第 8 章　软件质量保证

软件质量保证是建立一套有计划、有系统的方法，来向管理层保证拟定出的标准、步骤和方法能够被所有项目正确采用。软件质量保证的目的是使软件过程对于管理人员来说是可见的。软件质量保证通过对软件产品和活动进行评审和审计来验证软件是否合乎标准。软件质量保证组在项目开始时就一起参与建立计划、标准和过程，以使软件项目满足机构方针的要求。

8.1　软件质量概述

简单地说，软件质量就是指软件的使用性能以及其他配套价值到底是怎么样的。例如，软件的使用寿命、正确性、易用性等，都是软件质量的体现。

8.1.1　软件质量的定义

关于软件质量的定义，标准有很多。ANSI/IEEE Std 729—1983 标准对软件质量的定义为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。M. J. Fisher 则把软件质量定义为“所有描述计算机软件优秀程度的特性的组合”。也就是说，为满足软件的各项精确定义的功能、性能需求，符合文档化的开发标准，需要相应地给出一些质量特性及其组合，作为在软件开发与维护中的重要考虑因素。如果这些质量特性及其组合都能在产品中得到满足，则这个软件产品就具有较高的质量。

软件质量主要反映了以下 3 方面的问题：

(1)软件需求是度量软件质量的基础，不符合需求的软件就不具备质量。

(2)在各种标准中定义了一些开发准则，用来指导软件人员用工程化的方法来开发软件。如果不遵守这些开发准则，软件质量就得不到保证。

(3)往往会有一些隐含的需求没有明确地提出来，如软件应具备良好的可维护性。如果软件只满足那些明确定义了的需求，而没有满足这些隐含的需求，软件质量也不能保证。

软件质量是各种特性的复杂组合，它随着应用和用户提出的质量要求的不同而不同。因此，有必要讨论各种质量特性，以及制定评价质量的准则。

8.1.2　软件质量的特性与度量

一般来说，可以按照以下几方面度量软件质量的特性：

1)功能性

由功能及与其指定的性质有关的一组属性组成，包括：

(1)适合性：与能否提供一组规定功能及这组功能的适合程度有关的软件属性。

(2)准确性:与能否得到正确或相符的结果或效果有关的软件属性。

(3)互用性:与同其他指定系统进行交互的能力有关的软件属性。

(4)依从性:使软件遵循有关的标准、约定、法规及类似规定的软件属性。

(5)安全性:与防止对程序及数据可能存在非授权、故意或意外访问的能力有关的软件属性,即为安全性。

2)可靠性

可靠性由与在规定的一段时间和条件下,软件维持其性能水平的能力有关的一组属性组成,包括:

(1)成熟性:与由软件故障引起失效的频度有关的软件属性。

(2)容错性:与在出现软件故障或违反指定接口的情况下,维持规定性能水平能力有关的软件属性。

(3)易恢复性:与在失效发生后,重建其性能水平并恢复直接受影响数据的能力,以及为达此目的所需的时间和能力有关的软件属性。

3)易用性

易用性由与一组规定或潜在用户为使用软件所需的努力和对这样的使用所作的评价有关的一组属性组成,包括:

(1)易理解性:与用户为认识逻辑概念及其应用范围所需的努力有关的软件属性。

(2)易学性:与用户为学习软件应用所需的努力有关的软件属性。

(3)易操作性:与用户为操作和运行控制所需的努力有关的软件属性。

4)效率

在规定的条件下,软件的性能水平与所使用资源之间存在一定的关系,与这个关系有关的一组属性是衡量效率的标准,这些属性包括:

(1)时间特性:与软件执行时的响应和处理时间以及吞吐量有关的软件属性。

(2)资源特性:与软件执行时所使用的资源数及占用时间有关的软件属性。

5)可维护性

可维护性由与为进行指定的修改而所需的努力有关的一组属性组成,包括:

(1)易分析性:与诊断缺陷或失效原因及判定待修改的部分所需的努力有关的软件属性。

(2)易改变性:与进行修改、排除错误或适应环境变化所需的努力有关的软件属性。

(3)稳定性:与由修改造成的未预料结果的风险有关的软件属性。

(4)易测试性:与确认已修改软件所需的工作量有关的软件属性。

6)可移植性

可移植性由与软件可从某一环境转移到另一环境的能力有关的一组属性组成,包括:

