第一章 引 言

1.1 软件可靠性问题的重要性

一个问题如果无关宏旨，便不值得众人关注。软件可靠性问题如果不重要，便不值得投入大量的人力、物力去研究。然而事实恰恰是，软件可靠性问题在软件工程实践中极为重要，乃至对生产活动和社会活动产生显著影响。我们可以从多个侧面反映软件可靠性问题的重要性。

(1) 软件失效可能造成灾难性后果。一个熟知的例子是由于控制系统 Fortran 程序少一个逗点，致使控制系统未能发出正确的指令，从而使美国的一次宇宙飞行失败。目前银行业务均利用计算机系统进行财务结算，一旦其中软件失效，将造成重大经济损失。对于采用主动控制技术(ACT)的现代化飞机，飞行安全极大地依赖于软件可靠性和余度管理策略，一旦软件失效，可能是机毁人亡。

(2) 软件失效经常发生，或者说，在整个计算机系统失效数额中占有很大比重。Bell 实验室曾对一个 AT&T 运行支持系统作统计，发现 80%的失效与软件有关[1]。究其原因，是软件太复杂了。一个小小的程序，其可能的路径可以是天文数字，以致于在软件开发过程中难于对其作穷尽的测试，或者说难于完全排除软件缺陷。为了说明软件复杂性，让我们考虑一个由 10 至 20 条高级语言构成的程序[2]，其控制流程图如图 1.1.1 所示。其中每个结点或圆圈代表一段可能以转移语句结束的顺序执行语句，每条边或弧则表示两段程序间的控制转移。程序含有一个最少重复 20 次的循环语句，而在循环体内，则有一些嵌套的条件语句。假设程序中所有判断都是相互独立的(尽管实际情况可能不是这样)，由于有 5 条贯穿循环体的路径，即 $c\to d\to e\to f\to h\to m$；$c\to d\to e\to f\to i\to m$；$c\to d\to e\to g\to j\to m$；$c\to d\to e\to g\to k\to m$；$c\to d\to l\to m$；那么从点 A 到点 B 的所有独立路径数为 $5^{20}+5^{19}+\cdots+5^{1}$，即大约为 10^{14}或 10^{16}亿。如果考虑程序输入数据的变化，那情况就更为复杂了。

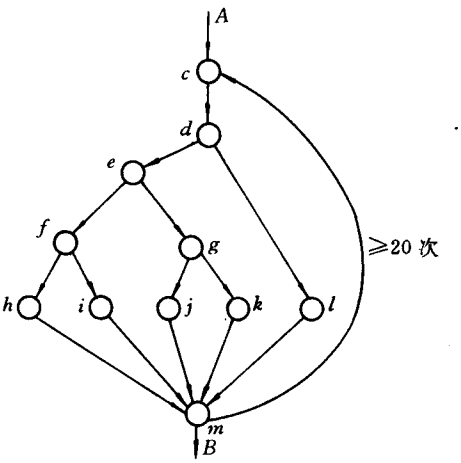

图 1.1.1 一个简单程序的控制流程图

(3) 相对于硬件可靠性技术，软件可靠性技术很不成熟，这就加剧了软件可靠性问题的重要性。譬如在硬件可靠性领域，故障树分析(FTA)、失效模式与效应分析(FMEA)技术比较成熟，容错技术也有广泛应用，有三模块冗余(TRM)系统、用不太可靠元件构造高

可靠系统等等，而在软件可靠性领域，相应的技术似乎尚未定型。

(4) 在硬件元器件价格急剧下降的今天，软件费用却有增无减。M. L. Shooman 曾粗略描述了软件开发费用在整个系统开发费用中所占比例逐年上升的趋势[3]，如图 1.1.2 示，软件可靠性问题是造成这一趋势的主要原因之一。

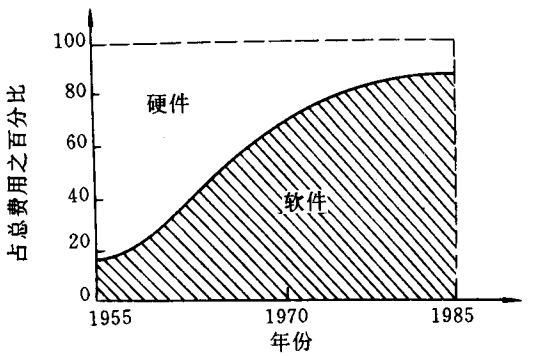

图 1.1.2 软件费用与硬件费用变化趋势

(5) 计算机技术获得日益广泛应用，对生产活动和社会生活产生显著影响，从而要求在生产活动和社会生活中考虑和重视软件可靠性问题。

(6) 软件在计算机系统中起着日益重要的作用，正是软件使得计算机系统的广泛应用成为可能，并使计算机系统蕴藏着巨大潜力和魅力，这就增加了软件可靠性问题在软件工程领域乃至整个计算机工程领域的重要性。

1.2 软件工程与软件可靠性工程

软件工程与软件可靠性工程既有密切关系，又有重要区别。认识它们之间的异同点，对促进软件可靠性工程的发展十分必要。因此，明确软件工程与软件可靠性工程各自含义，并回顾各自的发展历程是有益的。

