ICS 35.080
L 77

中华人民共和国国家标准

GB/T 28171—2011

嵌入式软件可靠性测试方法

Embedded software reliability testing method

2011-12-30 发布　　2012-06-01 实施

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会　发布

目　　次

前　言

本标准按照 GB/T 1.1—2009 给出的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准由全国信息技术标准化技术委员会(SAC/TC 28)提出并归口。

本标准起草单位:中国电子技术标准化研究所、珠海南方软件产品检测中心、珠海许继电气有限公司、炬力集成电路设计有限公司、上海博为峰软件技术有限公司、沈阳软件公共技术服务平台有限公司、深圳市吉阳自动化科技有限公司、上海博泰悦臻电子设备制造有限公司、广东宝莱特医用科技股份有限公司、上海嵌入式系统应用工程技术研究中心、珠海优特电力科技股份有限公司、上海超算并行软件有限责任公司、上海鲁齐信息科技有限公司。

本标准主要起草人:侯建华、陈勇、秦卫东、杨丽春、王兴念、潘海洋、王忠福、张展新、徐锋光、阳如坤、应臻恺、张旸旸、史旭光。

引　言

嵌入式系统是指以应用为中心，以计算机技术为基础，软硬件可剪裁，适应应用系统对功能、可靠性、成本、体积和功耗严格要求的专门计算机系统。嵌入式技术并不是一个独立的学科，它是伴随着微电子技术和计算机技术的发展，微控制芯片功能越来越强大，而嵌入微控制芯片的设备和系统越来越多而发展起来的。嵌入式系统几乎包括了生活中所有的电器设备，如：MP3、手机、数字电视机、汽车、微波炉、数字相机、电梯、空调、自动售货机、工业自动化仪表与医疗仪器等。

虽然大多数软件测试方法都可以直接或间接地用于嵌入式软件的测试，但嵌入式软件可靠性测试与通用软件可靠性测试有着较大差别，这是由于嵌入式系统软硬件功能界限模糊，软件对硬件的依赖性和专用性较强，对实时性、安全性要求较高，目前针对嵌入式软件的测试和调试工具较少等。这些都使得嵌入式软件的测试相比通用计算机软件测试可继承性较差。

本标准参考了国内外相关资料，结合嵌入式软件可靠性测试的实践和特点而制定。

嵌入式软件可靠性测试方法

1 范围

本标准规定了嵌入式软件生存周期内软件产品的可靠性测试方法、过程和准则。

本标准适用于嵌入式软件生存周期全过程，可用于嵌入式软件测试中的可靠性增长测试和可靠性确认测试要求。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

GB/T 9386—2008 计算机软件测试文档编制规范

GB/T 11457—2006 信息技术 软件工程术语

GB/T 15532—2008 计算机软件测试规范

GB/T 16260.2—2006 软件工程 产品质量 第2部分：外部度量

GB/T 16260.3—2006 软件工程 产品质量 第3部分：内部度量

3 术语和定义

GB/T 11457—2006 界定的以及下列术语和定义适用于本文件。

3.1

软件可靠性 software reliability

特定数目的自然单元中或特定任务时间内软件无失效执行的概率。

3.2

偏离 deviation

嵌入式软件执行中的系统行为相对预期行为的偏差。

3.3

级联 cascaded

直接由初始行为产生的行为。例如，级联偏离、级联失效、级联偏差。

3.4

失效 failure

系统运行行为对用户要求的偏离。

3.5

失效强度 failure intensity

单位时间出现的失效次数。

注：是表示可靠性的另一种方式。

3.6

操作 operation

持续一段时间，结束时将控制权还给系统的一种逻辑任务。操作与软件的功能或特征相关，例如，

用户命令的执行、对输入的响应处理、系统事务处理。

3.7

操作剖面　operational profile

操作及其出现的概率的集合。

3.8

操作模式　operational mode

随时间或资源与输入的不同而有较大差别的操作的集合。

4　测试目的

嵌入式软件可靠性测试的目的是：

——通过嵌入式软件可靠性测试有效地发现程序中影响软件可靠性的缺陷，实现可靠性增长；

——验证嵌入式软件是否满足嵌入式系统开发合同或项目开发计划、系统与子系统设计文档、软件需求规格说明和软件设计说明所规定的软件可靠性要求、可靠性的定量要求；

——评估当前嵌入式软件可靠性的水平，预测未来可能达到的水平，从而为嵌入式软件开发管理提供决策依据；

——通过嵌入式软件可靠性测试，为用户平衡可靠性、时间开发和开发费用提供参考。

5　测试环境

嵌入式软件可靠性测试的测试环境如下：

——具备嵌入式软件运行的目标环境，或高度一致(除位置、结构、接口等部分外其他环境与目标环境一致)的仿真环境；

——具备与嵌入式系统应用验证相关的必要的测试仪器仪表，例如，频率源、波形发生器、标准电压电流源、规约分析器等；

——具备嵌入式系统运行的温度、湿度、电磁兼容、振动、冲击等环境；

——具备一些专用的测试工具；

——具备操作剖面所需要的全部外部输入输出的环境支持。

6　测试内容

嵌入式软件可靠性测试的内容主要包括：可靠性增长测试和可靠性确认测试。

可靠性增长测试以迭代的方式进行，根据测试过程中检出和跟踪的失效，使用基于可靠性增长模型和统计推理的可靠性评估方法，进行失效强度的估计，然后消除缺陷再测试，使可靠性达到目标要求从而结束测试。

可靠性确认测试是产品发布或交付前为确定合同约定或者具体标准规定的可靠性指标得到满足而组织实施的测试。

7　测试方法

7.1　总则

GB/T 15532—2008 中确定的系统测试方法适用于本标准，一般采用黑盒测试技术进行嵌入式软

件可靠性测试项目的测试。

进行嵌入式软件可靠性测试，首先明确可靠性目标。没有规定可靠性目标时，应按照 7.2 中的方法进行可靠性目标的确定。

可靠性目标确定后，编制测试计划、开发操作剖面、进行测试准备、执行可靠性测试、分析评估，最后给出可靠性测试报告。

7.2 可靠性目标的确定

7.2.1 确定失效程度

对一个嵌入式软件，根据产品的使用范围、对象，确定失效的严重程度。一般根据对人员生命、成本和系统能力的影响来区分失效的严重程度；失效严重程度级别用于对失效数据的分析，在测试过程中判定是否需要查找缺陷和解决。表 1 给出失效程度的级别。

表 1 嵌入式软件失效的程度级别

失效程度级别	对失效的描述
1	不能进行一项或多项关键操作
2	不能进行一项或多项重要操作
3	不能进行一项或多项操作，但是有补救办法
4	一项或多项操作中的小缺陷

7.2.2 为嵌入式软件建立失效强度目标

根据嵌入式系统的使用对象，为嵌入式软件建立失效强度目标，表 2 为推荐的失效强度目标、失效间隔时间和失效影响的对照表。

表 2 失效强度目标、失效间隔时间与失效影响

失效造成的影响	典型失效强度目标	失效间隔时间
造成人员伤亡或千万元以上经济损失	10^{-6}	114 a
没有人身伤害，10 万元以上经济损失	10^{-4}	10 000 h
没有人身伤害，万元以上经济损失	10^{-3}	1 000 h
没有人身伤害，千元以上经济损失	10^{-2}	100 h
没有人身伤害，少量经济损失	10^{-1}	10 h
没有人身伤害，轻微或无经济损失	1	1 h

7.2.3 选择通用度量

由于嵌入式系统一般是连续运行的，因此嵌入式软件在时间度量上，普通时间和执行时间是一致的，所以，选择普通时间作为通用度量，本测试方法采用小时(h)作为时间单位。当然也可以采用自然单元作为通用度量。表 3 为 1 h 任务时间的可靠性与特定周期等效失效强度的对照表。

表 3　1 h 任务时间的可靠性与特定周期等效失效强度的对照

1 h 任务时间的可靠性	特定周期失效强度
0.368	每 1 h 1 次失效
0.9	每 1 000 h 105 次失效
0.959	每天 1 次失效
0.99	每 1 000 h 10 次失效
0.994	每周 1 次失效
0.998 6	每月 1 次失效
0.999	每 1 000 h 1 次失效
0.999 89	每年 1 次失效