(1)适应性:与软件无须专门配置就可以适应不同运行环境有关的软件属性。

(2)易安装性:与在指定环境下安装软件所需的努力有关的软件属性。

(3)遵循性:使软件遵循与可移植性有关的标准或约定。

(4)易替换性:与软件在该软件环境中替代指定的其他软件的可能性和工作量有关的软件属性。

总而言之,软件产品的度量独立于软件开发过程。软件的质量是由一系列质量特性组成的,每一个质量特性又由一些子特性构成。

8.1.3 软件质量保证

为了使软件质量能够得到有效控制，需要采取一定的措施和方法。所谓软件质量保证，就是建立一套有计划、有系统的方法，来向管理层保证所拟定出的标准、步骤、实践和方法能够被所有项目正确采用。

软件质量保证的目的是使软件过程对于管理人员来说是可见的。一般来说，软件质量保证组在项目开始时就一起参与建立计划、标准和过程。这些将使软件项目满足机构方针的要求。

软件质量保证的主要手段，是对软件产品和活动进行评审和审计，以此来验证软件是否合乎标准。下面介绍在软件开发各个阶段实施的软件质量保证检验项目。

1. 需求分析阶段

需求分析阶段主要检测开发目的、目标值、开发量（程序、文档）、所需资源、各阶段的产品和作业内容、开发体制等的合理性。

2. 设计阶段

设计阶段的检验项目包括产品的量（计划量、实际量）、评审量、差错数、检出差错的内容、评审方法和覆盖性、出错原因、处理情况及对该阶段的影响、评审结束和阶段结束的判断标准等。

3. 实现阶段

实现阶段的检验项目包括设计阶段的所有检验项目，另外还包括计算机使用时间、测试环境、测试项目设定种类、测试用例的设计方法等。

4. 验收阶段

验收阶段主要检查说明书（检查与被检查程序有关的用户文档等）和程序（为了评价和保证程序质量，采用各种黑盒测试或白盒测试手段进行检查）。

5. 运行维护阶段

运行维护阶段主要是掌握用户使用产品的情况，并反馈给开发部门。

8.2 软件质量度量模型

通过将软件质量特性进行分层定义，Barry W. Boehm 在《软件风险管理》中第一次提出了软件质量度量的层次模型。McCall 等人又进一步将软件质量分解到能够度量的程度，提出 McCall 模型，该模型分为软件质量要素、衡量标准和量度标准 3 个层次，这几个层次又是由一些比较低层的（如模块化、数据通用性等）标准决定的。除了 McCall 质量度量模型以外，国际标准化组织 ISO 定义了 ISO 质量度量模型。

8.2.1 McCall 质量度量模型

McCall 等在 1979 年提出的软件质量度量模型认为，特性是软件质量的反映，软件属性可用作评价准则，通过定量化地度量软件属性来判定软件质量的优劣。McCall 质量度量模型是一个 3 层框架，其软件质量概念基于 11 个特性之上，分别对应软件产品的运行、修正、