1.2.1 软件危机

在 50 年代末 60 年代初，计算机硬件从晶体管到集成电路，得到飞速进步，并朝着超大规模集成(VLSI)方向发展，计算机的进步主要源于硬件的进步。那时软件的重要性还不显著，尽管软件开发环境得以改善，如高级语言的产生，软件开发仍处于“手工作坊”阶段，极大地依赖于开发人员的编程技巧，且主要关注的是软件功能。但随着计算机应用的发展，软件需求量剧增。与此同时，硬件技术不断成熟。于是软件开发技术成为妨碍计算机进步的瓶颈，在 60 年代末突出表现为三方面的问题：软件质量不高、开发进度延缓以及开发费用昂贵，形成所谓的“软件危机”。更具体地说，存在以下几方面的问题[4]。

(1) 软件技术及进度估计往往相当不准确，大大挫伤了开发者的信心，导致企图走捷径而使质量下降，引起用户不满。

(2) 即使在计划、规范书、设计、编码和测试中使用了经过证明的方法，但仍没有系统地开发软件。

(3) 不适当的软件文档。计算机程序应该有一整套的文档，这套文档不是在软件开发之后，而是在软件开发过程中编制出来的。它们作为软件开发的里程碑，能帮助管理人员控制并核查进度。

(4) 软件质量值得怀疑。由于不可能一致地使用经过证实的软件质量保证技术(复查、审查及测试)，导致了质量保证方面的各种问题。

(5) 软件经常是"不可维护"的。要在许多程序中纠正潜在的缺陷是很困难的。要使这些程序适用于新的计算机或者增强用户所要求的功能，实际上也是不可能的。可重复使用的、通用性较强的软件，仍然是一个急于寻求而又尚未达到的目标。

面对严重的软件危机，出路何在呢？

1.2.2 软件工程

软件危机的根本原因是软件开发和生产过程采用"手工作坊"模式，将软件开发和生产过程与程序编制混为一谈，甚至将软件与程序混为一谈。因此软件危机的出路在于软件开发过程和生产过程的工程化，这就是软件工程。"软件工程"这一术语是在1968年和1969年北大西洋公约组织(NATO)主办的两次研讨会上讨论软件危机时正式提出的，意指应用于计算机软件的定义、开发和维护的一整套方法、工具、文档、实践标准和工序。软件工程旨在使软件开发生产过程规范化、工程化，以提高软件质量、加速开发进度、降低开发费用，实现软件生产从手工作坊到工程化、社会化的转变。显而易见，提高软件质量、加速开发进度、降低开发费用是软件工程的三大目标。

那么如何实现软件工程的三大目标呢？

1.2.3 软件生存期

软件生存期概念是软件工程最重要的概念之一，是实现软件工程的根本保证。软件生存期可以根据不同的模型加以界定，如原型模型、瀑布模型，但最被广泛接受的、实践最多的是瀑布模型。它将软件从计划制定到报废这段过程划分为若干阶段，包括需求分析阶段、概要设计阶段、详细设计阶段、实现阶段、测试阶段、安装验收阶段、运行维护阶段、报废阶段。其中前六个阶段合称为软件开发阶段。

1. 需求分析阶段

这阶段包括两部分内容，一是软件计划制定，二是需求分析。开发单位必须制定项目开发计划，其主要内容是：

① 用简练的语言写出开发项目的各项主要工作。

② 说明为完成开发项目应具备的条件，如硬件资源和软件资源等。

③ 说明完成开发项目的约束条件，如时间、人力、经费和运行环境等限制。

④ 具体说明成本估算。

⑤ 制定每项工作的进度要求。

⑥ 说明开发组织内部的分工及其职责。

需求分析的任务是确定软件开发的主要任务，并根据任务要求，确定软件的主要功能。必须分析系统中影响软件或软件需求的各种有关部分，并确定该软件与系统中其它各部分的连接方式，包括与外部设备的连接和相应接口，以及与其它程序或系统的连接，必须用文字、图表和数学公式等方法详细地描述软件每个功能的特性。

需求分析阶段的输出是软件需求规范说明书，用于详细描述软件各个功能及约束条件。

2. 概要设计阶段

这阶段是根据软件需求规范将软件设计过程进行到模块级。必须确定分程序和模块的功能。在功能划分中必须全部满足软件需求。必须确定程序在各种运行方式下的控制流和数据流，并确定每个分程序及其模块的所有输入、输出和处理功能。必须确定软件与其它系统的接口，了解其它系统的设计要求、接口的目的和要交换的数据，明确数据量、频率、速率、格式、内容、转换要求和约定等。必须确定各模块之间的详细接口信息。

这阶段的输出是概要设计说明书，详细说明本阶段的设计结果，供详细设计阶段之用。

3. 详细设计阶段

这阶段对模块进行过程描述，设计模块的内部细节(包括算法和数据结构)，为编写源代码提供必要的说明，写出详细设计说明书。

4. 实现阶段

这阶段包括两个部分，一是程序编码，二是单元测试。程序编码必须根据软件详细设计说明书和编程标准进行。编程标准意指编程语言约定、编程格式约定、控制结构约定等一系列内容。

必须对所开发的程序单元进行测试。程序单元可以是模块中可独立执行的一部分，也可以是整个模块。测试前，开发者应该完成代码逐步审查。测试时，应确保每一条可执行的源代码语句至少执行一次，并满足功能要求。

