### 摘要

所有软件活动包括：根本任务，即打造构成抽象软件实体的复杂概念结构；次要任务，即使用编程语言表达这些实体，在空间和时间限制下将它们映射成机器语言。

没有任何技术或管理上的进展，能够独立地许诺在生产率、可靠性或简洁性上取得数量级的提高。

在关注根本任务时，建议：

1. 仔细地进行市场调研，避免开发已上市的产品；
2. 在获取和制定软件需求时，将快速原型开发作为迭代计划的一部分；
3. 有机地更新软件，随着系统的运行、使用和测试，逐渐添加越来越多的功能；
4. 不断挑选和培养接触新生代的概念设计人员。

软件的概念结构包括：数据集合、数据条目之间的关系、算法和功能调用等。

软件开发中困难的部分时规格说明、设计和测试这些概念上的结构，而不是对概念进行表达和对实现逼真程度进行验证。

### 根本困难

软件的**复杂性**是根本属性之一。软件实体的扩展不仅仅是相同元素重复添加，而必须是不同元素实体的添加。大多数情况下，这些元素以非线性递增的方式交互，因此整个软件复杂度要比非线性增长多得多。

软件的**一致性**要求软件必须遵循人为惯例和系统。许多情况下，最新开发的软件必须遵循各种接口；另一些情况下，软件的开发目标就是兼容性。

软件具有**可变性**一部分原因在于软件包含了很多功能，而功能是最容易感受变更压力的部分。另外的原因是软件是纯粹思维活动的产物，可以无限扩展因此能够很容易地进行修改。软件产品扎根与文化的母体中，如各种应用、用户、自然及社会规律、计算机硬件等。后者持续不断地变化这，这些变化无情地强迫着软件也随之变化。

软件具有**不可见性**即是不可见的和无法可视化的，它的客观存在不具有空间的形体特征。因此当我们试图用图形来描述软件结构时，发现它是很多相互关联、重叠在一起的图形、这些图形可能代表控制流程、数据流、依赖关系、时间序列和名字空间的相互关系等。实际上，建立概念控制的一种方法是强制将关联分割，直到可以层次化为一个或多个图形。

### 以往解决次要困难的一些突破

**高级语言**减轻了一些次要的软件复杂度。机器语言的复杂度高，高级语言所达到的抽象程度体现了抽象程序所需的要素，避免了更低级的元素，消除了并不是程序所固有的整个级别的复杂度。

**分时**保证了系统响应的及时性，从而使我们能维持对复杂度的一个总体把握，在提高程序员的生产率和产品质量方面起到了很大作用。

**统一编程环境**主要通过提供集成库、统一文件格式、管道和过滤器解决了共同使用程序的次要困难。它使得概念性结构理论上的相互调用、提供输入和互相使用，在实现中可以非常容易地实现。

### 银弹的希望

**Ada和其他高级语言。**Ada语言的重要特征就是其嵌入式风格，模块化设计，编译检查，平行处理，异常处理及泛型编程。Ada在95年加入了对面向对象设计的支持，包括动态分配等。Ada的最大贡献在于编程人员培训方式的转变，即需要对开发人员进行现代软件设计技术培训。

**面向对象编程。**必须区别两个不同的概念：抽象数据类型和层次化类型。它们的出现允许设计人员表达自己设计的内在特性，而不需要表达大量句法上的内容。

**人工智能。**人工智能可能给软件生产率和质量带来数量级上的增长。专家系统通过建议接口规则、制订测试策略、记录各种 bug 产生的频率、提供优化建议等等来提示软件生产率。

**“自动”编程。**从一段对系统的描述自动生成代码的技术在近几年逐渐走向成熟，并且会成为未来编程的趋势。

### 针对概念上根本问题的颇具前途的方法

所有针对软件开发过程中次要困难的技术工作基本上能表达成以下的生产率公式： 任务时间 = ∑（频率）i×（时间）。准确地表达复杂概念结构是软件概念上的根本问题。以下几点可能有希望解决该问题。

**需求精炼和快速原型。**快速原型将开发作为迭代需求过程中的一部分，在迭代的过程中客户和设计人员、开发人员进行多次广泛的交流，逐渐完善需求。

**增量开发——增长，而非搭建系统。**增量开发将系统分为多个构件，每个构件独立进行设计、开发、测试等。增量开发的方式首先要使得系统能够运行，即使未完成任何有用功能，接着系统一点一点被充实，子系统轮流被开发，或者是在更低的层次调用程序、模块等。