《软件工程实践》平时作业

学　　院：	计算机与信息技术学院
专　　业：	计算机科学与技术
班　　级：	计科 1602
姓　　名：	麻锦涛
学　　号：	16281262

日期：2019.6.23

对软件本质的认识

麻锦涛 16281262 计科1602

软件是一种固化的思维，这一点决定了许多的事情。从特质上来看，既然软件是固化的思维，那就必然同时具备思维以及思维所承载之物之特质。既然思维自身的特质是复合的，那么作为固化思维的软件，其特质必然也是复合的：这也就意味着在软件这一大的范畴里，两种矛盾的说法同时成立，并不是什么值得惊讶的事情。只要思维承载的东西蕴含着彼此对立的东西，那么两种对立的观点同属于软件这一范畴，并且同时争取，一点也不稀奇。这很可能是大家吵来吵去的一个根本原因，因为我们总是喜欢用自己的经历来定义软件是什么以及判断标准，但如果这种经历来自完全不同的两个领域，并且互相矛盾，那就只能吵架。实际上只有基于软件是一种思维这样特质推导出来的东西才更有普适性。软件就是按既定要求进行的运算、储存、读取、传输。既定要求的输入包括键盘输入、端口输入、网络传输，其他方式输入。运算结果的输出包括屏幕显示、打印输出、网络传输，其他方式输出。

信息作为一种客观存在，它一直都在积极地发挥着人类意识或没有意识到的重大作用。科学技术在近两个世纪所取得的空前进步，使人们终于认识到，信息是与物质和能源可以相提并论的用以维系人类社会存在及发展的三大要素之一。信息给人类社会带来了多元化的变化，吃喝住行等等方面，丰富了人民的生活。信息是非常重要的，信息首先重要在它对决策起作用，能够集中精力在更能够出业绩、出成效的方向，而不是走错路、浪费时间、白费力气。信息其次重要在它让我们更完全，不盲信。哪怕你明知真实的信息却利用它去欺骗了别人（利用信息不对称获取利益），你也不会是在这种相对关系中，那个更傻的人。然而，我们是不是都需要让自己不至于去落入那种傻欺的境地。

同时，信息对于国家的发展具有重大的意义，在工业发达国家，信息经济正迅速发展成为指导现代经济的主要经济，并且对世界各国的经济发展都产生了重大的影响和推进。近些年来，我国信息产业的发展异常迅速，信息经济产值的快速增长已很好地证明了信息在经济发展中所起的巨大作用。

信息首先重要在它对决策起作用，能够集中精力在更能够出业绩、出成效的方向，而不是走错路、浪费时间、白费力气。信息其次重要在它让我们更完全，不盲信。

因为通过学习和摸索做到这样，在跟不同行业以及相关行业的人士交流过程中，我了解到一些极为 有用的信息，这种信息于我又在别处进一步得到了验证，这些有用的信息配合相关的工作经历，可能，会对我将来的工作产生很大的影响和帮助。 而这些信息，不管是由经历换来，还是由交流换来，都是我愈加看重的珍贵“财富”的一部分。

三年来我们对于课程的学习，厌倦过，焦虑过，但既然当初选择了它，我们就没有退路。可以说计算机这个专业是一个很有前景的学科，是不是随意轻视任何一门课程。干一行、爱一行，既然选择了它，就要对它有始有终。也许现在我们的未来还是一个未知数，但我们有目标有理想，我们不可能一帆风顺的到达彼岸，所以脚踏实地才是最根本的原则。

当然成为一名工程师也不是一件容易的事，并不是一朝就能练成的，这需要

我们规划好大学时光。

首先要正确给自己定位，确定自己的短期目标和长期目标。短期目标可以是自己在大学四年的学习，甚至细化至每个学期的学习。长期目标就是自己的理想，成为一名工程师，这是自己的终极目标，每个短期目标都是为它服务的。

抓住在学校里的各种实践的机会，要为自己积累经验，就业时经验比什么都有用。

对于一个立志成为架构师的人，最重要的其实是强烈的好奇心和学习精神.没有比强烈的好奇心和学习精神更好的武器了，它是成功的工程师乃至在各行各业的成功者们永攀高峰的源泉和动力所在。

打好专业基础。牢记学习编程的秘诀是编程编程再编程 。

软件危机＆抽象层次

麻锦涛 16281262 计科 1602

一、软件危机

软件危机是指计算机软件开发和维护过程中所遇到的一系列严重问题。
典型表现有：
（1）对软件开发成本和进度的估计常常很不准确
（2）软件产品的质量往往靠不住
（3）用户对已完成的软件系统不满意的现象经常发生
（4）软件常常是不可维护的
（5）软件中没有适当的文档资料
（6）软件成本在计算机系统总成本所占的比例逐年上升
（7）软件开发生产率提高的速度，往往跟不上计算机应用迅速普及深入的趋势
原因：
（1）软件本身独有的特点确实给开发和维护带来了困难
（2）与软件开发和维护的许多错误认识和做法的形成有关
（3）程序只是完整软件的一个组成部分
（4）轻视是一个最大的错误

二、抽象层次

开发中的抽象

在软件开发里面，最重要的抽象就可能是分层了。分层随处可见，例如我们的系统就是分层的。最早的程序是直接运行在硬件上的，开发成本非常高。然后慢慢开始有了操作系统，操作系统提供了资源管理、进程调度、输入输出等所有程序都需要的基础功能，开发程序时调用操作系统的接口就可以了。再后来发现操作系统也不够，于是又有了各种运行环境（如 JVM）。

编程语言也是一种分层的抽象。机器理解的其实是机器语言，即各种二进制的指令。但我们不可能直接用机器语言编程，于是我们发明了汇编语言、C 语言以及 Java 等各种高级语言，一直到 Ruby、Python 等动态语言。
开发中，我们应该也都听说过各种分层模型。例如经典的三层模型（展现层、业务逻辑层、数据层），还有 MVC 模型等。有一句名言：“软件领域的任何问题，都可以通过增加一个间接的中间层来解决”。分层架构的核心其实就是抽象的分层，每一层的抽象只需要而且只能关注本层相关的信息，从而简化整个系统的设计。

其实软件开发本身，就是一个不断抽象的过程。我们把业务需求抽象成数据模型、模块、服务和系统，面向对象开发时我们抽象出类和对象，面向过程开发时我们抽象出方法和函数。也即是说，上面提到的模型、模块、服务、系统、类、对象、方法、函数等，都是一种抽象。可想而知，设计一个好的抽象，对我们软件开发有多么重要。

总结

软件开发并不是仅仅只依靠抽象能力就能完成的，最终我们还是要把我们抽象出来的架构、模型等，落地到真正的代码层面，那就还需要逻辑思维能力、系统分析能力等。

对软件局部化的认识

麻锦涛 16281262 计科 1602

软件工程的原则有抽象、信息隐藏、模块化和局部化等。推迟实现是软件方法学的基本指导思想。软件开发过程应该理性地“推迟实现”，即把逻辑设计与物理设计清楚地划分开来，尽可能推迟软件的物理实现。对于大中型的软件项目，在软件开发过程中，如果过早而仓促地考虑程序的具体实现细节，可能会导致大量返工和损失。为了正确、高效地进行软件开发，必须在开发的前期安排好问题定义、需求分析和设计等环节，进行周密、细致的软件实现的前期工作，并明确规定这些环节都只考虑目标系统的逻辑模型，不涉及软件的物理实现。

逐步求精。逐步求精（也称逐步细化）是基于承认人类思维能力的局限性提出的，一般认为人类思维与理解问题的能力限制在 7±2 大小的范围内，求解一个复杂问题采用有条理的从抽象到具体的逐步分解与细化方法和过程进行。这是人类把复杂问题趋于简单化控制和管理的有效策略。逐步求精（细化）解决复杂问题的策略是软件工程方法学中的一项通用技术，与分解、抽象和信息隐蔽等概念紧密相关。

分解与抽象。分解是把复杂问题趋于简单化处理的有效策略。论证分解，即

“分而治之”的有效性。若将一个复杂问题分解成若干容易解决的小问题，就能够减少解决问题所需要的总工作量。分解必须是科学而合理的，否则可能会增加解决问题的难度和工作量。

抽象是人类在认识或求解复杂问题的过程中，科学而合理地进行复杂系统分解的基本策略之一。抽象是把一些事物（状态或过程）中存在的相似的方面（忽略它们的差异）概括成“共性”的。抽象的主要思想是抽取事物的本质特性，而暂不考虑它们的细节。这是一种分层次的渐进过程。软件工程方法学中广泛采用分层次的从抽象到具体的逐步求精技术。建立模型（建模）是软件工程常用的方法和技术之一。软件工程中，整个软件开发过程需要建模，软件开发过程的各个阶段也需要建模。不同的软件开发方法的最主要的区别表现在它们的模型不同。所以，软件开发过程的一系列模型的建立标准、描述形式、应用规范等是软件开发方法最核心的研究内容。

