### 导致软件衰退的因素

本质上导致软件运行中衰退的因素有三方面：

1. **环境动态多变**：软件**运行环境**包括软件运行所需要的各种条件，主要包括计算资源、存储资源、通信资源、支撑服务（操作系统服务和平台服务）资源。软件网络化使得软件的运行环境逐渐开放、动态和多变，随着虚拟化技术和云计算平台的发展，资源的按需分配模式将使软件运行环境更动态多变。而软件对环境具有较高的依赖性，其稳定性和性能也必然会受到环境动态变化的影响。
2. **隐藏缺陷间歇性暴露**：软件规模、复杂度、重用度、模块化程度的逐渐提高，不可避免地造成一些缺陷因测试不到位或未造成代码执行中断和产生错误结果而隐藏起来，未被消除。尤其是具有瞬时性、**间歇发生**的bug ，以及一些复用、购买的组件或服务资源中的缺陷（其源代码不可见而被视作“黑匣子”），更是难以消除。此外，软件的更新、升级和修改配置等维护行为也会引入新的、未知的缺陷，这些缺陷在软件运行过程中，**随机**地被激活，会逐渐暴露、引发错误。（影响正确性、稳定性）
3. **环境与软件交互的不确定性高（随机性）**：软件（尤其是网络应用软件）的一个典型特征是其**负载**具有高度动态性和不确定性：用户输入随机（**输入环境**）、使用强度随机，软件与环境交互过程中，**资源的分配、释放**频繁，造成资源的耗损。（配伍禁忌？）对bug的激活具有不确定性。

上述因素导致运行的软件在与环境的不断交互中，引发资源竞争、耗损，稳定性、正确性、性能逐渐下降。而这些因素又难以避免，因此，软件衰退像人的老化一样，是不可阻止和避免的，它正在成为软件健康面临的威胁的新趋势。

### 软件衰退的特点

与其他故障现象不同，软件衰退是成熟软件在运行时受到环境的变化和隐蔽的、间歇性的、暂时性的缺陷等因素威胁，长期累积的过程，它因这些因素的难以避免而必然存在，其导致的稳定性、性能下降或失效迟早会发生。上述这些威胁具有环境敏感性，能够激活它们的相同情景是随机的、很难重现，因此极难通过测试来再现和排除它们。由于软件自身的动态性及其对环境的依赖性和未知缺陷层出不穷，偶发性、不确定性和环境依赖性成为了软件衰退的三大特征：

1. **偶发性**：衰退相关的内外部威胁（fault等）具有隐蔽性、间歇性，这类bug的激活依赖于软件自身的内部环境和外部环境（运行环境、输入环境），因此衰退时隐时现。
2. **不确定性**[58]**：**软件运行在开放、动态和多变的环境中，其功能和结构日渐复杂化、多样化且在不断动态变化（更新、升级等），未知的会引发衰退的缺陷也层出不穷，衰退的表现和模式、趋势也随之动态多变且多样；从软件开始运行到触发导致衰退的因素、再到发生性能下降或失效的时间和间隔是随机的，因为触发衰退的条件及其强度、软件的使用强度、运行刨面、软件相应的进程执行的工作量和工作类型等因素具有随机性。
3. **环境依赖性（环境多样性）**：在不同的环境中，软件衰退的表现不同；换一个运行环境，软件衰退的影响可能就不一样。

上述特点使得衰退软件的健康情况每况愈下，管理员个人却很难招架。研究显示，多种软件系统已受到软件衰退的威胁，如电话通讯及其计费系统[1, 3]、事务处理服务器[6]、web服务器软件[28, 31]、航天器控制系统[22]、爱国者导弹防御系统等安全相关的系统[16]（查权威调查数据）中间件、cluster servers [5, 14].