转移 3 个阶段，如图 8-1 所示。

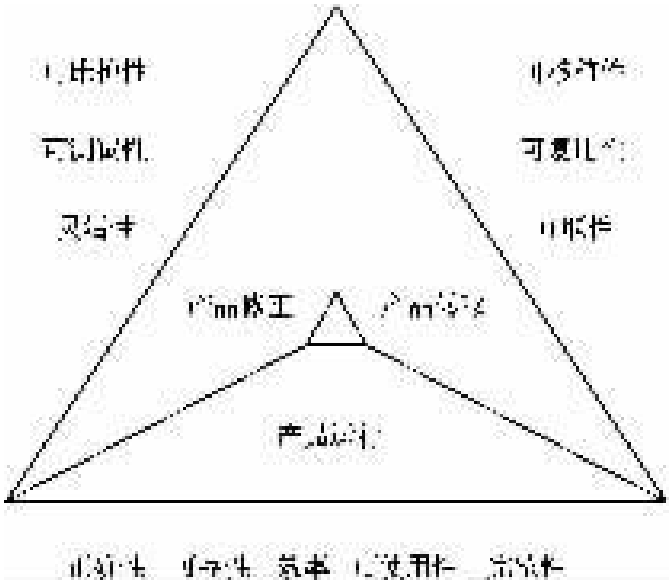

图 8-1　McCall 质量度量模型

McCall 质量度量模型中的 11 个特性的含义如下：

(1)正确性：表示在预定环境下，软件满足设计规格说明及用户预期目标的程度。它要求软件没有错误。

(2)可靠性：表示软件按照设计要求，在规定的时间和条件下不出故障，持续运行的程度。

(3)效率：表示为了完成预定功能，软件系统所需的计算机资源量。

(4)完整性：表示软件为了某一目的而保护数据，避免其受到偶然或有意的破坏、改动或遗失的能力。

(5)可使用性：表示对于一个软件系统，用户学习、使用软件及为程序准备输入和解释输出所需工作量的大小。

(6)可维护性：表示为满足新的用户要求，当环境发生了变化或运行中发现了新的错误时，对一个已投入运行的软件进行相应诊断和修改所需工作量的大小。

(7)可测试性：表示测试软件，以确保其能够执行预定功能所需工作量的大小。

(8)灵活性：表示修改或改进一个已投入运行的软件所需工作量的大小。

(9)可移植性：表示将一个软件系统从一个计算机环境移植到另一个计算机环境中运行时所需工作量的大小。

(10)可复用性：表示一个软件(或软件的部件)能再次用于其他应用(该应用的功能与此软件或软件部件的功能有联系)的程度。

(11)互联性：将一个软件和其他系统连接所需工作量的大小。如果这个软件要联网或与其他系统通信，或要把其他系统纳入到自己的控制之下，则必须有使系统间可以相互连接的接口。互联性很重要，它又称为相互操作性。

8.2.2　ISO 软件质量评价模型

国际标准化组织 ISO 制定的 ISO 质量度量模型由 3 层组成，分别是：

(1)高层，称为“软件质量需求评价准则”。

(2)中层，称为“软件质量设计评价准则”。

(3)低层,称为“软件质量度量评价准则”。

ISO 认为,应对高层和中层建立国际标准,而低层可由各使用单位视实际情况制定相关标准。同时,按照 ISO/IEC 9126 质量特性国际标准,第一层(即 ISO 质量度量模型中的高层)称为质量特性,第二层(即 ISO 质量度量模型中的中层)称为质量子特性,第三层(即 ISO 质量度量模型中的低层)称为度量。该标准定义了 6 个质量特性,即功能性、可靠性、可维护性、效率、可使用性、可移植性,并推荐了 21 个子特性,包括适合性、准确性、互用性、依从性、安全性、成熟性、容错性、可恢复性、可理解性、易学习性、操作性、时间特性、资源特性、可分析性、可变更性、稳定性、可测试性、适应性、可安装性、一致性、可替换性,但不作为标准。

8.3 软件复杂性

与软件质量密切相关的一个因素,是软件的复杂程度。很显然,软件的复杂程度越高,就越容易出现软件错误和缺陷,从而影响到软件的质量。本节主要介绍如何度量和控制软件的复杂程度,即软件复杂性。

8.3.1 软件复杂性的基本概念

顾名思义,软件复杂性是指软件的复杂程度。很显然,不同类型的软件其复杂程度也大不相同。软件复杂性度量的参数很多,一般来说,主要包括:

(1)规模:即软件总的指令或源程序行数。

(2)难度:通常由程序中出现的操作数的数量来表示。

(3)结构:通常用与程序结构有关的度量来表示。

(4)智能度:即算法的难易程度。

软件的复杂性是导致软件错误的重要原因,而软件的可靠性与其复杂性密不可分,当复杂性超过一定限度时,软件缺陷或错误便急剧上升。此外,软件的可维护性等质量特性也与之有极大关系。1979 年,Belady 和 Lehman 提出的软件维护模型表明,软件的维护开销除与维护人员的素质等相关外,软件复杂性也是维护工作量的一个重要因素。由此可见,软件复杂性度量与控制是软件可靠性工程亟待解决的一个重要问题。

8.3.2 软件复杂性的度量方法

软件复杂性度量是对软件复杂性的定量描述,是软件复杂性分析和控制的基础,它针对不同对象,从不同角度选用不同方法来描述软件的复杂性。

软件复杂性主要表现在程序的复杂性。程序的复杂性主要指模块内程序的复杂性,它直接关系到软件开发费用、开发周期和软件内部潜伏错误的多少,同时也可用于度量软件的可理解性。

要求软件复杂性度量满足以下假设:

(1)软件复杂性度量可以用来计算任何一个程序的复杂性。

(2)对于不合理的程序,如长度动态增加的程序,或者原则上无法排错的程序,则不应当使用软件复杂性度量进行复杂性计算。

(3)程序中指令条数、附加存储量、计算时间增多不会减少程序的复杂性。

下面将介绍 3 种比较经典的软件复杂性度量方法。

1. 长度度量

度量程序的复杂性,最简单的方法就是统计程序的源代码行数。这种度量方法有一个重要的隐含假定是:书写错误和语法错误在全部错误中占主导地位。然而,由于这类错误严格来讲是私有的,不应把它们计入错误总数之中,在这种情况下,这种度量方法的前提就不存在。因而,代码行数度量法是一种很粗糙的方法,在实际应用中很少使用。

2. Halstead 度量

Halstead 度量法是通过计算程序中运算符和操作数的数量来度量程序的复杂性。设 n_1 表示程序中不同运算符的个数,n_2 表示程序中不同操作数的个数,N_1 表示程序中实际运算符的总数,N_2 表示程序中实际操作数的总数。令 H 表示程序的预测长度,Halstead 给出 H 的计算公式为:$H=n_1\log_2 n_1+n_2\log_2 n_2$;令 N 表示实际的程序长度,其定义为:$N=N_1+N_2$。

Halstead 度量的重要结论之一是:程序的实际长度 N 与预测长度非常接近,这表明即使程序还未编写完也能预先估算出程序的实际长度 N。Halstead 还给出了另外一些计算公式,包括:

(1)程序容量:$V=N\ \log_2(n_1+n_2)$。

(2)程序级别:$L=(2/n_1)\times(n_2/N_2)$。

(3)编制程序所用的工作量:$E=V/L$。

(4)程序中的错误数预测值:$B=N\ \log_2(n_1+n_2)/3\ 000$。

Halstead 度量实际上只考虑了程序的数据流,而没有考虑程序的控制流,因而也不能从根本上反映程序的复杂性。

3. McCabe 度量

该度量方法由 McCabe 提出,又称为环路复杂性度量方法,其步骤是:

(1)将程序流程图中的每个处理符号看作一个节点,流程图中的流程线看作有向弧,从而将流程图转换为一个有向图。

(2)从图的入口点到出口点加一条用虚线表示的有向边,使图成为强连通图。

(3)定义环路复杂性的计算公式为:

$$V(G)=m-n+p$$

其中,$V(G)$是有向图 G 中的环路数,m 是图 G 中的弧数,n 是图 G 中的节点数,p 是图 G 中的强连通分量。

也可简单地将环路复杂性看作有向图中有向弧所封闭的区域个数。

例如,如图 8-2(a)所示的程序流程图对应的有向图如图 8-2(b)所示。因为节点数$n=8$,弧数 $m=10$,强连通分量 $p=1$,因此,环路复杂性 $V(G)=3$。

实践表明,模块复杂性与模块中存在的软件错误数或缺陷数,以及为了发现并改正它们所需的时间之间存在一种明显的关系。McCabe 指出,$V(G)$可用作最大模块复杂性的定量指标。通过大量软件工程数据研究发现,$V(G)=10$ 是模块复杂性的实际上限,当 $V(G)$超过这个值时,模块复杂性的测试就变得相当困难,甚至是不可能的。