这阶段的输出是一系列可供软件综合测试之用的程序单元。每个程序单元应满足以下条件：

① 正确地通过编译或汇编。

② 完成代码逐步审查。

③ 完成单元测试。

④ 经验证完全满足设计要求(包括所有必要的输入和输入要求)。

⑤ 满足软件质量保证要求。

⑥ 置于承办单位软件配置管理之下，并放入程序库中。

5. 测试阶段

这阶段的工作包括制定测试计划、进行软件综合测试和系统综合测试。测试计划必须至少包括测试目的、内容、进度、条件、步骤、结果分析和评价准则等。软件综合测试则是检测软件设计是否与软件结构一致，是否满足软件需求规范。通过测试的软件至少应该满足下列条件：

① 各模块正确地连接。

② 满足各项功能和性能要求。

③ 对非正常输入有容错能力。

④ 人机界面全面正确。

⑤ 满足全部操作要求，包括启动、从外部设备输入数据、程序装入、重新启动、从显示控制台或其它控制台上监督和控制系统的操作等。

⑥ 与软件需求规范中规定的所有设备均正确地连接。

⑦ 除了在存储器中需要分配绝对地址的程序段以外，必须有定位能力。

当开发的软件是系统的一个组成部分时，则应该参加有关的系统综合测试。

这阶段的输出有两个，一是供安装验收之用的软件，二是软件测试报告，用于说明所验证的软件的功能及其存在的缺陷。

这阶段的一个显著特征是软件缺陷一旦被发现、诊断，则必然要求被剔除，从而使软件残留缺陷数不断减少，软件可靠性呈增长趋势。

6. 安装验收阶段

这阶段根据软件验收规范(包含于软件需求分析规范或软件测试报告之中)对软件进行一系列测试，以确定软件是否满足验收规范，从而制定是否接受或拒绝该软件。这阶段的一个显著特征是验收测试过程中暂不剔除已被发现的软件缺陷，因此软件可靠性保持不变。

7. 运行维护阶段

软件一旦通过验收测试，即可投入实际环境运行，在运行过程中可对软件进行维护。软件维护大体上可分为四类：

① 正确性维护：分析和纠正软件在用户使用后所暴露的程序缺陷。

② 完善性维护：用户要求对程序特性进行修改或增强。

③ 适应性维护：由外部环境的改变(如新的操作系统、不同的硬件)引起的软件修改。

④ 预防性维护：预先准备上述某项或各项的维护活动。

8. 报废阶段

软件生存期的最后一个阶段是软件报废，从系统中剔除该软件，乃至永久不再使用。

1.2.4 软件工程发展历程

“软件工程”这一术语在60年代末被正式提出，但这并不意味着在此之前不存在软件工程，只是那时“软件工程”的概念不明确、不系统，人们对其重要性认识不足，软件工程实践尚处于雏形阶段。事实上，有了电子计算机，便有了软件工程的实践活动，初期，只是主要为程序编码，因此，谈及软件工程发展历程，应该从电子计算机诞生之日算起。

按照 Musa 和 Everett 的说法[5]，软件工程的发展经历了以下几个阶段：

① 功能阶段：这一阶段软件开发考虑的主要因素是软件功能。

② 进度阶段：这一阶段软件开发进度成为软件开发考虑的第二个主要因素。

③ 费用阶段：这一阶段软件费用成为软件开发考虑的第三个主要因素。

④ 目前处于一个崭新阶段，软件可靠性正逐渐成为软件开发考虑的第四个主要因素。

1.2.5 软件可靠性工程发展历程

有了软件，便有了软件可靠性工程实践。因为任何软件设计和程序编码均需要排除软件错误和缺陷，只是那时的工程实践活动不系统，概念不明确，对其重要性认识不足。软件可靠性问题获得重视是60年代末的事。那时软件危机被广泛讨论，软件不可靠是造成软件危机的重要原因之一。Jelinski-Moranda模型[6]于1972年正式提出，它标志着软件可靠性系统研究的开始。在70年代，软件可靠性的理论研究获得很大发展，一方面提出了数十种软件可靠性模型，另一方面是软件容错的研究，提出了分别与硬件静态冗余和动态冗余相对应的软件N文本方法和恢复块方法。在80年代，软件可靠性理论研究停滞不前，没有质的飞跃。但软件可靠性的工程实践经验得到不断积累，不少软件可靠性技术在软件工程实践中得以应用。某些技术达到实用化程序，如软件可靠性建模技术、管理技术等。可以说，软件可靠性从研究阶段逐渐迈向工程化。进入90年代后，由于软件可靠性问题的重要性进一步提高和软件可靠性工程实践范畴的不断拓展，软件可靠性逐渐成为软件开发考虑的第四个主要因素，软件可靠性工程在软件工程领域逐渐取得相对独立的地位，成为一个生机勃勃的分支。

1.2.6 软件工程与软件可靠性工程

为了揭示软件工程与软件可靠性工程之间的关系，有必要说明软件质量的含义。软件质量是指软件产品中能满足给定需求的各种特性的总和[7]。这些特性叫做质量特性，它包括以下几个方面：

