软件工程

软件可靠性：软件可靠性 (software reliability )是软件产品在规定的条件下和规定的时间区间完成规定功能的能力。规定的条件是指直接与软件运行相关的使用该软件的计算机系统的状态和软件的输入条件，或统称为软件运行时的外部输入条件；规定的时间区间是指软件的实际运行时间区间；规定功能是指为提供给定的服务，软件产品所必须具备的功能。软件可靠性不但与软件存在的缺陷和（或）差错有关，而且与系统输入和系统使用有关。软件可靠性的概率度量称软可靠度

软件可维护性：软件可维护性即维护人员对该软件进行维护的难易程度,具体包括理解、改正、改动和改进该软件的难易程度。

决定可维护性的因素:1.系统的大小2.系统的年龄3.结构合理性

可维护性可通过7个质量特性来衡量:可理解性 、[可测试性](https://baike.baidu.com/item/%E5%8F%AF%E6%B5%8B%E8%AF%95%E6%80%A7)、可修改性、可靠性、可移植性、可使用性、效率

　一**.计算机软件可维护性一般方法**

　　（1）提升软件工具模块化和质量技术

　　在软件开发过程，有效方法之一是提高软件质量和降低成本，其有效技术也是提高可维护性。它的优点是，如果需要改变一个功能模块，只需要改变这个模块，不会影响到其他模块;如果程序需要添加一些功能，只需完成这些功能，增加一个新的模块或模块层;程序测试和重复测量更容易，序列错误很容易发现和改正，以提高程序的运行效率。采用结构化程序设计技术，以提高现有系统的可维护性。这种办法需要掌握更换模块的外部特征，不需要把握其内部运作的状态。它可以帮助其减少新的错误，并有机会提供一个结构化的模块，并逐步取代非结构化的模块，运用自动重建结构和重新格式化的工具。

　 （2）创建精密的软件品质目标和优先级

　　程序的维护性应该是可以理解的、可靠的、可修改和测试的、可移植的、可以使用和效率高的。为了实现这些目标，要求付出的代价很大，也未必是可行的。一些质量特性存在互补性，如可理解性和可测试性、可理解性和可修改性等。然而，其他一些质量特性互相矛盾，如效率和可移植性、效率和可变性。因此，各品质特性的维护性要求可以得到满足，但它们相对重要性应遵循程序使用作用和计算环境变化而变化。

　　（3）选有可维护的程序设计语言

根据程序可维护性，选择程序设计语言，其影响是极大的。低层次的语言就是机器语言和汇编语言，这非常难以理解和掌握，也更难以对其进行维护。高级语言更容易理解，具有更好的可维护性，而低层次语言相对要差，但作为高层次语言，难易程度不一样也是可以理解的。一些第四代语言是过程化语言，而有些是非程序语言。不管是什么语言，程序编制出来都很容易理解和修改，但存在指令数量可能会少一个数量级，而语言编制数量级要多一个，其开发速度会快多倍。

（4）加强计算机软件可维护性的研究

在软件维护过程，经常遇到一些问题，如频繁的员工流失率，已离开的原有开发商;缺乏文档资料，很难了解其他人开发体系;不符合程序或文档的文件不适当，并很难理解，软件结构不合理，难以修改或修改后容易出现错误。该软件易于开发，但其难以维持，通用性较差，这是以前设计软件比较常见的通病问题，也是在同一个系统或重复开发的原由。重复开发会加强其系统功能，但单位人力、物力和财力资源会被浪费，而且还影响系统的正常使用。在软件开发过程，应充分和适当地思索，其系统通用性和自我维护能力，以避免系统开发重复是十分必要的，而且软件开发过程是需要重点留意的地方。

　　如果要设计多功能易于维护的软件，就必须有以灵活、通用和易维护为主旨的设计方法和思路。体系共性和个性方法分析，实现了对系统自维护功能的具体保证。在实际应用进程中使用此系统，并且得到用户的好评。在该系统自维护功能概念基础上，调整其参数，其实可以做一个小的开发工具，进而可以开发类似的系统管理。这至少表明，引入该系统自维护功能定义，为系统使用和维护管理带来了极大方便。