失效强度和可靠性转换见式(1)和式(2)：

$$\lambda = -\frac{\ln R}{t} \qquad \cdots\cdots(1)$$

$$R = \exp(-\lambda t) \qquad \cdots\cdots(2)$$

式中：

λ ——失效强度；

R——可靠性；

t ——自然或时间单位数。

7.3　开发操作剖面

7.3.1　综述

使用表格或图形方式构造嵌入式软件的操作剖面。测试人员在系统体系结构设计人员、软件工程师的参与下构造操作剖面。

开发操作剖面时，确定输入及相关数据域，分析系统的可靠性需求，对所有可能的操作模式进行分类列表；分析影响软件操作模式的全部外界条件及其对软件运行的影响程度；对各种功能需求之间的相关性进行分析和组合，对于密切相关的功能模块进行合并，对于部分相关的功能模块给出相应的输入变量的组合方法。

7.3.2　确定操作模式

对于不同的嵌入式软件，操作模式会显著不同，确定操作模式宜按表 4 的方法进行。

表 4　操作模式的确定

确定操作模式	确定方法
主要时间和次要时间	某天或一天的某段时间，处理的事务或事务频度显著不同，处理事务量的显著差别
不同的用户类型	管理员、一般使用者、新手等
输入的显著差别	大量和多变的输入
电源的极限	电源方面的要求

表 4（续）

确定操作模式	确定方法
温度的极限	温度方面的要求
电磁的极限	电磁兼容方面要求
其他环境的重大变化	湿度、振动、噪声等方面的要求
行业规范要求的模式	电信、电力、银行等行业的要求

7.3.3 确定操作的发起者

操作的发起者包括用户、外部条件、嵌入式系统自身等，宜按表 5 的确定方法进行识别。

表 5 操作发起者的确定

确定发起者	确定方法
使用用户	使用者操作、远程登录等
外部条件	外部输入，例如输入一个量、收到输入信号
嵌入式系统自身	嵌入式系统软硬件判断出的条件，例如内存异常、中断信号、内部变量

7.3.4 选择表示方法

操作剖面的表示方法有表格法和图示法。可根据使用习惯来选择。

本标准推荐采用表格方法，即用列表的方式，列出所有操作模式、操作发起者、操作、操作出现率等信息。

7.3.5 创建操作表

创建操作表需要参考用户需求、软件使用说明、行业规定及相关标准要求等，通常以操作的启动者来划分。在定义输入空间、操作覆盖输入空间时，应考虑全覆盖。表 6 举例说明如何创建一个操作表。

表 6 操作表

确定发起者	操　　作
A(外部输入一个信号)	输出一个信号到外部 1
	输出一个信号到外部 2
B(使用者输入一个指令)	显示一行信息
C(内部数据存储满)	提示数据满
	系统信号灯亮

7.3.6 确定出现率

根据用户需求、软件规格、使用说明、经验等信息，确定每种操作的出现率。表 7 举例说明如何确定出现率。

表 7　出现率

操　　作	出现率(每小时出现次数)
输出一个信号到外部 1	10
输出一个信号到外部 2	10
显示一行信息	1 000
提示数据满	0.001
系统信号灯亮	0.001
合计	1 020.002

7.3.7　确定出现概率

将每个操作的出现率除以总出现率。表 8 举例说明如何确定出现概率。

表 8　出现概率

操　　作	出现概率
输出一个信号到外部 1	0.009 8
输出一个信号到外部 2	0.009 8
显示一行信息	0.980 4
提示数据满	0.000 000 98
系统信号灯亮	0.000 000 98
合计	1

7.4　测试准备

7.4.1　综述

本活动主要根据概率分布信息和测试计划生成对应的测试用例输入文件，计算或给出每一测试用例预期的输出结果，构建测试环境，选择或开发测试工具，进行测试用例设计；生成测试用例时，一定要保证测试的覆盖率。

确定输入及相关数据域，分析系统的可靠性需求，对所有可能的操作模式进行分类列表；分析影响软件操作模式的全部外界条件及其对软件运行的影响程度；对各种功能需求之间的相关性进行分析和组合，对于密切相关的功能模块进行合并，对于部分相关的功能模块给出相应的输入变量的组合方法。

附录 C 列举了一些嵌入式软件可靠性测试用例的实例，可作为测试用例设计参考。

7.4.2　测试用例准备

测试用例准备活动主要包括：

——估计当前版本所需的新测试用例的数量；

——在要测试的系统之间分配新测试用例的数量；

——在每个系统的新操作之间分配新测试用例的数量；

——指定新测试用例；

——开发新增测试用例，将新测试用例添加到以前版本的测试用例上；
——新开发的软件的第一版可靠性测试用例基于前阶段的测试用例来设计。

7.4.2.1 估计需要测试用例的数量

估计需要测试用例的数量，要考虑时间和成本因素，取这两个数量的最小值作为计划准备的测试用例的数量。

时间的计算，用可用的时间乘以可用的人员数，再除以准备一个测试用例的平均时间。

成本的计算，用建立测试用例的预算除以每个测试用例的平均准备成本。

对于再次测试，例如回归测试，只计算新增加的测试用例数量。

7.4.2.2 分配测试用例

为每个操作分配测试用例数量。对于回归测试，只分配新修改的操作的测试用例。

根据操作出现率，进行下列工作：

——确定很少出现的关键操作（概率小但应测到），为每个这样的操作分配测试用例数量。关键操作是指失效会造成人身伤害、重大损失的操作，对这些操作要分配充足的测试用例。

——确定偶发性的操作，分配一个测试用例，偶发性的操作是指出现概率非常低的操作，这样做的目的是保证至少为这样的操作分配一个测试用例。

——根据操作概率，把剩下的测试用例分配给剩余的其他操作。

7.4.2.3 指定测试用例

为每个操作指定测试用例。

从操作的直接输入变量的值域中，找出具有相似失效行为的值域，形成输入变量值域组合。

在选择了输入变量值域组合之后，从组成值域的集合成员中，随机选择输入变量，作为测试用例。

选择了测试用例后，编制测试用例的脚本。

7.4.3 测试过程准备

为每个操作模式准备一个测试过程，指定或调整测试操作剖面和操作出现率，调整主要发生在回归测试或需求变更的测试过程。

7.5 执行测试

7.5.1 综述

在执行可靠性测试时注意以下方面：

——被测软件的测试环境（包括硬件配置和软件支持环境）和预期的实际使用环境应完全一致。

——测试时按测试计划和顺序对每一个测试用例进行测试，判断软件输出是否符合预期结果。

——测试时应记录测试结果、运行时间和判断结果。如果软件失效，那么还应记录失效现象和时间，以供失效分析。

7.5.2 分配测试时间

分配测试时间宜按照以下方法进行：

——在要测试的系统之间分配测试时间；

——对进行可靠性增长测试的功能测试、回归测试、负载测试之间分配测试时间；要分配足够的时间进行功能测试，以便充分执行测试用例，并对前一版本进行回归测试，然后把剩下的时间分