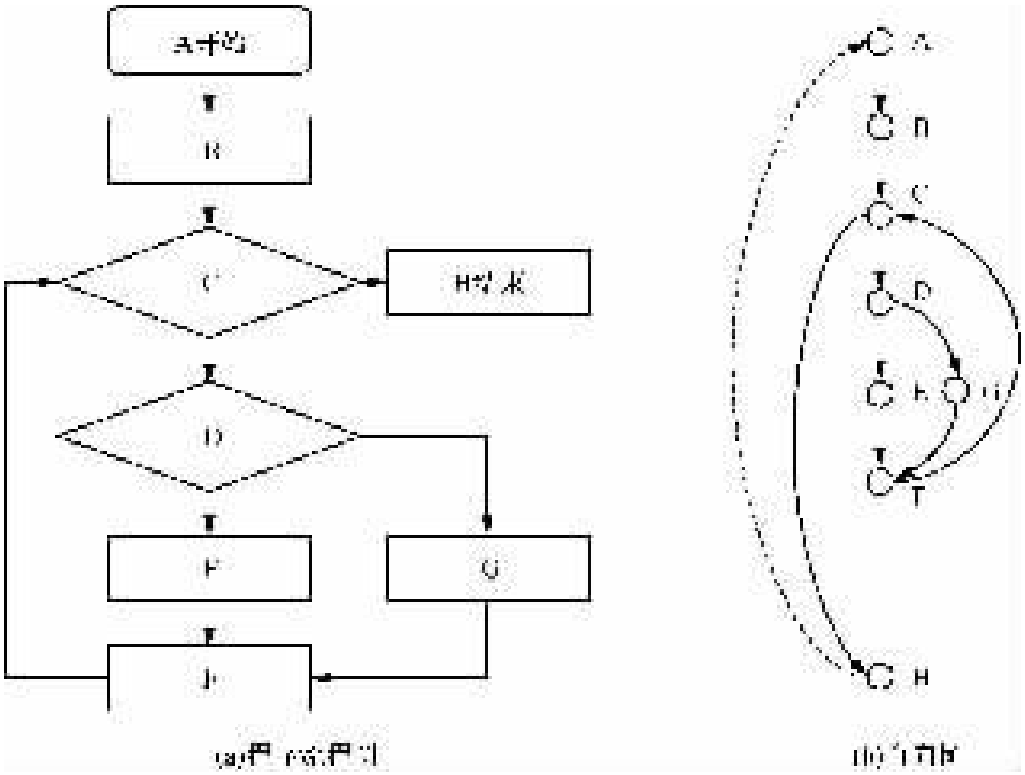

图 8-2 McCabe 复杂性度量的例子

8.3.3 软件复杂性控制

软件复杂性度量的目的是为软件复杂性的定量分析和控制提供依据，可以通过控制软件复杂性来改善和提高软件的可靠性。设计过程是软件复杂性形成的根源，因此，最有效的控制软件复杂性的方法是在软件设计过程中就对软件复杂性进行有效控制，使之保持在一个合理的范围内。

软件总体复杂性控制是一个系统工程，它是在对软件进行各种复杂性度量的基础上，综合考虑软件开发成本、可靠性要求等一系列因素之后，对软件复杂性的一种平衡。

下述内容构成了软件复杂性度量的基本度量准则集，是软件复杂性控制的基本出发点：

(1)控制结构和数据结构复杂的程序较复杂。

(2)转向语句使用不当的程序较复杂。

(3)非局部量较多的程序较复杂。

(4)按地址调用参数比按值调用参数复杂。

(5)模块及过程之间联系密切的程序较复杂。

(6)嵌套深度越大，程序越复杂。

(7)循环结构的复杂性大于选择结构和顺序结构的复杂性。

(8)宽度是软件复杂性形成的主要原因。

软件复杂性度量力图对这些准则进行定义和定量描述，找出影响和制约软件复杂性的所有关系。而复杂性控制试图在软件设计中，通过对所有影响软件复杂性，进而影响软件可靠性的因素进行控制，将它们限制在最小的范围内来改善和提高软件的可靠性。目前，尽管有不少度量与控制方法，但它们的能力还很有限。因此，对软件复杂性精确度量与有效控制的进一步研究与实践，无疑为软件可靠性工程的研究与实践提出了新的课题。

8.4　软件可靠性

软件的可靠性是用于衡量一个软件(计算机程序)好坏的很重要的评价指标之一。软件的可靠性与硬件的可靠性有许多相似之处,更有许多差异,这种差异是由软、硬件故障机理的差异造成的。软件可靠性在术语内涵、指标选择、设计分析手段以及提高软件可靠性的方法与途径等方面具有其自身的特点。

8.4.1　软件可靠性的定义

1.软件故障及其特征

常用以下 3 个术语来描述软件未正常运行的情况:

(1)缺陷:是指软件的内在缺陷。

(2)错误:缺陷在一定条件下暴露,导致系统运行中出现可感知的不正常、不正确和不按规范执行的状态。

(3)故障:由于对错误未作任何纠正,而导致系统的输出不满足预定的要求。

缺陷可能导致错误,并造成系统故障,即缺陷是一切错误的根源,因此存在下面的传递关系:缺陷—错误—故障。

有缺陷的软件只有在特定条件下才能导致出错,在一般情况下是能够正常工作的。发生过故障的软件通常仍然是可用的,只有当软件频繁发生故障,或公认已经“陈旧”时,软件才被废弃,这一版本软件的寿命也就此终结。