　　领域计算机管理在企业中的应用范围更为广泛，程序更加深入。计算机管理应用程序可大致分为两个主要方面，即工业控制和信息化管理。信息化管理水平，即使市场上有很多的软件支持，但考虑到企业的实际情况，要能更好地适应自己的管理模式，并更有效地管理自己的信息，一些核心应用系统主要由企业自主开发达成。在大多数的软件管理上，报表类软件为大部分，它们处理的报表主要是计算总的检查验证等。

　　执行软件设计上，自我维护功能定义的引入，使软件更具生命力。系统自维护功能给系统设计带来了一定难度，需要采取一些额外系统资源占用，但随着计算机技术的发展，到今天为止，它的资源不被认为是一个重要因素，考虑到对系统维护性影响，达到自我维护功能是值得的。对于其他软件系统的开发，如在开发中能够充分考虑系统的共性和个性，添加到系统自我维护功能的观念，体现技术应用，且实现更好的发展。

二.软件工程可靠性实现

1、避错设计

避错设计是传统的可靠性设计技术,在长期的可靠性实践中产生了大量的理论并积累了丰富的经验,是实现实时软件高可靠性目标的有力保证。它在充分应用软件工程技术、方法、工具和加强软件工程管理的基础上,针对实时软件的具体特点,采用形式化设计、抗干扰设计、软硬件相结合等技术和方法。此外,彻底的软件测试是保证实时软件可靠性的重要手段。

（1）.实时操作系统与BIOS设计

实时操作系统和BIOS的设计,不仅是保证系统响应及时性及实时任务处理等的根本,而且是设计实时应用软件的基础。

a.实时操作系统设计

实时操作系统不仅要覆盖系统的常规功能,还必须具有任务或时间的实时处理能力,要求能支持半导体盘、Watchdog和XIP程序的上电直接运行等。实时操作系统设计的基本目标是,提高系统引导速度、运行快速性和代码可靠性。

为提高系统和代码的运行速度,一般避免使用可靠性较低的软硬驱设备,将重复执行的代码放置在可靠性较高且不易丢失数据的固态存储器中。这样不仅提高了系统的引导速度,还提高了可靠性。而且,ROM盘可以以I/O或局部映射的方式访问,不占用实模式主存空间,并可在保护模式下对其进行访问。

XIP方式的提出,是建立在程序运行的高可靠性和新的存储结构要求的基础上,操作系统应是XIP格式的。目前,一些XIP格式的实时操作系统已开发成功,它们具有精简的内核和OM的可配置性。选择实时操作系统时,可根据要求,使用相应的配置工具,即可获得合适的配置结果。例如,为保证兼容性和系统配置的简洁性,可将一个XIP DOS配置到64KB之内,仅包括一个实时运行内核和最小化的命令解释器。

对可靠性要求更高的实时系统,常采用容错操作系统。它不仅在软件结构上保证了可靠性,而且对容错计算管理层中的错误检测与诊断、系统重组与降级、错误恢复与重构,以及软件故障检测、限制与恢复等提供支持,从而进一步提高了可靠性。

b.BIOS设计

由于BIOS直接与硬件挂接,因此其设计直接影响系统的性能。通常,为保证BIOS的功能及系统运行的可靠性,BIOS的设计应遵循简洁性、可裁剪性、可调试性、可调整性和独立性,以及可靠性和可维护性等原则。遵照这些原则,首先,BIOS在软件结构上采用模块化设计思想并在建立一个可覆盖所有应用的功能模块集的基础上,建立模块的筛选层;其次,在BIOS代码中,为各个阶段可能的软硬件调试与维护预留适当的调试出口、寄存器状态监视、嵌入式dbug等,为调试提供方便。当然,这些功能也应是可筛选的。此外,在确保BIOS代码覆盖系统基本硬件资源和确保BIOS代码集基本配置的基础上,BIOS代码应尽量小,以便在内存的F段节省出一定的空间,以备特殊用途。

