不可能有如此宏大的雄心与抱负。一般来说，我们可以用以下思路来进行我们对流程改进的分析：清除低效环节、简化业务瓶颈、整合可用资源，以及将繁琐任务自动化。

清除低效环节，就是清除那些耗费成本高而收效又低的环节，最典型的就是过量的库存。过量的库存原因很多，有可能是供销环节没有处理好而造成的过量采购，或者生产过剩，也可能是生产计划没有制订好而产生活动间的等待。除此之外，还有重复的活动，等等。

简化业务瓶颈，就是分析业务流程中影响整体进程的瓶颈业务，并有效地简化它。如很多业务审批流程中都有一个受理环节。大量业务都集中在一两个人来集中受理，根本忙不过来，造成整个流程的效率下降。解决的办法有两个：一个是采用信息化的手段进行批量受理，加快处理效率；另一个是将受理环节的任务分散到更多岗位中，降低受理人员的工作量。

整合可用资源，就是更大范围地整合各个部门、不同职能的人员与社会资源，更加协同地来完成任务，这也是计算机信息化管理最拿手的方面。制造业的供应链管理是最典型的例子，因为实在太经典了我就不累赘了。医院系统也是一个不错的例子：完成了身体检查，医生就立即知道了检查结果；医生开完药，收费处就知道收多少费，药房就知道拿什么药。

最后是自动化繁重操作。在财务系统中开了销售单，就直接开发票了，并且直接形成报税数据；在网上报完税就知道该缴多少钱，甚至不用去税务局，直接上银行缴，等等等等，不胜枚举。繁重操作自动化，正是信息化系统价值的体现。

用例说明

当我们进行业务流程分析时，只空对空而不落到纸面上是不可以的。过去，在面向过程的时代，我们绘制 DFD 图、流程图，以及编写流程说明来描绘这一部分分析；而现在，在面向对象的时代，我们则是绘制行动图、状态图，以及编写用例说明来完成这部分工作。

在这部分工作中，编写用例说明应当是最主要的工作，之后在一些关键部分辅之以行动图、状态图。现在我们来看看用例说明应当怎样编写。

毫不疑问，做用例分析首先是要绘制出用例图（前面已经说过了）。图形的最大优势是能够形象生动地描述我们的分析，但它最大的缺点是会遗失许多的细节信息，因此我们必须要对它进行进一步的文字描述。

用例标识			用例名称	
创建人			创建日期	
版本			用例类型	
用例描述				
参与者				
触发事件				
前置条件				
事件流	基本流程			
	扩展流程			
	异常流程			
后置条件				
假设与约束				
非功能需求				
补充规格说明书			优先级	
业务需求列表				

创建人	版本	描述	创建日期

由于不同的人对用例的理解不同，格式也不尽相同，但一些基本的元素是一样的。以上表格是我采用的用例说明格式，其中用例名称、用例描述、参与者、前置条件、事件流、后置条件是公认的、用例说明的基本元素。

用例标识：就是用例的编号，一般采用“项目编号-子系统编号-模块编号-序号”来编号。

用例名称：没啥可说的，就是用例图中该用例的名称。注意用例的命名规则：用例名称通常是一个动词短语或短句，而不是一个名词短语。它可以是一个动词（如：自动考核），一个动宾短语（如：提取存款），一个被动句（如：发票填报），或者一个主谓句（如：用户提款，这个不推荐，因为主语就是参与者，显得有些多余）。

用例类型：在我看来，不同类型的用例，其用例说明的格式是不一样的。以上给出的是“业务操作”类用例的格式，它更加着重地在描述业务操作的流程。而“查询报表”类用例则没有什么流程，它更加着重地在描述报表格式及显示内容（后面再给出）。还有用例类型还包括“子用例”、“扩展用例”。

用例描述：对该用例的功能定义、要实现的业务需求，以及谁（参与者）应该如何使用进行描述。同时，这部分还可以整体概述实现业务需求的主要流程，以及与其它用例、其它外部系统的关系。通过用例描述，阅读者可以对该用例有一个整体的认识。

参与者：用例图中该用例的参与者，通常是业务操作的触发者和施与对象（如外部系统）。

触发事件：谁干了什么，触发了这个用例。

前置条件：在触发该用例相关操作前必须达到的条件。

事件流：这是用例说明中最重要的部分，它详细描述了该用例可能出现的所有流程。
1. 基本流程：另一个名称更能表达它的意义：最佳流程（The Best Flow）。它描述的是该用例以最佳的、最正常的方式流转，没有出现任何异常，并且最终成功完成操作的流程。基本流程在编写时，应当通过数字对流程中的每一步进行编号。
2. 扩展流程：或者叫“分支流程”，它描述的是基本流程在流转过程中可能出现的所有分支。扩展流程最大的特点就是，它应当是在基本流程的某一步骤发生的分支，因此它的编号规则是“基本流程号+序号”。基本流程号就是发生分支的那一个基本流程的编号。在同一个基本流程上发生多个分支时，它们的序号从 1 依次开始编号。
另一种情况是，某个扩展流程本身拥有多个步骤，这时应当在“基本流程号+自身序号”的基础上再添加序号，如“2.1.1”。
扩展流程在描述时，应当首先描述进入这个分支的条件，即“如果××则××”、“当××时××”。
3. 异常流程：就是发生异常情况时的处理流程。注意，用例说明是站在用例角度进行的说明，因此这里并不是我们通常一样的发生程序异常的处理流程，而是用户在处理业务操作时

发生的异常情况，如：如果顾客不能提供身份证，则 ••••••

后置条件：又称为“成功保证”，就是执行基本流程获得成功以后所达到的状态（条件）。后置条件往往体现的是执行该用例的最终目的。如：完成用户档案的填写并提交。

非功能需求：可以简单归纳为“URPS+”，即可用性（Usability）、可靠性（Reliability）、性能（Performance）、可支持性（Supportability）以及其它（+）。这一部分的需求分析相当重要而又最容易被忽略，后面我们再详细讨论。

假设与约束：就是隐藏于业务功能中的各项规则与条件，如各种逻辑条件、计算公式、环境限制等等。

优先级：没啥可说的，最关键的是怎么去评定。这里我卖一个官子，在需求评审阶段，我会给大家一个比较准确而又可操作的评定方法。

除此之外，我还往往在每一个用例说明的后面与该用例相关的需求列表，便于需求跟踪。用例分析实质是需求人员的一份设计。既然是设计就可能出现偏差，最终偏离原始的需求（这种情况特别容易出现在日后的升级维护中）。因此，将需求列表附在用例后面，便于日后的需求评审与确认。当每次需要升级时，则添加上新的需求，或对以往的需求进行更新。

查询报表分析

在我以往的用例分析中，使用这样格式的用例模式，对于大多数业务操作流程来说是得心应手的，但对于有些功能来说总感觉不对劲。感觉不对劲的，就是那些查询、汇总与报表功能。对于这部分功能，需要我们描述的不是什么操作流程，而更重要的是那些数据项、数据来源、报表格式、数据链接，以及使用者、使用频率的说明。而这些，在以往的用例说明格式中统统都没有，怎么办呢？俗话说“东西是死的人是活的”，把我们的用例格式改改吧。

用例标识		用例名称	
创建人		创建日期	
版本号		用例类型	
用例描述			
参与者			
报表作用			
报表内容			
输出列			
使用频率			
数据链接			
非功能需求			
假设与约束			
补充规格说明书			
备注		优先级	

业务需求列表			
创建人	版本	描述	创建日期

这是我设计的查询报表类用例的格式，同时还可以在后面配上报表的格式。你也可以根据需要设计你自己的格式，用例不是什么阳春白雪的高级玩意儿，而是沟通你、用户、开发设计人员的桥梁。该说明的都说到了，该分析的都分析了，大家都能看明白，并以此为根据去完成各自的工作，这才是用例说明的实质，其它神马都是浮云。

报表作用：就是描述参与者使用这个报表做什么。如果有多个参与者，每一个都应当描述。

报表内容：用简短的话描述一下。

输出列：罗列报表的输出列，如果需要的话，还应对输出列进行说明，或描述它的数据来源。

使用频率：参与者使用它的频率，便于设计者考虑报表的查询效率。

数据链接：哪些数据项有链接，链接到什么报表，或显示什么数据。

最后依然是那个需求列表，便于业务需求的跟踪。

查询报表的需求分析与一般的业务操作的需求分析存在着巨大的差异。而许多需求分析人员没有认识到这一点，这往往导致对查询报表的分析不到位，为项目的研发带来风险，因此在这里我们认真探讨一下。

一个有效的报表，往往不是对数字的简单堆砌，它通过一组一组的数据，揭示的都是一些客观规律、复杂活动与发展趋势。客户方的领导，特别是那些中层和高层领导，通过对这些报表的阅读，就可以掌握他们的工作进程、加强他们的人员管理、发现他们的管理漏洞、指导他们的战略决策。总之一句话，每个报表都有他们的设计意图。

比如说，一份工作月报，领导希望看到的，是按时间、按项目、按部门统计的各项工作的进展情况，以及有哪些异常情况，以便领导监控各项工作能够顺利完成；一份销售报表，领导希望看到的，是按产品、按区域、按顾客类型统计的各项产品的销售情况，以便领导制订销售计划与各种营销战略。没有弄清楚一个报表的真实意图，就不算真正理解了这个报表的业务需求。

同时，报表的数据项应当都是来源与系统中各项操作的结果数据。许多业务系统的操作流程都是纷繁复杂的，其中还包括各种情况。更复杂的，一些商业智能与分析决策系统，报表所需的各种数据，甚至来源与各种各样的外部系统。分析一个报表的数据来源，就是在梳理各种业务流、数据流，以及各种数据间的关系。如果这方面的分析不到位，最终设计出来的报表往往是不准确的。

另外，用户使用报表的频率，常常决定了报表设计的方式。如果报表中的数据总是在实时变化，并且用户总是在密切关注这些数据的变化，那么报表必须设计成实时查询的；如果用户并不是十分关注数据的实时变化，并且总是以天（或者月，或者年）来查看报表，则报表可以设计成按天（或者月，或者年）来预运算统计数字，使得报表查询效率显著提高，可以保证更多的并发访问。

最后，一个报表的核心就是展现给客户的报表格式，以及报表与报表间的各种链接。需求人员在进行需求分析阶段，应当准确地与客户敲定这些格式，并最终在用例说明中体现出来。报表格式是否体现客户的意图，报表数据项是否都能在系统中取到，数据间的逻辑关系是否正确，报表格式是否技术可行，都是需求分析人员在前期就必须要分析到位的内容。否则，报表是项目后期可能出现频繁需求变更的重灾区。