信息隐蔽。信息隐蔽是指使一些关系密切的软件元素尽可能彼此靠近，使信息最大限度地局部化。软件模块中使用局部数据元素就是局部化的一个例子。信息隐蔽的指导思想始终贯穿在软件工程的面向过程、面向功能和面向对象的软件开发方法的发展中。

质量保证。质量保证是为保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动。质量保证是面向消费者的活动，是为了使产品实现用户要求的功能，站在用户立场上来掌握产品质量的。同样道理，这种观点也适用于软件开发。软件质量保证要求软件项目的实施活动符合产品开发中的对应的需求、过程描述、标准及规程。质量保证的基本思路是提倡“预防”而不是事后“补救”，强调“全过程控制”和“全员参与”。 软件质量是“软件与明确和隐含地定义的需求相一致的程度”。更具体地说，软件质量是软件与明确叙述的功能和性能需求、文档中明确描述的开发标准以及任何专业开发的软件产品具有的隐含特征相一致的程度。

基本原则

美国著名的软件工程专家 B. W. Boehm 综合软件领域专家的意见，并总结了多家公司开发软件的经验，提出了软件工程的 7 条基本原则。Boehm 认为，这 7 条基本原则是确保软件产品质量和开发效率的原理的最小集合。它们是相互独立的，是缺一不可的最小集合；同时，它们又是相当完备的。
分阶段的软件开发。根据这条基本原则，可以把软件生存周期划分为若干个阶段，并相应地提定出切实可行的计划，然后严格按照计划对软件开发与维护进行管理。在软件开发与维护的生存周期中，需要完成许多性质各异的工作。Boehm 认为，在整个软件生存周期中应指定并严格执行 6 类计划：项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划、运行与维护计划。

坚持进行阶段评审。统计结果显示：在软件生命周期各阶段中，编码阶段之前的错误约占 63%，而编码错误仅占 37%。并且，错误发现得越晚，更正它付

出的代价就会越大，要差 2～3 个数量级甚至更高。因此，软件的质量保证工作不能等到编码结束以后再进行，必须坚持进行严格的阶段评审，以便尽早地发现错误。

实行严格的产品控制。实践告诉我们，需求的改动往往是不可避免的。这就要求我们要采用科学的产品控制技术来尽可能达到这种要求。实行基准配置管理（又称为变动控制），即凡是修改软件的建议，尤其是涉及基本配置的修改建议，都必须按规定进行严格的评审，评审通过后才能实施。基准配置指的是经过阶段评审后的软件配置成分及各阶段产生的文档或程序代码等。当需求变动时，其他各个阶段的文档或代码都要随之相应变动，以保证软件的一致性。

采用先进的程序设计技术。先进的程序设计技术既可以提高软件开发与维护的效率，又可以提高软件的质量和减少维护的成本。

明确责任。软件是一种看不见、摸不着的逻辑产品。软件开发小组的工作进展情况可见性差，难于评价和管理。为更好地进行管理，应根据软件开发的总目标及完成期限，尽量明确地规定开发小组的责任和产品验收标准，从而能够清楚地审查。

开发小组的人员应少而精。开发人员的素质和数量是影响软件质量和开发效率的重要因素，应该少而精。事实上，高素质开发人员的工作效率比低素质开发人员的工作效率要高几倍到几十倍，开发工作中犯的错误也要少得多。

不断改进开发过程。软件过程不只是软件开发的活动序列，还是软件开发的最佳实践。在软件过程管理中，首先要定义过程，然后合理地描述过程，进而建立企业过程库，并成为企业可以重用的资源。对于过程，要不断地进行改进，以不断地改善和规范过程，帮助提高企业的生产效率。

遵从上述七条基本原则，就能够较好地实现软件的工程化生产。但是，它们只是对现有的经验的总结和归纳，并不能保证赶上技术不断前进发展的步伐。因此，我们不仅要积极采纳新的软件开发技术，还要注意不断总结经验，收集进度和消耗等数据，进行出错类型和问题报告统计，评估新的软件技术的效果，指明必须着重注意的问题及应该采取的工具和技术。

生命周期的认识根源和实践根源

麻锦涛 16281262 计科 1602

同任何事物一样，一个软件产品或者软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，一般称为软件生命周期。软件生命周期是由软件定义，软件开发和运维（也称为软件维护）3 个时期组成的，每一个时期又可以划分为若干个阶段。

软件定义时期的任务是：确定软件开发工程必须完成的总体目标，确定工程的可行性，导出实现工程目标应该采用的策略及系统必须完成的功能；估计完成该项工程需要的资源和成本，并且制定工程进度表。这个时期的工作通常又称为系统分析，由系统分析员负责完成。软件定义时期又可以进一步划分为 3 个阶段，

即问题定义，可行性研究和需求分析。

开发时期具体设计和实现在前一个时期定义的软件，它通常由下述 4 个阶段组成：总体设计，详细设计，编码和单元测试，综合测试。

其中前两个阶段又称为系统设计，后两个阶段又称为系统实现。维护时期的主要任务是使软件持久地满足用户的需求。具体来说，当软件在使用过程中发现错误时应该加以改正；当环境改变时应该修改软件以适应新的环境；当用户有新的要求时，应该及时改进软件以满足用户的新需求。通常对维护时期不再进一步划分阶段，但那时每一次维护活动本质上都是一次压缩和简化了的定义和开发过程。

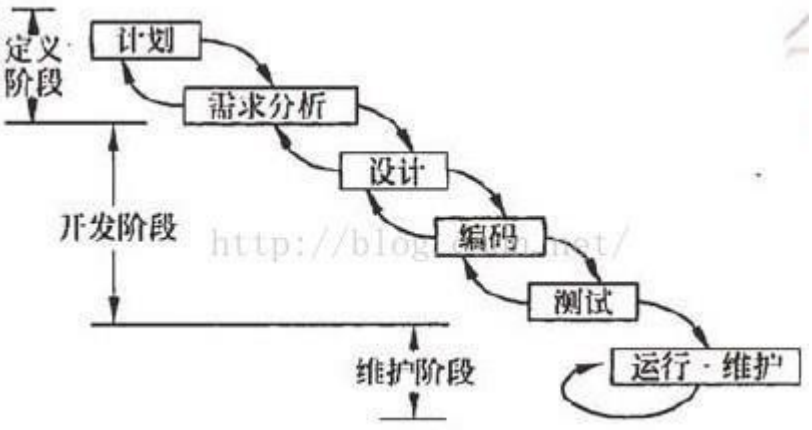
总的来说，如果按照进一步的划分，软件的生命周期可划分为较小的问题定义，可行性研究，需求分析，总体设计，详细设计，编码和单元测试，综合测试，软件维护 8 个阶段。

软件生命周期模型：

软件生命周期模型是指人们为了开发更好的软件而归纳总结的软件生命周期的典型实践参考。为了使规模大、结构复杂和管理复杂的软件开发变的容易控制和管理，人们把整个软件生命周期划分为若干阶段，使得每个阶段有明确的任务，整理出软件生命周期模型。软件生命周期模型主要包括了：瀑布模型、增量模型、原型模型、螺旋模型、喷泉模型等，详细信息如下：

(一) 瀑布模型

瀑布模型是一个经典的软件生命周期模型，一般将软件开发分为可行性分析（计划）、需求分析、软件设计（概要设计、详细设计）、编码（含单元测试）、测试、运行维护等几个阶段，如图所示



瀑布模型是一种广为使用的生命周期模型，有时候软件生命周期模型指的就是瀑布模型，它主要有以下特点：

1．阶段间具有顺序性和依赖性

必须等前一阶段的工作完成之后才能开始后裔阶段的工作；建议阶段的输出

文档就是后一阶段的输出文档。只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。

2．推迟实现的观点

在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。清楚的区分逻辑与物理设计，尽可能地推迟程序的物理实现。