(1) 功能度　软件的功能度是指程序的执行满足其在软件需求规范中规定的各项功能需求的能力。

(2) 可靠性　软件可靠性的一种定义是指在规定的运行环境中，在规定的运行时间内或规定的运行次数下，程序运行各种不同测试用例的成功概率。

(3) 易使用性　软件的易使用性是指人们学习、操作、准备输入和解释程序输出(输出结果和故障信息)的难易程度。

(4) 时间经济性　软件的时间经济性是指在规定或隐含的条件下，其程序完成规定功能所需要的时间。

(5) 资源经济性　软件的资源经济性是指在规定或隐含的条件下，其程序完成规定功能所需要的内存空间、外存空间和其它外部设备的数量和时间。

(6) 可维护性　软件的可维护性是指对已交付的软件进行正确性修改、适应性修改或完善性修改的难易程度。

(7) 可移植性　软件的可移植性是指在规定条件下将一个程序从一个环境移到另一个环境进行运行的难易程度。

(8) 安全性　软件的安全性是指软件的各项配置能控制非授权人员对其进行存取和识别的能力。

(9) 可再用性　软件的可再用性是指其程序的一部分可用于构成其它软件的能力。

(10) 可装接性　软件的可装接性是指其程序或程序的一部分与其它程序或其它系

统进行连接的难易程度。

我们知道，提高软件质量、加速软件开发进度、降低软件费用是软件工程的三大目标，而软件可靠性是软件质量的基本特性之一，一个不可靠的软件必是一个质量不高的软件。因此，提高软件可靠性是软件工程的子目标之一。显而易见，软件可靠性工程的目标在于保证和提高软件可靠性，或者说，软件可靠性工程旨在开发利用以保证和提高软件可靠性为主要目标的软件技术。显然，这些软件技术须以工程化方式加以开发应用，因此，软件可靠性工程是一个工程领域。可见，软件可靠性工程是软件工程的一个分支。没有软件工程，便没有软件高可靠性。

但是，软件工程与软件可靠性工程有着明显不同。软件可靠性工程只有一个主要目标，而软件工程兼顾其三大目标，因此有着不同的覆盖范畴[8]。譬如软件容错的实现将延缓软件开发进度，并增加软件开发费用，但它旨在提高软件可靠性，因此软件容错隶属于软件可靠性工程范畴。另一方面，软件结构化设计方法同时有利于软件质量、软件开发进度和软件费用，但不能说软件可靠性是其考虑的最主要因素。因此，结构化设计方法隶属于软件工程范畴，而不能成为软件可靠性研究的专门对象。

应当说明，本书讨论软件可靠性工程的专门问题，不讨论软件工程的一般问题和技术。但这绝不意味着软件工程不重要，事实上软件工程是保障软件可靠性的基础。我们只是强调，为了保证软件的高可靠性，除了遵循软件工程的一般规范之外，还需要某些专门的软件可靠性技术和方法。这些专门的技术和方法是构成本书的基本内容。

1.3 软件可靠性工程的基本问题

可靠性研究面向故障或失效，没有故障或失效，便没有可靠性研究及其工程实践。任何现实对象都难免出现故障或失效，可靠性的一种含义是指不出现故障或失效的机会（概率或可能性）。

软件可靠性工程的主要目标是保证和提高软件可靠性。为达到这一目标，显然首先要弄清软件为什么会出现故障或失效（这类似于研究硬件可靠性物理）。只有这样，才有可能在软件开发过程中减少导致软件故障或失效的隐患，且一旦出现软件故障或失效，有可能采取有效措施加以清除。但是软件是开发出来的，满足可靠性要求的软件也是开发出来的，因此，软件可靠性工程的核心问题是如何开发可靠的软件。而有了软件，又该如何检验其是否满足可靠性要求？这是软件可靠性工程的又一个问题。

综上所述，软件可靠性工程有以下三个基本问题[8]：

① 软件为什么失效（软件可靠性物理）。

② 如何开发可靠的软件。

③ 如何检验软件可靠性。

1.4 软件可靠性的含义

软件是程序、可执行数据和文档的统称。

软件可靠性这一术语有两种不同的理解：广义的和狭义的[8]。广义的理解是指一切旨在避免、减少、处理、度量软件故障(错误、缺陷、失效)的分析、设计、测试等方法、技术和实践活动。于是有诸多相关术语，如软件可靠性度量、软件可靠性设计、软件可靠性建模、软件可靠性测试、软件可靠性管理等等。

软件可靠性的狭义理解则指软件无失效运行的定量度量，尤其是那些面向用户的定量度量(这时软件意指程序)，主要有下述几个方面。

1. 软件可靠度

软件可靠度表示在规定的运行环境中和规定的时间内软件无失效运行的机会。这里需注意以下几点：

(1) 软件对象必须明确，即须指明它与其它软件的界限。

(2) 软件失效须明确定义。

(3) 必须假设硬件无故障(失效)和软件有关变量的输入值正确。

(4) 运行环境包括硬件环境、软件支持环境和确定的软件输入域。

(5) 规定的时间必须指明时间基准，可以是日历时间、时钟时间、CPU 运行时间或其它时间基准。

(6)软件无失效运行的机会通常以概率度量，但也可以模糊数学中的可能性加以度量。

(7) 上述定义是在时间域上进行的，这时软件可靠度是一种动态度量。也可以在数据域上将软件可靠度定义为一种静态度量，表示软件成功执行一个回合的概率。软件回合(run)是指软件在规定环境下的一个基本执行过程，如给定一组输入数据到软件给定相应的输出数据这一过程。软件回合是软件运行的最小的、不可分的执行单位，软件的运行过程由一系列软件回合组成(参见 6.9 节)。