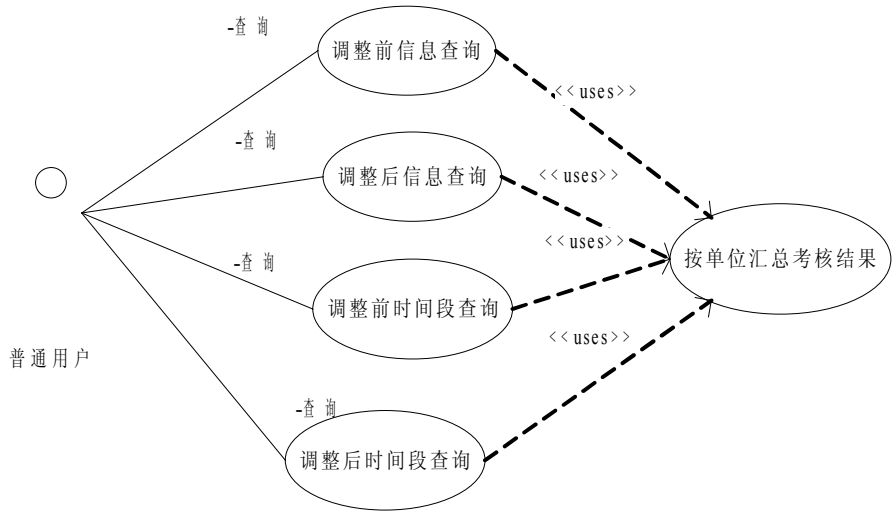
所有这些分析，都体现在了我提供给大家的用例说明格式中。报表作用体现的是报表对于不同用户的真实意图；输出列体现的是对各个数据项及其数据来源的说明；假设与约束罗列的

是报表中各个数据项的运算公式、数据规则与约束；还有使用频率、数据链接、非功能需求，以及最后的界面原型，等等。只要我们把这些都分析到了，我们的查询报表就分析到位了。

子用例与扩展用例

用例模型作为 UML 中 4+1 视图中非常重要的一员，非常集中地体现了面向对象的分析与设计思想。用例模型将现实世界中连续的一个一个业务流程，按照场景划分到了一个一个的用例中。由于场景的出现，使得用例中的业务流程存在着高度的内聚性，从而成为了日后各种对象的雏形。同时，在用例分析中，又将那些存在于各个用例中的，相同或相近的业务操作提取出来，形成一个一个的子用例或扩展用例，又体现了面向对象设计中的复用性。现在我们来谈谈用例分析中的子用例与扩展用例吧。

前面我们在用例说明中提到了基本流程。基本流程就是所有步骤都非常理想地正确执行，并最终完成所有操作的那个“最佳流程”。在基本流程中，可能有些步骤是多个用例都共有的，可以相互共享的流程。将这部分流程提取出来形成的就是子用例。子用例应当是在逻辑上相对独立的一系统流程组成的用例。这个用例应当是抽象的，没有自己的参与者，只有在调用它的用例中，才能真正明确它的使用者。



如图是一个子用例使用的例子。图中，用例“调整前信息查询”、“调整后信息查询”、“调整前时间段查询”、“调整后时间段查询”都用到了“按单位汇总考核结果”。它们是一种使用关系或者包含关系，因此被绘制成一条虚线，从使用者指向被使用者，并标注为 use 或 include。

另外，在用例中还存在许多扩展流和异常流。当系统在运行到基本流程中某个步骤时，由于满足了某个分支条件或异常条件，这时系统就从基本流程流转到了扩展流或异常流中。扩展流和异常流其实不那么泾渭分明。在业务逻辑上扩展流依然是一种正常的操作，仅仅只是正常操作的另一个操作，而异常流其本身就是有什么东西不对劲了，需要进行一些异常处理，比如用户密码输错了、用户忘带身份证了，等等。扩展流和异常流最终都可能回到基本流程中，也可能不能回来，而从另一个结束点结束。

与子用例相似，扩展流和异常流中的流程如果相对独立、可以为其它流程所共享，则可以提取出来，形成一个单独的用例，叫扩展用例。如果扩展用例是直接从基本流程中某个环节扩展出来，则该环节被成为扩展点，进入扩展用例的条件叫扩展条件。在用例图中，扩展关系被绘制成一根虚线，从扩展用例指向被扩展的用例，并标注为 extend。

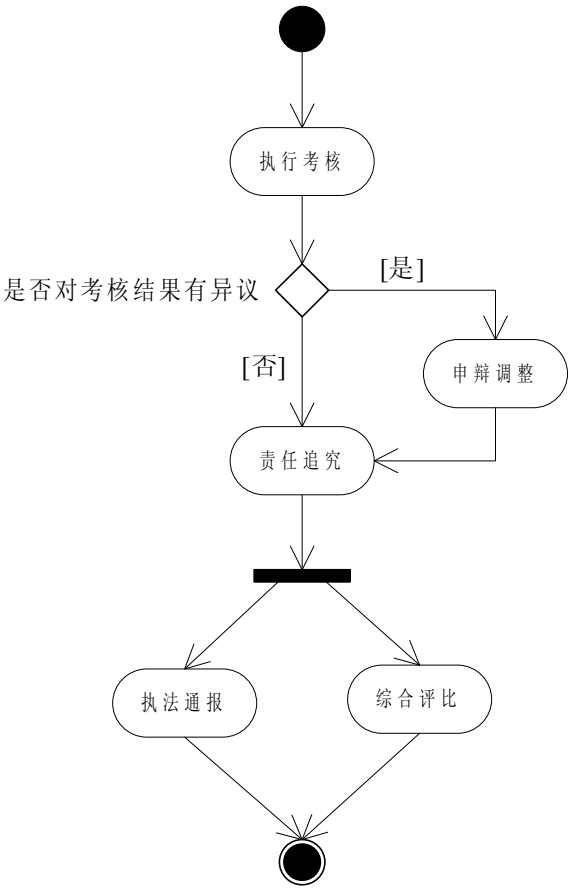
用例分析中对子用例与扩展用例的分析，使我们对系统的设计，从一开始就将公共的、可共享的部分提取出来，使我们在日后的设计与开发中得以很好地复用，提高了系统的内聚并降低了系统的耦合，是一个优秀软件设计的开始。

行动图与状态图

前面，我们耗费了大量的篇幅来讨论用例分析及用例图。用例图，无疑是功能分析、角色分析，以及流程分析的利器，它将我们要开发的系统，清晰而详尽地描述出来。但是，正如任何事物都有两面性，用例图也不例外，也有自己不利的一面。在我看来，这集中体现在两个方面：只见树木不见森林、不生动形象。

什么叫“只见树木不见森林”呢？就是说，用例说明中对业务流程的描述，过早地将系统的整体流程，分散到了各个用例中了，丢失了对业务流程的整体描述。不生动形象，则是说用例说明中对流程的描述都是用枯燥无味的文字来表述的，缺乏生动形象的图形表示。针对这些不足，UML 的另外两种视图，可以有效地弥补用例图的缺陷。它们就是行动图与状态图。

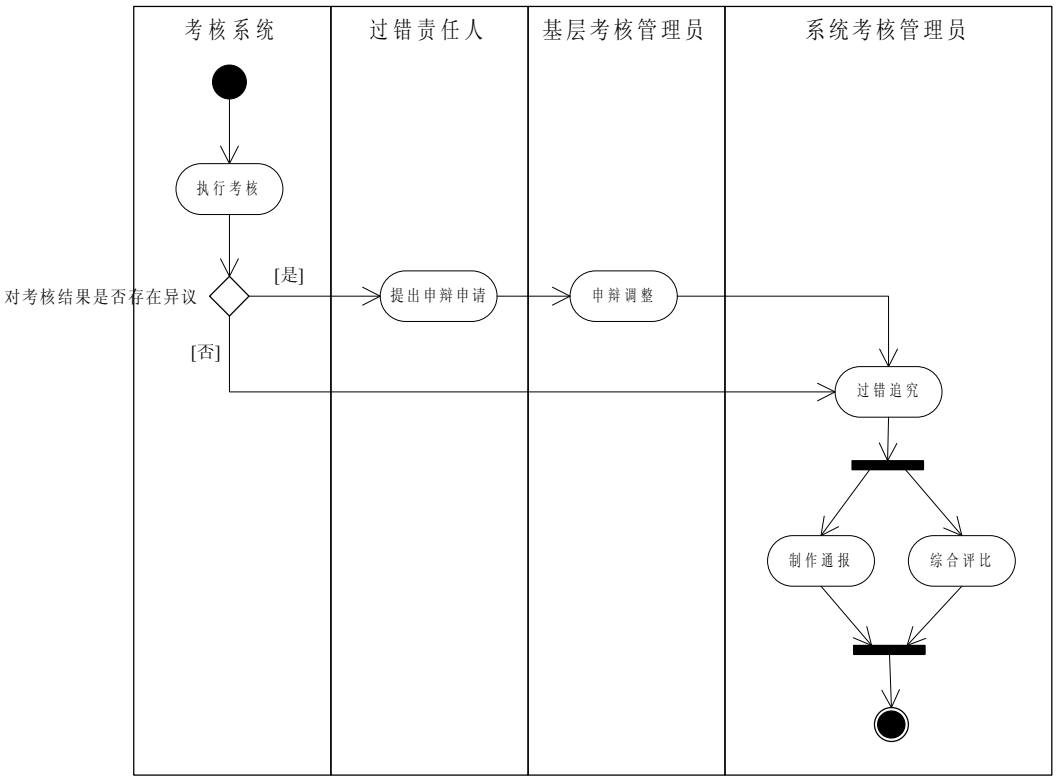
行动图（Active Diagram），比较类似于我们过去绘制的流程图，是 UML 中描述流程与分支的视图。在行动图中，往往是从一个实心圆的起始节点开始的。最频繁使用的则是活动节点了，它表示的是业务流程中的一项活动。活动节点可以表述为一个活动短语（如下订单），可以表述为一个表达式（如 len=a.length+x），还可以表述为一个消息（如 send(msg)）。同时，将各个活动节点连接起来的一个个实线箭头，表明了各种活动之间的流转顺序。



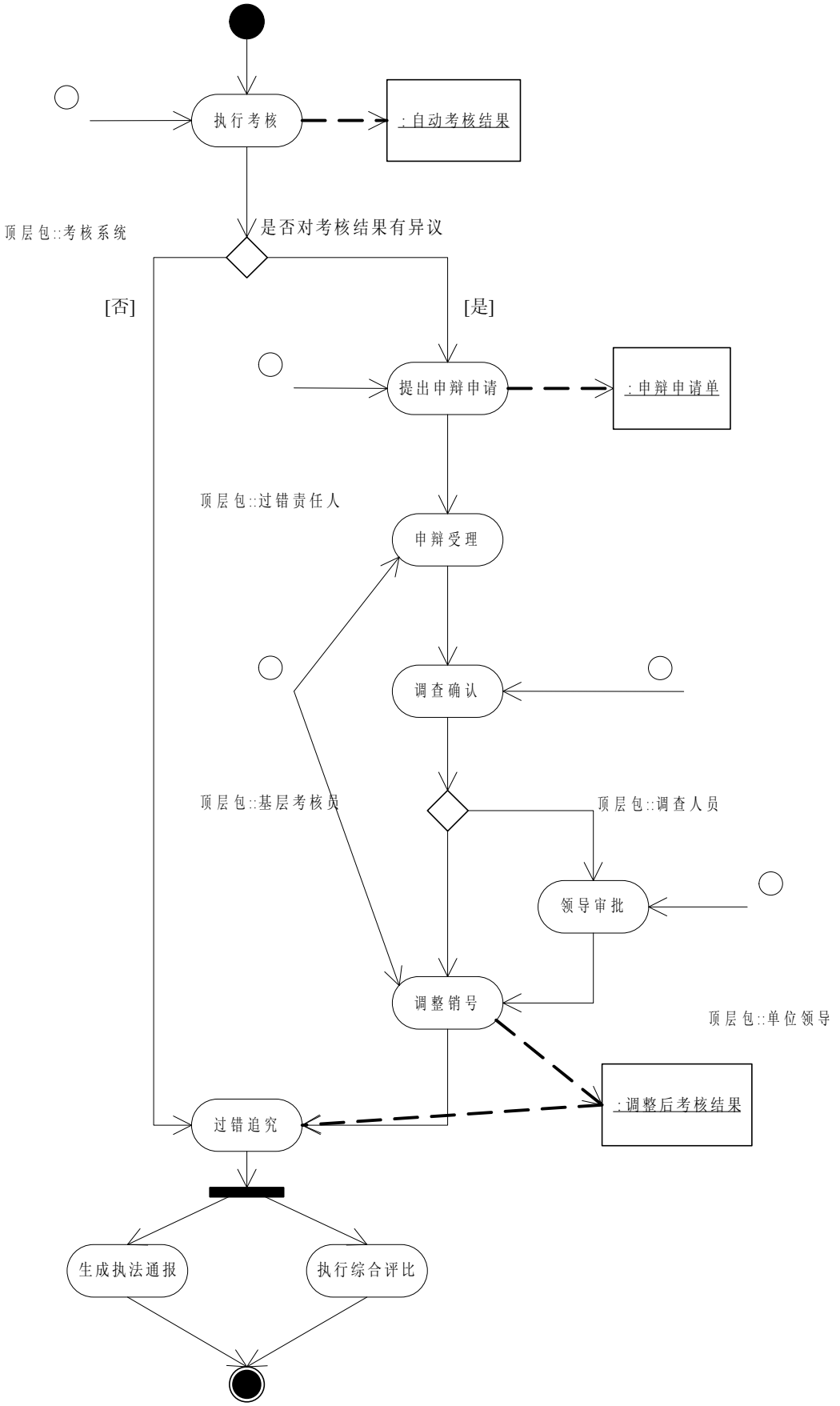
在各种业务流程中，毫无疑问会有许多的分支。在行动图中，分支用一个菱形来表示。一个指向菱形的箭头，表示流程进入分支，另外两个或多个从菱形伸出的箭头，则表示不同条件下的分支流。而菱形本身，则表示为一个条件判断语句。