3．质量保证的观点

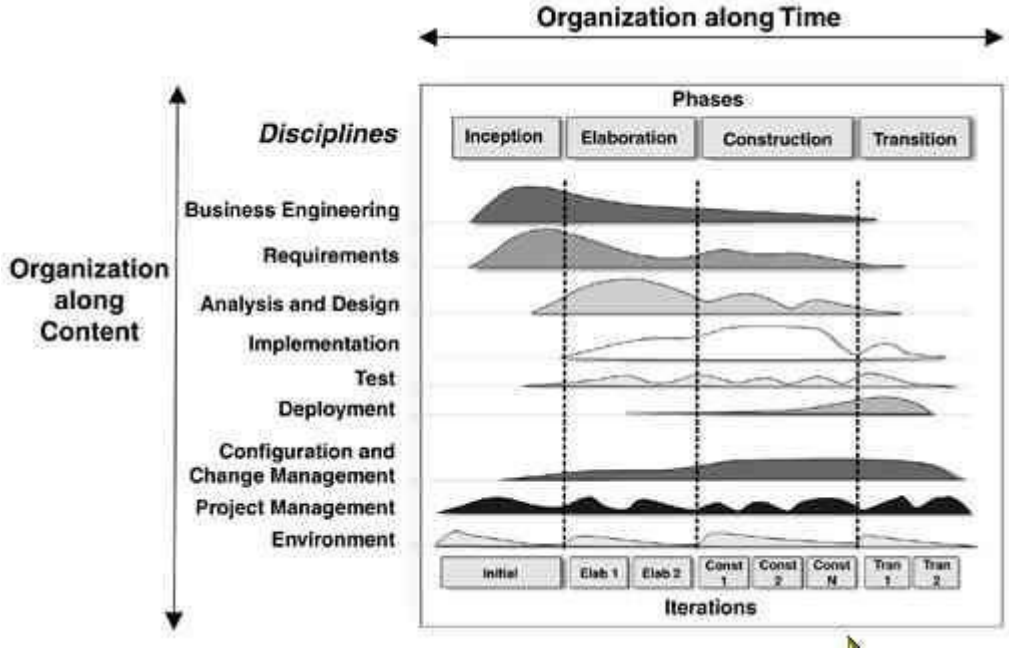
为了保证软件的优质以及高产，每个阶段都必须完成规定的文档，没有交出合格的文档就没有完成该阶段的任务，它有利于开发人员之间的交流，同时也是维护时期的重要依据。

每个阶段结束之前都要对所完成的文档进行评审，以便于尽早地发现问题，改正错误。及时审查时保证软件质量、降低软件成本的重要措施。

由于瀑布模型可以强迫开发人员使用规范的方法（例如，结构化技术）；严格规定了每个阶段必须提交的文档；要求每个阶段交出的所有产品都必须进过质量保证小组的仔细验证。

(二) 增量和迭代模型

增量模型也称为渐增模型，使用增量模型进行开发时，把软件产品作为一系列的增量构建来设计，编码、集成和测试。每个构建由多个相互作用的模块构成，并且能够完成特定的功能。增量迭代是 RUP 统一过程常采用的软件开发生命周期模型.增量和迭代有区别，但两者又经常一起使用.所以这里要先解释下增量和迭代的概念.假设现在要开发 A,B,C,D 四个大的业务功能,每个功能都需要开发两周的时间.则对于增量方法而言可以将四个功能分为两次增量来完成,第一个增量完成 A,B 功能,第二 次增量完成 C,D 功能;而对于迭代开发来将则是分两次迭代来开发,第一次迭代完成 A,B,C,D 四个基本业务功能但不含复杂的业务逻辑,而第二个功能再逐 渐细化补充完整相关



的业务逻辑.在第一个月过去后采用增量开始时候 A,B 全部开发完成而 C,D 还一点都没有动;而采用迭代开发的时候 A,B,C,D 四个的 基础功能都已经完成.

RUP 强调的每次迭代都包含了需求,设计和开发,测试等各个过程, 而且每次迭代完成后都是一 个可以交付的原型.迭代不是并行,在每次迭代过程中仍然要遵循需求->设计->开发的瀑布过程.迭代周期的长度跟项目的周期和规模有很大的关 系.小型项目可以一周一次迭代,而对于大型项目则可以 2-4 周一次迭代.如果项目没有一个很好的架构师,很难规划出每次迭代的内容和要达到的目标,验证相 关的交付和产出.因此迭代模型虽然能够很好的满足与用户的交付,需求的变化,但确是一个很难真正用好的模型. 就对风险的消除上,增量和迭代模型都能够很好的控制前期的风险并解决.但迭代模型在这方面更有优势.迭代模型更多的可以从总体方面去系统的思考问题,从最早就可以给出相对完善的框架或原型, 后期的每次迭代都是针对上次迭代的逐步精化.

业界比较标准的增量模型往往要求在软件需求规格说明书全部出来后后续的设计开发再进行增量.同时每个增量也可以是独立发布的小版本.由于系统的总体设计 往往对一个系统的架构和可扩展性有重大的影响,因此我们推荐的增量最好是在架构设计完成后再开始进行增量,这样可以更好的保证系统的健壮性和可扩展性.

(三) 快速原型模型

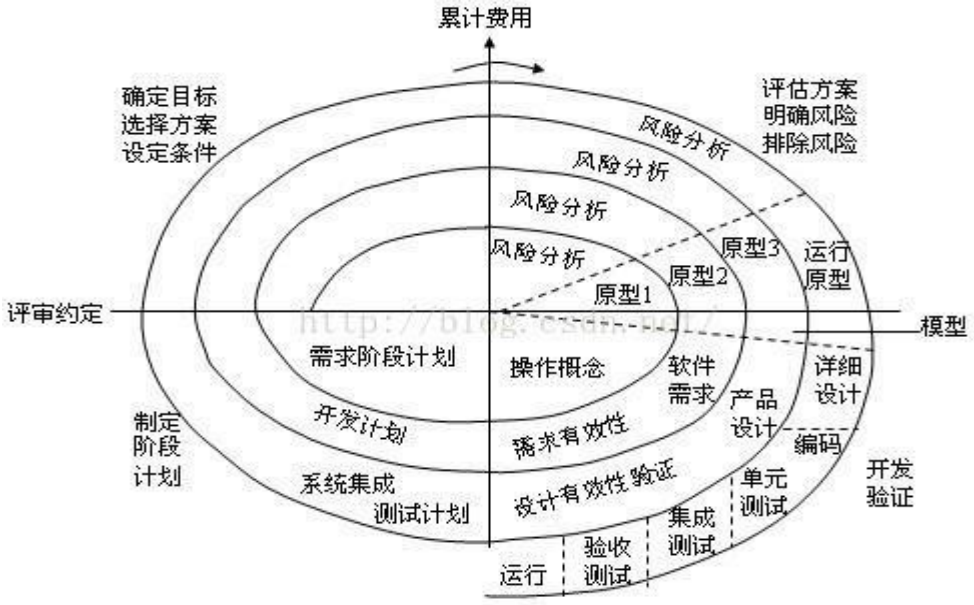
特点：快速建立起能够在计算机上运行的程序（最终产品功能的一个子集）。

优点： 软件产品的开发基本上是线性的

缺点： 必须迅速地构建原型然后根据用户意见循序的修改原型 适用范围：用户需求不明确，需要通过构建原型来清楚的了解用户的真实需求。

(四) 螺旋模型

螺旋模型是一个演化软件过程模型，将原型实现的迭代特征与线性顺序（瀑布）模型中控制的和系统化的方面结合起来。使得软件的增量版本的快速开发成为可能。在螺旋模型中，软件开发是一系列的增量发布。螺旋模型的整个开发过程如图所示



图中的螺旋线代表随着时间推进的工作进展；开发过程具有周期性重复的螺旋线形状。4 个象限分别标志每个周期所划分的 4 个阶段：制定计划、风险分析、实施工程和客户评估。螺旋模型要点：统一了瀑布模型与原型模型，与增量模型相似，更强调风险分析。

1.软件分多个版本开发，每个版本就是一次螺旋。

2.每个版本按照这样的顺序进行：

1）确定软件目标，选取定实施方案，弄清项目开发的限制条件；

（图中左上象限）

2）分析所选取方案，考虑如何识别和消除风险；（图中右上象限）

3）实施软件开发；（图中右下象限）

4）评价开发工作，提出修正建议，调整计划。（图中右下象限、左下象限）

3.需求不是一次获取和实现的，通过多个螺旋来完善。

4.计划也不是一次成型的，每次螺旋都需要调整。

该模型在实际工作中实用性还是相当高的，但可能是该模式很多资料都说得不太清楚，让很多人会有一些误解。

几种生命周期模型间的比较

软件生命周期模型规定了把生命周期划分成哪些阶段及各个阶段的执行顺序，因此，也称为过程模型。所以几种生命周期模型都规定了各个阶段应该执行的顺序，但是它们各自规定的生命周期中应该执行的顺序是不一样的。另外，不同的生命周期模型它们还有各自独特的要求，比如：瀑布模型的思想是通过完整的顺序性和前一阶段输出作为后一阶段的输入的思想，并尽可能推迟具体实现来达到软件开发的优质。快速原型分析的第一步便是快速建立一个能够反映用户主