(8) 有时将软件运行环境简单理解为软件运行剖面(Operational profile)，软件运行剖面是指软件的数据环境，由软件数据输入域及各种输入数据组合状态出现的机会确定。

2. 软件失效强度

软件失效强度的物理解释是单位时间内软件发生失效的机会。在概率范畴内，它与软件可靠度(动态度量)有明确的数学关系。设 $R(t)$表示$[0,t]$时间内软件不发生失效的概率，那么 t 时刻软件失效强度 $\lambda(t)$为：

$$\lambda(t) = -\frac{\frac{\mathrm{d}R(t)}{\mathrm{d}t}}{R(t)}$$

或

$$R(t) = \mathrm{e}^{-\int_0^t \lambda(t)\mathrm{d}t}$$

3. 软件平均失效时间(MTTF)

软件平均失效时间是指软件投入运行到出现一个新失效的平均(期望)时间，在概率范畴有：

$$\mathrm{MTTF} = \int_0^{\infty} R(t)\mathrm{d}t$$

上述度量与硬件可靠性中的相应概念本质上是一致的。软件可靠性还有许多度量，有面向用户的度量，也有面向开发人员的度量；有技术度量，也有管理度量；有定量度量，也有定性度量，参见 3.2 节中的阐述。

参 考 文 献

[1] Everett W. W., Software Reliability Measurement, IEEE J. Selected Areas in Communication, Vol. 8, No. 2, 1990, pp247—250

[2] Meyers G. J., The Art of Software Testing, John Wiley & Sons, 1979

[3] Shooman M. L., Software Reliability: A Historical Perspective, IEEE Trans. Reliability, Vol. R-33, No. 1, 1984, pp48—55

[4] 朱三元编译，软件工程指南，上海翻译出版公司，1985

[5] Musa J. D., W. W. Everett, Software Reliability Engineering: Technology for the 1990s, IEEE Software, Vol. 7, No. 6, 1990, pp36—43

[6] Jelinski Z., P. B. Moranda, Software Reliability Research, in: W. Freiberger (ed), Statistical Computer Performance Evaluation, Academic Press, 1972, pp465—484

[7] 航空工业部，软件质量度量准则规范，1988

[8] 蔡开元，软件可靠性概要，系统工程与电子技术，Vol. 15, No. 4, 1993, pp47—54

第二章　软件为什么失效（软件可靠性物理）

2.1　硬件与软件

硬件与软件有许多不同点，这已有许多论述[1-3]。但从可靠性角度来看，它们主要有三个不同点[4]：

(1) 复杂性　软件内部逻辑高度复杂，图 1.1.1 给出的例子从一个侧面说明了这个问题，而硬件内部逻辑较为简单，这就在很大程度上决定了设计错误是导致软件失效的主要原因，而导致硬件失效的可能性则很小。

(2) 物理退化　软件不存在物理退化现象，硬件失效则主要由于物理退化所致。这就决定了软件正确性与软件可靠性密切相关，一个正确的软件任何时刻均可靠。然而一个正确的硬件元器件或系统则可能在某个时刻失效。

(3) 唯一性　软件是唯一的，软件拷贝不改变软件本身，而任何两个硬件不可能绝对相同。这就是为什么概率方法在硬件可靠性领域取得巨大成功，而在软件可靠性领域不令人满意的原因[5]。

2.2　软件失效机理

弄清软件失效机理是软件可靠性物理的根本目标。由于软件内部逻辑复杂，运行环境动态变化，且不同的软件差异可能很大，因而软件失效机理可能有不同的表现形式。譬如有的失效过程比较简单，易于追踪分析，而有的失效过程可能非常复杂，难于甚至不可能加以详尽描述和分析，尤其是运行于高度复杂实时环境中的大型软件。但总的说来，软件失效机理可描述为：软件错误——→软件缺陷——→软件故障——→软件失效。

(1) 软件错误(Software error)　在可以预见的时期内，软件仍将由人来开发。在整个软件生存期的各个阶段，都贯穿着人的直接或间接的干预。然而，人难免犯错误，这必然给软件留下不良的痕迹。软件错误是指在软件生存期内的不希望或不可接受的人为错误，其结果是导致软件缺陷的产生。可见软件错误是一种人为过程，相对于软件本身，是一种外部行为。

下面举例说明。已知成都中心气象台1978年元月上旬的气温统计如表2.2.1所

表 2.2.1　气温统计数据　　单位：℃

时间＼日期	1日	2日	3日	4日	5日	6日	7日	8日	9日	10日
2时	5.4	5.1	5.5	5.0	0.6	3.4	1.1	5.1	6.8	4.9
8时	4.8	4.5	4.3	3.3	−1.2	3.8	0.0	4.3	3.1	3.8
14时	10.1	9.7	6.7	9.4	9.8	6.4	9.2	10.0	10.1	9.7
20时	6.5	5.9	6.2	3.6	4.7	3.3	4.8	7.1	5.8	7.4

示[6]。设这些数据是按时间顺序存入数据文件 CD 的要求每日的平均温度。

那么一个正确的程序是(FORTRAN 77)：

```
C     EXAMPLE
      DIMENSION T(4,10), MT(10)
      REAL ML
      OPEN(4,FILE='CD')
      READ(4,*) ((T(I,J),I=1,4),J=1,10)
      DO 20 J=1,10
      S=0.0
      DO 10 I=1,4
10    S=S+T(I,J)
20    MT(J)=S/4.0
      WRITE (4,*) MT
      CLOSE(4)
      STOP
      END
```