另外，业务中的各个流程还会分岔与汇合的情况。分岔，表示在某个时间点上，同时开始两个业务流程，这两个业务流程是同步进行的。分岔用一个入箭头，一根横杠，与两个出箭头表示。汇合，则表示，只有在两个流程都完成的情况下，才会进入下一流程，否则只能等待。汇合则用两个入箭头，一根横杠，与一个出箭头表示。

最后，用一个或多个带环的实心圆，表示的是活动图的终止节点，代表了业务流程的终结。以上这些元素，就组成了一个基本的活动图。然而，基本的活动图还不能完整的反映我们的业务流程，因此我们还需要在基本活动图的基础上增加元素。现在我们来看看泳道与业务对象流。



如图就是一个带泳道的活动图，图中每个泳道代表一个参与者的业务操作，而整个图形表述了多个参与者间的协作过程。起初我比较爱绘制这样的活动图，但后来常常感到绘制泳道是一件比较繁琐的事情。既然如此，我们就改改吧。



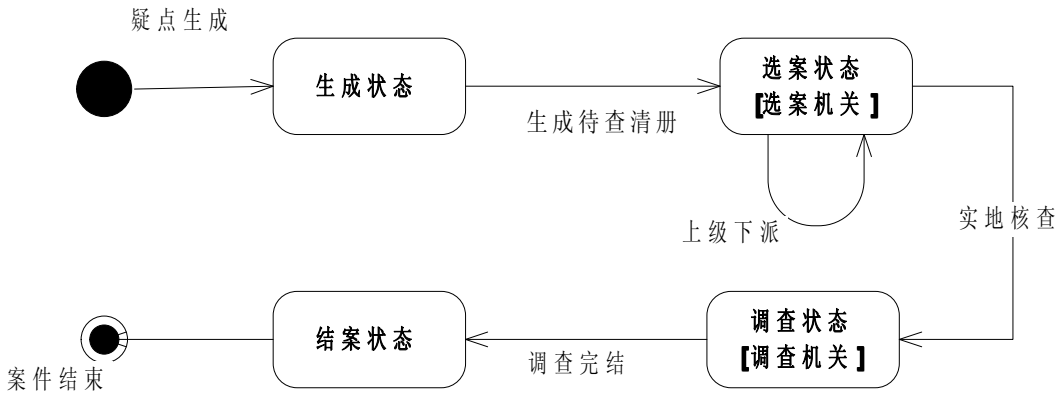
这张图才是我最爱使用的行动图。图中，将参与者由繁琐的泳道改为了用例图中的小人。同

时，在这张图中还增加了对象流与对象。图中，自动考核结果、申辩申请单、调整后考核结果，都是数据对象，是该流程中相关环节操作的结果。从活动节点指向对象的虚线箭头，则表示了一个对象流，如“申辩申请”活动指向“申辩申请单”的虚线箭头，表示了申辩申请活动的最终结果是产生申辩申请单；从“调整后考核结果”指向“过错追究”的虚线箭头，表示过错追究活动读取了调整后考核结果。

当然，活动图还有其它的元素，但我个人认为其实并不实用，使用以上元素就足以表述我们的业务流程了。活动图打破了子系统与子系统的壁垒、用例与用例的壁垒，使我们能够从整体上了解整个系统的流程，因此常常使用在对整个系统的概述、对整个子系统的概述，以及对整个功能模块的概述中。同时，与其它视图一样，活动图也应当有它的文字说明，以便对图中的每个活动节点、分支进行描述。但对于一些流程相对简单，甚至没有什么流程的查询报表类功能模块，绘制它们的活动图则显得有些牵强附会，因此我们要灵活掌握。

除了活动图，我们似乎对需求的描述还缺少点儿什么，那就是对关键对象中流程中状态变化的描述，在这种情况下，我们的状态图就上场了。

在使用状态图时，一个非常关键的概念就是，一定是对某个关键对象的状态变化的描述，而这些状态变化一定是在某个业务流程的大背景下进行的。下图是一个疑点数据整个生命周期的状态变化图。图中，与行动图一样，一个实心圆点代表的是流程的开始，圆边的方框代表的是对象生命周期中的各个状态，状态节点间的实线箭头代表的是状态的切换，箭头的文字描述是触发状态切换的事件。与行动图一样，状态图可以有分支、分岔、汇合，并最后以一个或多个带环的实心圆结束，代表对象生命周期的终结。

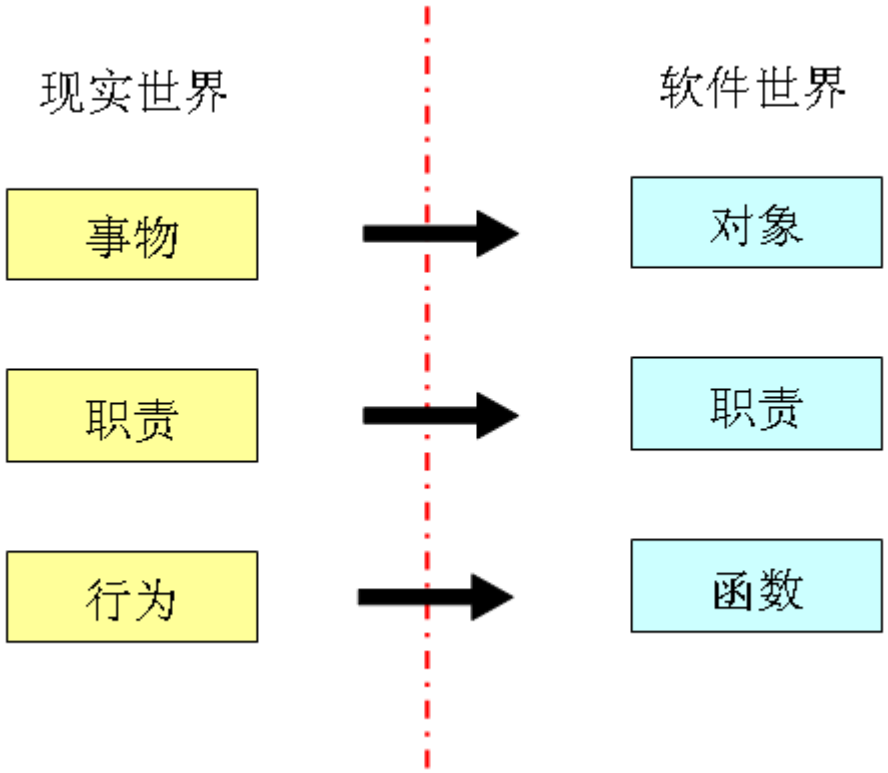


在需求分析中，状态图并不是必须的，它仅仅出现在你认为需要对某个对象的状态进行说明的时候。

业务领域分析

在需求分析工作中，最后一项分析工作就是业务领域分析啦。业务领域分析，就是对需求分析中涉及到的业务实体，以及它们相互之间关联关系的分析。前面我们谈到了功能角色分析，或者说用例分析，它是从整体的角度对整个系统人机交互的分析与整理。随后我们谈到了业务流程分析，它是在对系统人机交互的分析与整理的基础上，更加细致的去分析和整理那些业务流程，以及组成这些流程的一个个业务操作。业务流程分析是对系统进行的一种动态的分析，分析的是那些行为，那些操作。但是，所有的行为，所有的操作，最终施与的对象都是那些实体。这句话怎么理解呢？比如，我们执行填写操作，施与的对象必然是那些表单，最终产生的结果必然是形成一份完整的表单，表单就是那个行为施与的对象。再比如，我们执行查询操作，施与的对象必然是一个报表，最终产生的结果必然是查看到了这个报表的结果。这里的表单、报表，都是存在于系统的静态实体，它们中的大多数也最终以数据结构的形式持久化保存于系统的数据库中。因此，系统中应当有哪些实体，这些实体都有哪些属性，被赋予了哪些行为，它们之间的相互关系是怎样的，就成为了业务领域分析的重要内容，而业务领域分析也就成为了对系统进行的一种静态分析。

我们的软件系统，毫不夸张地说，就是对现实世界的真实模拟。现实世界中的事物，在软件世界中就被模拟成一个对象。该事物在现实世界中赋予什么职责，在软件世界中就赋予什么职责；在现实世界中拥有什么特性，在软件世界中就拥有什么属性；在现实世界中拥有什么行为，在软件世界中就拥有什么函数；在现实世界中与哪些事物存在怎样的关系，在软件世界中就应当与它们发生怎样的关联。这正是面向对象编程的核心思想。



我们进行业务领域分析，就是基于这样一个思想进行的。什么叫业务领域，就是客户所在的

知识领域，譬如财务人员所在的是财务领域，税务人员所在的是税务领域，营销人员所在的是销售领域。不同的知识领域拥有各自不同的领域知识，需求分析人员就应该通过客户中的领域专家去学习这些知识、掌握这些要点，并最终体现在我们的需求分析中。然而，这必然是一个长期的过程。从这个角度说，业务领域分析不仅出现在需求分析阶段，还应当贯穿与设计阶段、开发阶段、测试阶段，甚至延续到后期的维护与升级。从另一个角度讲，现在的软件研发概念，已经不再是一锤子的买卖，而是延续到数年的不断升级完善中了。而软件的升级完善，从本质上说就是对业务领域不断深入的认识。我们对业务领域的认识深入一点儿，我们的软件系统就完善一分，再深入一点儿，就再完善一分。这就是世界级软件分析大师 Eric Evans 提出的领域驱动设计的核心思想。