要需求的原型系统，让用户在计算机上使用它，通过实践来了解目标系统的概貌。增量模型是在完成了总体设计之后，一个构建一个构建逐个完成，并还强制要求了每一个构建的设计文档的说明，以及风险分析和测试。螺旋模型，实际上是为了降低软件风险，在每一个阶段都
设置了风险分析，然后通过逐个添加构建的方法来完成软件的开发。

可行性研究的地位和作用：

可行性研究阶段的主要作用是对在问题定义阶段所确定的问题是否有行得通的解决方案进行研究和分析。系统分析员需要进行一次大大压缩恶化简化了的系统分析和设计过程，也就是在较为抽象的较高层次进行分析和设计。

可行性研究的任务并不是去具体解决问题，而是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。可行性研究的结果是客户做出是否继续进行这项工程的决定的重要依据，通常只有那些投资可能缺德较大消息的那些工程才值得继续进行下去。可行性研究的意义在于对于那些不值得继续进行研究的项目或者工程应当及时停止，因为在再可行性研究后的阶段需要投入更多的人力和物力，及时终止不值得投资的工程项目，可以避免更大的浪费和损失。

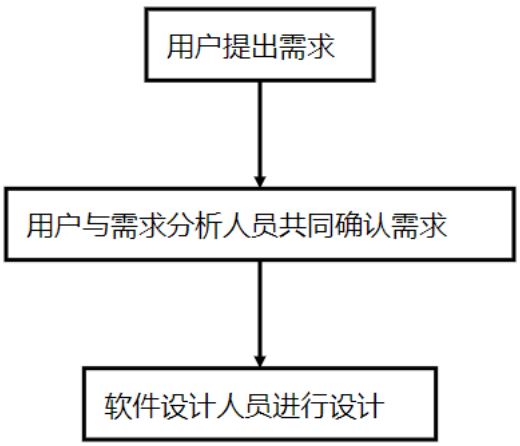
需求规格说明＆运行概念主要描述哪几个方面

麻锦涛 16281262 计科 1602

一、需求规格说明

软件 project 里对于软件项目各个过程须要输出的文档都有明白的定义，可是每一个方法论又是不太一样，比方 cmmi 和敏捷的类似性就不大。当提到用户故事那应该是敏捷而不是 cmmi，当说起需求规格说明书就应该不是敏捷的。

叫什么名字事实上我不太关注。抛开这些定义，一个真实的需求分析到软件设计的过程是如何呢？



cmmi 在上述的第一步，会输出《用户需求规格说明书》，第二步会输出《软

件需求规格说明书》，第三步会输出《软件概要设计说明书》。
当然，用户需求称为客户需求也行，软件需求规格说明叫做软件规格需求说明都能够。这些我不关注。

我想表达的是，《用户需求规格说明书》是业务人员写的。《软件需求规格说明书》是技术人员写的，假设是甲乙方的项目，那就是甲方的技术部写。而《软件概要设计说明书》是乙方输出。

因此。假设假设我并非甲方人员。让我写《需求规格说明书》，不管是用户需求还是软件需求，都是不合适的。回到主题。明显须要我写的仅仅能是《软件需求规格说明书》。可是项目计划里简简单单的写上需求规格说明书是不恰当的。

软件需求规格说明书有什么

先说说没有什么：一定不涉及技术元素，比方选择什么技术路线。选择什么编程语言等等。

也没有什么数据库的设计。
一定要技术人员和用户都看得懂。这样用户能够依据这个来验收，技术人员也能够依据它来进行概要设计。
当然。也要依据用户的需求书来验收，可是毕竟它和软件系统脱离的，有些关于系统操作类的精确事项无法描写叙述到。

举个样例，比方有四个不同的业务人员各自提出需求。他们之间的需求肯定有类似的地方，也有不同的地方。那么《软件需求规格》就须要把同样点归并，不同点如何体现编写出来，而且与四个业务人员确认，这样合并能否满足了这个需求。假设有一些现存的老系统，那么也须要说明对老系统进行什么样的改造（非技术类说明）。

二、运行概念主要描述哪几个方面

1.《运行概念说明》(OCD)从以下几方面描述一个建议的系统：说明它能满足用户什么需要，它与现有系统或过程的关系，以及它的使用方式等。

2.OCD 旨在需方、开发方、支持方和用户代理之间对所建议的系统的运行机理取得共识。取决于使用的目的,OCD 可专注于向开发者表述用户的需求；或专注于向用户或其他感兴趣的对象表达开发者的思路。术语“系统”也可理解为系统的一部分。

运行概念说明的正文的格式如下：

1 引言

本章应分为以下几条。

1.1 标识

本条应包含本文档适用系统和软件的完整标识，(若适用)包括标识号、标题、缩略词语，版本号和发行号。

1.2 系统概述

本条应简述本文档适用的系统和软件的用途。它应描述系统和软件的一般特性；概述系统开发、运行和维护的历史；标识项目的投资方、需方、用户、开发方和支持机构；标识当前和计划的运行现场；列出其他有关的文档。

1.3 文档概述

本条应概括本文档的用途和内容，并描述与它的使用有关的保密性或私密性要求。

2 引用文件

本章应列出本文档引用所有文档的编号、标题、修订版本和日期，本章也应标识不能通过正常的供货渠道获得的所有文档的来源。

3 现行系统或状态

本章分以下几条描述现行系统或状态。

3.1 背景、目标和范围

本条应描述现行系统或状态的背景、任务或目标和范围。

3.2 运作策略和约束

本条应描述适用于现行系统或状态的运作策略和约束。

3.3 现行系统或状态描述

本条应给出现行系统或状态的描述，标识出不同运行状态和方式的差异(例如：常规、维护、训练、降级、紧急事件和备选场点)。状态和方式之间的区别是任意的，可以仅按状态描述系统，也可以仅按方式、方式中的状态、状态中的方式或其他有用的方式描述系统。如果系统不是以多种状态和方式运行，本条应如实陈述，而不需要进行人为的区分。描述包括以下内容：

a.运行环境及其特性；

b.主要的系统部件及这些部件之间的互连；

c.与外部系统或过程的接口；

d.现行系统的能力/功能；

e.描述输入、输出、数据流、手工和自动处理的图表和相应的说明，使用户能够充分理解现行系统和状态；

f.性能特性，如：速度、吞吐量、容量和频率；

g.质量属性，例如：可靠性、可维护性、可用性、灵活性、可移植性、易用性和有效性；

h.安全性、保密性、私密性和紧急情况下运行连续性的规定。

3.4 用户或相关人员

本条应描述系统用户的类型或当前状态下所涉及到的人员(若适用)，包括：组织结构、培训/技能、职责、活动及彼此之间的交互。

3.5 支持概念

本条应简述现行系统的支持概念，若适用于本文档，应包括：支持机构，设施，设备，支持软件，修复/替换准则，维护等级和周期，存储、分发和供应方法。

4 变更理由和种类

本章应分以下几条。

4.1 变更理由

本条应：

a.描述需要新建或修改系统的用户需求、威胁、任务、目标、环境、接口、人员或其他因素的新的或修改的方面；

b.简述现行系统或现状不能满足上述因素的不足和约束。

4.2 所需变更的说明

本条应简述能满足 4.1 中所标识因素的新建的或修改的能力/功能、处理、接口或其他所需的变更。

4.3 变更之间的优先级别

本条应指出所需变更之间优先级别。例如：把每个变更标识为本质的、期望的或可选的，对期望的和可选的变更给出优先级别。

4.4 考虑过但未纳入的变更

应指出曾考虑过，但未包含在 4.2 中的变更，说明未包括它们的理由。

4.5 假设和约束

对于适用于本章所标识的变更，本条应标识有关假设和约束。

5 新系统或修改后系统的概念

本章应分以下各条描述新系统或修改后系统。

5.1 背景、目标和范围

本条应描述新系统或修改后系统的背景、任务或目标和范围。

5.2 运行策略和约束

本条应描述适用于新系统或修改后系统的运行策略和约束。

5.3 新系统或修改后系统的描述

本条应提供对新系统或修改后系统的描述，标识出不同运行状态和方式的差异(例如：常规、维护、培训、降级、紧急事件、备选场点等)。状态和方式之间的区别是任意的，可以仅按状态描述系统，也可以仅按方式、方式中的状态、状态中的方式或其他有用的方式描述系统。如果系统的运行不是以多种状态和方式，本条应如实陈述，而不需要进行人为的区分。(若适用)描述包括以下的内容：

a.运行环境及其特性；

b.主要系统部件及这些部件之间的互连；

c.与外部系统或过程的接口；

d.新系统或修改后系统的能力/功能；

e.描述输入、输出、数据流、手工和自动处理的图表和相应的说明，使用户能够充分理解新系统或修改后系统；

f.性能特性，如：速度、吞吐量、容量和频率；

g.质量属性，例如：可靠性、可维护性、可用性、灵活性、可移植性、易用性和有效性；

h.安全性、保密性、私密性和紧急情况下运行连续性的规定。

5.4 用户/受影响人员

本条应描述新系统或修改后系统的用户类型，(若适用)包括：组织结构、培训技能、职责和相互之间的交互。

5.5 支持概念

本条应简述新系统或修改后系统的支持概念，(若适用)应包括：支持机构，设施，设备，支持软件，维修/更换标准，维护等级和周期，存储、分配和供应方法。

6 运行场景

本章应描述一个或几个运行场景，举例说明新系统或修改后系统的作用、与用户的交互、与其他系统的接口和为该系统标识的所有状态或方式。(若适用)场景应包括：事件、动作、触发器、信息和交互等。可以使用影像等其他媒体来提供这些信息中的部分或全部。

