

怎样成为优秀软件模型设计者

我们期待自己成为一个优秀的软件模型设计者，但是，要怎样做，又从哪里开始呢？

将下列原则应用到你的软件工程中，你会获得立杆见影的成果。

1. 人远比技术重要

你开发软件是为了供别人使用，没有人使用的软件只是没有意义的数据的集合而已。许多在软件方面很有成就的行家在他们事业的初期却表现平平，因为他们那时候将主要精力都集中在技术上。显然，构件（components），EJB（Enterprise Java Beans）和代理（agent）是很有趣的东西。但是对于用户来说，如果你设计的软件很难使用或者不能满足他们的需求，后台用再好的技术也于事无补。多花点时间到软件需求和设计一个使用户能很容易理解的界面上。

2. 理解你要实现的东西

好的软件设计人员把大多数时间花费在建立系统模型上，偶尔写一些源代码，但那只不过是为了验证设计过程中所遇到的问题。这将使他们的设计方案更加可行。

3. 谦虚是必须的品格

你不可能知道一切，你甚至要很努力才能获得足够用的知识。软件开发是一项复杂而艰巨的工作，因为软件开发所用到的工具和技术是在不断更新的。而且，一个人也不可能了解软件开发的所有过程。在日常生活中你每天接触到的新鲜事物可能不会太多。但是对于从事软件开发的人来说，每天可以学习很多新东西（如果愿意的话）。

4. 需求就是需求

如果你没有任何需求，你就不要动手开发任何软件。成功的软件取决于时间（在用户要求的时间内完成）、预算和是否满足用户的需求。如果你不能确切知道用户需要的是什么，或者软件的需求定义，那么你的工程注定会失败。

5. 需求其实很少改变，改变的是你对需求的理解

Object ToolSmiths公司（www.objecttoolsmiths.com）的Doug Smith常喜欢说：“分析是一门科学，设计是一门艺术”。他的意思是说在众多的“正确”分析模型中只存在一个最“正确”分析模型可以完全满足解决某个具体问题的需要（我理解的意思是需求分析需要一丝不苟、精确地完成，而设计的时候反而可以发挥创造力和想象力 - 译者注）。

如果需求经常改动，很可能是你没有作好需求分析，并不是需求真的改变了。

你可以抱怨用户不能告诉你他们想得到什么，但是不要忘记，收集需求信息是你工作。

你可以说是新来的开发人员把事情搞得一团糟，但是，你应该确定在工程的第一天就告诉他们应该做什么和怎样去做。

如果你觉得公司不让你与用户充分接触，那只能说明公司的管理层并不是真正支持你的项目。

你可以抱怨公司有关软件工程的管理制度不合理，但你必须了解大多同行公司是怎么做的。

你可以借口说你们的竞争对手的成功是因为他们有了一个新的理念，但是为什么你没先想到呢？

需求真正改变的情况很少，但是没有做好需求分析工作的理由却很多。

6. 经常阅读

在这个每日都在发生变化的产业中，你不可能在已取得的成就上陶醉太久。

每个月至少读 2、3 本专业杂志或者 1 本专业书籍。保持不落伍需要付出很多的时间和金钱，但会使你成为一个很有实力的竞争者。

7. 降低软件模块间的耦合度

高耦合度的系统是很难维护的。一处的修改引起另一处甚至更多处的变动。

你可以通过以下方法降低程序的耦合度：隐藏实现细节，强制构件接口定义，不使用公用数据结构，不让应用程序直接操作数据库（我的经验法则是：当应用程序员在写 SQL 代码的时候，你的程序的耦合度就已经很高了）。

耦合度低的软件可以很容易被重用、维护和扩充。

8. 提高软件的内聚性

如果一个软件的模块只实现一个功能，那么该模块具有高内聚性。高内聚性的软件更容易维护和改进。

判断一个模块是否有高的内聚性，看一看你是否能够用一个简单的句子描述它的功能就行了。如果你用了一段话或者你需要使用类似“和”、“或”等连词，则说明你需要将该模块细化。