2.软件可靠性

软件可靠性是软件系统的固有特性之一,它表明了一个软件系统按照用户的要求和设计的目标执行其功能的正确程度。软件可靠性与软件缺陷有关,也与系统输入和系统使用有关。从理论上说,可靠的软件系统应该是正确、完整、一致和健壮的。但是,实际上任何软件都不可能达到百分之百的正确,而且软件的可靠性也无法做到精确度量。一般情况下,只能通过对软件系统进行测试来度量其可靠性。

软件可靠性的定义是:软件可靠性是软件系统在规定的时间内及规定的环境条件下,完成规定功能的能力。根据这个定义,软件可靠性包含了以下 3 个要素:

(1)规定的时间。软件可靠性只是体现在软件运行阶段,因此将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起(开启但空闲)的累计时间。由于软件运行的环境与程序路径选取的随机性,软件的失效为随机事件,所以“运行时间”属于随机变量。

(2)规定的环境条件。环境条件指软件的运行环境,它涉及软件系统运行时所需的各种支持要素,如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。在不同环境条件下运行的软件的可靠性是不同的。具体地说,“规定的环境条件”主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求,并假定其他一切因素都是理想的。有了明确规定的环境条件,还可以有效判断软件失效的责任在用户方,还是研发方。

(3)规定的功能。软件可靠性还与规定的任务和功能有关。由于要完成的任务不同,软件的运行过程和调用的子模块也会不同,其可靠性也就可能不同。要准确度量软件系统的

可靠性，就必须首先明确它的任务和功能。

8.4.2 软件可靠性的指标

软件可靠性与可用性的定量指标，是指能够以数字概念来描述可靠性的数学表达式中所使用的量。

软件可靠性的常用指标包括：

(1)平均失效间隔时间(Mean Time to Failure，MTTF)：即两次失效之间的平均操作时间。

(2)平均修复时间(Mean Time to Restoration，MTTR)：即修复一个故障平均使用的时间。

(3)有效性＝MTTF/(MTTF＋MTTR)。

(4)初期故障率：一般以软件交付使用方后的3个月内为初期故障期，初期故障率指软件在初期故障期内单位时间的故障数，一般以每100小时的故障数为单位，可以用初期故障率来评价交付使用的软件的质量，并预测软件可靠性何时基本稳定。

(5)偶然故障率：一般以软件交付给使用方的4个月后为偶然故障期。偶然故障率指软件在偶然故障期内单位时间的故障数，一般以每千小时的故障数为单位，它反映了软件处于稳定状态时的质量。

(6)使用方误用率：使用方不按照软件规范及说明等文件来使用软件而造成的错误叫“使用方误用”。在总使用次数中，使用方误用次数占的百分比叫“使用方误用率”。造成使用方误用的原因之一是使用方对说明理解不深，操作不熟练，但也可能是说明没有讲解清楚，引起误解等。

(7)用户提出补充要求数：由于软件在开发过程中未能充分满足用户需要，或者用户在软件开发时所提要求不全面，软件开始使用后用户又提出补充要求，需要生产方对软件进行修改、完善。

8.4.3 软件可靠性模型

软件可靠性模型是指为预计或估算软件的可靠性所建立的可靠性框图和数学模型。建立可靠性模型是为了将复杂系统的可靠性逐级分解为简单系统的可靠性，以便于定量预计、分配、估算和评价复杂系统的可靠性。

软件可靠性模型的分类方法包括：

(1)随机性分类法：根据随机过程的假设(如过程的确定性或非确定性、马氏过程、泊松过程等)进行分类。

(2)按软件出现的故障数进行分类：主要有错误计数模型和非计数模型，可数性或不可数性模型。

(3)按模型参数的估计方法进行分类：主要有贝叶斯方法或非贝叶斯方法，最大似然估计法或最小二乘法，另外还有线性模型等。

(4)按模型使用的时间方式分类：主要有日历时间和执行时间模型。

(5)按修复过程分类：主要指强调软件系统修复过程的一类模型，如完全修复型和不完全修复型的模型，完全排错型和不完全排错型的模型。

(6)按对软件的内部结构是否了解进行分类：可分为黑箱模型和白箱模型。主要根据对

软件内部结构的了解程度，以及对它们的结构能加以利用的程度来进行分类。

8.5　软 件 评 审

在软件工程中，评审是一种非常重要的工程活动，这个活动已经在欧美的软件工程中被实践和证实了 20 年。评审是 M. E. Fagan 于 1976 年在 IBM 创造出的一种方法，其第一次出现应该是发表在 IBM Systems Journal 上的一篇文章《Design and code inspection to reduce errors in program development》中。本节主要介绍软件评审的概念、具体内容和原则。