因此，我们进行业务领域分析，就是通过与用户进行交流，掌握领域知识，然后绘制成业务领域模型，去指导我们软件开发的过程。日后我们去设计开发系统时，应当设计哪些类，类中都应当有什么属性和行为，以及怎样去设计数据库，都是以这个领域模型为基础的，虽然有时并不完全与领域模型完全一致。过去，没有一个切实可行的方法来指导我们的业务领域分析，而现在，我们可以通过两种分析方法一步步进行：原文分析法与领域驱动设计。下面我们对这两种方式进行详细分析。

原文分析法

原文分析法（Textual Analysis），是在用例说明与流程分析的基础上进行的业务领域分析，是一项在需求研讨会后整理和分析需求的工作。当我们完成了用例图的绘制，为每个用例编写出用例说明以后，原文分析的工作就可以开始了。要讲解原文分析，我们还是用一个实例更简单明了：

用例标识		XM201117-ZXKH-01	用例名称	自动考核
创建人		某某	创建日期	2011-11-23
版本号		V1.0	用例类型	业务操作
用例描述		系统中设计了许多考核指标，每个考核指标就是对某个执法过程的考核。每个考核指标都有一个过错判定标准。正确的执法行为定义为分子，错误的执法行为定义为过错，所有被考核的执法行为定义为分母，而一个指标的考核结果就是该指标所有分子除以分母形成的正确率的统计结果。同时，每个考核指标根据定义都有一个或多个不同的过错行为。不同的过错行为，根据其性质的轻重，定义了不同的分数。正确的加分，错误的扣分。另外，系统还要对限期完成类执法行为提前进行预警。 系统则每天对所有开启的考核指标进行自动考核，并对其打分，产生当天的考核结果。该结果是指从本月一日截止到当天所有执法情况进行的考核。直到月底最后一天，完成当月的最终考核，考核结果将不再被更改。		
参与者		系统、执法人员		
触发事件		系统触发器每日开始触发自动考核		
前置条件		无		
事件流	基流	1. 系统触发器每日开始触发自动考核，对每个指标当月的执法行为依次执行考核： 1.1 首先根据该指标的定义，对从本月 1 日起，截止当天应当预警的所有执法行为进行预警； 1.2 其次根据该指标的定义，对从本月 1 日起，截止当天应当完成的所有执法行为进行采集； 1.3 根据指标的过错标准，判断每个执法行为是否正确。一个指标可能存		

		在多个过错行为，每个过错行为都有自己的过错判断标准，只有所有过错行为都不是过错，才能判定该执法行为是正确，否则就是过错。 1.4 根据执法行为是否是过错，是什么样的过错，对每个执法行为进行打分； 2. 对所有指标的考核结果，按照时间（年度、月份）、指标（考核指标、过错行为）、机关（省、地市、区县、科所等级别）三个维度对考核结果进行统计，产生分子数、分母数、过错户、过错数、正确率等统计数据。 3. 到了月底最后一天，完成当月的最终考核结果，考核结果将不再被更改。
	分支流	无
	替代流	无
后置条件		每天完成所有指标的考核，产生当天的考核结果； 月末产生当月最终的考核结果，并可以在调整后查询中进行查询。
非功能需求		1. 每天的自动考核必须在当天晚上完成 2. 自动考核结果必须准确
假设与约束		1. 系统对限期完成类执法行为提前进行预警。如果被考核人看到考核结果中的预警信息，并在当天完成了预警信息提示的相关工作，则系统对第二天的考核结果中不再显示该预警； 2. 系统对可补救类执法行为可以进行补救。如果被考核人的某项执法行为被考核为过错，并且根据该指标的定义，被考核人在当天对错误行为予以了更正，则系统对第二天的考核结果中不再认定该执法行为为过错。已跨月则不能予以补救。
补充规格说明书		无
备注		无
优先级		高

业务需求列表			
创建人	版本	描述	创建日期
某某	1.0	系统对所有考核指标进行自动考核	2011-11-23
某某	1.0	对限期完成类执法行为提前进行预警	2011-11-23
某某	1.0	执法人员对可补救类过错能够进行补救	2011-11-23

某某	1.0	每月最后一天完成当月的最终考核，考核结果不再被更改	2011-11-23

这是一个实际项目的用例说明。在进行原文分析的时候，我们首先要做的事情就是对用例说明中事件流部分的文字描述，提取其中的名词。在这个实例中都有些什么名词呢？这些名词我在用例中用蓝色标注了出来，经过整理就是这些：触发器、考核指标（简称指标）、执法行为、指标定义、过错标准（过错判断标准）、过错行为、考核结果、年度、月份、机关、分子数、分母数、过错数、正确率。

领域模型中的实体，往往就在我们通过原文分析提取出来的这些名词中，但需要我们进行进一步分析。并不是所有名词都可以成为实体，那么哪些可以呢，而哪些又不能呢？首先，系统外的参与者不能。系统外的参与者是触发本系统某个事件的人或者物，但它本身存在于系统之外，比如用户使用鼠标点击了一个按钮，而领域模型是描述系统之内的事物，因此系统外的参与者应当被排除。本例中的触发器就是系统外的参与者（参见《功能角色分析与用例图》），它应当被排除。

其次，系统之内的事物转化到领域模型中，可能会变成两种东西：实体与实体中的属性。什么变成实体而什么变成实体中的属性呢？自身有自己的属性，可以成为系统中行为的执行者或施与者的，才是实体。比如考核指标就是实体，因为它有它的考核标准、过错行为、分子数、分母数、过错数、正确率等属性，它在系统中会去执行考核，所以是实体；分子数是不是实体呢？它仅仅是一个数据，没有自己的属性和方法。另一个判断是实体还是属性的方法就是判断它将如何持久化。如果一个事物被持久化到数据库中时是一个表，则是一个实体；如果仅仅是表中的一个字段，则是一个属性。

然而，是实体还是属性并不是那么绝对，关键看系统对这个事物进行怎样的处理。比如过错标准是一个实体还是一个属性呢？如果我们在系统中仅仅是一个文字描述则是考核指标中的一个属性，如果需要对它进行分解，有它的判断公式，需要让它去执行判断，则应当是一个实体。在需求分析的初期，可以先将其设计成一个属性，待日后的细化阶段再进行调整。

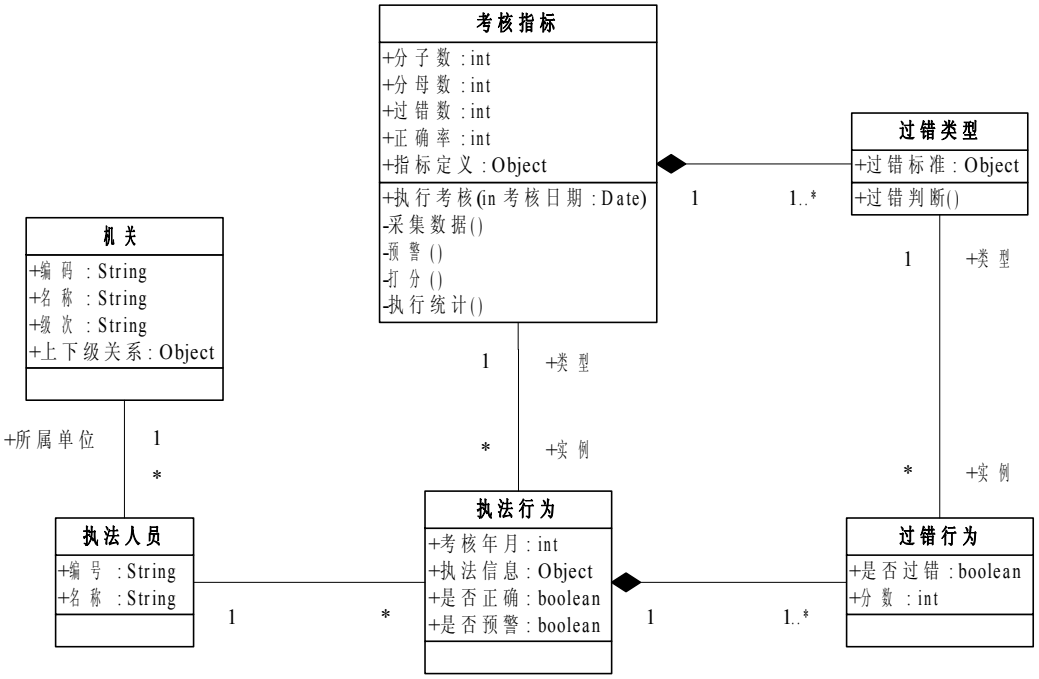
另外一个非常重要、值得我们着重关注的地方是名词的多义性。在本例中，我们考察一下“过错行为”这个名词。“一种过错行为”与“一个过错行为”显然不是一个概念。“一种过错行为”代表的是一种类型，有它的过错定义与判断标准；“一个过错行为”则代表的是一个实例，一个执法行为中的某个错误的行为。正因为它们概念上的差异，我们在领域模型中将其分为“过错类型”与“过错行为”。

经过一番分析，我们绘制出了一个基本的领域模型。毫无疑问，这个领域模型使用的是一个类图，实体在图中就是一个个的类。同时，我们将各个类之间的关系标注出来：一对一、一对多、多对多、聚集、组合、继承，等等。为了提高模型的可读性，我们在必要时可以标注关系的名称。如考核指标与执法行为之间是类型与实例的关系，等等。

现在，让我们重新回到原文分析。这次要分析的不是用例说明中的名词，而是动词，在本例中我用红色标注出来。最后，我们整理出这些动词：触发、执行考核、预警、采集、判断、是过错、是正确、打分、统计。

对用例说明中的动词分析，是为了定义各个实体之间的各种行为。同样，并不是所有动词都

是实体的行为。参与者的行为显然不是实体的行为，应该被排除掉，如：实例中的“触发”。还有一些动词是某个行为的一个细节，如：“是过错”、“是正确”，被合并到“过错判断”中。最后，将行为添加到行为的执行者中。最后绘制出这样一个领域模型：



领域模型有别于后期的分析模型，其中最关键的就是目的，它的目的仅仅是分析需求，因此在很多地方会比较模糊而不考虑技术实现，比如本例中的“指标定义”、“过错标准”。另外一个比较关键的地方就是，系统中的行为到底由谁来执行，这个标准常常是说起来容易做起来难。我给大家的建议是参考 GRASP 中的“信息专家”模式。

GRASP 是一种职责驱动设计的系统分析方法，它的“信息专家”模式是这样描述的：应当将系统中的行为交给信息专家去执行，而信息专家就是掌握着执行该行为所需数据的实体。在本例中，由于考核指标掌握着指标的定义，还有那些执法行为，所以它可以执行考核，而过错类型则掌握着过错标准，因此可以执行过错的判断。注意，这里的“执行”什么行为，是软件意义上的概念，即一个类可以拥有什么行为，而非现实世界的概念。要知道现实世界中的事物是不可能有主动执行什么操作的能力的。

过去我们拿到需求不知道该怎样去业务领域分析。有了原文分析方法，给了我们一个简单可行、易于操作的方法，让我们准确而高效地完成业务领域分析。

领域驱动设计

2007 年，世界级的软件分析大师 Eric Evans 发表了他的经典著作《领域驱动设计》，进而形成了一套独特的软件分析与设计方法，简称为 DDD（Domain-Driven Design）。在领域驱动设计思想中，有许多是涉及到需求分析领域的先进方法，我把它归纳为有效建模、统一语言和持续学习。

