链滴

设计的载体是什么

作者：localvar

原文链接：https://ld246.com/article/1522382383093

来源网站：链滴

也是一篇发在公司内刊上的文章。

从参加工作开始，我就一直困惑于如何写设计文档。公司的各种规定和规范看起来很美，但真的执行来却总让我头痛不已；同时，我也觉得自己喜欢的方式有些自由散漫，满足不了商业化开发的要求。到前不久，偶然在网上看到了一篇题为《源代码就是设计》的文章后，我终于感觉自己把这个问题想清楚些了，所以写了本文，与大家交流。这也许是篇偏激的文章，但我相信它提到的很多问题是我们以回避的。

公司一直把文档作为设计的主要部分，并且，我认为，公司对这份文档的要求至少包含以下几点：

1. 能够指导程序员编码。最好是设计师可以完全脱离编码，完成设计后能立即投入下一个项目。
2. 提供评审和QA监督的依据，以便尽早发现不合格项，尽早纠正。
3. 作为项目维护的依据。

但仔细想想会发现，文档并非做到这几点的必要条件。换句话说，有了文档不见得能做到，没有文档不见得做不到。

首先，只要设计没有细化到本身就是代码的程度，程序员就不可能仅参照设计编码。这是因为，文档—即使是写作过程中使用了很多软件设计工具的文档——也是一种更接近自然语言的东西，而自然语有一个天生的弊端——二意性。所以，不论它写的有多好，程序员理解起来也不会与设计师所想的完一致。而让设计师完全脱离编码也是一个不太现实的想法，一个设计师或许能在一个项目中这样做，如果每个项目是都是如此的话，他日后肯定会眼高手低，那时他的设计就根本没法用了。我们也许听过一些新闻，说国外某项目的主设计师根本没写过代码。我不否认这些新闻，但我认为我们绝不能照这种模式，因为那些设计师往往是某方面的世界级权威，而人比人是要死的。

其次，评审和QA方面的目标也基本没有可能达到。因为大多数情况下这两个过程是“外行监督内行。我这里并不是说参加评审的同事和项目组的QA水平不足，只是说相对于项目组成员，他们对项目投入要少的多，对项目细节的了解也少的多。这种情况下还想让他们提出评审问题或监督项目执行，求未免高了一些。以XYZ产品为例，评审阶段发现的问题数一般不多，到了Y项目更是可怜，只有组级要求的几分之一了。而如果再仔细观察一下这些问题，还会发现其中绝大部分无关紧要，真正价值大的往往只有一两个。至于QA，由于其监督的主要是过程，所以更容易糊弄，并且极有可能是项目的故意糊弄（评审上的一般是无意的）。我相信这种糊弄在各个公司都不罕见，前一段时间来培训单测试的讲师就讲到过他们从前（应该是在华为）在项目中糊弄QA的事例。作为反例，公司的统计数已经表明，代码走查这个内行监督外行的过程发现问题的效率是最高的。

第三，已有的经验也说明设计文档在项目维护过程中没多大用。从公司内部看，到目前为止我还不知哪个项目是主要靠设计文档维护的。项目出问题后，我们首先想到的一般是找参与过项目的人，然后是文档和其它的东西。我想这个过程应该能说明：人，而不是文档，才是项目维护中最重要的因素。然，我们也可以说这是因为文档写的不够完美，不过我希望你读完本文后能同意“完美的文档是不可的”这个观点。再者，从我接触过的开源项目看，项目的维护也不需要文档，开源项目一般没有太好设计文档，但人们还是能仅仅通过源码和注释等就参与进去。另外微软的文档应该是非常好了，可开时我还是要经常去看看MFC、ATL的源代码，因为那才是最准确的第一手资料，MSDN上描述的不太楚的东西，看看源码就一目了然了。