配给负载测试；

——对进行负载测试的操作模式之间分配测试时间；

——进行确认测试，将所有的时间都分配给负载测试；

——时间分配以小时计。

估计测试需要的时间按式(3)：

$$t=\frac{T_N}{\lambda_F} \qquad \cdots\cdots(3)$$

式中：

t ——用自然或时钟时间单元表示的测试时间；

T_N——规范化度量(MTTF 数)，见式(4)；

λ_F ——失效强度目标。

$$T_N=\frac{\ln\frac{\beta}{1-\alpha}}{1-\gamma} \qquad \cdots\cdots(4)$$

式中：

γ——分辨率；

α——提供商风险，即错误地认为失效强度目标没有达到但实际上已经达到的概率；

β——客户风险，即错误地认为失效强度目标已经达到，但实际上没有达到的概率。

7.5.3 调用测试

对于可靠性增长测试，首先执行功能测试，然后进行负载测试，按照测试用例进行全覆盖测试。

在每次对软件做了较大修改后，要进行回归测试。

功能测试按照分配的测试用例，顺序调用测试用例进行测试。

回归测试应全面检查需求变化处。另外随机测试和回归测试的要求不同，需要注意区分，按影响域调用相关测试用例。

7.5.4 标识出现的失效

在测试中应对测试的输出进行分析，确定失效、失效时间和失效强度。

7.5.4.1 分析测试输出的偏离

采用自动化工具或以人工对测试结果审查，确定执行结果与相对预期行为的偏差。

在分析偏差的过程中，不计算级联。

7.5.4.2 确定哪些偏离是失效

确定出现的偏离是否为失效。

硬件错误引起的失效不作为嵌入式软件的失效统计，但对于需要实现软件容错、避免严重错误的失效，应统计在内。

7.5.4.3 估计失效发生的时间

估计失效发生的时间采用统一的时间度量，即普通时间，以小时为单位。以出现顺序，累加度量单元，包括所有操作模式的功能测试、回归测试、负载测试。

对于同时出现的多个失效记录，会导致多个零失效间隔，应在记录的时间间隔内，估计随机的时间，用于替代这些零间隔。

7.5.4.4 指派失效严重程度类

对失效，确定出失效的严重等级。

7.5.4.5 形成测试记录

按照执行的测试用例，记录测试过程、测试结果、运行时间、失效时间、失效现象，形成测试记录。表9为测试记录的示例。

表9 测试记录示例

事　　件	时　　间	失效的时间间隔/min
开始测试	2010年1月1日8时00分	0
失效1	2010年1月1日8时35分	35
失效2	2010年1月1日9时10分	35
失效3	2010年1月1日13时20分	250
失效4	2010年1月1日15时30分	130
测试结束	2010年1月1日16时00分	30

7.6 失效数据的分析评估

7.6.1 综述

本活动进行失效分析、可靠性估计及判定；整理测试记录，并将结果写入测试报告。测试结果不满足给定的可靠性目标时，应在完成纠正错误后组织实施回归测试。

7.6.2 可靠性增长测试

7.6.2.1 失效数据分析

在进行失效数据分析时，必须明确以下问题：

——对某一失效，决定不查找和解决的，该失效不统计；

——重复的失效不统计；

——人为操作失误或外界环境异常引起的失效需要统计；

——硬件错误引起的失效不统计，但由于软件没有进行冗余、容错的失效应统计。

可靠性增长测试过程，要根据失效数据定期进行趋势分析和评估。可靠性增长测试过程中发现的所有缺陷都应纠正，且纠正时应确保不引入新的缺陷。

在测试过程中，应详细记录失效的时间、现象，以分析失效的根源和纠正缺陷。表10为可靠性增长测试过程中推荐的评估频率。

表10 可靠性增长测试过程的评估频率

剩余测试长度	分析评估频率
＜1个月	每天一次
1个月～3个月	每周二次
＞3个月	每周一次

对失效数据进行趋势分析，以指导可靠性模型的选择、失效强度估计。趋势分析采用定期评估FI/FIO比的方法。

FI/FIO 是可证明失效强度与失效强度目标之比，FI/FIO 比 λ_D 为：

$$\lambda_D = \frac{T_{N,A}(n)}{t_t \lambda_F} \qquad \cdots\cdots(5)$$

式中：

$T_{N,A}(n)$——已经出现的失效数对应的继续和接受边界的规范化度量(见式(9))；

t_t ——测试结束时的时间单位数；

λ_F ——失效强度目标。

在可靠性预测和趋势分析时，遵循以下方法：

——FI/FIO 比估计的精确度取决于样本规模(发生失效的数量)；

——当 FI/FIO 比大于 2 时，检查最近的 5 个值，查看是否出现稳定、相当明显的上升趋势，如果是，继续测试；不是时，分析原因，做好软件的变更控制和测试控制；

——当规范化失效强度小于 0.5 时，结束测试；

——当 FI/FIO 比大于 15，在测试时间内不可能达到失效强度目标时，应推迟发布软件，商议调整失效强度目标，修改错误后再测试。

在测试过程中，定期评估和预测可靠性，以指导下一步的测试工作，主要有：

——估计当前的可靠性，包括 MTTF、当前失效率、当前失效强度等；

——预测现在达到的可靠性水平；

——预测还需要增加的测试，预测何时能够达到可靠性目标。

根据分析结果，可适当调整测试资源，增加投入，使可靠性增长测试在预定时间内达到目标。

7.6.2.2 失效强度估计

根据对失效数据的趋势分析，选择可靠性模型，估计失效强度。可靠性模型选择可参见附录 B。由于可靠性模型较多，在可靠性模型的选择上本方法推荐两种作为参考，这两种模型为：基本(Musa)和对数泊松(Musa-Okumoto)执行时间模型。

基本和对数泊松执行时间模型，又分别称为指数和对数模型。这两种模型都使用执行时间，在嵌入式软件测试估计时，日历时间即为执行时间(这是基于嵌入式软件是连续运行的假设)。

出现失效的函数的基本泊松模型的失效强度 λ 为：

$$\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{\nu_0}\right) \qquad \cdots\cdots(6)$$

式中：

λ_0——开始执行的初始失效强度；

μ ——给定时间点上的平均或预期发生的失效数；

ν_0——在无限时间内发生的失效总数。

出现失效的函数的对数泊松模型的失效强度 λ 为：

$$\lambda(\mu) = \lambda_0 \exp(-\vartheta\mu) \qquad \cdots\cdots(7)$$

式中：

λ_0 ——开始执行的初始失效强度；

μ ——给定时间点上的平均或预期发生的失效数；

ϑ ——失效强度延迟参数。

对于这两种模型，需要在执行开始时，确定 λ_0、ν_0、ϑ，应根据嵌入式软件自身的特征预测，如软件的源代码长度、可执行代码的编译效率、软件的复杂度、对失效强度的要求等；也可以在嵌入式软件系统测试阶段，通过收集失效的数量进行估计。

7.6.3 可靠性确认测试

7.6.3.1 综述

可靠性确认测试是为确认在给定统计置信度下，对嵌入式软件当前的可靠性水平是否满足用户需要而进行的测试，即确认是否满足所规定的可靠性目标，本标准推荐采用失效执行时间和可靠性示图来进行确认测试。