有人说：大师所站的高度实在太高了，是生活在太空里的，所以我们要追随大师就只有因为缺氧而死掉。我认为这句话说得非常生动，学习大师真的不是一件容易的事，把大师的思想落实到我们的工作中更难，常常还伴随着一些不小的风险，学习伊大师也是一样的。

伊大师一上来就提出了要有效建模的思想，我当时立马就晕菜了。按照这个思想，我们应当在业务研讨会上，与客户讨论业务需求的时候就开始现场建模了。这！怎么可能呢？客户看得懂那些专业的、抽象的模型吗？我们能拿着模型与客户交流吗？这是不是在浪费时间？

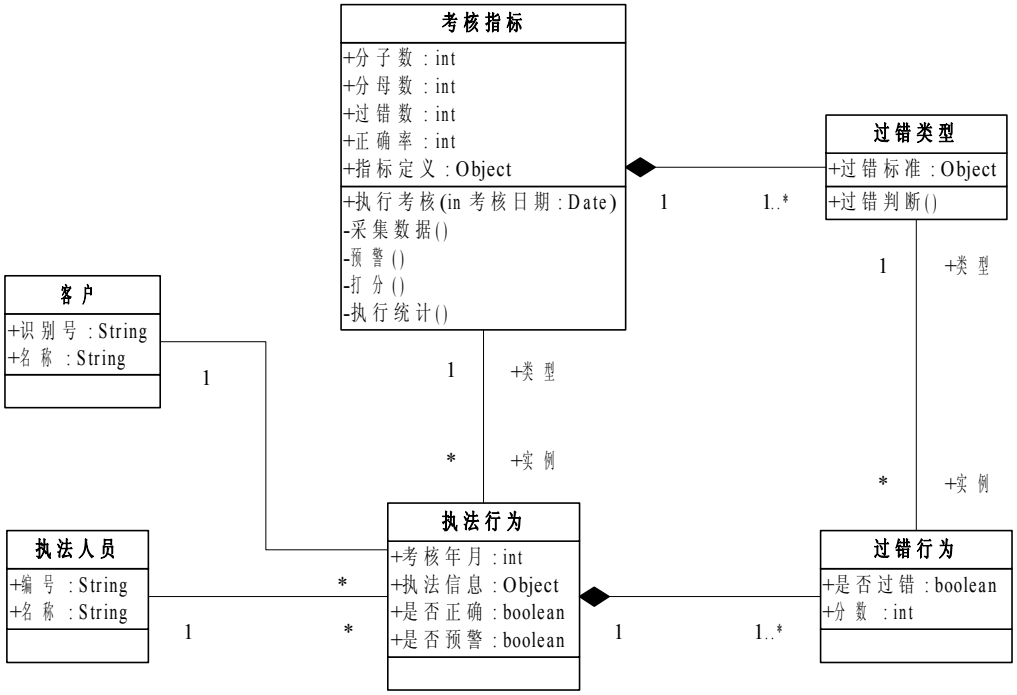
的确，伊大师提出了有效建模思想，与其它很多诸如在会后分析整理时进行的原文分析方法大相径庭。同时，这个思想认为，我们应当与客户代表形成一种统一的语言，一种混合语言。这种语言，既有软件技术中的元素，又有业务领域中的术语，同时，它又是技术人员与业务人员都能理解的语言。使用这个语言，技术人员与业务人员就是在用同一语言在沟通与讨论问题，这种沟通的障碍就得以消除。

道理简单实践难，什么是有效的建模，什么是统一的语言呢？经过无数的实践与尝试，我逐渐开始明白了。首先，什么是有效的建模呢？当我们作为非专业人员去看一个建筑设计师绘制的图纸时，我们一看就明白这是一栋楼房，那是一座桥梁，为什么？因为图纸形象生动，没有那么多专业术语，我们一看就明白了。软件中的设计图也是一样的道理。

当我们站在技术人员的视角去绘制设计图时，客户必然看不懂，因为图中使用的都是专业的术语、专业的符号，表达的都是专业的设计思想。反过来，如果我们站在业务人员的视角去绘制设计图时，情况就不一样了。如果一个用例图，图中的功能都是客户日常经常做的业务操作，并且命名都是业务人员能够理解的业务术语，试问客户会不理解吗？同样，在领域模型中，我们按照客户的思路，运用客户的术语，去绘制一个一个的对象，按照他们的思路去描绘对象间的关系，描绘对象间的操作，他们真的就会看不明白吗？这说得似乎有一些抽象，我们举一个实际的例子吧。

有一次，我与客户在讨论一个考核系统，首先客户描述着他们的需求：
客户：我们这个考核系统是由许多个考核指标组成的，每个考核指标就标志着我们的某项工作的完成情况。每个考核指标中有一个分母数，标志某段时间所有应当完成的工作数量，有一个分子数，标志这段时间正确完成的工作数量，最后还有一个过错数，标志那些错误的，或者没有按时完成的工作数量。
需求人员：为什么是分子分母？
客户：因为最后要计算正确率，用正确率来考核一个单位完成工作的情况。
这样，我们在纸上绘制出一个考核指标，在属性中写下分母数、分子数、过错数、正确率。

需求人员：那么每个考核指标都有一个过错判断标准了？
客户：当然啦，每个考核指标都有它的过错判断标准。一个考核指标可能会有多个过错行为，每一个过错行为都有各自的过错判断标准，任何一个过错了，这个执法行为就算过错啦。
需求人员：先等等，你刚才提到执法行为了。执法行为和考核指标是什么关系？
客户：哦，执法行为嘛，就是执法人员对某个用户执行的一次业务操作。考核指标中的分母数就是所有执法行为的个数；分子数就是正确的执法个数；过错数就是错误的执法个数。
这样，我们就绘制出这样一个草图：



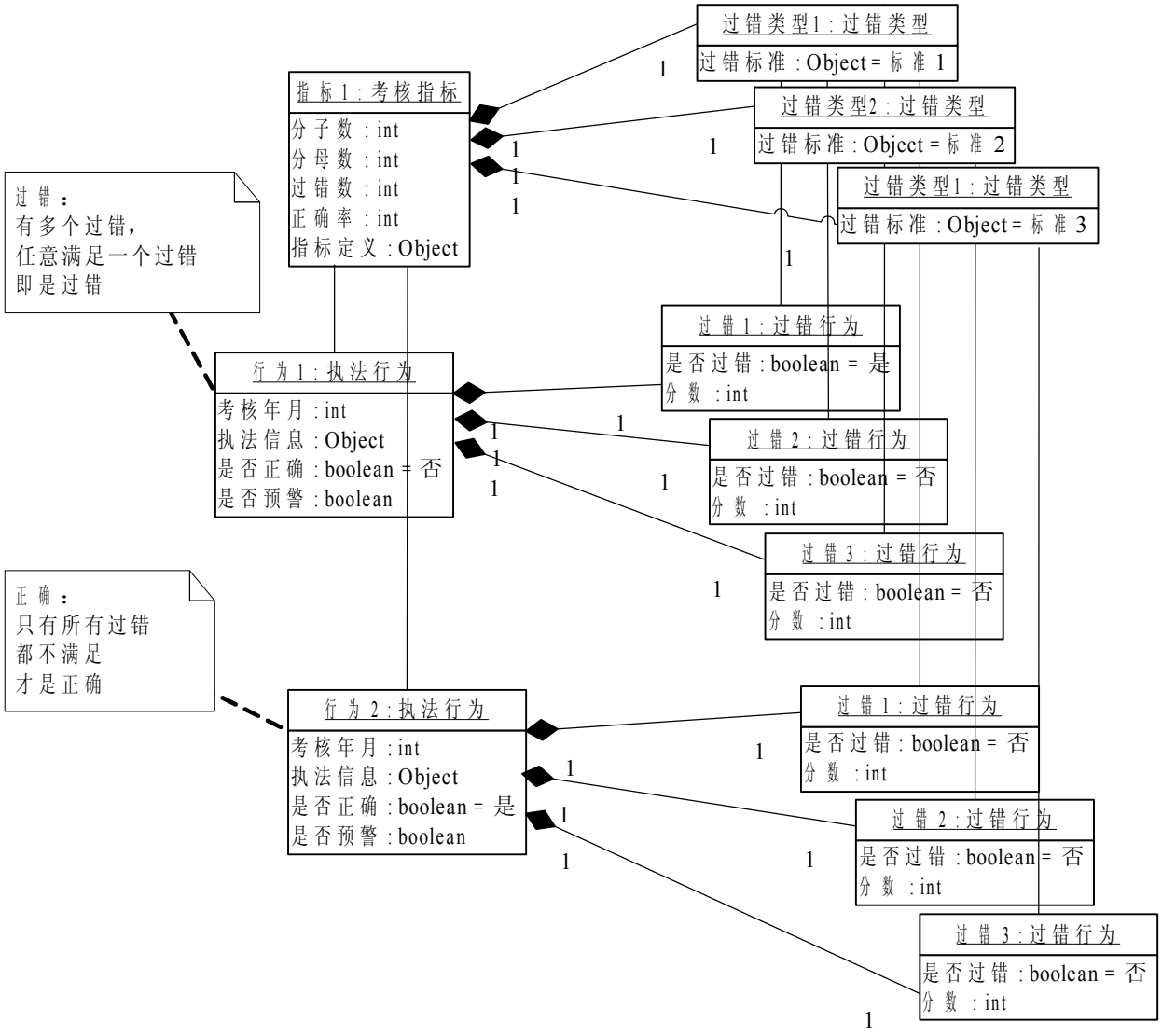
客户看了这个草图有些不同明白：过错类型是什么东西？
需求人员：过错类型就是某种类型的过错行为呀，它定义了某种过错行为，有它的过错判断标准。下面这个过错行为就是那些具体的过错，比如张三今天犯了什么错，李四明天犯了什么错。
客户：哦，明白。这两个箭头怎么跟其它箭头不一样，后面还跟了个菱形框？
需求人员：哦，这代表的是包含关系，表示一个考核指标包含了多个类型的过错行为呀。