在企事业、电子商务等信息系统中，由于软件衰退的固有性及环境依赖性，集群服务器系统中各server承载的软件具有类似的衰退概率和模式，造成稳定性、性能下降，甚至整个信息系统的崩溃。这引起了贝尔实验室、IBM、摩托罗拉全球软件中心、Sun等软件开发商和用户的高度重视。

### 软件抗衰方法的现状

从导致软件衰退的因素的分析可以看出：软件衰退难以在软件的设计、开发、测试和在运行维护阶段避免和阻止[41]。在这种情况下，软件运行阶段的容错方法是对抗它的有效手段。目前，传统的容错方法（如设计多样性（代码N版本）、数据多样性）无法耐受软件衰退问题；业界选择牺牲成本（预算可能会高4-10倍）来换取业务，采用硬件支持的集群技术对关键应用模块（的代码、数据、硬件）进行冗余、备份、负载均衡，暂且不说很多应用系统限制使用集群[12]，这种做法是在软件发生挂起和/或崩溃失效后进行恢复和负载转移，属于被动式容错管理，一来正在处理的业务会丢失，二来被动式恢复至少需要3-5分钟，此间该软件无法服务会造成集群中其他版本超载[8][12]，三来无法提前处理服务效率和性能低下。而且，这种被动式容错只能较好的避免硬件错误，无法避免衰退问题、以及冗余版本也出现衰退/失效的情况。因此，主动的、预防性的、自适应的容错管理是增加软件可用性、降低软件失效导致的损失的关键策略之一，软件抗衰就是这样的方法。

软件抗衰是一种主动积极的、预防式的应对软件衰退现象的故障管理方法，属于一种环境多样性容错方法，它致力于有计划地、主动地清除运行中软件的内部错误，给软件换一种 “干净的”运行环境，使软件尽可能恢复到初始健康状态，以减小未来软件发生意外中断、业务遭到严重损害。这个过程可能会造成业务短暂中断，但其带来的损失远比被动式地恢复意外失效造成的损失小得多。

软件抗衰包含两层含义：首先是指抗衰活动要在性能严重下降、失效发生前采取；其次，抗衰技术本身应具有自主智能性。目前，软件抗衰方法总体上可以分为三大类：基于时间模型的方法、基于度量的方法和基于时间模型与度量的混合方法，这些方法主要是为预测软件衰退导致的失效而提出的。

基于时间模型的抗衰方法是目前实际应用中常用的抗衰方法。该方法是一种预防软件失效的定期抗衰方法：假设软件失效率固定，依赖于已知失效时间和负载分布数据，采用随机Petri网、马尔科夫等数学模型对软件建模，再用随机过程、统计分析等数学方法求得软件运行状态的分布，进一步计算最优的抗衰时间和周期。软件运行时，到抗衰时间点后，按固定周期执行抗衰。由于抗衰时间和间隔是采用数学方法根据已知失效分布数据求解得到，因此具有一定的统计正确性。但该方法最大的不足是宏观、缺乏灵活性，不顾软件的实际运行情况，因为软件衰退过程和失效率一般不是固定不变的，且失效时间分布数据是未知、难以获取的。在软件网络化和资源虚拟化、按需分配的大背景下，动态多变、状态多样的软件，其衰退的表现模式、趋势越来越动态多变，该方法面临着越来越多的挑战。

考虑到系统资源耗尽是软件衰退的一大影响，基于度量的抗衰方法不是去求解特定的抗衰周期，而是通过刻画系统资源的耗费情况来预测衰退导致的失效，是一种视情抗衰方法。该方法首先建立系统资源耗费的趋势，求解系统资源耗费的速度，之后根据实时度量的资源耗费数据，结合当前的负载，预测资源耗尽的时间作为软件会失效的时间。基于度量的方法更符合软件的实际运行情况，最大的优点是可以预测部分未知的软件衰退情况，但不足是这些方法容易漏掉偶发性强的软件衰退，因其异常数据难以获取。且该方法一般基于特定的软件衰退影响（如内存泄露）进行预测，不能自适应的感知和预测动态、多样的软件衰退现象，无法正确区分负载改变等正常现象和真正的性能异常。

基于时间模型和度量的混合抗衰方法，采用基于时间模型的抗衰方法常用的建模方法，计算软件运行状态的大致分布，在此基础上计算需要监测的关键点的时间与间隔；软件运行时，在关键点开始监测软件运行情况，并根据当前软件衰退状态所处的阶段，决定采取的抗衰的粒度。尽管该类型方法一定程度上弥补了基于时间模型的抗衰方法的灵活性，避免了基于度量的预测模型过高的计算代价，但其预测关键点是固定的，难以适应不确定性软件衰退现象，无法正确区分负载改变等正常现象和真正的性能衰退；而且，由于其抗衰预测本质上属于基于度量的方法，在预测偶发性强、动态、多样的软件衰退时，也面临着易被逃避的问题。这也是本来重点要解决的问题。