其实，我觉得，把文档看成设计不光满足不了公司的要求，还会带来其他的一些弊端，比如工作效率一致性等。

作为开发人员，我想没有几个人喜欢写文档，我们常说“兴趣就是动力”，那没了兴趣肯定也就没动了，所以写文档的效率也就可想而知了。以我自己做例子，我认为自己写文档的效率比写代码低的多即使抛开主观因素，由于中英文输入效率的差别和画图与打字的效率差别，写文档的也要写代码慢不。所以，一份“完美的设计文档”可能会大幅增加项目成本，以至于不能公司不会接受。

再进一步，就算能写出来一份“完美的设计文档”，我们也不可能一劳永逸，因为设计会变更。大家

往非常关注需求变更，但设计变更发生的更频繁，道理很简单：需求变更会导致设计变更，需求不变设计也可能变更。设计发生变更的时候，代码或许已经完成了，加上工期的要求等，开发人员很有可会先改代码后改文档，这时问题就出现了：开发人员会“忘了”改文档。或者即使他没忘，由于这份档的初稿很“完美”，他要修改的地方会很多，工作量会很大，甚至要花数倍于他改代码所用的时间行，再考虑上人的惰性、其他工作的压力等因素，他敷衍了事应该是再正常不过的了。而我们前面已分析过，这种敷衍是很难被发现的。反正不管是因为什么原因，经过变更后，我们的“完美”文档就代码对不上了。

XYZ是一个以数据处理为主要功能的项目，所以我在刚开始时考虑了很长时间“在不能保证数据的正性时应该怎么办”这个问题，最后的决定是把这些数据扔掉，因为我认为“错误的数据不如没有数据。把这个原则应用到项目上来应该就是：错误的文档不如没有文档，即使这份文档看起来很“完美”。

回到《源代码就是设计》这篇文章上来。实际上，仅仅是它的标题就让我眼前一亮了。从上学时的《件工程》教科书，到工作后的实际项目，先入为主的“设计是文档”让我一直没有仔细想过设计应该什么。这篇文章给了我一个思考的机会，并且经过仔细比较后我发现：代码是比文档更好的设计载体甚至是天生的设计载体。

我们经常把做软件和盖房子做对比，并把程序员比作建筑工人，这也带来了所谓“软件蓝领”的称呼但我要说这个对比把程序员和建筑工人放在一块是不恰当的，最简单的理由是建筑工人把砖块磊整齐非常重要，程序员把代码排整齐了则没那么重要（虽然也很有用）。我认为：程序员应该对应建筑设师，编码应该对应画图纸。你也许会问：这个对比中，磊砖块对应什么？别着急，软件开发中还有一大家已经见的太多以至于被忽视了的过程——编译，它对应着磊砖块，也就是把设计变成现实的这个程，只不过由于软件开发的自动化程度太高，它完全被计算机完成了而已。如果我们把盖房子换成加机器零件，这个对比的正确性就更明显了，现在自控机床已经能自动加工零件了，而以前，这个工作要很多人工的。或许未来会有一种机器能根据图纸自动盖房子。我们发现了编译这个之前一直被忽视的过程之后，应该会注意到，在把文档看成设计的那个对比中，盖房子没有与编译对应的过程，这也明它是不恰当的。

把代码看成设计的一个显而易见的好处是消除了设计和实现的不一致性，编译代码速度很快，时间成极低并且基本不需要人工，所以保证这二者的一致是非常容易的。另外它也提高了工作效率，因为主方面开发人员更喜欢写代码，客观方面英文输入速度比中文快得多。