经过一番交流，我们已经明白客户的意思了，客户也明白我们画的图了。大家对彼此的交流都比较满意。

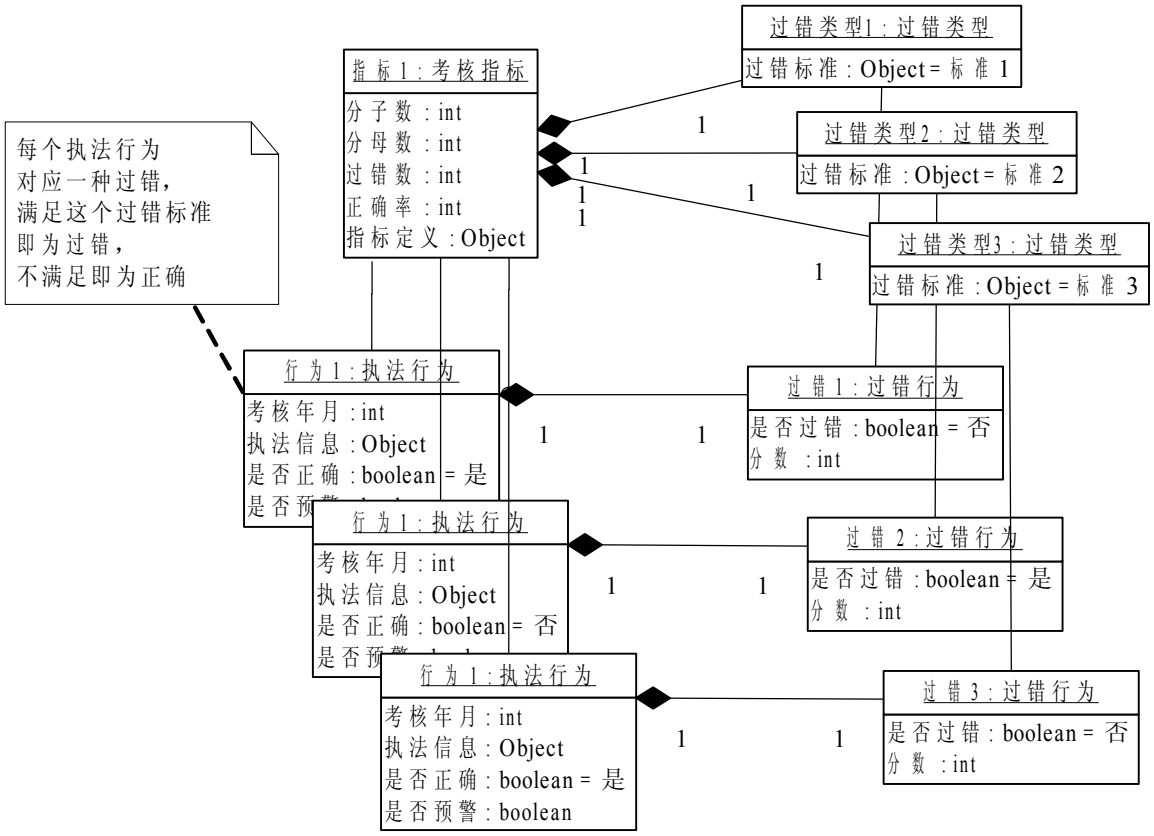
所有的爱情都是以浪漫开始的，需求分析也不如此。随着需求分析的不断深入，我们发现问题了。在这张图中，我们把执法行为与过错行为仅仅描述为一对多的包含关系，似乎没有什么特别的。但对大量考核指标具体需求的分析，我们才发现其实不是这样简单。当考核指标只有一种过错行为的时候，那非常简单，这个过错行为对就是对，错就是错。但当考核指标存在多种过错行为的时候，情况就复杂了，必须分成三种情况：

1. 一个执法行为同时包含多种过错行为，每个过错行为就是一个考核点，任意一个考核点错了整个就判错，只有所有考核点都正确才判正确。这种情况就像填一个表单，所有数据项

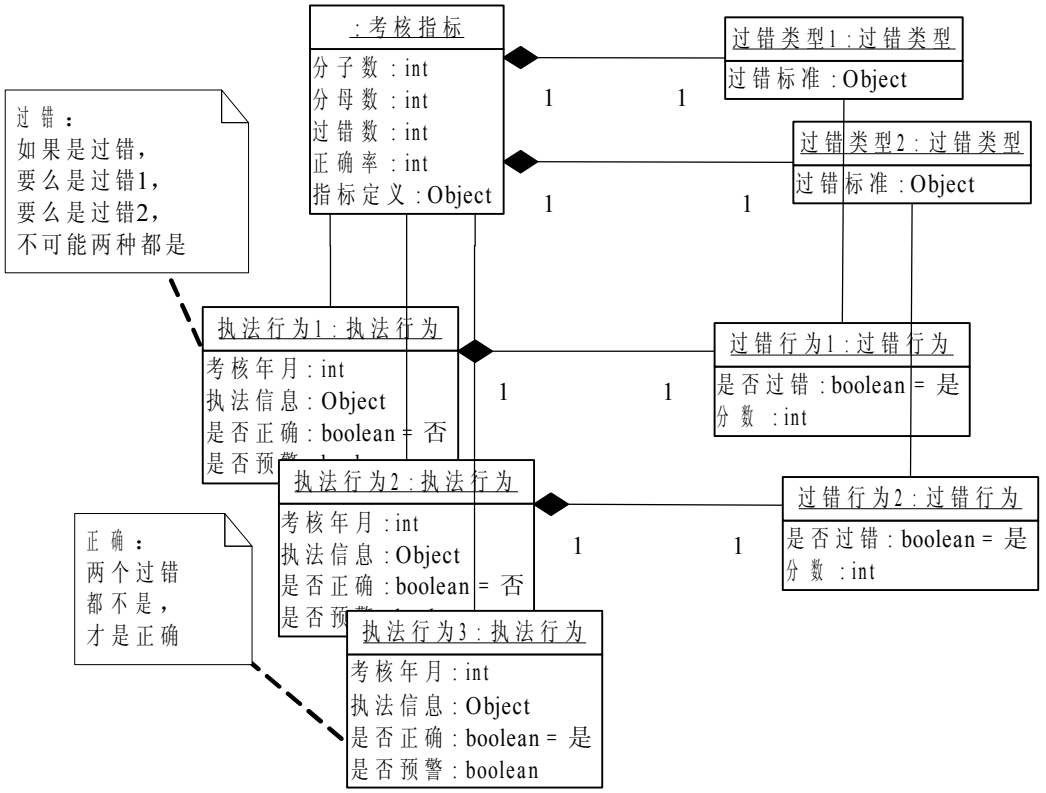
都填对了才算对，任意一个错了就算错，然后画出这样一个对象图：



2. 虽然一个考核指标定义了多个过错行为，但它把所有执法行为分为多个类型，每个类型的执法行为只对应一个过错行为，这个过错行为对就是对，错就是错：



3. 最后一种就是那些限期完成的考核指标，正确的行为只有一个：按时完成的行为，过错行为却有两个：逾期完成与逾期未完成。



虽然图形有些复杂，但这正是代表了现实世界业务的复杂性。我们拿着这些图与客户进行了简单的描述，由于图中的所有元素都是用客户熟悉的术语描述的，因此他们很快就能够理解。同时，开发人员拿到这样一个图，很快就制订了四套不同的方案，来分别解决四种不同的情况。

随后，在对这四种情况更加深入的分析时，我们发现第一种情况在计算过错数时似乎有一些问题。在第一种情况中，一个执法行为包含了多个过错行为，如果出现了过错，过错数算几？假如一个执法行为包含三个过错行为，如果都做对了，分子数算 1；但假如有 2 个过错行为错了，过错数算 2？还有那一个正确的行为呢？这似乎有些矛盾！当我们向业务人员提出这个问题时，他也有些懵了，这样的结果似乎是我们双方都没有预料到的。经过反复的思考与讨论，最后我们做出这样的决定：将原有的过错数拆分成过错户与过错数。在以上情况中，如果一个执法行为有 2 个过错行为错了，过错户为 1，过错数为 2。试想，如果不对需求进行如此深入分析与理解，能发现这样的问题吗？如果不及早发现这样的问题，是否会给项目后期带来巨大的风险？

应该说，从最初的粗浅认识，深入到后来对四种情况的认识，正是体现了 DDD 的另一个思想：持续学习。需求人员在开始一个新的管理系统的分析工作时，都有可能面临着一个全新的业务领域。在这个领域中，他们不可能一夜成为专家，也不必要成为专家。他们需要时间去学习领域知识，但这并不意味着学习所有的领域知识，而是与软件相关的领域知识。做财务软件，你不必考财务师，但你必须要学会与财务软件相关的财务知识。你对领域模型的认识被延伸到了整个软件生命周期中，包括之后一次一次的升级完善。你每认识深入一点儿，就可能会体现到你的分析设计中，并最终体现在开发的软件中。你对领域知识认识再深入一点儿，软件就再完善一分。

非功能需求

我曾经看过许多关于需求分析的书籍，老外写的，国人写的，都有。但我总体就是一个感觉：累。各种各样的分析、各种各样的视图，让人眼花缭乱。为什么会这样呢？不得不说，需求分析是一个太宽泛的概念了，不同的行业（商业的、管理的、游戏的），不同类型的软件（底层的、桌面的、网络应用的），不同的设计方式（面向过程的、面向对象的），需求分析的过程都存在着巨大的差异。要制订放之四海而皆准的方法谈何容易。即使同一类型的软件，它们也存在着各自的特点，有的问题大多数软件都不用考虑，而它必须考虑。正因为如此，许多关于需求分析的方法和书籍描述得挺复杂的。

但我要说，我们做需求分析应当化繁为简，不必去拘泥于那些过程。怎样化繁为简？寻找适合自己的，避免做过度分析和设计，这种思想也是敏捷开发的精髓。比如我所从事的管理软件的研发，关注业务流程、关注业务实体、关注规则约束，功能方面的需求就分析完成了大半。然后再关注查询报表、关注外部接口、关注打印导出等细小功能，功能方面就差不多了。

但是，我不得不说，需求分析人员最容易忽略的部分就是非功能需求。非功能需求更加靠近的是技术，是设计，是实现，是架构师关注的内容，是需求人员最不擅长的方面，这也是非功能需求为什么常常被忽略的重要原因。正因为如此，架构师应当尽早参与到项目中，参与到需求分析中，尽早分析需求的技术可行性，尽早考虑性能、安全性、可靠性等非功能需求，尽早开始架构设计。

在非功能需求分析中另一个非常常见的错误，就是将非功能需求仅仅归结为一些放之四海而皆准的原则，比如专门拿出一章来描述报表查询效率要怎样、系统易用性要怎样。诚然，这些原则性的东西是十分必要的，但许多非功能需求不能仅仅停留在这些基本原则上，要落实到对一个一个功能的分析中。

说这么多虚的，咱们还是上实例吧。还是这个考核系统，每天在上班后 1 小时内，将有 90%的用户会上线查看自己的考核结果。因此，在进行考核结果查询功能的分析中，我们写下了这样的话：查询必须高效（预计查询数据量：xxx），并且支持高并发操作（预计并发用户峰值：xxx）。有了这些描述，设计和开发人员会着重注意该功能的性能问题，测试人员也可以着重进行该部分的性能测试。

在另一个项目中，用户需要对大量的数据进行选择，进而完成制作清册、下派、回退等操作。在前期的需求分析中，需求人员没有仔细分析这些操作的易用性，没有提供给用户批量选择等功能，直到试运行时才发现。当时系统到各基层试运行时，激起了巨大的民怨，给项目带来了巨大的负面影响。多亏我们及时发现问题，加班加点完成了相关操作的批处理功能，才使项目得以顺利推行。如此看来，非功能需求对于一个软件项目是多么重要。因此，我建议，在需求分析的细化阶段，需求分析人员应当与架构师一起，一项一项地去分析每个功能的非功能需求，并在用例说明中记录下有特殊非功能需求的功能，使我们对非功能需求的分析落到实处。

那么哪些是非功能需求呢？我们可以简单归纳为“URPS+”，即可用性（Usability）、可靠性（Reliability）、性能（Performance）、可支持性（Supportability）以及其它（+）。而这 5 部

分我们可以进一步细化。

可用性是一个非常宽泛的概念，它泛指那些能让用户顺利使用系统的指标，包括易用性（易操作、易理解）、准确性、安全性（权限体系、访问限制）、兼容性（服务器、客户端的兼容度），等等。