7 影响综述

本章应分以下几条。

7.1 操作方面的影响

本条应描述对用户、需方、开发方和支持机构的可预计的操作影响。这些影响包括：与计算机操作中心接口的变化；过程的变化；新数据源的使用；输入到系统中的数据在数量、类型和时序上的变化；数据保留需要的变化；以及在紧急

情况等条件下新的运行方式。

7.2 组织影响

本条应描述对用户、需方、开发方和支持机构的可预计的组织影响。这些影响包括：职责的修改；职责或岗位的增加和撤销；培训或再培训的需要；在各种运行方式下人员在数量、技能级别、岗位标识和地点等方面的变化。

7.3 开发期间的影响

本条应描述在系统开发工作中对用户、需方、开发方和支持机构可预计的影响。这些影响包括：关于新系统的会议/讨论；数据库的开发和修改；培训；新系统和现有系统的并行运行；对新系统测试期间的影响以及帮助或监控开发所需的其他活动。

8 建议系统的分析

8.1 优点概述

本条应对新系统或修改后系统的优点提供定性和定量的总结，包括：新增的功能、增强的功能、改善的性能，(若适用)以及与在 4.1 中确认的缺陷的关系。

8.2 缺点/限制概述

本条应对新系统或修改后系统的缺点或约束进行定性和定量概述，包括：降级或缺少的功能；降级或达不到期望的性能；高于期望的计算机硬件资源的使用；出现了不想要的运行影响；与用户假设的冲突；其他的约束条件。

8.3 应考虑的变通和折衷

本条应标识和描述对系统或其特性应考虑的主要的变通办法，在其间所做的折衷以及作出决定的依据。

9 注解

本章应包含有助于理解本文档的一般信息(例如背景信息、词汇表、原理)。本章应包含为理解本文档所需要的术语和定义，所有缩略语和它们在文档中的含义的字母序列表。

附录

附录可用来提供那些为便于文档维护而单独出版的信息(例如图表、分类数据)。为便于处理，附录可单独装订成册。附录应按字母顺序(A,B 等)编排。

需求分析中，怎么看待主要矛盾和价值

麻锦涛 16281262 计科 1602

需求分析是软件定义时期的最后一个阶段。

需求分析的基本任务：

准确地回答“系统必须做什么”这个问题。

需求分析的任务还不是确定系统怎样完成它的工作，而仅仅是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求 。

在需求分析阶段结束之前，系统分析员应该写出软件需求规格说明书，以书面形式准确地描述软件需求。

尽管目前有许多不同的用语需求分析的结构化分析方法，但是，所有这些分析方法都遵守下述准则：

（1）必须理解并描述问题的信息域，根据这条准则应该建立数据模型。

（2）必须定义软件应完成的功能，这条准则要求建立功能模型。

（3）必须描述作为外部事件结果的软件行为，这条准则要求建立行为模型。

（4）必须对描述信息、功能和行为的模型进行分解，用层次的方式展示细节。

需求分析的任务

确定对系统的综合要求

虽然功能需求是对软件系统的一项基本需求，但却并不是唯一的需求。通常对软件系统有下述几方面的综合要求：

1.功能需求

这方面的需求制定系统必须提供的服务。通过需求分析应该划分出系统必须完成的所有功能。

2.性能需求

性能需求制定系统必须满足的定时约束或容量约束。

3.可靠性和可用性需求

可靠性需求定量地制定系统的可靠性。可用性与可靠性密切相关，它量化了用户可以使用系统的程度。

4.出错处理需求

这类需求说明系统对环境错误应该怎样响应。

5.接口需求

接口需求描述应用系统与它的环境通信的格式。常见的接口需求有：用户接口需求；硬件接口需求；软件接口需求；通信接口需求。

6.约束

设计约束或实现约束描述在设计或实现应用系统时应遵守的限制条件。常见的约束有：精度；工具和语言约束；设计约束；应该使用的标准；应该使用的硬件平台。

7.逆向需求

逆向需求说明软件系统不应该做什么。

8.将来可能提出的需求

应该明确地列出那些虽然不属于当前系统开发范畴，但是根据分析将来很可能会提出来的需求。这样做的目的是，在设计过程中对系统将来可能的扩充和修改项做准备，以便一旦确实需要时能比较容易地进行这种扩充和修改。

分析系统的数据要求

分析系统的数据要求通常采用建立数据模型的方法。

导出系统的逻辑模型

综合上述两项分析的结果可以导出系统的详细的逻辑模型，通常用数据流图、实体联系图（ER 图）、状态转换图、数据字典和主要的处理算法描述这个逻辑模型。

修正系统开发计划

根据在分析过程中获得的对系统的更深入更具体的了解，可以比较准确地估计系统的成本和进度，修正以前制定的开发计划。

与用户沟通获取需求的方法

访谈是最早开始使用的获取用户需求的技术，也是迄今为止仍然广泛使用

的需求分析技术。

访谈有两种基本形式，分别是正式的和非正式的访谈。正式访谈时，系统分析员将提出一些事先准备好的具体问题。在非正式访谈中，分析员将提出一些用户可以自由回答的开放性问题，以鼓励被访问人员说出自己的想法。

情景分析技术

情景分析技术的用处主要体现在下述两个方面：

（1）他能在某种程度上演示目标系统的行为，从而便于用户理解，而且还可能进一步揭示出一些分析员目前还不知道的需求。

（2）由于情景分析较易为用户所理解，使用这种技术能保证用户在需求分析过程中始终扮演一个积极主动的角色。需求分析的目标是获知用户的真实需求，而这一信息的唯一来源是用户，因此，让用户起积极主动的作用对需求分析工作获得成功是至关重要的。

简易的应用规格说明技术

这种方法提倡用户与开发者密切合作，共同标识问题，提出解决方案要素，商讨不同方案并指定基本需求。

使用简易的应用规格说明技术分析需求的典型过程如下：

①进行初步的访谈，通过用户对基本问题的回答，初步确定待解决的问题的范围和解决方案。

②开发者和用户分别写出“产品需求”。

分析建模与规格说明

分析建模

为了更好地理解复杂事物，人们常常采用建立事物模型的方法。所谓模型，就是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。

需求分析过程中应该建立 3 种模型，它们分别是数据模型、功能模型、行为模型。

软件需求规格说明

通过需求分析出了创建分析模型之外，还应该写出软件需求规格说明书，它是需求分析阶段得出的最主要的文档。

通常用自然语言完整、准确、具体地描述系统的数据要求、功能需求、性能需求。可靠性和可用性要求、出错处理需求、接口需求、约束、逆向需求以及将来可能提出的需求。

实体-联系图

为了把用弧度数据要求清楚、准确地描述出来，系统分析员通常建立一个概念性的数据模型（也称为信息模型）。概念性数据模型是一种面向问题的数据模型，是按照用户的观点对数据建立的模型。它描述了从用户角度看到的数据，它反映了用户的现实环境，而且与在软件系统中的实现方法无关。