只有高内聚性的模块才可能被重用。

9. 考虑软件的移植性

移植是软件开发中一项具体而又实际的工作，不要相信某些软件工具的广告宣传（比如 java 的宣传口号 write once run many ? 译者注）。

即使仅仅对软件进行常规升级，也要把这看得和向另一个操作系统或数据库移植一样重要。

记得从 16 位 Windows 移植到 32 位 windows 的“乐趣”吗？当你使用了某个操作系统的特性，如它的进程间通信(IPC)策略，或用某数据库专有语言写了存储过程。你的软件和那个特定的产品结合度就已经很高了。

好的软件设计者把那些特有的实现细节打包隐藏起来，所以，当那些特性该变的时候，你的仅仅需要更新那个包就可以了。

10. 接受变化

这是一句老话了：唯一不变的只有变化。

你应该将所有系统将可能发生的变化以及潜在需求记录下来，以便将来能够实现（参见“Architecting for Change”，Thinking Objectively, May 1999）

通过在建模期间考虑这些假设的情况，你就有可能开发出足够强壮且容易维护的软件。设计强壮的软件是你最基本的目标。

11. 不要低估对软件规模的需求

Internet 带给我们的最大的教训是你必须在软件开发的最初阶段就考虑软件规模的可扩充性。

今天只有 100 人的部门使用的应用程序，明天可能会被有好几万人的组织使用，下月，通过因特网可能会有几百万人使用它。

在软件设计的初期，根据在用例模型中定义的必须支持的基本事务处理，确定软件的基本功能。然后，在建造系统的时候再逐步加入比较常用的功能。

在设计开始考虑软件的规模需求，避免在用户群突然增大的情况下，重写软件。

12. 性能仅仅是很多设计因素之一

关注软件设计中的一个重要因素—性能，这好象也是用户最关心的事情。一个性能不佳的软件将不可避免被重写。

但是你的设计还必须具有可靠性，可用性，便携性和可扩展性。你应该在工程开始就应该定义并区分好这些因素，以便在工作中恰当使用。性能可以是，也可以不是优先级最高的因素，我的观点是，给每个设计因素应有的考虑。

13. 管理接口

“UML User Guide” (Grady Booch, Ivar Jacobson 和 Jim Rumbaugh, Addison Wesley, 1999) 中指出, 你应该在开发阶段的早期就定义软件模块之间的接口。

这有助于你的开发人员全面理解软件的设计结构并取得一致意见, 让各模块开发小组相对独立的工作。一旦模块的接口确定之后, 模块怎样实现就不是很重要了。

从根本上说, 如果你不能够定义你的模块“从外部看上去会是什么样子”, 你肯定也不清楚模块内要实现什么。

14. 走近路需要更长的时间

在软件开发中没有捷径可以走。

缩短你的在需求分析上花的时间, 结果只能是开发出来的软件不能满足用户的需求, 必须被重写。

在软件建模上每节省一周, 在将来的编码阶段可能会多花几周时间, 因为你在全面思考之前就动手写程序。

你为了节省一天的测试时间而漏掉了一个 bug, 在将来的维护阶段, 可能需要花几周甚至几个月的时间去修复。与其如此, 还不如重新安排一下项目计划。

避免走捷径, 只做一次但要做对 (do it once by doing it right)。

15. 别信赖任何人

产品和服务销售公司不是你的朋友, 你的大部分员工和高层管理人员也不是。

大部分产品供应商希望把你牢牢绑在他们的产品上, 可能是操作系统, 数据库或者某个开发工具。

大部分的顾问和承包商只关心你的钱并不是你的工程 (停止向他们付款, 看一看他们会在周围呆多长时间)。

大部分程序员认为他们自己比其他人更优秀, 他们可能抛弃你设计的模型而用自己认为更好的。

只有良好的沟通才能解决这些问题。

要明确的是, 不要只依靠一家产品或服务提供商, 即使你的公司 (或组织) 已经在建模、文档和过程等方面向那个公司投入了很多钱。

16. 证明你的设计在实践中可行

在设计的时候应当先建立一个技术原型, 或者称为“端到端”原型。以证明你的设计是能够工作的。

你应该在开发工作的早期做这些事情, 因为, 如果软件的设计方案是不可行的, 在编码实现阶段无论采取什么措施都于事无补。技术原型将证明你的设计的可行性, 从而, 你的设计将更容易获得支持。

