很明显的。修复局部问题的工作量很清晰，并且往往不大。但是，更大范围的修复工作常常会被忽视，除非软件结构很简单，或者文档书写得非常详细。其次，维护人员常常不是编写代码的开发人员，而是一些初级程序员或者新手。

作为引入新 bug 的一个后果，程序每条语句的维护需要的系统测试比其他编程要多。理论上，在每次修复之后，必须重新运行先前所有的测试用例，从而确保系统不会以更隐蔽的方式被破坏。实际情况中，回归测试必须接近上述理想状况，所以它的成本非常高。

显然，使用能消除或至少能指明副作用的程序设计方法，会在维护成本上有很大的回报。同样，设计实现的人员越少、接口越少，产生的错误也就越少。

前进一步，后退一步

Lehman 和 Belady 研究了大型操作系统的一系列发布版本的历史。[6] 他们发现模块总数量随版本号的增加呈线性增长，但是受到影响的模块数量随版本号的增加呈指数增长。所有修改都倾向于破坏系统的架构，增加了系统的混乱程度(熵)。用在修复原有设计上瑕疵的工作量越来越少，而早期维护活动本身所引起的漏洞的修复工作越来越多。随着时间的推移，系统变得越来越无序，修复工作迟早会失去根基。每一步前进都伴随着一步后退。尽管系统在理论上一直可用，但实际上，整个系统已经面目全非，无法再成为下一步进展的基础。而且，机器在变化，配置在变化，用户的需求在变化，现实系统不可能永远可用。崭新的、基于原有系统的重新设计是完全必要的。

通过对统计模型的研究，关于软件系统，Belady 和 Lehman 得到了更具普遍意义、为所有经验支持的结论。“事物在最初总是最好的，”