虽然有不少好处，但从文档到代码的这个转变有点太大，恐怕大家会非常怀疑它是否真的可行。下面就来分析一下大家可能的疑问。

第一个问题是设计师和程序员如何协作，也就是说设计师怎么让程序员按照自己的想法去做事情。我先来看设计到底是怎么进行的，乍一看大家可能认为设计是“想”出来的，但实际上它更多的是“试出来的。任何一个项目都会有一些设计师之前没遇到过的问题，区别只是多少而已。对这些问题，设师可能会在脑子里想一些方法，然后他就要去试验这些方法是否真的可行，而试验成功之时，也是针这些问题的设计完成之时，并且，往往也是相关代码写完之时。基于这一点，我认为设计师在一个项中的工作是完成系统框架和技术难点的开发，程序员则去做具体细节的开发。细节中的一些问题可能程序员没有遇到过的，解决这些问题的过程，不管是完全靠自己还是求助于设计师，对程序员来说也一个做“设计”的过程。所以，这种模式下，是不必严格区分设计师和程序员的，大家都是设计师也是程序员，而这也正好符合“源代码是设计”这一观点。

第二个问题是如何审查监督。我的看法是把这项工作交给项目经理或主设计师，而不做严格的外部审和监督。自己监督自己也许听起来不那么可行，但我相信大多数人，尤其是项目经理和主设计师，对目是负责的，所以这样做效果不一定就不好。用人不疑，疑人不用，这句老话用在这里应该很合适。且，我们前面已经分析过，如果项目组成员成心欺骗，外部监督机制也不见得有多大效果。

第三个问题是项目的风险会不会增大。设计做的不好一直被认为是项目拖期的主要原因之一，现在这（按原来的观点）看起来根本没有设计的模式会不会加大这个风险呢？我觉得不会，因为它把编码提了，而编码阶段是最容易发现问题的阶段，所以设计上的问题会被更早的发现，更早的修正，从而降项目风险。另外，代码重构（在这个模式中相当于设计变更）的代价可能也没有想象的那么大，我自经常重构大量代码，我觉得花的时间并不多。最后说句实话，XYZ相关的项目一直都是先编码后文档

但它并没有因此发生过拖期，而且从公司的统计数字看，它的生产率也一直是高于组织级标准。

第四个问题是项目如何维护。前面我已经论证了文档不可能作为项目维护的依据，或者至少是主要依，所以没有文档也不会让这个情况再变坏多少。另外，不做开发的人可能会觉得文档更易读，但开发员就不是这样了，他们受过专业训练，对代码的条件反射更强烈，就像音乐家看到123会首先想到哆咪一样。因此，项目维护是可以基于代码进行的，而且，稍后我会说明这种新的开发模式也并非没有何文档。

前面的文字应该已经基本证明了“源代码就是设计”的可行性，但一个项目如果只有代码肯定也不行因为代码的组织很松散，开发人员很难通过它们形成对系统的总体认识。所以，一些文档是必须的。面我就对“应该有哪些文档”和“这些文档应该怎么写”发表一下看法。

第一份应该有的文档是需求，虽然它是需求阶段的输出，但我也要在这里强调它。我认为需求是一个目中最重要的文档，它规定了系统开发出来后应该是什么样的，必须全面而详细。有经验的开发人员往看到需求就知道“应该如何实现这个系统”或者“这个系统应该是怎么实现的”了。

第二份文档是系统的总体结构图和一些简要说明，它应该在系统框架开发完成后输出，以便指导项目后续开发。这份文档可能只要很少的几页就够了。

第三份文档是通讯协议格式和文件格式等，这些内容是结构性的，而不是逻辑性的，所以文档能更清的把它们描述出来。数据库格式也可以写，但考虑到可以在大多数DBMS中直接看到数据库结构，它必要性并不大。这些文档可以在相关部分开发完毕后输出，也可以等整个项目开发完了再输出。

第四份文档一些描述系统关键点实现方法的论文。由于针对单个问题，所以它们的主题很明确，不大能因为太松散而失去价值。这些论文可以等项目结束后再写。

总之，我认为设计相关的文档应该等代码完成之后再写，这样最大程度的避免了文档和代码的不一致这些文档的篇幅也不宜过大，把一些框架性的东西说清即可，描述具体细节是代码的事。另外我觉得司也可以考虑为项目组或个人建立内部BLOG，并鼓励大家发表相关文章，以促进交流。