如果由于人为错误，将求平均温度当作求最高温度，这就是一个软件错误，其结果是得到如下不可接受的程序：

```
C     EXAMPLE
      DIMENSION T(4,10),MT(10)
      REAL MT
      OPEN(4,FILE='CD')
      READ(4,*)((T(I,J),I=1,4),J=1,10)
      DO 20 J=1,10
      S=T(1,J)
      DO 10 I=2,4
      IF(S,LT,T(I,J))THEN
      S=T(I,J)
      ENDIF
10    CONTINUE
20    MT(J)=S
      WRITE(4,*)MT
      CLOSE(4)
      STOP
      END
```

(2) 软件缺陷(software defect)　软件缺陷是存在于软件(文档、数据、程序)之中的那些不希望或不可接受的偏差，如少一逗点、多一语句等等。其结果是软件运行于某一特定条件时出现软件故障，这时称软件缺陷被激活。当软件意指程序时，软件缺陷(defect)与软件(程序)污点(bug)同义。

现在，我们仍考虑表 2.2.1 的例子。由于将求平均温度当作求最高温度，导致产生一个软件缺陷。该缺陷由以下语句构成：

```
      S=T(1,J)
      DO 10 I=2,4
      IF(S.LT.T(I,J)) THEN
      S=T(I,J)
      ENDIF
10    CONTINUE
20    MT(J)=S
```

在数据文件 CD 中，如果第一个数据与实际统计数据不符，譬如是 0.4 而不是应有的 5.4，那么我们说软件又存在一个缺陷。

可见软件缺陷是存在于软件内部的、静态的一种形式。

(3) 软件故障(software fault)　软件故障是指软件运行过程中出现的一种不希望或不可接受的内部状态。譬如软件处于执行一个多余循环过程时，我们说软件出现故障。此时若无适当措施(容错)加以及时处理，便产生软件失效。显然，软件故障是一种动态行为。

让我们具体考虑一个软件故障过程。假设求最高温度的程序为：

```
C     EXAMPLE
      DIMENSION T(4,10), MT(10)
      OPEN(4,FILE='CD')
      READ(4,*)((T(I,J),I=1,4),J=1,10)
      DO 20 J=1,10
      S=0.0
      DO 10 I=1,4
10    S=S+T(I,J)
20    MT(J)=S/4.0
      WRITE(4,*)MT
      CLOSE(4)
      STOP
      END
```

显然这程序有一个缺陷，即没有对数组 MT 进行数型声明。这样在程序执行过程中 MT 被当作整型数组而不是实型数组。一旦程序执行到语句 MT(J)=S/4.0 时，便进入故障状态。可见程序总共将产生 10 次故障。

(4) 软件失效(software failure)　软件失效是指软件运行时产生的一种不希望或不可接受的外部行为结果。

在上述的软件故障例子中，由于没有容错措施，即没有限制和排除软件故障的措施，所以将得到一个不可接受的结果(平均温度)：

6,6,5,5,3,4,3,6,6,6

这便是一个软件失效，而正确的结果(平均温度)应该是：

6.7,6.3,5.7,5.3,3.5,4.2,3.8,6.6,6.5,6.5

综上所述,软件错误是一种人为错误。一个软件错误必定产生一个或多个软件缺陷。当一个软件缺陷被激活时,便产生一个软件故障。同一个软件缺陷在不同条件下被激活,可能产生不同的软件故障。软件故障如果没有及时的容错措施加以处理,便不可避免地导致软件失效。同一个软件故障在不同条件下可能产生不同的软件失效。

如果从软件输入输出关系出发,则可以用图 2.2.1 表示软件缺陷与软件失效的关系。如果有缺陷存在于软件之中,那么当输入数据为一定组合时,软件缺陷便被激活,从而导致出现某些不可接受的输出结果。

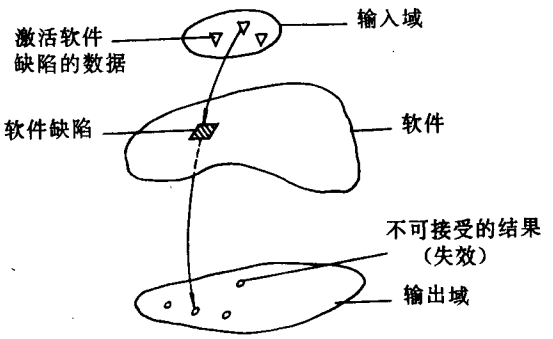

图 2.2.1 软件输入输出关系

2.3 软件错误(缺陷、故障、失效)分类

在软件生存期中存在和产生形形色色的软件错误、缺陷、故障和失效。不同的软件,其错误、缺陷、故障和失效无论在表现形式、性质乃至数量上都可能大不相同,试图对它们作一个全面而详细的阐述是不现实的。B. Beizer 曾对软件错误、缺陷、故障、失效给出一个较好的阐述[7],只是未区别对待错误、缺陷、故障及失效。诚如 2.2 节所述,错误、缺陷、故障及失效的含义不同,所以有必要加以区别对待。下面我们将它们进行大致的分类。当然,分类准则不同,所得的分类结果亦不尽相同。