为了能从根本上保证程序指令的高可靠执行,实时软件已逐步从加载运行的重定位\*.ee格式,转向就地运行的预定位\*.bin格式。因此,不论实时操作系统,还是BIOS,都应该是XI格式的。它也是程序固化和就地执行的前提。当然,用高级语言、或用高级语言与汇编语言混合编制实时应用软件时,代码与数据的分离和定位是XIP格式的关键。

(2).实时应用软件设计

目前,软件开发有很多有效的形式化方法。它们针对不同的软件系统、管理模式和开发使用环境发挥不同的作用。其中,基于生命周期模型的开发方法对软件工程管理和提高软件产品可靠性最有效。它遵循软件生命周期的划分,明确规定每个阶段的任务。在软件开发中,按照工程管理的原则清晰地划分软件的时间和任务阶段,严格按要求完成每个阶段的任务并产生该阶段文档,同时对相应阶段进行评审或测试,进而保证软件产品的可靠性。在目前尚无更有效的软件工程方法和开发模型的情况下,严格遵循软件工程原理,按照软件生命周期模型进行软件工程设计,是保证实时软件可靠性的基础。

a.健壮性设计

实时软件仅有正确性远远不够,还必须具有一定的防止错误输入的能力,在发生故障时应能有效地控制事故的蔓延,并进行报警输出处理,使之具有较强的健壮性。

提高软件健壮性的措施有:

* 检查输入数据的数据类型,防止操作失误。
* 模块调用时检查参数的合法性,控制事故蔓延。
* 降低模块之间的耦合度,简化软件的复杂性,实现信息隐蔽。

这些措施虽然明显地提高了软件的健壮性,但没有从根本上解决问题,且工作量极大。在常规软件设计中,数据结构与其操作分离,使它们之间存在着潜在的不一致性,不利于改进软件健壮性。面向对象程序设计把数据结构与其操作封装在一个对象中,不允许其它类直接访问它的数据,改变了传统的数据访问方式,从而彻底消除了潜在的不一致性,提高了软件健壮性。

b.抗干扰设计

实时软件一般是嵌入式软件,其可靠性常常受到嵌入环境和外部干扰的制约。因此,进行软件抗干扰设计势在必行。容错设计、冗余设计、抽象复算、指令复机、纠错编码、设备重复、自动诊断、自动重组、自动修复系统等技术都是有效的抗干扰设计方法。

实时软件因受干扰而使程序"跑飞"或"死锁"时,可重新启动,并初始化。程序的限界运行也是处理程序因干扰而"跑飞"的有效方法。程序运行的时间监视是处理因干扰导致"死锁"的一种有效方法。限时运行方法常用于已知子程序或程序功能块运行时间的情况。软件陷阱是在程序中的适当地方加入陷阱入口/出口语句,当因干扰而发生程序"跑飞"时,就可能落入预设的陷阱。陷阱的出口由设计人员预先设定,这样程序运行就进入可控阶段。有时,实时软件的数据采集会因环境的电磁等干扰而使所采集的数据中含有干扰成分。为此,可在实时软件中植入数字滤波器,对数据进行平滑处理,以提高数据精度。

此外,软件工具的应用也是实时软件可靠性的重要保证。目前,在软件需求分析、软件设计、软件测试、正确性证明、软件验收、软件维护及软件工程管理等各个阶段都有相关的工具予以支持。软件工具根据任务需求,严格按既定的标准和规范工作。在软件开发中,应尽可能使用优秀的软件开发工具。同时,对软件的部分功能和性能测试,软件工具也是极其有效、甚至必不可少的。

在软件生命周期的各个阶段、尤其在需求分析和软件设计等重要阶段进行严格的评审和测试,是发现错误、提高可靠性的有效办法。此外,软件可靠性问题不仅来自于软件设计,更大程度上来自于无约束的随意修改。因此,在实时软件的可靠性设计中,要严格技术状态管理,建立软件修改报告制度,按规定履行更改手续,保持软件技术状态的一致性、可操作性和可检查性。

