利用马克思主义方法分析，软件危机发生，客观上是由软件本身的特点造成的，主观上是软件开发与维护的方法不正确造成的。

**1.现阶段软件设计文档不规范，无法验证、分析、证明，导致产品可能存在错误、不能很好满足用户需求，造成后期的反复修改**

软件开发的过程是不可见的构思具体化、可见化。软件开发的本质是一项工业化的作业，软件的研制过程包含了软件的开发,测试,维护,和升级等明确标准的环节，但具体到其中的每个步骤，即使使用了形如量化空间等衡量方法，各项指标难以量化。

同时，软件开发包括了软件的开发,测试,维护,和升级等环节，不同阶段的程序语言可能迥异，即使是同一阶段，不同的程序员写的内容也具有很浓的个人风格。为此，想要实现软件开发的团队化作业，就需要一个被大多数人认可且高可读性的说明文档，否则高昂的交流成本将让软件开发的难度指数型增加。

常见的软件开发设计的文档大致包括了可行性分析报告、项目开发计划等十余种，而通过在国家标准化管理委员会官网上搜索，有很大部分缺失对应的文件。

**2.模块内聚性，逻辑内聚性模块的代码，内聚性低，但又无法避免，导致软件产品质量，增加维护难度**

如果一个模块执行多个完全不相关的动作，那么这个模块就有偶然内聚性，当一个模块执行一系列相关的动作,且其中一个动作是作为其他动作选择模块，称其有逻辑内聚性。

对于前者，第一个问题是在改正性维护和完善性维护方面，这些模块降低了产品的可维护性；第二个问题是这些模块不能重用，这和我们期待的软件开发工业化、标准化冲突。

对于后者，也有两个问题：其一，接口部分难于理解，其二，多个动作的代码可能会缠绕在一起，从而导致严重的维护问题，甚至难于在其他产品中重用这样的模块。

各个动作之间没有任何逻辑关系，它们可能只是或是由于客户的需求、或是因为历史原因和方便管理而被放在同一个模块中。

**3.程序代码无法证明其正确性，只能通过测试验证**

计算机科学中“停机问题”这个不可解问题。停机问题指出不存在一个算法可以判断任意给定的程序是否会停止运行。换句话说，我们无法编写一个通用的程序来检查其他程序是否总是产生正确的结果。

此外，程序的正确性还取决于输入数据的完整性和正确性。即使程序本身是正确的，如果输入数据存在问题，也可能导致错误的结果。因此，在软件开发过程中，我们需要对程序进行充分的测试，以确保其在各种情况下都能产生正确的结果。

总之，由于停机问题的存在以及输入数据的影响，我们无法证明程序代码总是正确的。相反，我们需要通过测试来验证程序的正确性。

**4.客户的需求是具体并具有浓厚的个性化色彩的、且随时随地在变化,很难用某一种"标准的,通用的"软件就能满足全部客户需求。**

首先，用户对软件产品的质量要求是满足需求、可靠、可操作性好、可维护性强。软件产品是厂家适应客户的模式。是否正确分析客户的需求是着手开发软件的第一步,如果分析不恰当,那么在此基础上研制的软件也就报废了。

另外,客户的需求是动态的,即使依靠着标准文档和各种法律手段，需求的变化也是客观存在着，和不同的需求和程序实现之间也有不少的冲突。

为此，改变软件功能的难度远大于用户改变需求的难度。

**5.基于软件标准化的构件技术和对人工智能的期待**

目前，软件规模和复杂度都大大增强了，但软件开发依然是可以通过一个个的模块和模块间的接口搭建起来。实现软件开发模块化的同时，也重视模块的复用，这不但是软件开发工业化的一个重要特征，更是在一定程度上提高了软件的生产率,有利于缓解软件危机。

构件是构成软件系统"即插即用"的软件成分,是可以独立制造,分发,销售,装配的二进制软件单元。构件是具有一定功能,能独立工作,并能和其他构件装配起来协同工作的程序体,它的使用与其开发,生产无关。

首先进行领域分析,得到软件系统的需求描述,再根据软件需求建立实现的应用框架。

将复杂的系统向下依次划分为若干子系统,子模块和构件,将现有的和创新的构件以某种有机,合理方式组织起来,再进行优化,调试,测试,最后实现软件系统。

从根本上改变的了软件的开发模式,大大提高了软件的重用率,有效利用了已有资源。模块化的软件系统具有更高的灵活性,有利于维护和升级。不需要掌握构件的编程技巧,只需要熟悉装配过程即可。

但是，这依然是治标不治本的行为，软件标准化不能从根本上解决软件危机。这种裱糊式的手段仍然是针对形式而非内容,不能从根本上解决软件危机。

大幅度提高软件生产率必须依靠一种比编制代码更高层次,更注重全程自动化的设计手段,而不能单纯依靠软件的标准化。

全程自动化的设计手段可能需要希冀于人工智能的发展，目前的人工智能可以读取理解程序员的大纲式指令和对每个细节的简单指令。