7.6.3.2 失效执行时间确认测试

给定客户风险 β 和 MTBF 的检验下限 θ，由式(8)计算出嵌入式软件的可靠性测试时间 T。

根据 T，按照测试用例执行测试，在 T 时间内无失效时接受软件，发生失效时拒绝软件。

$$T=-\theta\ln\beta \qquad \cdots\cdots(8)$$

式中：

θ ——MTBF 检验下限值；

β ——客户风险。

这种确认测试适用于 MTBF 在 1 000 h 以内的嵌入式软件可靠性确认测试，对于 MTBF 大于 1 000 h 的嵌入式软件，应采用可靠性示图确认测试。

7.6.3.3 可靠性示图确认测试

根据客户风险和开发风险级别构造可靠性示图，失效被绘制在图上，根据失效落入的区域，判定被测嵌入式软件被接受、拒绝还是继续测试。

首先与客户及提供商协商确定提供商风险 α 和客户风险 β，参照附录 A 的方法，画出可靠性示图。

提供商风险 α 和客户风险 β 一般选取 20%以下，最大不宜超过 30%；对于可靠性要求很高的嵌入式软件应在 10%以内。

绘制可靠性示图时，需计算出继续和接受边界、继续和拒绝边界，并绘出边界线。边界线由式(9)和式(10)求出：

$$T_{N,A}(n)=\frac{A-n\ln\gamma}{1-\gamma} \qquad \cdots\cdots(9)$$

式中：

$T_{N,A}$ ——已经出现的失效数对应的继续和接受边界的规范化度量；

n ——失效数；

A ——由式(11)计算得出；

γ ——分辨率，即最大可接受失效强度与失效强度目标的比值，γ 的取值范围 1.1～2.0 之间。可靠性要求越高，取值应越小。

$$T_{N,B}(n)=\frac{B-n\ln\gamma}{1-\gamma} \qquad \cdots\cdots(10)$$

式中：

$T_{N,B}$ ——已经出现的失效数对应的继续和拒绝边界的规范化度量；

n ——失效数；

B ——由式(12)计算得出；

γ ——分辨率，即最大可接受失效强度与失效强度目标的比值，γ 的取值范围 1.1～2.0 之间。可靠性要求越高，取值应越小。

$$A=\ln\frac{\beta}{1-\alpha} \qquad \cdots\cdots(11)$$

$$B=\ln\frac{1-\beta}{\alpha} \qquad \cdots\cdots(12)$$

图 1 表示的可靠性示图中客户风险 $\beta=0.1$，提供商风险 $\alpha=0.1$，分辨率 $\gamma=2$。

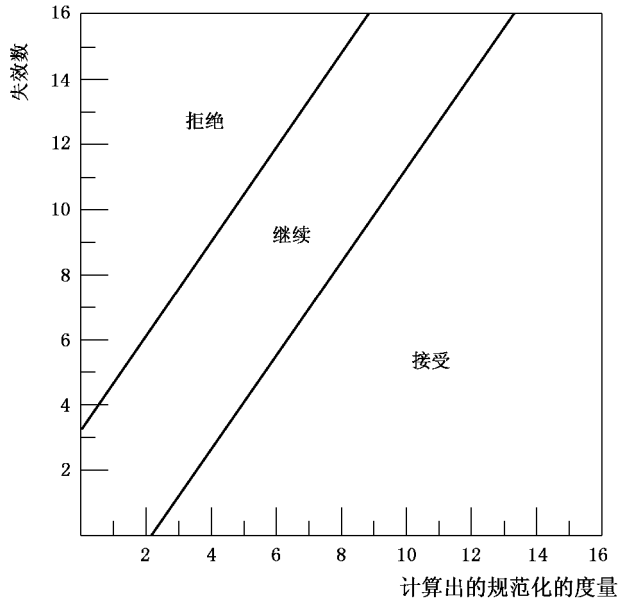

图 1　经过规范化度量(MTTF)

可靠性示图的绘制方法和不同分辨率、风险的参数计算，参见附录 A。

根据失效数和失效发生的时间单元，计算出经过规范化的度量(MTTF)。纵坐标为失效数，横坐标为计算出的规范化的度量，标注在可靠性示图上。按照下面的方法进行判决：

——该点落在继续区域时，继续测试；

——落在拒绝区域时，结束测试，拒绝软件；

——落在接受区域时，结束测试，接受软件；

——直到测试结束一直在继续区域时，计算 FI/FIO 比，FI/FIO 比大于 5 而且出现了失效时，拒绝软件；FI/FIO 比小于 5 时则接受软件。

7.7　可靠性测试报告

可靠性测试完毕，应给出一份可靠性测试报告，一般包括：

——测试环境、测试设备情况；

——可靠性目标及确定情况；

——操作剖面确定和构建情况；

——可靠性增长测试过程、时间，测试用例数量，测试覆盖率；

——可靠性确认测试过程、时间，测试用例数量，测试覆盖率；

——失效强度估计结果和分析；

——可靠性模型拟合、拒绝或接受结论；

——测试中发现的问题和处理建议；

——被测软件的版本信息。

测试报告的编制应符合 GB/T 9386—2008 的要求。

测试覆盖率等指标的计算宜采用 GB/T 16260.2—2006 和 16260.3—2006 的方法度量，在进行可靠性增长测试时，可以按照该标准的计算方法，计算出相应的指标，并加入到报告中。

附 录 A
（资料性附录）
可靠性示图绘制

在采用可靠性示图确认测试时，为了绘出可靠性示图，需求出继续和接受边界、继续和拒绝边界，并绘出边界线，推荐按下面的方法绘出可靠性示图。

$n=0$ 时的 T_N，与横轴的交点：

$$T_{N,A}(0)=\frac{A}{1-\gamma} \qquad \cdots\cdots(A.1)$$

$$T_{N,B}(0)=\frac{B}{1-\gamma} \qquad \cdots\cdots(A.2)$$

$n=16$ 时的 T_N，与横轴的交点：

$$T_{N,A}(16)=\frac{A-16\ln\gamma}{1-\gamma} \qquad \cdots\cdots(A.3)$$

$$T_{N,B}(16)=\frac{B-16\ln\gamma}{1-\gamma} \qquad \cdots\cdots(A.4)$$

$T_N=0$ 时，与纵轴的交点：

$$n_A(0)=\frac{A}{\ln\gamma} \qquad \cdots\cdots(A.5)$$

$$n_B(0)=\frac{B}{\ln\gamma} \qquad \cdots\cdots(A.6)$$

$T_N=16$ 时，与纵轴的交点：

$$n_A(16)=\frac{A-16(1-\gamma)}{\ln\gamma} \qquad \cdots\cdots(A.7)$$

$$n_B(16)=\frac{B-16(1-\gamma)}{\ln\gamma} \qquad \cdots\cdots(A.8)$$

为了使用的方便，给出典型的参数，见表 A.1 和表 A.2。

表 A.1 区域与边界的各个横轴和纵轴的交点值

交点位置	分辨率 γ		
	2	1.5	1.1
$n=0$ 点的横轴	$-A$，$-B$	$-2A$，$-2B$	$-10A$，$-10B$
$n=16$ 点的横轴	$-A+11.1$，$-B+11.1$	$-2A+13.0$，$-2B+13.0$	$-10A+15.2$，$-10B+15.2$
$T_N=0$ 点的纵轴	$1.44A$，$1.44B$	$2.47A$，$2.47B$	$10.5A$，$10.5B$
$T_N=16$ 点的纵轴	$(A+16)/0.693$， $(B+16)/0.693$	$(A+8)/0.405$， $(B+8)/0.405$	$(A+1.6)/0.095\ 3$， $(B+1.6)/0.095\ 3$

表 A.2 各种提供商风险和客户风险水平条件下的 *A* 值和 *B* 值

提供商风险	参数	客户风险			
		0.1	0.05	0.01	0.001
0.1	A	−2.20	−2.89	−4.50	−6.80
	B	2.20	2.25	2.29	2.30

表 A.2（续）

提供商风险	参数	客户风险			
		0.1	0.05	0.01	0.001
0.05	A	−2.25	−2.94	−4.55	−6.86
	B	2.89	2.94	2.99	2.99
0.01	A	−2.29	−2.99	−4.60	−6.90
	B	4.50	4.55	4.60	4.60
0.001	A	−2.30	−2.99	−4.60	−6.91
	B	6.80	6.86	6.90	6.91

可以看出，A 随客户风险迅速变化，但随提供商风险变化很小，它决定接受边界与横轴线在 $n=0$ 的交点，因此接受边界会随客户风险急剧变化，随提供商风险有很小的变化。

B 随提供商风险迅速变化，但随客户风险变化很小，它决定拒绝边界与纵轴线在 $T_N=0$ 的交点，因此拒绝边界会随提供商风险急剧变化，随客户风险有很小的变化。

在确认测试时，可以根据需要，画出所需要的客户风险和提供商风险所有组合的可靠性示图，也可以只画出客户和提供商风险对称的可靠性示图。当二者不对称时使用足够近似的图。

随着分辨率、客户风险水平或提供商风险水平的降低，继续区域会拓宽，因此如果要降低在估计失效强度中所能够容忍的错误，或降低错误决策的风险，达到拒绝或接受区域则需要更多的测试。

附 录 B
（资料性附录）
可靠性模型选择

根据对失效数据的趋势分析，选择可靠性模型，估计失效强度。表 B.1 给出了分析结果与可靠性模型选择的参考对照。

表 B.1 根据趋势选择模型

编号	失效趋势	可靠性估计模型选择
1	可靠性增长	J-M 模型、G-O 模型、M-O 模型等
2	可靠性下降	选择允许失效强度上升的模型
3	可靠性先降后升	Yamada、Ohba、Osaki S 模型
4	可靠性稳定	HPP 模型、失效时间服从指数分布的模型

附 录 C
（资料性附录）
可靠性测试用例分析设计实例

C.1 故障注入测试

故障注入测试是针对嵌入式系统的可靠性测试的常用手段。故障注入测试项目的测试顺序应综合考虑该故障出现的概率和测试项目之间的互补关系。故障注入测试分为如下几步：

a） 故障注入。在这一步把缺陷、故障和失效插入到代码中去，这时候要确定插入的位置，可能还要添加适当的代码。

b） 执行测试。要用输入必要的数据、产生内部的或外部的事件、发送特定的消息等方法把前面插入的故障激活。

c） 测试结果收集。包括收集测试前设置的观察点处观察到的数据、现象等，另外还要收集被测系统出现的各种异常现象，如死机、复位。

d） 测试结果评估。即根据测试过程中收集到的各种测试数据，结合用户的实际使用需求和设计需求，判断当前出现的各种现象是否属于正常。在实际操作中，可以采用下面的两条准则来判断是否需要启动修改流程：