17. 应用已知的模式

目前, 我们有大量现成的分析和设计模式以及问题的解决方案可以使用。

一般来说, 好的模型设计和开发人员, 都会避免重新设计已经成熟的并被广泛应用的东西。

<http://www.ambyssoft.com/processPatternsPage.html> 收藏了许多开发模式的信息。

18. 研究每个模型的长处和弱点

目前有很多种类的模型可以使用, 如下图所示。用例捕获的是系统行为需求, 数据模型则描述支持一个系统运行所需要的数据构成。你可能会试图在用例中加入实际数据描述, 但是, 这对开发者不是非常有用。同样, 数据模型对描述软件需求来说是无用的。每个模型在你建模过程中有其相应的位置, 但是, 你需要明白在什么地方, 什么时候使用它们。

19. 在现有任务中应用多个模型

当你收集需求的时候，考虑使用用例模型，用户界面模型和领域级的类模型。

当你设计软件的时候，应该考虑制作类模型，顺序图、状态图、协作图和最终的软件实际物理模型。程序设计人员应该慢慢意识到，仅仅使用一个模型而实现的软件要么不能够很好地满足用户的需求，要么很难扩展。

20. 教育你的听众

你花了很大力气建立一个很成熟的系统模型，而你的听众却不能理解它们，甚至更糟一连为什么要先建立模型都不知道。那么你的工作是毫无意义的。

教给你开发人员基本的建模知识；否则，他们会只看看你画的漂亮图表，然后继续编写不规范的程序。

另外，你还需要告诉你的用户一些需求建模的基础知识。给他们解释你的用例 (uses case) 和用户界面模型，以使他们能够明白你要表达的东西。当每个人都能使用一个通用的设计语言的时候（比如 UML-译者注），你的团队才能实现真正的合作。

21. 带工具的傻瓜还是傻瓜

你给我 CAD/CAM 工具，请我设计一座桥。但是，如果那座桥建成的话，我肯定不想当第一个从桥上过的人，因为我对建筑一窍不通。

使用一个很优秀的 CASE 工具并不能使你成为一个建模专家，只能使你成为一个优秀 CASE 工具的使用者。成为一个优秀的建模专家需要多年的积累，不会是一周针对某个价值几千美元工具的培训。一个优秀的 CASE 工具是很重要，但你必须学习使用它，并能够使用它设计它支持的模型。

22. 理解完整的过程

好的设计人员应该理解整个软件过程，尽管他们可能不是精通全部实现细节。

软件开发是一个很复杂的过程，还记得《object-oriented software process》第 36 页的内容吗？除了编程、建模、测试等你擅长工作外，还有很多工作要做。

好的设计者需要考虑全局。必须从长远考虑如何使软件满足用户需要，如何提供维护和技术支持等。

23. 常做测试，早做测试

如果测试对你的软件来说是无所谓的，那么你的软件多半也没什么必要被开发出来。

建立一个技术原型供技术评审使用，以检验你的软件模型。

在软件生命周期中，越晚发现的错误越难修改，修改成本越昂贵。尽可能早的做测试是很值得的。

24. 把你的工作归档

不值得归档的工作往往也不值得做。归档你的设想，以及根据设想做出的决定；归档软件模型中很重要但不很明显的部分。给每个模型一些概要描述以使别人很快明白模型所表达的内容。

25. 技术会变，基本原理不会

如果有人说“使用某种开发语言、某个工具或某某技术，我们就不需要再做需求分析，建模，编码或测试”。不要相信，这只能说明他还缺乏经验。抛开技术和人的因素，实际上软件开发的基本原理自 20 世纪 70 年代以来就没有改变过。你必须还定义需求，建模，编码，测试，配置，面对风险，发布产品，管理工作人员等等。

软件建模技术是需要多年的实际工作才能完全掌握的。好在你可以从我的建议开始，完善你们自己的软件开发经验。

以鸡汤开始，加入自己的蔬菜。然后，开始享受你自己的丰盛晚餐吧。