数据模型中包含 3 种相互关联的信息：数据对象、数据对象的属性及数据对象彼此间相互连接的关系。

数据对象

数据对象是对软件必须理解的附和信息的抽象。所谓复合信息是指具有一系列不同性质或属性的事物，仅有单个值的事物（例如宽度）不是数据对象。

数据对象可以是外部实体。总之，可以由一组属性来定义 ID 实体都可以

被认为是数据对象。

数据对象彼此间是有关联的。

数据对象之封装了数据而没有对施加于数据上的操作的引用，这是对数据对象与面向对象范型中的“类”或“对象”的显著区别。

属性

属性定义了数据对象的性质。

应该根据对所要解决的问题的理解，来确定特定数据对象的一组合适的属性。

联系

客观世界中的事物彼此间往往是有联系的。

数据对象彼此之间相互连接的方式称为联系，也称为关系。联系可分为以下 3 中类型：

（1）一对一联系（1：1）

（2）一对多联系（1：N）

（3）多对多联系（M：N）

联系也可能有属性。

实体-联系图的符号

通常，使用实体-联系图（entity-relationship diagram）来建立数据模型。可以把实体-联系图简称为 ER 图，相应地可把用 ER 图描绘的数据模型称为 ER 模型。

ER 图中包含了实体（即数据对象）、关系和属性 3 中基本成分，通常用矩形框代表实体，用连接相关实体的菱形框标识关系，用椭圆型或圆角矩形标识实体（或关系）的属性，并用直线把实体（或关系）与其属性连接起来。

数据规范化

软件系统经常使用各种长期保存的信息，这些信息通常以一定方式组织并存储在数据库或文件中，为减少数据冗余，避免出现插入异常或删除异常，简化修改数据的过程，通常需要把数据结构规范化。

通常用“范式（normal forms）”定义消除数据冗余的程度。第一范式（1NF）数据冗余程度最大，第五范式（5NF）数据冗余程度最小。但是，第一，范式级别越高，存储同样数据就需要分解成更多张表，因此，“存储自身”的过程也就越复杂。第二，随着范式级别的提高，数据的存储结构与机遇问题域的结构间的匹配程度也随之下降，因此，在需求变化时数据的稳定性较差。第三，范式级别提高则需要访问的表增多，因此性能（速度）将下降。从实用角度看来，在大多数场合选用第三范式都比较恰当。

通常按照属性间的依赖情况区分范式化的程度。下面给出第一、第二和第三范式的定义：

（1）第一范式　　每个属性值都必须是原子值，即仅仅是一个简单值而不含内部结构。

（2）第二范式　　满足第一范式条件，而且每个非关键字属性都由整个关键字决定（而不是由关键字的一部分来决定）

（3）第三范式　　符合第二范式的条件，每个非关键字属性都仅由关键字决定，而且一个非关键字属性不能仅仅是对另一个非关键字属性的进一步描述（即一个非关键字属性值不依赖于另一个非关键字属性值）。

3.6 状态转换图

状态转换图（简称为状态图）通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为。

3.6.1 状态

状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。

在状态图中定义的状态主要有：初态（即初始状态）、状态（即最终状态）和中间状态。在一张状态图中只能有一个初态，而终态则可以有 0 至多个。

状态图既可以表示系统循环运行过程，也可以表示系统单程生命期。

事件

事件是在某个特定时刻发生的事情，它是对引起系统做动作或（和）从一个状态转换到另一个状态的外界事件的抽象。简而言之，事件就是引起系统做动作或（和）转换状态的控制信息。

符号

在状态图中，初态用实心圆表示，终态用一对同心圆（内圆为实心圆）表示。

中间状态用圆角矩形表示，可以用两条水平横线把它划分成上、中、下 3 个部分。上部分为状态的名称，这部分是必须的；中间部分为状态变量的名字和值，这部分是可选的；下面部分是活动表，这部分也是可选的。

验证软件需求

软件需求的验证从以下 4 个方面：

（1）一致性

所有需求必须是一直的，任何一条需求不能和其他需求互相矛盾。

（2）完整性

需求必须是完整的，规格说明书应该包含用户需求的每一个功能或性能。

（3）现实性

指定的需求应该是用现有的硬件技术和软件技术基本上可以实现的。

（4）有效性

必须证明需求是正确有效的，确实能解决用户面对的问题。

逐步求精方法结构化方法的认识

麻锦涛 16281262 计科 1602

结构化方法是强调开发方法的结构合理性以及所开发软件的结构合理性的软件开发方法，也称为新生命周期法，是生命周期法的继承与发展，是生命周期法与结构化程序设计思想的结合。其基本思想是用系统工程的思想和工程化得方法，根据用户至上的原则，自始自终按照结构化、模块化，自顶向下地对系统进行分析与设计。

结构化分析方法是面向数据流进行需求分析的方法，采用自顶向下、逐层分解，建立系统的处理流程，以数据流图和数据字典为主要工具，建立系同在的逻辑模型。

对于结构化分析，我认为它是整体和细节的桥梁，有效地把一个软件整体分成几个模块，不同的块负责不同的内容，他们分别既相互独立有有一定联系，比如数据输入，数据处理，数据输出；然后在模块的架构上逐步细化，丰富功能，其可以大致分为传入模块、传出模块、变换模块、协调模块。传入模块的功能是取得输入数据，经过某些处理，再将其传送给其他模块；传出模块的功能输出数据，在输出之前可能进行某些处理，数据可能被传输到系统的外部，也可能会输出到其他模块进行进一步的处理，但最终目的是输出到系统的外部；变换模块的功能是从上级调用模块取得数据，进行特定的数据处理，转换成其他形式，再将加工结果返回给调用模块；协调模块的功能是通过调用，协调和管理其他模块来完成特定的功能。

综上所述，我认为结构化设计方法存在以下优点：（1）减少设计复杂性。将大的任务转化为多个小任务，使得复杂问题简单化。（2）结构独立。将程序划分为多个相对独立模块。（3）模块功能单一化，可单独执行。（4）采用自顶向下。逐步求精的方法。

对软件工程抽象与逐步求精的理解的认识

麻锦涛 16281262 计科 1602

软件设计的困难随着问题的规模和复杂性不断增大，抽象是管理、控制复杂性的基本策略。“抽象”是一个心理学概念，它要求人们将注意力集中在某一个层次上思考问题，而忽略那些低层次的细节。使用抽象技术便于人们用“问题域“本来的概念和术语描述问题，而无须过早地转换为那些不熟悉的结构。软件设计过程应当是在不同抽象级别考虑、处理问题的过程。最初，应该在最高抽象级别上，用面向问题域的语言叙述问题，概括问题解的形式，而后不断地具体化，不断地更接近计算机域的语言描述问题；最后，在最低的抽象级别上给出可直接实现的问题解，即程序。

软件工程过程的每一步都是对较高一级抽象的解作一次比较具体化的描述。在系统定义阶段，软件系统被描述为基于计算机的大系统的一个组成部分，在软件规划和需求分析阶段，软件用问题域约定的习惯用语表达；从概要设计过度到详细设计时，抽象级别再一次降低；编码完成后则达到了抽象的最低级。在上述由高级抽象到低级抽象的转换过程中，伴随着一连串的过程抽象和数据抽象。过程抽象把完成一个特定功能的动作序列抽象为一个过程名和参数表，以后通过指定过程名和实际参数调用此过程；数据抽象把一个数据对象的定义（或描述）抽象为一个数据类型名，用此类型名可定义多个具有相同性质的数据对象。

”逐步求精“是与”抽象“密切相关的一个概念，可视为一种早起的自顶向下设计策略，其主要思想是，针对某个功能的宏观描述，使用逐步求精的方法不断的分解，逐步确立过程细节，直到该过程能用程序语言描述的算法实现为止。因为求精的每一步都是用更为详细的描述代替上一层次的抽象描述，所以在整个设计过程中，产生的具有不同详细程度的各种描述，组成了系统层次结构。层次结构的上一层还是下一层的抽象，下一层是上一层的求精。在过程求精的同时自然伴随着数据的求精，无论是过程还是数据，每个求精步骤都蕴含着某些设计决

策，因此设计人员必须掌握一些基本的准则和各种可能的候选方法。

在软件设计过程中，抽象与逐步求精一般都是结合起来进行应用，在建立了较高层次的抽象模型后，对其进行求精得到更加具体的抽象模型，然后再进行求精……由此一直到达最终的软件实现。该过程在面向对象软件开发过程中更加直观，因为面向对象的概念层抽象与最终类组成的系统实现具有自然的映射关系。其开发过程可以从概念建模逐步精化为最终的软件实现。