1） 故障注入后引起用户不可接受的严重故障；

2） 故障停止插入后系统无法恢复，仍然处于故障状态。

针对嵌入式软件可靠性测试的故障注入方法，本附录列举了常用的插入测试项目，通过这些项目可构造大量的可靠性测试用例。

C.2 内存过载测试

当一个计算机系统的内存占用率为80%～100%时，视为内存过载。内存过载测试主要是测试系统在内存即将用完的时候，系统运行的可靠性。

故障产生原因有大流量冲击、内存丢失、算法缺陷等。故障可能产生的后果如复位、空转、系统内部数据状态不一致等。

对于一般系统，内存过载时能自动降低业务处理量，如果持续时间过长，则允许单板复位。对于高可用系统，要求能区分是大流量冲击还是内存丢失造成的内存过载。如果是大流量冲击造成的过载，应能自动降低业务处理量；如果是内存丢失造成的过载，应能复位单板，但对业务不造成影响。

采用内存丢失的办法实现内存过载的测试步骤：

a） 丢失空闲内存，使内存占用率达到80%，运行一段时间，观察系统的运行状态；

b） 在前面的基础上再丢失10%的内存，运行一段时间，观察系统的运行状态；

c） 把剩余的所有内存全部申请出来丢掉，观察系统的运行情况。

C.3 CPU过载测试

当一个计算机系统的CPU占用率达到80%～100%时，称为CPU过载。CPU过载测试主要是测试系统在高CPU占用率状态下系统的可靠性。

故障产生原因有大流量冲击、算法效率低下、CPU锁死等；故障可能产生的后果有：对实时事件响

应不及时、影响提供业务的能力等。

通过创建一个任务，用循环的方式强制消耗掉一部分 CPU 资源，循环次数可以动态调整，以调整 CPU 占用率来模拟故障产生。

CPU 过载时应能自动启动流控，降低业务流量，同时，系统运行的各种业务不应出错，输入输出的各类消息顺序保持不变。

测试步骤：

a) 创建一个最高优先级的 CPU 过载测试任务，调整循环次数，使 CPU 过载，观察系统的运行情况；

b) 降低过载测试任务的优先级，使原来优先级较高的任务可以正常运行，优先级较低的任务受到影响，运行一段时间，观察系统的运行情况；

c) 重复步骤 b)，使各优先级的任务均受到过载干扰。

C.4 队列过载测试

对于有限长度队列，队列的长度达到最大长度的 80%～100%时，称为有限长度的队列过载。

对于无限长度队列，队列长度远远大于正常运行时队列长度时，例如 10 倍，称为无限长度的队列过载。

队列过载测试主要是为了测试队列缓冲造成的延时对系统的影响，对于有限长度队列，还可以验证队列满的情况下程序运行的情况。

故障产生原因有大流量冲击、处理队列的任务受到阻塞、发生了死锁等；故障可能产生的后果有：系统响应超时、系统内部各模块之间数据状态不一致等。

在测试中可以采用挂起处理队列的任务的办法实现队列过载。

队列过载不应造成系统的业务运行出错，造成的延时也应在可以接受的程度，对于有限长度队列，能正确处理队列溢出的情况，不会产生内存越界或其他错误。

测试步骤：

a) 运行程序，输入各种必要的数据，使被测队列有一个持续的进队列操作。

b) 挂起处理队列的任务。对于有限长度队列，使队列长度达到 80%时解挂该任务，恢复运行；对于无限长度队列，使队列长度达到正常队列长度的 8 倍，然后解挂该任务，观察系统的运行情况。

c) 过一段时间后，再次挂起该任务。对于有限长度队列，使队列长度达到最大长度，此时继续进行入队列的操作，观察系统的运行状态；对于无限长度的队列，使队列长度达到正常长度的 10 倍。然后解挂该任务，运行一段时间，观察这一过程中系统的运行情况。

C.5 资源丢失测试

资源丢失是指资源没有正在被使用，也不在对应的资源管理程序中，即该资源已经不受控了。资源丢失测试的目的是验证在发生资源丢失故障时软件是否能从故障中恢复过来，在故障恢复过程中是否还引入了其他的故障。

故障产生原因有重入、申请出来的资源使用后没有释放、在某些路径下资源没有释放等；故障可能产生的后果如复位、输出错误、软件内部状态错误、业务无法完成或业务性能降低等。

采用把资源申请出来后不释放的方法来模拟故障产生。

对于一般系统，要求在资源耗尽以后能复位，这时要注意耗尽型资源和非耗尽型资源的差异，复位条件的检测要特殊设计；对于高可用系统，要求能恢复资源丢失造成的影响，并且不影响业务的正常运作。

测试步骤：

a） 申请一个资源；

b） 经过一段时间后，检查软件是否能发现出现资源丢失，观察软件是否采取了什么恢复措施，该措施是否有效；

c） 重复申请一批资源但不释放，过一段时间后，检查软件是否能发现出现资源丢失故障，故障软件是否采取了相应的措施，以及该措施是否有效。

C.6 释放错误资源测试

释放错误资源测试就是把伪造的非法资源通过资源释放函数试图加入到空闲资源池的测试过程。释放错误资源测试的目的是为了验证资源管理程序是否对非法资源有合理的保护措施。

故障产生原因有资源在使用过程中修改了资源的标记属性，如地址指针、ID 号等；故障可能产生的后果有：资源管理程序遭到破坏、资源丢失、业务中断等。

采用伪造一个非法的资源并用资源释放函数释放该非法资源，例如对于内存，可以先申请出一块内存，然后把地址加 1 再释放，来模拟故障。

测试过程中，返回了出错信息，资源管理程序内部应没有真正增加这些非法的资源。

测试步骤：

a） 伪造非法资源；

b） 用资源释放函数释放这些非法资源；

c） 观察资源管理程序是否返回出错，检查资源管理程序的内部数据看是否把非法的资源加进去了。

C.7 重复释放资源测试

重复释放资源是指同一个资源被释放两次或两次以上。重复释放资源测试的目的是为了验证资源管理程序是否具有处理资源重复释放故障的能力。

故障产生原因一般是编程错误。故障可能产生的后果有：资源管理程序内部存在多个同一资源、软件提供的业务出现错误等。

采用修改代码，把申请成功的资源释放 2 次来模拟故障。

测试过程中，第 2 次或以后的释放均返回失败，系统应给予告警或提示。

测试步骤：

a） 把申请成功的资源释放 2 次；

b） 检查是否 2 次都释放成功；

c） 检查资源管理程序内部的数据记录中是否存在 2 个相同的资源。

C.8 资源释放时机错误的测试

资源释放时机错误是指资源被提前释放了，即资源被释放后仍然在继续使用这个资源。本测试的目的是检查资源被提前释放后对系统造成的影响是否可以接受。

故障产生原因一般是资源被提前释放；故障可能产生的后果有：数据遭到破坏、资源遗漏、提供的业务出现错误等。

通过修改代码，把释放资源的操作提前来模拟故障。

对于一般系统，如果资源提前释放造成了更严重的故障，系统应能复位；对于高可用系统，它能检测

到资源提前释放造成的故障，并对其影响能恢复。

测试步骤：

a） 提前释放资源；

b） 观察对系统的影响。

C.9 资源申请失败测试

测试目的是验证程序对资源申请失败的处理是否正确。

故障产生原因可能有：资源已经用完了、资源管理程序出现了故障、编程错误等。故障可能产生的后果有：业务没有完成、访问了非法资源等。

采用修改代码，使资源申请函数在需要的时候可以产生一个异常返回值来模拟故障。

测试过程中，软件的主要业务不应出现错误。

测试步骤：

a） 修改代码，在代码中增加一个路径控制开关。当开关关闭时，资源可以正常申请；当开关打开时，申请的资源都不成功。

b） 在系统正常运行后，把上述的控制开关打开，加入故障。

c） 观察故障对系统的影响。

C.10 临界资源重入测试

由两个或两个以上的任务共享的资源称为临界资源，比如多个任务间共享的内存、全局变量、物理端口等。测试目的是验证系统是否具有临界资源重入故障的处理机制以及处理的结果是否符合需求。