8.5.1　软件评审的概念

对软件工程来说，软件评审是一个“过滤器”。在软件开发的各个阶段都要采用评审的方法，以发现软件中的缺陷，然后加以改正。

可以把“质量”理解为“用户满意程度”。为使用户满意，软件必须满足以下两个条件：

(1)设计规格说明书要符合用户的要求。

(2)程序要按照设计规格说明书所规定的情况正确执行。

8.5.2　软件评审的阶段和内容

软件评审一般可按照设计质量和程序质量两方面来进行。

1. 设计质量的评审

设计质量的评审包括以下内容：

(1)评价软件的规格说明是否合乎客户的要求，即总体设计思想和设计方针是否明确，需求规格说明是否得到了客户或单位上级机关的批准，需求规格说明与软件的概要设计规格说明是否一致等。

(2)评审可靠性，即是否能避免输入异常(错误或超载等)、硬件失效及软件失效。一旦发生异常，应能及时采取代替或恢复手段。

(3)评审保密措施实施情况，即是否具有检查系统使用资格的功能；在检查出有违反对特定数据、特殊功能的使用资格的情况时，能否向系统管理人员报告有关信息；是否提供对系统内重要数据加密的功能等。

(4)评审操作特性实施情况，即操作命令和操作信息的恰当性，输入数据与输入控制语句的恰当性，输出数据的恰当性，应答时间的恰当性等。

(5)评审性能实现情况，即是否达到所规定性能的目标值。

(6)评审软件是否具有可修改性、可扩充性、可互换性和可移植性。

(7)评审软件是否具有可测试性。

(8)评审软件是否具有复用性。

2. 程序质量的评审

程序质量的评审通常是从开发者的角度进行评审，它与开发技术有关。程序质量评审是着眼于对软件本身的结构、与运行环境的接口、变更带来的影响等进行的评审活动。

1)软件的结构

软件的结构包括功能结构、功能的通用性、模块的层次、模块结构、处理过程的结构等。其中,在功能结构方面需要检查的项目有:

(1)数据结构:包括数据名和定义,构成该数据的数据项,数据间的关系。

(2)功能结构:包括功能名和定义,构成该功能的子功能,功能与子功能之间的关系。

(3)数据结构和功能结构之间的对应关系:包括数据元素与功能元素之间的对应关系,数据结构与功能结构的一致性。

在模块结构方面需要检查的项目有:

(1)控制流结构:规定了处理模块之间的流程关系,检查处理模块之间的控制转移关系与控制转移形式(调用方式)。

(2)数据流结构:规定了数据模块是如何被处理模块加工的流程关系;检查处理模块与数据模块之间的对应关系;处理模块与数据模块之间的存取关系,如建立、删除、查询、修改等。

(3)模块结构与功能结构之间的对应关系:包括功能结构与控制流结构的对应关系,功能结构与数据流结构的对应关系,每个模块的定义(包括功能、输入与输出数据)。

2)与运行环境的接口

与运行环境的接口包括与硬件的接口及与用户的接口。

随着软件运行环境的变更,软件的规格也随着不断变更。运行环境变更时的影响范围需要从以下 3 个方面来分析:

(1)与运行环境的接口。

(2)在每项设计工程规格内的影响。

(3)设计工程相互间的影响。

8.5.3 软件评审的原则和作用

从某种意义上说,软件中的多数错误是人为的。软件评审用于在软件生产过程中过滤软件错误。软件评审涉及评审的组织机构、管理、准则、类别、内容、文件和要求等。

一般要求在软件开发阶段的各个里程碑处进行软件评审。评审的类别主要有软件定义评审、软件需求评审、概要设计评审、详细设计评审、软件实现评审和软件验收评审等。

1.评审的原则

评审的原则包括:

(1)某阶段未通过阶段评审不得进入下一个软件研制阶段。

(2)评审时对事不对人,评审的是产品,而不是评审生产者。

(3)评审就要挑刺,找问题、缺陷和隐患。

(4)评审组的人员面越广越好,如包括用户、设计人员、测试人员等。

(5)评审组不做无休止的争论和辩驳,而是将争论点记录下来,供以后甄别。

(6)评审只是提出问题,没有解决问题的任务。

(7)使用“评审检查单”,以提高评审的效果。

2.评审的作用

评审的作用在于:

