“ 软件工程 ”概说 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/293/2021_2022__E2_80_9C_E8_BD_AF_E4_BB_B6_E5_c67_293059.htm 未接触软件工程之前一直都很想学这门课程，因为觉得这门课很NB，是那些有工程师称号的高手才摆弄的东西。但学过之后，最大的感触却是：软件工程方法一定要从娃娃抓起，否则到了后面坏习惯已经养成后再回过头来修正，那绝对是地狱般的磨难。下面就是我在近两个月的学习中一些总结和体会，希望对后来者有所补益。由于是初学这门课程，难免浅薄和有所错漏，还望大家多多指教。 软件工程的由来 据说上个世纪60年代的程序员都是天才，写程式就像写日记一样，吃过晚饭没事干随手就可以写几个出来玩，第二天还可以拿去卖钱。所以那时候程序员在大家眼中，跟那些搞美术，音乐的是一类的，被称为“ 艺术家 ”。 但事过境迁，就像任何人都不会嫌钱多一样，永远都不会有人嫌CPU快的。于是，随之而来的就是硬件的迅猛发展和越来越变态的软件。记得以前常去同学家拷游戏，通常几张软盘就可以搞定，而现在的游戏，两三张CD-ROM都算少的了。像如此庞大复杂的怪物，就算你是如何的天才，一个人肯定是搞不定的，否则，等你把程式写出来，人家Intel连奔腾N都开发出来了。既要开发大型的软件还要追求速度（这样才能赚钱），于是很自然地，合作的概念被提了出来。 在开始合作的初期，由于大家都习惯了当很有个性的“ 艺术家 ”，结果可想而知，一个是毕加索派的，而另一个是意大利印象派的，再加上一个画泼墨山水画的，要是像这样凑出来的东西都能不出问题的话，那么Bill早就转

行了。所以，那时侯的大型软件，据说“ 蓝屏 ”比WINDOWS 98还多。 马克思告诉我们，万物都是从量变到质变的。随着问题的不断涌现，一些master们开始尝试去总结经验，并归纳了一些规范去指导软件的分析，设计，实现，测试，维护，人员交流协作，项目预算及时限控制等方方面面，这就是软件工程的前身。 软件工程到现在已发展了30多年，可以说是相当成熟的了。现在开发软件，据说都是一大帮人排排坐，按着一整套的规章制度来干活。于是，软件开发成了“ 工程 ”，程序员也就沦为“ 工人 ”了。 最初提出问题的是Dijkstra.他于1968年写给ACM的一封题为Goto Statement Considered Harmful 的信中，指出了GOTO语句的负面作用，并提出了解决之道，其引发的一系列效应最终带来了软件工程的诞生。 软件工程的核心 无论是在上个世纪还是在现在，软件开发所涉及的工作基本上都没有变化，它们都起始于一个实际需要或某个灵感，然后就是分析，设计，编码，调试，维护。这些任务以某种方式动态地结合起来就构成了软件开发的整个过程，这就是所谓的“ 软件开发周期 ”。 但对于这些工作，具体怎样做，什么时候做，每个人都有自己的一套方式，甚至有的人有几套方式。这样，当几个人合在一起干活的时候，最终的结果就只能是一片混乱。所以就需要一套规则，大家都按规则来办事，问题就会少得多。好的规则就叫做规范，规范都是由一些master们根据经验总结的，又经过长时间的历练，不断地被补充修正，可以说都是精华，按照规范来干活，对于提高软件质量和工作效率自然大有帮助。 而软件工程，说白了，就是这样一套用于软件的团队开发，以提高软件质量和程序员工作效率为目的的规范。其核心

就是，对于软件开发的5个重要组成部分：需求分析，设计，编码，调试，维护，如何组织这5个部分的工作，以及如何完成每一个工作。简单来说，就是对于总体的组织和对于局部的实现。 规范只是提供一个好的例子，以描述一种思想，具体到每一个环节怎样实现，对于不同的公司或团体则是各有千秋，因为根本就不可能存在一套放之天下皆可行的标准。就像C ，也只是提供了一套标准，不同的编译器都有各自的实现，对标准的支持程度也互不相同。所以，在不同的公司或团体中，尽管核心思想都是大同小异，但具体到每一个步骤，往往都是不相同的。我手上就有一份GB8567-88的文档模板，对于那些顶多只有几千行的小程序来说，假如真按上面的要求全写上了，简直就是一种折磨！据说，当前业界最权威的标准是CMM. 软件开发过程的组织 如何组织软件开发过程中的每一个步骤，就是软件开发周期模型要解决的问题。其实开发软件，就像是解决一个逻辑问题。想想自己平时是怎样写程序的。首先是要有一个想法，即我写的这个程序是要干什么的；然后就是对要实现的核心功能大概构思一种或多种实现方法，并从中选出一种自认为是较好的；接下来就是将涉及的各种主要或次要功能分成各个模块；最后就是分模块来编码和DEBUG.在我看来，除了第一步外，其余的步骤应该是一个循环的过程。在编码的过程中，你总是需要不断地回过头来修改原先的模块设计，甚至最初选定的实现算法。例如，最简单的情况是，你通常都会突然发现在两个成员函数中有相同的代码，这时，程序员的直觉告诉你，你应该为你的类再添加一个private成员函数并将公共的代码放于其中；又或者是，你突然发现一个模块中的某个功能具有很高