可靠性就是系统可以可靠运行，包括系统成熟度（数据吞吐量、并发用户量、连续不停机性能等）、数据容错度、系统易恢复性，等等。

性能，我认为是需求分析阶段最主要的分析内容。用户对性能的要求没有止境，但现实却是残酷的。性能受到许多因素的影响，包括业务需求、软件设计、数据库设计、系统部署方式，等等。其中，业务需求和部署方式，对性能的影响是最大的，我们必须在需求分析阶段就想清楚，解决掉。有一次，客户提出了一个数据导出的功能，这看似一个非常普通的功能。但是经过仔细地分析我们发现，客户在执行数据导出前的查询时，如果选择时间跨度数年，查出的数据量可能达到数十万。要将数十万数据一次性地导入到一个 excel 文件中，这不论从运行效率、系统稳定性，还是技术可行性分析都是不可取的。最后，我们经过与客户的协商，一次性导出数据最大不超过 2 万，同时提供了分页导出的功能，可以让他们选择导出从第几页到第几页的数据。这样，如果数据量大，客户可以经过多次将数据导出，数据导出的性能得以保证。

系统部署架构对性能的影响也是巨大的。一个管理系统，是市级集中，还是省级集中，甚至全国集中，对性能的考量是不一样的。市级集中不会过于担心性能的问题；省级集中就必须要考量并发访问量，是否要建立集群；全国集中就必须考量是否使用消息队列，所有流程是否有性能瓶颈，以及采用什么技术架构更适于并发访问等等。而这一切都是系统架构师应当考量的内容。

最后一个内容，也是最容易被忽略的一个内容，就是可支持性。可支持性，就是软件的可维护性、易变更性。可支持性对于客户是透明的，不可见的，因此客户通常不关心这个。由于时间紧、人员素质参差不齐，这部分也常常为管理者所忽略。但试问，谁没有维护糟糕系统的痛苦经历？谁们的系统维护了数年经过数次升级后还能维护？在需求分析与设计阶段，可支持性实际上体现在，我们是否能有效识别系统可变的需求，并能够提供合理的方案。这体现的也是架构师的功底。一次分析和设计 ERP 软件的时候，我发现应付单需要生成凭证，随后又发现应收单、采购单、销售发票都要生成凭证。既然这么多单据需要生成凭证，是否还有其它我们还不知道的单据也要生成凭证，是否可以有一个统一的接口。果不其然，核销单、工资单、固定资产核定都需要生成凭证。最后我们设计成了一个统一的生成凭证接口。还有一次，我们发现客户报表在查询 SQL、过滤条件、显示列等部分经常变，因此设计成一套可配置的报表系统，大大提高了系统可维护性。

需求分析是一个撒大网的过程，而不是姜太公钓鱼的过程。功能需求固然重要，非功能需求同样重要。我们在进行非功能需求的分析时，除了制订整体的原则以外，还要落实到各个具体的功能中，将这些功能所潜在的、特殊的非功能需求挖掘出来，提前进行分析设计，对于可行性不高的应及时与客户商讨，才能有效地避免日后存在的这些方面的风险。

我们应当怎样做需求确认

需求分析是一个我们与客户不断沟通的过程，这个过程就如同我们与老板的一次对话。老板把你叫去，给你交待了一大堆任务。我们首先是仔细聆听任务的内容，然后整理个一二三四。然后我们复述一遍老板的意思：“老板，我复述一遍，您看看我理解得对不对。首先，您要求我×××，然后×××，最后×××。”老板：“恩，就是这意思，你照着办吧。”之后，我们开始了我们的工作。这个复述的步骤相当重要，因为人与人的沟通最大的问题就是失真。由于人在知识水平、观点看法、性格特质的不同，听者常常会误解对方的意思。有了复述的步骤，误解就会立即被纠正，沟通得以顺畅。在需求分析中，这个复述的步骤就是需求确认。

但与一次简单的沟通不同，需求分析是一系列复杂的沟通过程，它涉及到许多人，谈论的是许多的事物。因此，一次简单的口头复述不足以满足需求分析的需要。因此，需求确认是一系列的确认过程，每次确认都可能需要与不同的人，在不同层次的确认。最终应当形成到纸面，形成文档性的东西，双方签字确认。这个过程中可以采用的一个好的方法就是原型法，最终产物应当是需求列表与需求规格说明书，最后结束于一场需求评审会，或者签字确认会。

需求列表

当我对无数失败项目的分析总结之后，得出的一个重要的结论就是我们的项目需要对需求的跟踪。大家想想，当一个项目持续数月，经过数轮的需求分析与设计，再经过数轮的需求确认与变更。用户、需求分析员、系统架构师、设计人员、开发人员，甚至测试，一个一个的角色像走马灯一样加入进来。需求开始变得模糊不清，软件设计的初衷开始偏离。开发人员不知道依据哪个标准开发，测试人员不知道依据哪个标准测试，甚至一些需求被人所遗忘。最终，等到软件交付的时候，客户说这不是他们所需要的，项目走向了失败。问题出在哪里呢？问题就出在，不论我们如何分析与设计，我们都要如实记录原始的需求，并以此来验证我们最终的软件。这个如实记录原始需求的文档，就是需求列表。

需求列表，又称之为需求跟踪表，是最原始的、用户对业务需求的描述。它不掺杂任何需求分析人员对业务需求的分析与设计，而是以简短扼要的语句，以业务人员的口吻表述的，今后要开发的这个系统应当提供给他们的各项功能。

首先，需求列表不掺杂我们对业务需求的任何分析与设计，这是需求列表的核心，也是它存在的意义。从用例模型到领域模型我们不难发现，它是一个分析与设计的过程。需求分析员对业务需求进行捕获、认识、理解以后，需要结合软件专业知识进行分析设计，还要听取系统架构师和设计师对需求可行性的分析，最后才整理和编写出用例模型。在这样一个过程中，随着业务需求复杂度的提高，以及各种技术分析的掺杂，最终的结果很有可能偏离原有的业务需求。这种偏离常常表现为对业务需求正确性与完整性的偏离，即需求已经变味儿了，或者某些需求项目缺失。需求列表就是那个最开初的、最完整的、正确的业务需求。用这样一个列表来开始我们的分析，最后用它来验证我们的设计，使之成为我们的分析设计之旅树立的一个正确的航标。有了这样一个航标，就可以使我们最终能够到达一个正确的彼岸。

其次，需求列表应当是站在业务人员的视角，对业务需求的简明扼要的描述。一个纷繁复杂的、业务庞大的管理系统，经过整理以后，被分解成一个一个的需求项目。每个需求项目是

一句简明扼要的话。简明扼要意味着清晰易懂；分解成需求项目意味着分解复杂问题为简单问题。每一次与业务人员讨论完业务需求以后，我们就整理成这样一个需求列表，使我们与客户的讨论都有一个清晰明了的讨论结果。当下一次与业务人员讨论时，我们拿出我们上一次讨论的需求列表，又使下一次的讨论有一个基点，使业务讨论能以演进的方式推进下去，提高我们的工作效率。

然而，需求列表中应当剔除那些客户对系统设计的内容。前面我们提到，客户，特别是那些对信息化建设有一定经验的客户，容易提一些对系统设计的期望，比如什么功能应当做成什么样子，功能界面是怎样的。客户提的这些意见，也许不是最佳的，我们经过深入的分析设计以后，可能会提出一些更加合理的方案。因此，这样内容不能成为我们验证系统功能的基石，因而不应当写入需求列表中。需求列表描述的更应当是客户对软件功能的意图，即客户使用这个功能所达到的目的，而不是功能的具体实现。这一点我们在后面通过具体实例详细说明。

最后，需求列表也不是一步到位的，而是经过由粗到细逐渐整理形成的。一个大的需求项目可以分解为多个细的需求项目，进而形成一个树状的需求列表。需求列表应当细分到什么程度呢？将系统需求描述清楚为宜。简单需求不需过多的细分，而复杂需求则需要尽量写细一些。同时，需求列表也是一个不断变化的过程，日后的每一次升级维护都需要不断增添和修改需求列表，使其与实际系统保持一致。

现在我举一个具体实例来看看需求列表是怎样编写的吧。这是一个公司内部的评审系统，它分为制订评审计划、执行评审、制作评审报告与问题跟踪四部分。经过初次与评审人员的业务讨论以后，我们整理出这样一个需求列表：

1．评审发起人填写一份评审计划，详细记录评审时间、评审内容、评审者、评审地点，制订评审组长，并预计评审工作量，发起一个评审任务。
2．评审者在收到邮件后，进入评审任务中，对评审内容进行评审，同时填写并提交各自的评审意见。
3．评审组长汇总所有的评审意见，并在评审会上依次过所有的评审意见，对评审意见进行修改或删除，填写问题跟踪，形成此次评审会上最终的评审意见及问题跟踪表。
4．评审组长制作评审报告，并形成评审结论，以邮件的形式通知所有评审者。
5．所有评审者对评审报告进行回复意见，如果都选择同意，评审组长关闭此次评审。
6．评审组长跟踪所有问题，并可以依次关闭每个问题。

当然，在这个需求列表中，客户提出了一些名词，比如评审计划、评审意见、评审组长等。我们在整理需求列表的同时，应当注意整理这些名称，弄清它的内涵外延，以及它们相互之间的关系、作用。这将为我们后面的领域模型分析提供素材。毫无疑问，这样的需求列表过于粗略。因而在后面的业务讨论中，我们逐项对它们进行了细化：

1．评审发起人填写一份评审计划，详细记录评审时间、评审内容、评审者、评审地点，制订评审组长，并预计评审工作量，发起一个评审任务。
1.1 评审时间应当分为数个阶段分别制订时间计划，如评审准备、评审会议、评审报告；
1.2 评审内容应当可以上传数个文件，分别描述文件的内容、作者、编写日期、版本号，供评审者下载与查看；

1.3 填写评审者时，选择一个评审者为评审组长，评审发起人不能是评审组长；
1.4 评审地点与预计评审工作量只需直接填写；

在我们后面的用例分析中，我们对这段需求列表进行了大量的分析设计。但这些都是设计与实现，它们会出现在后面的用例分析及其模型中，却不应出现在需求列表中。在后来的升级开发中，客户又提出了发邮件通知的功能。将该功能描述出来，并添加到需求列表中：

1.5 评审计划提交以后，以邮件的形式发送给每个评审者，通知该评审任务。

有了这样的需求列表，当需求分析工作完成时，我们将一项一项检查用例模型是否满足需求列表的内容；当软件开发完成时，我们将一项一项检查软件功能是否满足需求列表的内容；当用户验收时，我们同样使用需求列表，一项一项检查我们的软件是否满足用户需求。

快速原型法

常常听到许多朋友跟我埋怨，需求分析之难，就在于用户自身就常常弄不清楚自己的需求。起初在需求确认的时候说得好好的，一到软件上线的时候就不是那么回事了，这可没法整。但我们只要坐下来仔细分析就会发现，在需求分析的时候我们跟用户是在空对空地讨论问题。用户不是专业人士，他也搞不清楚软件到底会做成啥样，所以你跟他确认的时候他就点头了。但是，用户不是傻子，当你软件上线时，他拿到了实物了，知道软件做成啥样了，一旦不满意他就开始提变更了。所以，需求分析的症结就在与这个实物。

既然症结在此，毫无疑问，我们就应当在需求分析阶段拿出实物，用实物与用户确认需求，这就是快速原型法的基本思想。快速原型法，简称原型法（Prototyping），是 20 世纪 80 年代提出的一种从设计思想、工具、手段都全新的系统开发方法。它摒弃了那种一步步周密细致地调查分析，然后逐步整理出文字档案，设计开发，最后才能让用户看到软件结果的繁琐作法。当我们捕获了一批业务需求以后，就立即使用快速可视化工具开发出一个原型，交给用户去试用、补充和修改。再提出一些新的需求以后，再开发一版新的原型。原型法的关键就是这个快速开发。不用考虑性能、美观、可靠，原型的目的就是模拟客户的需求，与客户进行确认的。整个需求分析的过程就是“捕获需求->原型开发->确认需求->再捕获需求”的过程。