(3).重入和并发

并行环境下的重入程序设计比单纯的递归调用更严格。如果设计不当,轻则产生数据计算错误,重则引起系统死锁,而且对时间敏感。但是,调试过程不一定能发现问题。

对可重入程序设计,可通过以下方法来改进其可靠性:

* 可重入程序使用的单元应使用堆栈或调用程序提供的临时工作单元。
* 可重入程序所使用的公共资源必须进行保护。
* 有临界区的可重入程序用封中断保护,待退出临界区时再打开中断。

并行是指程序执行的时间与其它程序有重叠现象,即多重任务与多进程并发运行。在并行处理中,公共资源保护是并发程序设计的重点。

4.结构冲突与回溯

实时软件的输入和输出数据之间可能很少、甚至没有结构上的对应关系,从而导致结构冲突。基本的解决办法是将冲突部分相互隔离开来,建立多个程序结构,再利用中间文件把程序结构联系起来,构成整体。Jackson法是目前较好的解决办法。此外,中间文件法、多道管理程序控制法、程序转换法和伙伴程序设计方法等也都是较好的办法。

在实时软件设计中所选用的循环和选择结构,在条件测试时,往往需要进行回溯。这样不仅影响其实时性,还可能导致软件的不可靠。软件设计中的回溯一般可用下列步骤解决:

* 先在分支点上确定测试条件,然后把问题简化成一个select结构。
* 在每一处可能判别的地方进行判别,加上quit...if,在此条件下退出此路径,然后再把select...or...改成posit...admit。
* 消除由select改变成posit所引起的副作用。

2、容错设计

完全或部分消除软件系统、尤其是软件故障后果特别严重的系统(如宇航系统等)的故障,是软件容错设计的基本目标。采用软件容错的目的在于,采取更积极的措施来降低因软件错误而造成的不良影响。实现软件容错的基本活动包括故障检测、损坏估计、故障恢复和缺陷处理。由于容错设计既容忍错误的存在,又能极大地提高可靠性,因此一开始就受到了实时软件、尤其是高可靠性要求的实时软件的青睐。软件容错自70年代受到重视以来,众多的研究基本沿袭了硬件容错的思想。目前,软件容错主要有以下两种方法:

(1).N版本程序设计

N版本程序设计是一种提高软件可靠性的屏蔽冗余技术,是由不同设计人员用不同算法与编程语言来实现的具有较高独立性但功能相同的软件。在每个程序中设置一个或多个交\*检测点,这N个软件在几台耦合疏松的计算机中分别运行,每执行到一个交\*检测点时便产生一个比较向量,并将比较向量传递给表决器,由表决器按多数表决或其它约定的策略做出输出决策。这种N版本程序设计方法于1972年由Elmendorf提出。随后,Arizenis和Chen分别于197年和1978年将其投入实际应用。

表决器是N版本程序结构的关键。由于表决程序规模较小,程序结构也不复杂,因此可以设计得很可靠,并可用程序正确性证明技术来证明其正确性。值得注意的是,表决器不仅能完成简单的表决功能,还必须允许因计算机字长等限制所造成的、不同程序的计算误差,并能在给定的误差范围内进行表决。表决器通常还要进行故障记录,这些记录将作为软件维护的依据。此外,表决器还应知道N个程序之间最大的运行时间差d,多数程序已输出结果后,表决器等待d时间后,如果还没有检测到输出,即判定该程序发生了差错。

根据多数表决的原理,N版本程序结构至少可以容忍(N-1)/2个程序中任意方式的软件故障。如果每个程序中的故障不会引起系统失效,则它可容忍多达N-2个故障。因此,N版本程序设计能有效地改善软件系统的可靠性。

小型程序通常没必要采用N版本程序结构。对于较大的程序,这种结构(即使N=3)对内存空间和执行时间的要求往往会超过用户的承受能力,从而限制了N版本程序结构的应用。但是,N版本程序设计技术是保证软件可靠性的最强有力的方法,对超高可靠性要求的实时系统是必要的。