2.3.1 软件错误分类

(1) 软件成分　软件是程序、可执行数据和文档的统称。因此按软件成分划分,软件错误包括程序错误、数据错误和文档错误。

(2) 生存期　软件生存期包括需求分析阶段、概要设计阶段、详细设计阶段、实现阶段、测试阶段、安装验收阶段、运行维护阶段及报废阶段。每个阶段均有人参与,因而可有相应的软件错误。于是软件错误可分为需求分析错误、概要设计错误、详细设计错误、实现错误、测试错误、安装验收错误、维护错误及报废错误。

(3) 后果　软件错误的直接后果是造成软件缺陷,因此可根据软件缺陷对软件错误

加以分类。

2.3.2 软件缺陷分类

(1) 软件成分　根据软件成分，软件缺陷可划分为程序缺陷、数据缺陷及文档缺陷。

(2) 生存期　根据软件生存期，软件缺陷可划分为需求分析缺陷、概要设计缺陷、详细设计缺陷、实现缺陷、测试缺陷(包括测试计划缺陷、测试用例缺陷、测试执行缺陷、测试结果分析缺陷以及根据测试结果对软件进行修改而带来的缺陷等)、安装与验收缺陷、维护缺陷、报废缺陷等。

(3) 表现形式　软件缺陷可有多种表现形式，但大致可分为语法缺陷和语义缺陷两大类。语法缺陷指软件不符合语言的语法要求，或不符合所应遵循的软件标准或约定。语义缺陷是指软件虽然满足语法要求，并遵循一定的软件标准或约定，但未能正确地表达所应表达的含义。就语义缺陷而言，还有许多形式，如初始化缺陷、接口缺陷等等。

(4) 原因　造成软件缺陷的直接原因是软件错误，因此可根据软件错误对软件缺陷进行分类。

(5) 结果　软件缺陷的直接后果是软件在一定条件下出现软件故障。因此可根据软件故障对软件缺陷进行分类。

2.3.3 软件故障分类

(1) 输入　软件故障是软件缺陷在一定输入情况下被激活的结果，因此可根据软件输入类型对软件故障加以分类。

(2) 动态形式　软件故障是软件内部的一种动态行为，因此可根据动态形式对软件故障加以分类。可能的形式包括控制流故障、计算故障、死循环、响应时间偏差、资源分配不当等等。

(3) 原因　软件故障的直接原因是软件缺陷，因此可根据软件缺陷对软件故障加以分类。

(4) 后果　软件故障的直接后果是无适当容错措施情况下造成软件失效，因此可根据软件失效对软件故障加以分类。

2.3.4 软件失效分类

(1) 输出　软件失效是软件的一种外部输出结果，因此可根据软件输出形式对软件失效加以分类。

(2) 功能与性能　每个软件都有功能与性能的要求，因此一种简单的分类是将软件失效划分为功能失效和性能失效。

(3) 后果　软件失效后果有大有小，可能极不相同，可划分为不同等级。一种划分方式是按文献[8]所述的方式，现介绍如下：

第一级：妨碍完成规定的操作、基本功能未完全实现、影响人员安全或导致国民经济重大损失。

第二级：对规定的操作或基本功能的实现产生有害的影响，并且不存在变通解决办法

(重新加载或重新启动该软件不属于变通解决办法)。

第三级:对规定的操作或基本功能的实现产生有害的影响,但存在合理的变通解决办法。

第四级:不便于操作,但不影响规定的操作或基本功能的实现。

第五级:其它。

2.4 软件可靠性因素

软件可靠性因素指软件生存期内影响软件可靠性的因素。显然,有许许多多因素可以影响软件可靠性,包括技术的、社会的、经济的、甚至文化的,因为在软件生存期的各个阶段均有人的干预,而人的行为受到各方面因素的影响。

但从技术角度来看,影响软件可靠性的因素主要包括:

(1) 运行环境(剖面) 软件可靠性定义相对于运行环境而言,同一软件在不同运行剖面下,其可靠性行为可能极不相同。让我们考虑一个极端例子。我们知道,软件故障是软件缺陷在一定输入情况下被激活的结果。于是可以将软件输入域划分为两个部分:G 和 F。G 中的输入不会激活软件缺陷,F 中的输入恒激活软件缺陷。如果运行剖面不包含 F 中的输入,则软件不会出现故障,其可靠性恒为 1。反之,如果运行剖面不包含 G 中的输入,则每一输入情况下均出现故障。如果没有容错措施,则导致软件失效,软件可靠性恒为 0。

(2) 软件规模 如果软件只含一条指令,那么谈论软件可靠性问题便失去意义。随着软件规模的增大,软件可靠性问题愈显突出。在我们考虑软件可靠性问题时,软件一般是指中型以上软件(4000～5000 条以上语句),这时可靠性问题难以对付。软件工程实践的一个侧面可以反映这一点,即单元测试一般由编程人员本人进行,而综合测试则需独立的测试人员。软件可靠性增长模型也主要应用于综合测试阶段。

(3) 软件内部结构 软件内部结构一般比较复杂,且动态变化,对可靠性的影响也不甚清楚。但总的说来,结构越复杂,软件复杂度越高,内含缺陷数越大,因而软件可靠度越低。