关于 pareto 原理更好的解释

和为什么需求确定测试

麻锦涛 16281262 计科 1602

一、关于 pareto 原理更好的解释

Walker Royce 扩展了 Barry Boehm 提出的有关软件项目管理的“二八定理”，构成了现代软件管理过程框架的理论基础。

1.80%的工程活动是由 20%的需求用掉的。

2.80%的软件成本是由 20%的构件用掉的。

3.80%的缺陷是由 20%的构件引起的。

4.80%的软件废品和返工是由 20 的缺陷引起的。

5.80%的资源是由 20%构件用掉的。

6.80%的工程活动是通过 20%的工具完成的。

7.80%的进展是 20%的人完成的。

同时我也认识了一句话：80%的 bug 隐藏在 20%的代码中。时隔多年，在去年才深有体会。研发的时候项目中有些代码写了之后从此就没有变过，而有些代码却经常变动。找了这句话的出处，原来有个二八定律。

二八定律也叫巴莱多定律，是 19 世纪末 20 世纪初意大利经济学家巴莱多发明的。他认为，在任何一组东西中，最重要的只占其中一小部分，约 20%，其余 80%的尽管是多数，却是次要的，因此又称二八法则。

深入了解了下这个法则，我想解决之道就是：利用马克思主义原理矛盾分析法中的抓住主要矛盾，从而花 80%的时间在 20%的代码上面。80%的缺陷出现在 20%的代码中；80%的 BUG 发现在 20%的时间中；80%的花费在 20%的错误代码上。

二、为什么需求确定测试

1、需求分析的必要性

如果要成功的做一个测试项目，首先必须了解测试规模、复杂程度与可能存在的风险，这些都需要经过详细的测试需求来了解。所谓知己知彼，百战不殆。测试需求不明确，只会造成获取的信息不正确，无法对所测软件有一个清晰全面的认识，测试计划就毫无根据可言，只凭感觉不做详细了解就下定论的项目是失败的。

测试需求分析越详细精准，表明对所测试软件的了解越深，对所要进行的任务内容就越清晰，就更有把握确保测试的质量与进度。
如果把测试活动比作软件生命周期，测试需求分析就相当于软件的需求规格，测试策略相当于软件的架构设计，测试用例相当于软件的详细设计，测试执行相当于软件的编码过程。只是在测试过程中，我们把”软件”两个字全部替换成了”测试”。这样，我们就明白了整个测试活动的依据来源于测试需求，所以需求分析是整个测试活动必不可少的环节。

2、不做需求分析的后果

不做需求分析或需求分析不到位，可能会产生很严重的问题，比如：
（1） 浪费时间和资源实现了用户不需要的需求；
（2） 遗漏了需求文档中没提到，但很重要的需求，导致客户满意度降低。
（3） 需求分析不到位，错误的估计了测试的工作量，导致延误发布周期，可能会降低发布质量。

黑盒测试、白盒测试和灰盒测试的内涵

麻锦涛 16281262 计科 1602

软件测试的方法和技术是多种多样的。 对于软件测试技术，可以从不同的角度加以分类：

从是否需要执行被测软件的角度，可分为静态测试和动态测试。从测试是否针对系统的内部结构和具体实现算法的角度来看，可分为白盒测试和黑盒测试；

1. 黑盒测试

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盆子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数锯而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

黑盒测试方法主要有等价类划分、边值分析、因—果图、错误推测等，主要用于软件确认测试。“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

软件的黑盒测试意味着测试要在软件的接口处进行。这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。因此黑盒测试

又叫功能测试或数据驱动测试。黑盒测试主要是为了发现以下几类错误：

1、是否有不正确或遗漏的功能？
2、在接口上，输入是否能正确的接受？能否输出正确的结果?
3、是否有数据结构错误或外部信息（例如数据文件）访问错误？
4、性能上是否能够满足要求？
5、是否有初始化或终止性错误？

2. 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能，白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

“白盒”法全面了解程序内部逻辑结构、对所有逻辑路径进行测试。“白盒”法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字。但即使每条路径都测试了仍然可能有错误。第一，穷举路径测试决不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出错。第三，穷举路径测试可能发现不了一些与数据相关的错误。

软件的白盒测试是对软件的过程性细节做细致的检查。这种方法是把测试对象看做一个打开的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序状态，确定实际状态是否与预期的状态一致。因此白盒测试又称为结构测试或逻辑驱动测试。白盒测试主要是想对程序模块进行如下检查：

1、对程序模块的所有独立的执行路径至少测试一遍。
2、对所有的逻辑判定，取“真”与取“假”的两种情况都能至少测一遍。
3、在循环的边界和运行的界限内执行循环体。
4、测试内部数据结构的有效性，等等。

3. 灰盒测试

灰盒测试，确实是介于二者之间的，可以这样理解，灰盒测试关注输出对于输入的正确性，同时也关注内部表现，但这种关注不象白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，有时候输出是正确的，但内部其实已经错误了，这种情况非常多，如果每次都通过白盒测试来操作，效率会很低，因此需要采取这样的一种灰盒的方法。

灰盒测试结合了白盒测试盒黑盒测试的要素.它考虑了用户端、特定的系统知识和操作环境。它在系统组件的协同性环境中评价应用软件的设计。

灰盒测试由方法和工具组成，这些方法和工具取材于应用程序的内部知识盒与之交互的环境，能够用于黑盒测试以增强测试效率、错误发现和错误分析的效率。

灰盒测试涉及输入和输出，但使用关于代码和程序操作等通常在测试人员视野之外的信息设计测试。

软件审查的地位和作用

麻锦涛 16281262 计科 1602

软件开发项目中包括很多的审查动作,贯穿于整个开发过程。个人认为审查主要有以下目的：

1.尽早排查出潜在的问题(Potential Risk/Issue)

经过其他人的参与，以不同的视角提出不同的看法，会有类似头脑风暴的效果，集思广议来查找工程师未能注意的问题。

2.保持良好且有效的双向沟通

很多时候沟通并不充分，总有许多以为明白，实际并不明白的情况。组织管理人员需要及时通过审查的方式，与开发人员进行有效的沟通。 同时，审查会使开发有更多的表达和相互沟通的机会，提高参与度，所以审查也可以提高开发人员的认同感和归属感。如果大家都能从审查中得到有效的指导和帮助，也会有助于提高审查效率。

3.通过以上两点，来达成最终正确的结果。

管理者需要对最终的结果负责，不能坐等问题发生，一定要先行加以预防，所以需要不断的进行审查。而组员通常对于所谓的审查会有一定的戒心，害怕批评或出丑，或者觉得没有必要，于是管理要确保在审查时做到对事不对人，并能针对问题提供有价值的指导。审查的对象是当前状态及产出，审查的目的在于确保最终的正确产出，审查的方式就是质询，而质询的基础就是审查制度。

审查制度可以很灵活,比如在项目开始定义出项目的角色及职责(ARCI矩阵),然后针对不同的 Milestones 或任务设置检查点(Check point),重要的环节加以定义,使得审查有据可循。如针对 Design Spec,我们需要宣布以下规则:

1. Design Spec 需要遵循公司统一规范。
2. Design Spec 中需要描述清楚 Block diagram, Interface defintions 等等。
3. Design Spec 什么时候提交给哪些人做审查。
4. 什么时候举行 Design Spec 的审查会议，哪些人会参加，等等。

这是针对工程师而言的。Design Spec 中的内容相应会有一个审查表(Check List)来进行审查，这就是规则。同样针对管理者也可以设定一个审查表,如:

1. 必须向所有组员说明一次公司相关的文档规范。
2. 必须定义清楚撰写及审查的时间安排，并向所有组员宣布，确保大家可以执行。
3. 必须提供相应的审查记录

对于需求分析、Functional Spec、代码管理、版本控制、发布控制、BUG 管理及成本控制，都可以细化相应的动作，加以审查。审查对象包括项目中的各个角色。

总结一下，项目中审查制度的建立有以下步骤:

1. 项目中角色及职责定义(R&R, Who)
2. 定义审查点 (Check Point, When)

3. 定义审查内容、范围及目标(Check List, What)
4. 定义出审查人员 (Who)

其正规性体现在：仔细划分错误类型，并把这些信息运用在后续阶段的文档审查中以及未来产品的审查中。审查之所以重要，是因为我们处在越来越讲求效率与生产力的环境，走越少的弯路，我们的企业才会更多的利润，员工才有更大的空间。这是管理者与工程师需要共同承担的责任。

审查的内在指导思想和软件工程的指导思想

麻锦涛 16281262 计科 1602

*SMART 原则是目标管理的重要参考法则，也对于这个的审查制度有很强的指导性。

上面的审查制度是常态性的，另外还有一些突发状况需要由管理者适时判断审查的必要性。

如果突然遇到之前未能预估到的技术难题时，管理者必须特别为这个问题设定应急计划和一个更为紧凑的审查机制。比如在应急计划如下：

1.召开会议，寻求资深工程师的帮助。[负责人及时间]

2.安排其他人员支援。[负责人及时间]