故障产生原因可能有：软件设计时没有考虑临界资源问题、临界资源太多，编码时只对部分临界资源进行了保护等。故障可能产生的后果有：软件内部数据状态混乱、业务中断、复位等。

测试时把已有的临界资源保护措施去掉，强制出现重入问题，来模拟故障。

对于一般系统，如果内部数据遭到严重破坏，系统应能复位重新启动；对于高可用系统，软件应能处理这一情况而不至于出现失效。

测试步骤：

a） 删除代码中的临界资源保护的措施；

b） 编译运行修改后的软件；

c） 观察系统的出错情况。

C.11 栈溢出测试

栈溢出是指栈指针越过了栈的边界。测试目的是检查程序对栈溢出是否有预防措施，当发生栈溢出后，程序是否有相应的检测、恢复过程。

故障产生原因可能有：使用了大数组作为局部变量、使用了大数组作为形式参数、递归调用层次过深、函数调用的层次过深等。故障可能产生的后果有：内存越界、程序异常等。

测试时的故障产生方法：

a） 使用定义大数组的局部变量方法实现栈溢出；

b） 使用内存拷贝的方法使栈越界；

c） 使用函数递归调用的方法使栈越界。

使用方法 b)和方法 c)时，某些操作系统可能把栈指针回卷到栈底。

对于一般系统，栈越界后如果造成了更严重的故障，要求能复位系统；对于高可用系统，栈越界后不应对系统提供的业务造成影响，一个可行的办法是采用硬件监视的方法在栈溢出时产生一个异常，防止栈溢出的产生。

测试步骤：

a） 在程序正常运行时，使用上面描述的方法使栈越界；

b） 观察栈越界后的结果。

C.12 任务强制挂起测试

任务强制挂起测试的目的是为了验证软件是否具有任务被异常挂起的检测功能，以及测试任务被异常挂起后软件的整个恢复过程是否正确，是否会引入其他的故障。

故障产生原因一般为程序设计问题。故障可能产生的后果是：被挂起的任务完成的功能无法完成，还可能引起进一步的其他故障。

测试时的故障产生方法：

a） 通过控制台调用挂起函数把被测任务强制挂起；

b） 修改代码，嵌入挂起任务的代码，重新编译软件；嵌入代码时，注意嵌入的代码应在整个软件正常运行一段时间后才能执行。

测试过程中，被挂起的任务能在一个合理的时间内解挂，解挂后各个任务之间的内部状态应保持一致，对业务造成的影响尽量小。

测试步骤：

a） 通过控制台或嵌入代码使被测任务强制挂起；

b） 观察系统的运行情况。

C.13 任务强制删除测试

任务强制删除测试的目的是为了验证软件是否具有任务被异常删除的检测功能，以及测试任务被异常删除后软件的整个恢复过程是否正确，是否会引入其他的故障。

故障产生原因一般为程序设计问题。故障可能产生的后果是：被删除的任务完成的功能无法完成，还可能引起进一步的其他故障。

测试时的故障产生方法：

a） 通过控制台调用任务删除函数把被测任务强制删除。

b） 修改代码，嵌入删除任务的代码，重新编译软件。嵌入代码时，应注意嵌入的代码应在整个软件正常运行一段时间后才能执行。

对于一般系统，要求能检测到任务被删除，并复位系统；对于高可用系统，要求被删除的任务能在一个比较短的时间内恢复，并对业务造成的影响尽量小，任务恢复后，各个任务之间的内部状态应保持协调一致。

测试步骤：

a） 通过控制台或嵌入代码使被测任务强制删除；

b） 观察系统的运行情况。

C.14 插入死循环测试

插入死循环是指在任务或中断服务程序中加入死循环代码。插入死循环测试的目的是为了验证任

务或中断进入死循环后软件的处理是否符合要求。

故障产生原因可能有：进入了死循环、发生了不能终结的函数递归调用、发生了函数的循环调用等。故障可能产生的后果是由任务或中断完成的功能无法完成。

采用在软件中加入死循环代码来模拟故障。

对于一般系统，要求能检测到发生死循环故障，并能复位，复位后系统能正常运行；对于高可用系统，要求能检测到发生死循环故障，并要求能重新启动或重新初始化该任务，重启后各个任务的内部状态能保持一致。

测试步骤：

a) 在被测任务或中断中加入死循环代码，重新编译程序；

b) 构造测试输入数据，使程序能执行到插入的代码；

c) 观察软件进入死循环后的处理情况。

C.15 任务延时测试

任务延时测试是指在被测任务中加入延时代码延长任务执行时间的一种测试过程。任务延时测试的目的是为了了解任务延时后对系统造成的影响是否满足要求。

故障产生原因可能有：程序在处理某些任务时处理量大、花的时间很长；程序在某些路径下需要等待某一事件，造成执行时间有一定程度的不确定性等。故障可能产生的后果有：业务性能下降、系统对某些事件的响应速度变慢、模块之间的消息时序上出现错误。

测试时的故障产生方法：

a) 在任务中加入空循环作延时处理；

b) 调用任务延时函数。

测试过程中，系统不导致严重故障或失效。当延时干扰取消后，程序应能恢复正常状态，各模块之间也不应出现时序上的混乱。

测试步骤：

a) 在任务中加入空循环作延时处理，重新编译程序；

b) 构造测试数据，激活被测任务；

c) 观察系统的性能是否出现明显下降，是否出现其他的故障，任务之间的交互时序是否出现变化以及这个变化带来了什么影响。

C.16 死锁测试

若一个进程集合中的每一个进程都在等待只能由本集合中的另一个进程才能引发的事件，则这种情况被视为死锁。死锁测试的目的是为了验证程序是否具有处理死锁的能力。

死锁的4个条件：

a) 互斥条件，每一资源或者被分配给一个进程，或者空闲；

b) 保持和等待条件，已分配到了一些资源的进程可以申请新的资源；

c) 非剥夺条件，已分配给一进程的资源不可剥夺，只能被占有它的进程显式地释放；

d) 循环等待条件，系统必然有一条由两个或两个以上的进程组成的循环链，链中的每一个进程都在等待相邻进程占用的资源。

故障可能产生的后果可能是：所有涉及的任务均被挂起，由这些任务完成的功能全部中断。

按照上述4个条件，任务A申请资源1和资源2，任务B申请资源2和资源1，为了能进入死锁，在申请完第1个资源后增加一个同步操作，使两个任务进入死锁。

对于一般系统，程序能检测到软件发生了故障并能复位系统；对于高可用系统，要求能检测到软件发生了故障而且在短时间内能正确恢复，并对业务不造成重大影响。

测试步骤：

a) 选择 2 个任务，按照下列步骤增加相应的代码；

b) 任务 A：申请信号量 1—申请信号量 3—释放信号量 1—等待信号量 2—申请信号量 4；

c) 任务 B：申请信号量 2—申请信号量 4—释放信号量 2—等待信号量 1—申请信号量 3；

d) 重新编译程序并运行，观察程序是否能检测到死锁并采取相应的措施。

C.17 频繁中断测试

当两个中断的间隔时间小于中断服务例程的处理时间时，称为发生了频繁中断。测试目的是验证软件在发生频繁中断情况下的运行是否稳定可靠，验证中断等待、中断嵌套功能是否达到设计要求。

故障产生原因可能有：中断服务例程执行时间过长、同时发生了多个中断、中断发生得太快等。故障可能产生的后果有：中断丢失、中断挂起、中断异常等，由此带来与该中断相关的业务可能受到影响。