(4) 软件可靠性设计技术 关于软件可靠性设计技术的外延并不明确,但一般是指软件设计阶段中采用的用以保证和提高软件可靠性为主要目标的软件技术,如失效模式与效应分析(FMEA),故障树分析(FTA)等。显然采用或不采用软件可靠性设计技术对软件可靠性必有影响。

(5) 软件(可靠性)测试与投入 研究表明,软件测试方法与资源投入对软件可靠性有不可忽视的影响。

(6) 软件可靠性管理 软件可靠性管理旨在系统管理软件生存期各阶段的可靠性活动,使之系统化、规范化、一体化,这样就可以避免许多人为错误,以提高软件可靠性。

(7) 软件开发人员能力和经验 显然,软件开发人员(包括测试人员)的能力愈强,经验愈丰富,所犯错误便可能愈少,所得软件产品质量愈高,相应的可靠性也愈高。

(8) 软件开发方法 软件工程表明,开发方法对软件可靠性有显著影响。与非结构化

方法比较，结构化方法可以明显减少软件缺陷数。形式化方法也有类似特征。

(9) 软件开发环境　研究表明，程序语言对软件可靠性有影响。譬如，结构化语言 Ada 优于 Fortran 语言(缺陷数减少)，而软件(测试)工具优劣则影响测试效果。

总之，有许许多多的因素影响软件可靠性，至今无法确定它们与软件可靠性之间的定量关系，甚至连它们与软件可靠性之间的定性关系也不甚清楚。

参 考 文 献

[1] Kline M. B., Software & Hardware R&M: What are the Differences? Proc. Ann. R&M Sym., 1980, pp179-185

[2] Soi I. M., K. Gopal, Hardware vs Software Reliability—— A Comparative Study, Microelectronics and Reliability, Vol. 20, No. 6, 1980, pp881—885

[3] Srivastava S., I. M. Soi, Hardware vs Software Maintainability: A Comparative Study, Microelectronics and Reliability, Vol. 22, No. 6, 1982, pp1077—1079

[4] 蔡开元，软件可靠性概要，系统工程与电子技术，Vol. 15, No. 4, 1993, pp47—54

[5] Cai K. Y., C. Y. Wen, M. L. Zhang, A Critical Review on Software Reliability Modeling, Reliability Engineering and System Safety, Vol. 32, 1991, pp357—371

[6] 丘玉圃，FORTRAN 程序设计，科学出版社，1981

[7] Beizer B., Software Testing Techniques, Van Nostrand Reinhold Company, 1983

[8] 航空工业部，软件开发规范，1988

第三章　如何开发可靠的软件

3.1 概　　述

纵谈软件可靠性，如果最终又开发不出可靠的软件，那纵谈便近乎空谈。如何开发可靠的软件是软件可靠性工程的核心问题。为此，首先需说明“可靠”的含义，即应给出可供选用的软件可靠性度量。这些度量应该是具体的，工程化的，可应用于软件的各种中间形态。其次，软件工程经验表明，如果在软件开发初期，尤其是软件设计阶段不考虑软件可靠性问题，则很难保证软件的可靠性，或者说，可靠性是“设计”到软件中去的，而不应是仅在设计之后的“事后诸葛亮”，因此需讨论各种软件可靠性设计技术。再则，应对整个软件开发过程实行有效的可靠性管理，以便对各阶段的软件可靠性活动进行监督。本章将讨论软件可靠性度量及软件可靠性管理，而软件可靠性设计则留在下一章专门讨论。

应当强调，为了开发可靠的软件，必须遵循软件工程的一般规范，我们这里所谈论的是，在遵循软件工程的一般规范的基础上，为了使软件高度可靠所应进行的一些“额外”的软件可靠性活动。

3.2　软件可靠性度量

在测试阶段收集测试数据，利用种种软件可靠性模型评估或预测软件可靠性，只能提供有关软件现有可靠性水平的信息，不能提高，也难以保证软件可靠性，只可谓“事后诸葛亮”。保证和提高软件可靠性的最重要手段是在软件设计阶段采取措施进行可靠性控制。为此，IEEE 曾于 1982 年组织几十位专家召开若干次研讨会，致力于软件可靠性度量工作[1,2]，并于 1989 年发表了与软件可靠性直接或间接有关的 39 种度量[3,4]。这是国外迄今为止在软件可靠性度量方面最重要的工作。Everett 曾专文讨论软件可靠性度量问题[5]，可惜他讨论的是软件可靠性度量在软件生存期各个阶段可能的作用及应用软件可靠性度量所应作的一些工作，并不讨论应当采用哪些软件可靠性度量。在国内，作者曾初步探讨了软件可靠性度量体系，并提出了一系列适用于软件生存期各阶段的可靠性度量[6]。这可能是迄今为止国内在软件可靠性度量方面的唯一工作。尽管关于软件质量度量的工作已经比较系统化了[7]，但关于软件可靠性度量的工作很不成熟。下面我们将尽可能地对软件可靠性度量给出一个系统阐述。

3.2.1　为什么需要软件可靠性度量

既然软件可靠性问题十分重要，软件失效后果可能是灾难性的，但要从数学上严格证明一个软件正确几乎是不可能的。事实上，有几个曾被“证明”为正确的软件，事后发现它们仍含有缺陷[8]。那么人们要问：一个软件是否可靠？在多大程度上可靠？或者是否已达