3.寻求第三方公司的支持。[负责人及时间]

4.网络寻求解答。[负责人及时间]

5.向相关人员报告细节问题，对于项目管理人员要考虑有无替代方案。[负责人及时间]

这是一种主动的自我审查方式。无论对于管理者还是工程师，一定要清楚，你的主管、客户和老板，都需要知道你在做什么！及时的汇报是非常必要的，这也是一种主动的审查，它的效率更高。所以我们可以要求承担重点开发任务的人员周期性提交状态报告，将做过的事情、遇到的问题、后续的计划表列出来。其他人可以另作要求。管理者则应主动为其提供相应支持，这样就可以保持一个较为良性的互动。

软件工程的指导思想：

为解决软件危机，把“软件”这种特殊商品的生产、管理过程纳入传统工程管理的轨道；

用计算机科学中的最新成果应用于软件工程中

用管理学的原理和方法进行软件生产管理

用工程学的观点进行核算，制定工程进度和实施方案

用数学方法建立软件的可靠模型和各种有效算法

结构化设计和面向设计对象

的区别和联系

麻锦涛 16281262 计科 1602

（1）结构化程序设计

结构化程序设计的基本思想是采用自顶向下、逐步细化的设计方法和单入单出的控制结构。其理念是将大型程序分解成小型、便于管理的任务。如果其中的一项任务仍然过大，则将它分解为更小的任务。这一过程将一直持续下去，直到将程序划分为小型的，易于编写的模块。

*这种设计方式有着显著的一些问题：

程序难以管理、数据修改存在问题、程序可重用性差、用户要求难以在系统分析阶段准确定义，致使系统在交付使用时产生许多问题。

用系统开发每个阶段的成果来进行控制，不能适应事物变化的要求。

这种问题的根源在于数据和数据的处理不能分离。

（2）面向对象程序设计

面向对象（ObjectOriented ）是认识事务的一种方法，是一种以对象为中心的思维方式面向将系统看成通过交互作用来完成特定功能的对象的集合。每个对象用自己的方法来管理数据。也就是说只有对象内部的代码能够操作对象内部的数据。

*面向对象的基本特征

1、抽象 2、封装 3、继承 4、多态 5、继承和多态 (3)面向对象程序设计优缺点

面向对象编程的优点：

易维护、质量高、效率高、易扩展

面向对象编程的缺点：运行效率会下降 10%左右

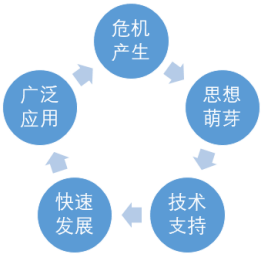
二者的联系和区别主要有以下几点：

（一）从起源上看

结构化方法与面向对象方法都起源于相应的程序设计思想和语言。20 世纪 60 年代后期，《程序结构理论》和《GOTO 陈述有害论》的提出，证明了任何程序的逻辑结构都可以用顺序结构、选择结构和循环结构来表示，确立了结构化程序设计思想，产生了如 FORTRAN、PASCAL、C 等语言。结构化方法把对程序的分析、设计，延伸至对项目工程的分析、设计，结合程序设计语言的技术支持，得以产生和发展。

20 世纪 80 年代，随着应用系统的日趋复杂、庞大，结构化开发方法在工程应用中出现了一些问题。同期，面向对象程序设计思想经过 20 年的研究和发展逐渐成熟，一大批面向对象语言相继出现，面向对象方法自产生就广受青睐。90 年代中期，互联网兴起，JAVA 语言因跨平台特性得以蓬勃发展；21 世纪初，不受限于时空的联合开发成为常态；今天，移动 APP 市场火爆，Andriod 开发成为热点。面向对象方法已经成为软件开发方法的中坚力量。

结构化方法和面型对象方法的起源和发展具有模式一致性：

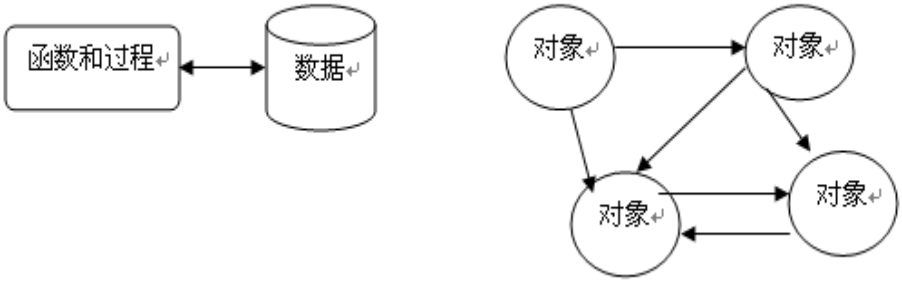


（二）从思想上看

结构化方法承袭了结构化程序设计的思想，把待解决的问题看作一个系统，用系统科学的思想方法来分析和解决问题。结构化方法遵循抽象原则、分解原则和模块化原则；以数据和功能为中心；以模块为基本单位；以算法为程序核心；强调逐步求精和信息隐藏。

面向对象方法的思想是模拟了客观世界的事物以及事物之间的联系。面向对象以类取代模块为基本单位；通过封装、继承和多态的机制，表征对象的数据和功能、联系和通信；通过对对象的管理和对象间的通讯完成信息处理与信息管理的计算和存储，实现软件功能。

对于结构化方法，模块由函数实现，完成对输入数据的加工和计算，数据和功能是分离的；而面向对象把数据和功能封装在对象中，形成一个整体。两种方法在数据和功能上的不同处理是其思想上的本质差别。



图二：结构化思想和面向对象思想

面向对象程序设计

从上例可见，结构化程序设计是一种过程式的“解题”的方式，程序关注且只关注对于输入数据，输出正确的结果，代码是算法的直接体现，代码效率高；面型对象程序设计是整体式的“建模”的方式，程序关注现实客体，而非某些数据，代码是功能的直接体现，复杂的算法往往是一两行库函数处理，代码效率低。

（三）从应用上看

结构化方法的实质是问题求解，结构化程序是由算法决定的，算法是程序员分析设计的，程序的执行过程主要是由程序员控制，而不是由用户控制；面向对象方法中，程序员设计的是对象属性及操作方法，但在什么时间、使用什么方式操作对象则是完全由用户交互控制。

结构化方法的建模工具难以表达交互性强的软件系统，程序设计融入系统的分析和设计中，处理大型系统时会过于复杂，甚至很难控制；面向对象方法的抽象机制提供了自然的建模方法，特别是能很好地把握对象之间复杂的相互关系。

结构化方法比较适合工程计算、实时数据的跟踪处理、各种自动控制系统等等； 面向对象分析更加适用于复杂的、由用户控制程序执行过程的应用软件，比如大型游戏软件以及各类管理信息系统软件[3]。

（四）二者之结合

经过上述分析，我们可知结构化方法和面向对象方法对于不同的软件系统各有优劣。结构化方法把解空间分数据和功能两部分，可以更加清晰地进行需求分析和功能分解，数据流图能够细致地说明数据在各个功能模块之间的流动和变化，更适于系统设计的前期阶段。设计人员清楚地了解数据和系统要求的操作后，面向对象方法能够把数据和功能以对象为单位封装成一个整体，更直观地表达对象的状态变化和对象间的交互，更加准确地分析功能的实现过程，更适于在软件后期细化系统的具体行为。基于此，设计的混合式软件开发方法如下：

1)使用 SA 进行需求分析，建立数据字典，构建总的和分层的数据流图。

2)使用模块结构图设计系统的独立功能块，做出模块内的程序流图。

3)结合数据流图，聚合同类模块，规约类，根据程序流图，设计类的属性和类的方法。

4)使用 OOD 建立系统的动态模型，分析对象的行为和协作。

5)总体面向对象程序设计，细节结构化程序设计优化，实现代码层。

使用混合式方法，我们能够充分利用两种方法的优点，扬长避短，提高开发的效果和效率。

无论是结构化方法，还是面向对象方法，都是用来解决日益矛盾的软件危机的系统方法。从直接开发，到结构化方法，再到面向对象方法，软件构件的愈发独立、可重用，开发在一个更高的层次进行，分析层、设计层和代码层关联性减少。这些都有利于系统开发员更加关注功能本身，提高软件质量。硬件性能的提高会使计算机的使用越发广泛，软件工作的环境更加复杂，软件的功能更加丰富，软件的性能更需提高，对软件开发方法提出了更多的要求，会涌现更高层次的新的方法。无论使用哪种开发方法，或者是混合哪几种开发方法，我们都要因地制宜，依据需求分析和系统要求，做出最好的选择或组合。