原型开发的快速与模拟到什么程度，是一对矛盾，我们要去把握。要快速开发，必然不可能和最终交付的软件系统一模一样，许多复杂问题被简化，非关键性流程被忽略，这就是所谓的模拟。因此，模拟到什么程度是关键，既能说明问题，又不耽误时间。根据我的经验，一般能拿出界面，并可以走通关键性流程就可以了。一些快速开发平台为快速原型法提供了可能。

当用户拿到原型可以自己操作时，需求研讨的气氛立即变得不太一样了。当用户享受原型给他们带来体验的快感时，需求被源源不断地被提出来。这时候的需求，就不再是枯燥无味的文字游戏，而是生动形象的图形界面。日后，如果项目采用迭代开发，让用户看着软件一点儿一点儿地成长，这又是多么美妙的体验啊。与此同时，你与用户的信任也在一步一步建立起来，软件风险在降低，项目将朝着正确方向前进。

快速原型法是美妙的，它给你与用户带来了从未有过的体验。但美妙的同时，也会带来一些的尴尬，不必要的误会，我们一定要注意。最常见的误会就是让用户将原型误以为最终交付的系统。开发一个系统需要持续数月，但你倒好，几天就搞定了，为什么还要在这个系统上投入大量资金呢？如果对方领导开始有这样的想法时，双方就开发费用进行的谈判就有一些不妙了。所以在给用户看到原型前，一定要跟用户解释清楚。

既然是原型，必要的校验、非正常操作的处理通通都被忽略。因此，当演示原型出错时，用户你可千万不要较真哟！这丑话可得说在前头，否则用户跟你较起真来，你在用户心目中的形象可就要大打折扣了。

总之，根据实际情况灵活运用原型法，可以更加顺畅地与用户确认需求。甚至在最后

编写需求规格说明书的时候，都可以将原型的截图放进去。都是与用户确认好的东西，又能提高需求规格说明书的准确与生动，何乐而不为呢？

需求规格说明书

曾经有项目组拿着用户编写的原始需求就开始开发，随后状况不断，一次令人崩溃的研发过程。拿着用户编写的原始需求，编写我们自己的需求规格说明书，之所以重要，就在于用户编写的原始需求，是脱离了技术实现，编写的一份十分理想的业务需求。理想与现实总是有差距，我们之所以要编写自己的需求规格说明书，就是要本着实事求是、切实可行的态度，去描述用户的业务需求。那些不可行的需求被摒弃，或者换成更加可行的解决方案。这就是需求规格说明书的重要作用。

从理论上讲，需求规格说明书（Requirement Specification）分为用户需求规格说明书和产品需求规格说明书。用户需求规格说明书是站在用户角度描述的系统业务需求，是用于与用户签字确认业务需求；产品需求规格说明书是站在开发人员角度描述的系统业务需求，是指导开发人员完成设计与开发的技术性文档。但是，我认为，用户需求规格说明书与产品需求规格说明书的差别并不大。领域驱动设计所提倡的就是要让用户、需求分析员、开发人员站在一个平台，使用统一的语言（一种混合语言），来表达大家都清楚明白的概念。从这个角度将，需求规格说明书就应当是一个，不区分用户需求规格说明书和产品需求规格说明书。

那么需求规格说明书怎么写呢？不同的公司、不同的人、不同的项目，特别是在需求分析中采用不同的方法，写出来的需求规格说明书格式都是不一样的。在这里，我给大家一个，采用 RUP 统一建模的方式分析需求，编写需求规格说明书的模板，供大家参考。

1．引言

1.1 编写目的

如题，描述你编写这篇文档的目的和作用。但最关键的是，详细说明哪些人可以使用这篇文档，做什么。需求规格说明书是用来做什么的？毫无疑问，首先供用户与开发公司确认软件开发的业务需求、功能范围。其次呢，当然就是指导设计与开发人员设计开发系统。当然，还包括测试人员设计测试，技服人员编写用户手册，以及其它相关人员熟悉系统。描述这些，可以帮助读者确定，阅读这篇文档是否可以从中获得帮助。

1.2 业务背景

描述业务背景，是为了读者了解与该文档相关的人与事。你可以罗列与文档相关的各种事件，也可以描写与项目相关的企业现状、问题分析与解决思路，以及触发开发该项目的大背景、政策法规，等等。

1.3 项目目标（或任务概述）

就是项目能为用户带来什么利益，解决用户什么问题，或者说怎样才算项目成功。前面提到过，这部分对项目成功作用巨大。

1.4 参考资料
参考资料的名称、作者、版本、编写日期。

1.5 名词定义
没啥可说的，就是文档中可能使用的各种术语或名词的定义与约定，大家可以根据需要删减。

2．整体概述
这部分是对系统整体框架性地进行描述。

2.1 整体流程分析
绘制的整体行动图，及其对它的说明。

2.2 整体用例分析
绘制的整体用例图，以及对每个用例的用例说明。如果项目比较大，存在多个子系统，可以将用例图改为构件图，详细描述每个子系统及其相互的接口调用。

2.3 角色分析
一个用例图，描述系统中所有的角色及其相互关系。在随后的说明中，详细说明每个角色的定义及其作用。

这部分还可以根据项目需要编写其它的内容，如部署方案、网络设备、功能结构、软件架构、关键点难点技术方案，等等。

3．功能需求
3.1 功能模块（子系统）
一个一个描述系统中的每个功能模块（或子系统），即整体用例分析中的每个用例。这部分是需求规格说明书最主要的部分。

3.1.1 用例图
绘制该模块的用例图（详见《功能角色分析与用例图》）。

3.1.2 用例说明
对用例图中的每个用例编写用例说明（详见《用例说明》）。

3.1.3 领域模型
为用例绘制领域模型，并编写领域模型说明，对每个实体进行说明。对实体的说明包括对实体的定义、属性说明、行为说明、实体关系说明等等。如果实体间关系复杂，还要使用对象图说明实体关系的所有情况（如《领域驱动设计》中的描述）。

4．非功能需求
这里描述的是软件对非功能需求的一般要求，即整体设计原则。那些与具体功能相关的非功能需求应该放在用例说明的“非功能需求”部分（详见《非功能需求》）。

5．接口需求
如果项目涉及到与外部系统的接口，则编写这部分需求。
5.1 接口方案
详细描述采用什么体系结构与外部系统的接口。
5.2 接口定义
接口的中文名、英文名、功能描述、参数、返回值、使用者、使用频率，等等。

评审与签字确认会

时间过得真快，经过一系列需求研讨、需求分析和整理确认，我们整理出了需求列表，编写出了需求规格说明书，一切似乎该到结束需求分析阶段的时候了。但是，敏捷大师的一句话让我们彻底心凉到了骨头里。敏捷大师说了，我们不可能在需求分析阶段完成所有的需求分析工作，它将延续到设计、开发，甚至测试阶段。

一直以来，我对这句话非常困惑。既然需求分析阶段不能完成所有的需求分析工作，那么完成多少才算结束呢？80%？60%？或者更少？大师没有给出一个标准。大师就是大师，生活在太空里的，我们慢慢理解吧。经过多年的实践，我慢慢理解了。我们说这种需求分析工作不可能完全完成，或者说日后用户的需求会变，其实并不是毫无规律可循的。通常，用户对需求的变更只发生在某些固定的范围内，弄清楚了这些范围，我们的问题就迎刃而解了。

1. 整体需求不变，具体细节变化。我们说需求是分层次的，整体框架、功能模块、每个操作的细节。如果用户变更到了将整个框架都推翻了，这个项目就别做了。所以整体框架是必须在需求分析阶段完成的，是日后不可能改变的。功能模块可能要变，但通常是某个部分在变，而更多的是那些具体操作的细节在变。

2. 界面风格与操作易用性是最容易发生变更的。我们说用户看到软件以后不满意，其实主要是对界面风格与操作性不满意，而不是软件功能。界面不够美观，操作不方便，不符合用户的操作习惯，都是造成用户不满意的地方。

3. 增加其它功能。软件是对现实的模拟，而现实也是复杂多变的。我们与用户在进行业务流程分析时，也许一些流程没有考虑到，或者还有特殊情况需要处理。这些是客户要求增加功能的主要动因。

经过以上分析，需求分析阶段要做到什么程度就可以清楚了：整体框架与功能模块必须确定下来，至于各个功能模块下的具体操作，尽量做，能到什么程度先到什么程度。至于界面风格与操作性，我们可以在日后迭代开发的每个迭代期，拿出样品以后再与用户确认。

OK，万事俱备只欠东风，当所有工作都完备以后，我们的需求分析工作开始进入最后收尾的阶段。我们说，需求分析阶段的产出物是需求列表与需求规格说明书，而最终结束的里程碑无疑就是需求评审会了，或者说与用户的签字确认会。

需求评审会的主要目的就是确认需求，以便以此开始我们的设计开发工作。从理论上说，需求评审会应当由用户代表，与项目经理、需求分析员、系统架构师、设计人员、测试人员、

QA 经理，还有公司相关领导参加。但实际上，让如此多不同角色的人聚集在一起开会是不现实的。因此，我们可以将需求评审会分为内部评审会与外部评审会两部分来开比较现实。

处理外部问题，必先要从内部统一思想。先召开一个内部评审会，听听系统架构师、设计人员、测试人员、QA 经理对需求分析工作的意见，然后由领导讲讲话，布置一下后面的工作，是十分有必要的。按照我的经验，系统架构师这时的作用相当重要，他应当仔细阅读需求，仔细思考技术是否可行，以及预测该系统是否能够达到用户方领导对该项目制订的目标。如果答案是否定，立即进行调整。

最后就是与用户的外部需求评审会了。外部需求评审会，也可称为签字确认会议，就是与用户就需求规格说明书进行评审，最后签字确认。用户签过字的东西，不可能完全抑制住用户的变更，但至少从很大程度上抑制住了用户的大改。然而，在召开外部需求评审会之前，我们建议大家就需求规格说明书，先与各个单位或部门的用户代表讨论并确定下来，避免在最终的签字确认会上出现分歧，影响工作进度。毕竟大家都不容易，工作一大堆，聚在一起不容易。

经过数月的分析讨论，最终在一片和谐的气氛中，双方领导在需求规格说明书上签字，项目开始进入一个新的轮回。在这个轮回中，是焦头烂额、不胜其苦，还是如履薄冰、最终顺利交付，是与许多因素有关的。但我想说，一份高质量的需求分析必定起到决定性的作用，必定为日后的软件开发扫清了许多许多的地雷。

后　序

有朋友说：老范，拜托你能不能精简点儿，看多了会头疼。说真的，我写给大家的都是干货，没有什么口水来占据篇幅，可以说一上来就进入主题。但是，我认为，问题不说清楚不说透，不如不说。文中一段一段的，都是在用真正的实例来说明深刻的道理，让人明白，知道如何去使用。这，就是我写作的初衷。

参考资料

1. Brett D. McLaughlin, Gary Pollice & David West. Head First Object-Oriented Analysis & Design. O'Reilly Media, Inc. ISBN: 978-7-5641-0743-7

2. 徐峰. 软件需求最佳实践：SERU 过程框架原理与应用. 电子工业出版社. ISBN:9787121073953. 2008-10
3. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley. August 20, 2003