采用在中断引脚产生频繁中断信号来模拟故障产生。

测试过程中，当中断队列不满时，不应发生中断丢失，同时不应发生中断挂起或异常；当取消频繁中断后，程序应能恢复正常运行时的状态。

测试步骤：

a) 测试出被测中断的一次正常中断的执行时间（t_1）；

b) 找到中断源对应的引脚，在引脚上插入中断信号发生器的输出信号，调节中断间隔时间（t_2），使其小于中断服务程序的执行时间（$t_1 > t_2 > 0.5t_1$），观察系统运行情况；

c) 继续降低中断信号的时间间隔，使 $0.5t_1 > t_2 > 0.25t_1$，即在处理中断服务程序时发生 2 个中断请求，观察系统运行情况；

d) 在中断服务程序中增加中断计数变量，重新编译运行，正常运行后，利用中断信号发生器在很短的时间内快速发送 1 000 个中断，观察系统运行情况。

C.18 重复中断测试

重复中断是指对同一次中断作了多次响应，即执行了多次的中断服务程序。重复中断测试的目的是为了测试重复执行中断服务程序对系统造成的影响。

故障产生原因可能是中断控制器发生了故障。故障可能产生的后果有：重复执行由该中断触发的操作、对于数据收发出现重包等。

采用在响应中断时，调用多次中断服务程序来模拟故障产生。

测试过程中，重复执行中断的服务程序不应对系统的业务造成影响。

测试步骤：

a) 修改代码，在响应中断时，调用多次中断服务程序，重新编译程序；

b) 按照实际条件触发该中断，观察程序的运行情况。

C.19 中断丢失测试

中断的响应次数比中断的触发次数少的情况称为中断丢失。测试目的是测试中断丢失对系统造成的影响，验证在发生中断丢失时，系统是否发生重大的功能缺陷。

故障产生原因可能是中断控制器故障。故障可能产生的后果有：对外部事件的响应不完整造成系

统的严重功能失效、对于数据收发造成丢包。

采用在中断服务程序中有选择性或随机地直接返回的方法来模拟测试。

测试过程中应不发生重大的功能失效，对后续的中断仍能正常处理。

测试步骤：

a) 修改中断服务程序，在入口处有选择性或随机地直接返回，重新编译程序；

b) 按照实际条件触发该中断，观察系统的运行情况。

C.20 中断挂起测试

中断挂起是指中断控制器由于某些原因造成不能响应新中断。中断挂起测试的目的是为了测试系统不能响应部分或全部中断后系统的运行情况。

故障产生原因可能是中断控制器发生故障。故障可能产生的后果是无法响应新产生的中断，因此由这些中断完成的功能均无法完成。

采用修改中断使能位或中断屏蔽位关闭被测中断来模拟测试。

程序应能检测到中断出了故障，采取相应的措施，并能在一个较短的时间内恢复正常。

测试步骤：

a) 运行程序，使被测程序正常运行，然后清除被测中断的使能位，关闭中断；

b) 按照实际条件触发该中断，观察系统的运行情况。

C.21 中断服务例程延时测试

测试目的是为了测试中断处理时间的延长对系统造成的影响。故障产生原因可能有：中断服务例程中执行到需要等待某些条件的语句、中断服务例程中某些路径很长导致执行时间大幅增长等。

故障可能产生的后果有：业务性能下降、系统对某些事件的响应速度变慢、中断与任务之间的输出时序上出现错误等。

采用在中断服务例程中加入空循环模拟延时来测试。

测试过程中，中断与任务之间的输出没有出现时序上的问题，业务性能不应出现严重下降，提供的业务不应出现错误。

测试步骤：

a) 在中断服务例程中加入空循环，重新编译程序；

b) 按照实际条件触发中断，使被测的中断服务例程能被执行；

c) 观察系统的运行情况。

C.22 单板启动过程中触发所有中断测试

测试目的是验证单板启动过程中是否把某些中断打开了。故障产生原因可能是单板启动过程中被触发中断。故障可能产生的后果是程序会异常运行。

采用模拟实际情况在单板启动过程中触发单板的每一个中断源来测试。

测试中程序应不异常，在启动过程中中断不起作用。

测试步骤：

a) 选择增加测试代码的位置。在主函数的初始化部分(主循环之前的代码)选择一个开中断语句之后的位置作为增加测试代码的位置。不管在代码中是否真正存在，主函数开始的地方总是默认有一个开中断语句。

b) 在步骤 a)中选择的位置增加一条死循环语句,死循环内部增加一个计数器或者增加一个闪灯的操作,重新编译程序。

c) 在程序进入该死循环后,逐一触发所有的中断源,检查是否异常,即检查计数器值是否在增加或者灯是否在继续闪亮。

d) 选择其他开中断语句之后的位置,重复步骤 b)和步骤 c),直到所有开中断语句均被测试过。

C.23 触发未使用中断测试

指程序运行过程中触发了没有使用的中断。测试目的是验证程序是否关闭了未使用中断。

故障产生原因是没有把未使用中断关闭或进行处理。故障可能产生的后果有:发生程序异常、影响程序的性能、未知故障等。

采用模拟实际条件触发未使用中断来测试。

程序不应异常,业务能正常处理,触发未使用中断对系统不造成任何影响,触发未使用中断时不应对中断标志位造成影响。

测试步骤:

a) 屏蔽所有未使用的中断源;

b) 运行程序,模拟实际条件触发所有未使用中断;

c) 观察程序运行状态,检查是否发生异常;

d) 检查中断标志位是否被置位。

C.24 接口数据长度错误测试

长度错误是指消息中各个长度域中的值发生错误。测试目的是测试程序对错误长度的消息的处理情况。

故障产生原因可能有:发送的消息长度有错、消息在传输过程中发生了错误等。故障可能产生的后果有:内存越界、使用了错误的数据、死机等。

采用发送错误长度的消息来模拟故障。

程序应能忽略消息内容并释放这些非法消息,而且程序的状态不应挂起在某个状态,当长时间收不到期望接收的消息时,应能恢复到初始状态。

测试步骤:

a) 修改代码,可以从后台直接发送十六进制格式的消息,重新编译程序并运行;

b) 发送 0 长度消息,即长度域的值是 0,观察程序的处理情况;

c) 发送超长消息,即长度值超过允许的最大值,观察程序的运行情况;

d) 发送消息内容中的长度域,使其值在允许范围之外,观察程序的运行情况。

C.25 接口数据类型错误测试

接口数据类型错误是指接口的消息类型发生的错误。测试目的是测试程序对错误类型的消息的处理情况。

故障产生原因可能有:发送的消息类型有错、消息在传输过程中发生了错误等。故障可能产生的后果有:使用了错误的数据、状态机故障等。

采用发送错误类型的消息来模拟故障。

程序应能忽略消息内容并释放这些非法消息,而且程序的状态不应挂起在某个状态,当长时间收不

到期望接收的消息时，应能恢复到初始状态。

测试步骤：

a） 修改代码，可以从后台直接发送十六进制格式的消息，重新编译程序并运行；

b） 发送消息类型为允许范围外的消息，观察程序的处理情况。

C.26 数据错误测试

数据错误是指传送了错误的消息数据。测试目的是测试程序对带有错误数据的消息的处理情况。

故障产生原因可能有发送的消息类型有错、消息在传输过程中发生了错误等。故障可能产生的后果有：软件处理了错误的数据、状态机故障等。

采用发送错误数据的消息来模拟故障。

程序应能忽略消息内容并释放这些非法消息，而且程序的状态不应挂起在某个状态，当长时间收不到期望接收的消息时，应能恢复到初始状态。

测试步骤：

a） 修改代码，可以从后台直接发送十六进制格式的消息，重新编译程序并运行；

b） 发送消息数据为允许范围外的消息，观察程序的处理情况。

C.27 消息数据的一致性错误测试

消息数据的一致性错误是指消息中的多个数据之间存在矛盾。测试目的是测试程序是否能区分出存在一致性错误的消息数据并给予正确的处理。

故障产生的原因有内存越界、算法错误、协议缺陷等。故障可能产生的后果有模块间配合出现错误、模块失效等。

采用强制修改消息数据来模拟故障。

程序应能忽略消息内容并释放这些非法消息，而且程序的状态不应挂起在某个状态，当长时间收不到期望接收的消息时，应能恢复到初始状态。

测试步骤：

a） 修改代码，可以从后台直接发送十六进制格式的消息，重新编译程序并运行；

b） 发送存在一致性错误数据的消息，观察程序的处理情况。

C.28 消息丢包测试

丢包是指发送模块发送的消息包没有到达该消息包对应的接收模块的现象。测试目的是验证丢包对系统造成的影响，是否导致系统出现业务方面的错误。

故障产生的原因可能有消息发送模块的发送队列满、发送模块出现故障、通信线路出现故障（如误码、链路闪断等）、底层驱动程序出现故障、接收队列满、协议缺陷等。故障可能产生的后果有影响系统业务、控制等方面的功能，造成模块或单板之间状态不一致，系统空转但不能处理业务等。