(1)技术把关,避免软件人员的想当然。

(2)概念沟通,吸收用户和总体设计人员参加,审查软件人员理解的正确性。

(3)集思广益,吸收有关的分系统人员参加,从不同侧面确认软件的协调性。

(4)总结汇报,使实时控制系统总指挥、总设计师了解软件生产的进度、问题和要求,以作出新的部署。

评审很容易走过场、走形式。评审效果的好坏与当事人(软件人员)密切相关。相关的开发人员不应对软件评审抱有消极或对立情绪,不要对自己编写的程序过于自信,对评审出的问题应进行整理、分类和汇总。

8.6　容 错 软 件

提高软件质量和可靠性的技术大致分为两类:一类是避开错误技术,即在开发的过程中不让差错潜入软件的技术;另一类是容错技术,即对某些无法避开的差错,使其影响减至最小的技术。

8.6.1　容错软件的定义

容错软件的定义为:规定功能的软件,在一定程度上对自身错误的作用(软件错误)具有屏蔽能力;在一定程度上能从错误状态自动恢复到正常状态;在发生错误时,仍然能在一定程度上完成预期的功能;在一定程度上具有容错能力。满足以上需求之一的软件则称之为容错软件。

8.6.2　容错的一般方法

1.结构冗余

(1)静态冗余。常用的有三模冗余(Triple Modular Redundancy, TMR)和多模冗余(Multiple Modular Redundancy,MMR)。

(2)动态冗余。动态冗余的主要方式是多重模块待机储备,即当系统检测到某工作模块出现错误时,就用一个备用的模块来顶替它,并重新运行。

(3)混合冗余。它兼有静态冗余和动态冗余的长处。

2.信息冗余

为检测或纠正信息在运算或传输中的错误需外加一部分信息,这种现象称为信息冗余。

3.时间冗余

时间冗余是指以重复执行指令(指令复执)或程序(程序复算)来消除瞬时错误带来的影响。

4.冗余附加技术

冗余附加技术是指实现上述冗余技术所需的资源和技术。

8.6.3　容错软件的设计过程

容错系统的设计过程如下:

(1)按设计任务要求进行常规设计,尽量保证设计的正确。按常规设计得到非容错结

构，它是容错系统构成的基础。在结构冗余中，不论是主模块还是备用模块的设计和实现，都要在费用许可的条件下，用调试的方法尽可能提高可靠性。

(2)对可能出现的错误分类，确定容错的范围。对可能发生的错误进行正确判断和分类。例如，对于硬件的瞬时错误，可以采用指令复执和程序复算；对于永久错误，则需要采用备份替换或者系统重构。对于软件来说，只有最大限度地弄清错误暴露的规律，才能对错误进行正确判断和分类，实现成功的容错。

(3)按照“成本—效率”最优原则，选用某种冗余手段(结构、信息、时间)来实现对各类错误的屏蔽。

(4)分析或验证上述冗余结构的容错效果，如果效果没有达到预期的程度，则应重新进行冗余结构设计。如此反复，直到有一个满意的结果为止。

本章小结

为了使软件质量得到保证，需要制定一套行之有效的质量保证方法。通常可以从功能性、可靠性、易用性、效率、可维护性、可移植性 6 方面来评判软件质量的标准。McCall 质量度量模型和 ISO 质量度量模型是两个常用的软件质量度量模型。与软件质量密切相关的一个因素是软件的复杂程度，通常称之为软件复杂性。常见的软件复杂性度量方法包括长度度量、Halstead 度量、McCabe 度量。软件的可靠性是用于衡量一个软件(计算机程序)好坏的一个很重要的评价指标。软件可靠性的指标包括平均失效间隔时间、平均修复时间、有效性等。软件评审用于在软件生产过程中过滤软件错误。通常从设计质量和程序质量两方面进行评审。提高软件质量和可靠性的技术大致分为两类：一类是避开错误技术，即在开发的过程中不让差错潜入软件的技术；另一类是容错技术，即对某些无法避开的差错，使其影响减至最小的技术。容错的一般方法包括结构冗余、信息冗余、时间冗余和冗余附加技术等。

习 题 8

简答题

1. 可以从哪几个方面来度量软件质量？
2. 常见的软件质量度量模型有哪些？它们之间有何异同？
3. 软件复杂性与哪些因素有关？如何有效控制软件的复杂性？
4. 软件可靠性的定义是什么？衡量软件可靠性的指标有哪些？
5. 软件容错有什么作用？如何实现软件容错？