N版本程序结构最好在多处理器系统上应用,N个程序可分配在不同的处理器上运行,使软硬件故障都能被容忍。N版本程序也可在同一处理器上先后执行,由于每个程序对硬件的使用部分或使用次序不同,一个硬件故障将在程序中表现为不同的差错或不表现差错。因此,即使在同一处理器上运行,N版本程序除能改善软件的可靠性外,还能容忍一定的硬件故障。

(2).恢复块设计

同硬件容错设计中的动态冗余技术类似,以一个静态冗余的N版本程序结构为核心,再用S个程序作后备,随时准备替换N版本程序中出现差错的程序,这样就构成了一个混合的动态冗余系统。但是,如果N版本程序结构退化到一个极端的情况,就可以得到一个特别有意义的动态冗余结构——恢复块结构。

程序的执行过程可以看成由一系列操作所构成,这些操作又可由更小的操作构成。恢复块设计就是选择一组操作作为容错设计单元,从而把普通的程序块变成恢复块。被选择用来构造恢复块的程序块可以是模块、子程序、程序段或过程等。一个恢复块含有若干个功能相同、设计有异的程序块,每一时刻有一个程序块处于运行状态。一旦出现故障,则以备用块加以替换,从而构成动态冗余。这种方法可在一台计算机上使用。

恢复块结构的可靠度极大地依赖于软件故障的恢复覆盖率,过低的恢复覆盖率将使恢复块结构失去意义。恢复块结构能较好地改善软件系统的可靠性。

恢复块结构中的恢复块可以不必存在内存中,并且一般情况下只运行一遍。因此,恢复块系统中的时间和空间要求很容意被接受。这就使恢复块系统有可能得到广泛应用。但是,如果覆盖率过低,则它对系统的可靠性几乎没有改进。

实时软件的可靠性设计是软件可靠性工程的重要内容。实时软件的避错设计是可靠性设计的根本。除本文给出的设计方法和保证技术外,形式化的实时软件设计、故障/安全性设计及软硬件相结合的可靠性设计都是非常有效的方法。

而容错设计是对避错设计的补充和提高。容错技术包括软件容错和硬件容错,由于时间约束的关系,一般采用硬件容错,主要是硬件冗余。容错技术的执行过程可分为分析错误、定位错误和错误恢复,其关键是要实时地分析错误。定位错误是个弹性很大的过程。对于实时性不太强的系统,可定位到很小的粒度(如插件、芯片等);对于强实时系统,需要及时进行处理,因此来不及精确定位,一般定位到机器一级。系统恢复的困难在于,如何使备份机快速投入运行,使任务执行不中断。因此,对不同软件的容错设计应区别对待。

建筑工程：

对于建筑而言，那应该怎样提高可靠性呢？

1. 从系统和整体的角度出发看待结构的安全问题
2. 应当加强设计单位、施工单位和监管单位之间的相互联系。设计合理施工规范、监管严格及时。
3. 科学合理的对一些过时的建筑设计规范进行修改，就我国当前诸多设计规范或多或少存在一些不足。

……

个人观点：

软件工程有很多地方起源于建筑工程，因此两者有很多相似之处。无论是软件工程还是建筑工程，两者的可靠性和可维护性对他们本身而言都是举足轻重的。

它们都十分重视安全这个问题！安全当然第一。其次便是要实现较高的可靠性和后期的维护，两者需要的预算。这是在工程前期需要做的。

那么，如何在保证可靠性和安全性的前提下降低成本呢？

我个人认为有以下几个方法：

1. 前期便设计好大概的方向甚至精确到每一砖每一瓦，避免后期因为前期设计的不规范导致推到重来所带来的损失。
2. 要想降低成本，就得发展技术，积极推广创新技术。虽然这样前期投入比较多，但是后期的回报确实惊人的。
3. 因地制宜，善于利用工程附近环境和自身特点，增加工程带来的附属收益。