采用在数据流中强制释放消息包来模拟故障。

丢包不应导致系统内部各模块之间或系统和与之连接的外部系统之间状态的不一致，不应导致业务或控制方面出现失效。

测试步骤：

a） 在程序中加入适当的代码，从原消息流中截取通过的所有消息，然后按一定的比率强制释放这些消息，比率可以动态调整，重新编译程序；

b) 运行程序，构造一些输入，使程序内部有消息流动，从小到大然后再从大到小调整丢包比率，观察系统的运行情况；

c) 关闭丢包功能，消除丢包故障，观察系统的运行情况，检查系统的各模块状态是否一致，是否能恢复回正常运行状态。

C.29 软误码测试

由于软件错误造成传输数据被修改的过程称为软误码。软误码测试的目的是为了测试错误的消息数据在系统上层程序中的处理情况。

故障产生的原因可能有发送消息的模块本身出错导致发送了一个错误的消息，发送、接收消息的子系统出现错误如内存越界、传输链路上发生了硬件错误、软件协议缺陷等。故障现象可能有内存越界、配置数据错误、执行了错误的命令、模块之间状态不一致等。

采用消息中的部分数据用其他的值代替，或把其中的某些位取反来模拟故障。

测试中涉及的模块有足够的保护，使错误的消息得不到执行，内部各模块的状态也保持一致，不出现业务上的故障。

测试步骤：

a) 增加测试代码，截取消息流中的消息包，按照一定的比率强制修改消息数据，重新编译程序；

b) 构造适当的输入，使步骤 a)中的消息流中有消息经过；

c) 观察系统的运行情况，检查系统是否出现告警、内存越界、死机，检查内部的各相关模块状态是否一致；

d) 关闭误码程序，观察系统的运行情况，检查系统是否可以恢复正常运行。

C.30 消息延时测试

延长消息传输时间的过程称为消息延时。消息延时测试的目的是为了测试消息超时到达接收模块后系统运行的情况。

故障产生的原因可能有传输通道上发生阻塞、底层驱动程序效率不够高或者发生了故障、系统非常繁忙等。故障现象可能有接收模块仍然把超时消息按正常消息处理、系统内部模块之间状态不一致等。

可以采用从消息通路上截取消息流，把消息发送到延时队列中去，经过一段时间后再把延时队列中的消息发送出去的方法，来模拟故障。

消息超时不应对程序造成严重影响，超时后程序能退回到一个恰当的状态重新开始，当超时扰动结束后，程序应能恢复正常状态。

测试步骤：

a) 修改代码，增加适当的程序，从消息通路上截取消息，把这些消息发送到延时队列中去，经过一段时间后，再把这些延时队列中的消息按原来的时间间隔发送出去，相当于每一个消息都延长了一个相同的时间。延时的时间可以调节。

b) 重新编译程序，构造测试输入数据，使被测的消息流中有消息经过。

c) 调节消息的延时时间，观察系统的运行情况。

C.31 消息重包测试

重包是指接收模块接收到多个相同消息的现象。测试目的是为了测试在接收端接收到多个相同消息后系统处理的情况。

故障现象有命令重复执行、模块间内部数据状态不一致等。采用在消息流通路上复制某些消息模拟重包的产生来测试。

测试中系统应不出错。

测试步骤：

a) 在消息流通路上选择适当的位置，增加测试代码，按照某一规则复制通过的消息，如按照某一比率或按照某一特征进行复制；

b) 重新编译程序，构造测试输入数据，使被测的消息流中有消息经过；

c) 观察程序的运行情况。

C.32 消息乱序测试

乱序是指多个消息的接收顺序和发送顺序相比发生了混乱。这项测试目的是测试消息乱序对系统造成的影响。

故障产生的原因如底层驱动程序存在缺陷、多个通信路径的速率不一样等。故障现象有逻辑错误、各操作的先后顺序混乱、内部数据状态混乱等。

测试时的故障产生方法：

消息顺序扰动原理如图 C.1 所示。初始化时，整个扰动池为空，当消息进入时，先计算插入位置。插入位置分两步计算，第一步，如果在随机工作方式下，按照扰动概率计算是否需要扰动；或者在部分扰动工作方式下，则判断是否符合扰动条件，如果符合，则也需要扰动。如果不需要扰动，则插入位置为 CurrentPos。如果需要扰动，进行第二步：在消息提前区和消息推后区随机选取一个位置为插入位置。提前区和推后区的大小由最大扰序幅度确定。如果计算出的位置已经有消息在位，则再就近找一个空位置填入，同时读出位置如果不空则把对应消息发送到下一个扰动处理模块，然后把该位置清空，最后把 CurrentPos 和 OutPos 向后移二个位置。每次移动二个位置是为了改善在强扰动下的扰动性能。

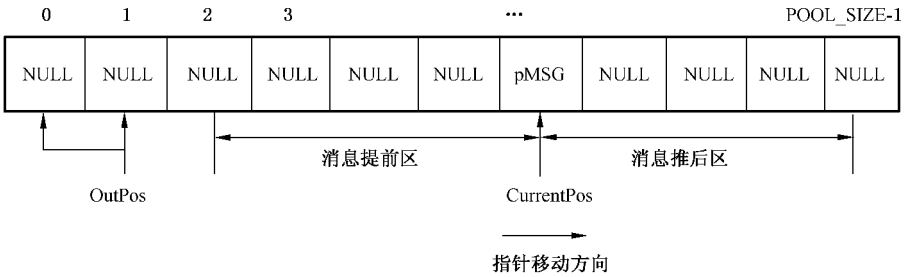

图 C.1 消息顺序扰动原理示意图

对于有先后顺序关系的一系列操作应设计相应的状态机来控制，避免多个操作的顺序错误。因此，消息顺序扰动不应对程序造成出错，程序的各种业务应能正常完成，当消息扰动停止后，系统内部的状态应能恢复到扰动前的正常状态。

测试步骤：

a) 在消息流通路上选择适当的位置，增加测试代码，按照上述方法实现消息的顺序扰动；

b) 重新编译程序，构造测试输入数据，使被测的消息流中有消息经过；

c) 观察程序的运行情况；

d) 停止消息的顺序扰动，观察系统的运行情况。

C.33 突发大流量测试

突发大流量是指在短时间内发送大量的消息或数据。这项测试目的是验证系统受到突发大流量冲

击时系统的稳定性与可靠性。

故障现象有系统复位、系统挂起、不能响应外部事件等。采用模拟实际情况在短时间内发送大量的消息或数据来测试。

测试中系统应不复位，业务能正常处理，在流量降低后，系统没有任何异常情况。

测试步骤：

a) 搭建好工具或增加适当的测试代码，使得可以向被测系统随意发送任意流量的消息或数据。

b) 运行程序，配置被测链路，为了达到最大的处理量，尽量把所有可以输入系统的端口都打开并给予对应的输入。

c) 调整流量，使流量达到最大，运行一段时间；再把流量调整到较低的水平，运行一段时间；然后又把流量调整到最大。如此反复多次，观察系统的运行情况。

C.34 长时间大流量测试

长时间测试是指持续时间远远大于正常测试时间的测试，分为 12 h、24 h、48 h、72 h 四档，根据实际情况决定选取哪一个档次的测试。这项测试的目的是验证在长时间大流量条件下系统的稳定性。

故障现象有系统崩溃、业务性能严重下降等。采用模拟实际情况在长时间内发送大量的消息或数据来测试。

测试中系统应不复位，业务能正常处理，在流量降低后，系统没有任何异常情况。

测试步骤：

a) 搭建好工具或增加适当的测试代码，使得可以向被测系统随意发送任意流量的消息或数据。

b) 运行程序，配置被测链路，为了达到最大的处理量，尽量把所有可以输入系统的端口都打开并给予对应的输入。

c) 调整流量，使流量达到最大，长时间运行，观察系统的运行情况。当运行时间达到规定的时间后，降低流量到 0，观察系统的运行情况。

参 考 文 献

[1] John D. Musa. 软件可靠性工程. 韩柯，译. 北京：机械工业出版社，2003

[2] 孙志安，裴晓黎，宋昕，戴忠健. 软件可靠性工程. 北京航空航天大学出版社，2009