的通用性，完全可以提取出来作为一个独立的功能组件，而你也确实应该这样做；要是倒霉一点的话，你很有可能会在最后调试的时候突然发现，你的程序跑得太慢了，连你自己都无法忍受。于是你找呀找，终于找到了80/20中的那段可恶的20，原来是用了一个O（N）的算法，这时你就得老老实实地换一个更好的算法。 总之，除非你是先知，否则，对于一个具有一定规模和复杂度的软件来说，在“ 设计编码 ”这个过程中，实在有太多的不可预知性和变化性，你根本不可能全盘地把握住每一个细节。当然，这是建立在我现时的水平之上的观点。我不知道是否成为高手以后会有所不同，因为我身边没有那样的人。 既然软件开发是一个具有不可预知性和变化性的动态的过程，那么，对其每一个步骤的组织，即周期模型，就必须包容它的这种性质。 现在来看一下最古老，最经典，同时也是最倍受批评的瀑布模型。 瀑布模型是一种线性模型，其最大的特点就是简单直观。它将软件开发过程规划为“ 分析设计编码测试维护 ”的线性过程，也就是说，你必须首先把你的软件要干的每一件工作都分析得彻彻底底，再对每一个模块，每一个接口，事无巨细，都设计得非常完美，然后才开始编码的工作，并且在编码的时候就像在对着图纸砌模型，根本不用再回头作任何修改，当然，是在把所有的代码都写完以后才开始测试的。 整个过程，光想一下就觉得冒冷汗！ 瀑布模型完全忽视了软件开发过程的动态变化。恐怕只有那些已经发展得非常成熟，且规模不大的系统，例如：用Access做后台，用VB画前端的数据库应用程序，才有瀑布模型一展拳脚的地方。 相比之下，现在常用的一些周期模型则更接近于人的自然思维，例如螺旋模型就是一

种我比较喜欢的模型。 软件开发过程的实现 具体到每一步的工作要怎样完成，我前面已提到过，是非常灵活的，只要把握住大体的方向就行。在进行分析，设计，编码，调试，维护这几部分的工作的时候，最核心的就是文档的编写。文档的作用在于以下3个方面：一是可以帮助整理思路。把要完成的目标，系统的结构，每一个模块的功能等整理一下，然后分门别类地写下来，这样在开发的过程中，就有据可依，在需要回过头来修改设计的时候，也有证可考。二是便于交流。想象一下开会时的情形。一大帮子人争先恐后，激烈辩论，然后会终人散，思想灵感也就随之散了，结果是开了半天会，什么也没讨论出来。这就是后来会议记录被发明出来的原因。在脑子里的东西一多，就会散而且乱，用语言表达的时候，很容易会丢三落四，别人也很难把握住你的思想。但经过整理写在纸上以后，则会清晰得多，无论是别人还是自己，看起来都可以一目了然。三是可以作为以后维护时的参考资料。有一句名言：“ 笔和纸永远都比大脑可靠 ”，意思就是说，放在大脑里的东西说不准哪天就忘了，但写在纸上的东西，只要不发生什么意外，一般是丢不了的。当过了一段时间，你需要再回过头来修改你的程序的时候，你就会发现，你以前写下的文档实在太有价值了。别指望你的源代码，对于复杂一点的程序来说，单纯的源代码几乎会扼杀掉你所有的时间。 至于文档怎样写，教科书上大多都是一条一条列得满满的，就像一些地方政府的规章制度一样，其实大可不必，只要能满足需要就行。如果是在公司，则每个公司大多都有一套自己内部的文档模板，个人没有选择的余地。而对于像我这种业余的，写个程序除了练练手艺，无非就是供

自己和亲朋好友玩玩，则根本没必要搞得过于复杂。以下就是我自己的一份文档模板的概要，麻雀虽小，但五脏俱全。可行性分析 就是关于当前项目能不能干的分析结果。主要考虑的方面包括：是否能把这个项目开发出来；假如可以的话，预计需要多少时间，能否满足客人的时间要求；需要多少人力和资金的投入；最重要的是，这个项目能否赚钱，能赚多少。还要对可能存在的风险进行评估，例如，万一项目主管被车撞了要怎么办。当然，这对于我来说毫无意义，我在这里写上只是为了保持完整而已。 项目描述 这是在决定立项以后，对当前项目的一份扼要说明。必须包括以下几个方面：（1）项目的名称或编号；（2）对客户方的描述；（3）对开发人员的描述；（4）工程任务的描述；（5）工程的输入和输出；（6）开发环境；（7）其他的附加条件。在这里，对工程任务的描述是从整体的角度来说的，例如：能对当前的象棋棋局进行分析并作出最优决策的人工智能系统。而工程的输入输出则可以这样 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com