GJB

中华人民共和国国家军用标准

FL 0114

GJB/Z 142－2004

军用软件安全性分析指南

Guide for military software safety analysis

2004－09－20 发布

2005－01－01 实施

中国人民解放军总装备部 批 准

目　次

前　言

本指导性技术文件的附录A、B、C、D、E、F是资料性附录。

本指导性技术文件由中国人民解放军总装备部司令部提出。

本指导性技术文件起草单位：总装备部装备论证研究中心。

本指导性技术文件主要起草人：张鲁峰、张剑波、许博义、赵　刚、黄敏桓。

军用软件安全性分析指南

1 范围

本指导性技术文件给出了在软件生存周期中实施软件安全性分析的指南。

本指导性技术文件适用于安全相关软件的获取、供应、开发、运行和维护。本指导性技术文件中固件被当作软件对待。本指导性技术文件不适用于现货软件，除非它包含在交付的产品中。本指导性技术文件仅适用于软件，标准中涉及的系统安全性分析仅被用来确定系统中包含的软件的安全性需求。

本指导性技术文件可被安全相关软件的需方、供方、开发者、维护者以及独立的评价者使用。

2 引用文件

下列文件中的有关条款通过引用而成为本指导性技术文件的条款。凡注日期或版次的引用文件，其后的任何修改单(不包括勘误的内容)或修订版本都不适用于本指导性技术文件，但提倡使用本指导性技术文件的各方探讨使用其最新版本的可能性。凡不注日期或版次的引用文件，其最新版本适用于本指导性技术文件。

GB/T 8566－2001 信息技术 软件生存周期过程

GB/T 11457 信息技术 软件工程术语

GB/T 18492－2001 信息技术 系统及软件完整性级别

GJB 900－90 系统安全性通用大纲

3 术语、定义和缩略语

3.1 定义

GB/T 11457 中确立的以及下列术语和定义适用于本指导性技术文件。

3.1.1 危险失效 dangerous failure

可导致安全相关系统进入危险状态或无法工作状态的失效。

3.1.2 受控设备 (EUC) equipment under control

用于制造、加工、运送、医疗或其它活动的装备、设备、仪器或机组。

3.1.3 EUC 控制系统 EUC control system

根据过程中或操作者输入的信号产生输出信号，以使 EUC 按照要求动作的系统。

3.1.4 影响分析 impact analysis

确定系统中一个功能或部件的改变将对该系统中其它功能或部件以及其它系统产生何种影响的活动。

3.1.5 事故 mishap

造成人员伤亡、职业病、设备损坏或财产损失的一个或一系列意外事件。

3.1.6 运行模式 mode of operation

安全相关系统使用的方式。根据安全相关系统运行的频率，运行模式分为下列两种：

a) 低频率模式：安全相关系统的运行频率不大于每年一次且不大于校验测试频率的两倍；

b) 高频率或连续模式：安全相关系统的运行频率大于每年一次或大于校验测试频率的两倍。包括为保持安全性施行连续控制的安全相关系统。

3.1.7 必要风险降低 necessary risk reduction

为了使风险水平不超过可容忍风险水平，采用 PE 安全相关系统，其它技术安全相关系统和外部安

全设施达到的必需的风险降低。

3.1.8 可编程电子系统 programmable electronic system

基于一个或多个可编程电子装置的，用于控制、防护或监视的系统，它包括系统中所有的要素，比如电源，传感器和其它输入装置，数据高速通路和其它通信路径，以及执行器和其它输出装置。

3.1.9 校验测试 proof test

用以发现安全相关系统失效的周期性测试，以便如有必要，系统可以回复到初始状态或尽可能接近初始状态。

3.1.10 安全性 safety

不发生事故的能力。

3.1.11 安全性评估 safety assessment

为了判定软件获得的安全性而进行的基于证据的调查。

3.1.12 安全功能 safety function

针对特定的危险事件，为达到或保持 EUC 的安全状态而实现的功能。

3.1.13 安全完整性 safety integrity

在规定的条件下、规定的时间内，安全相关系统成功实现所要求的安全功能的可能性。

3.1.14 安全完整性级别(SIL) safety integrity level

一种离散的等级(四种可能等级之一种)，用于规定分配给安全相关系统的安全功能的安全完整性要求。安全完整性级别 4 是最高的，安全完整性级别 1 是最低的。安全完整性级别决定了安全功能的目标失效测度。

3.1.15 安全相关软件 safety related software

在安全相关系统中，用于实现安全功能的软件。

3.1.16 安全相关系统 safety-related system

用于下列两目的之一的系统：

a) 执行要求的安全功能以达到或保持 EUC 的安全状态；

b) 使安全功能达到必要的安全完整性。

安全相关系统可采用广泛的技术实现，本指导性技术文件中将基于可编程电子技术的安全相关系统称为 PE 安全相关系统。

3.1.17 软件安全性 software safety

软件具有的不导致事故发生的能力。

3.1.18 软件安全性分析 software safety analysis

对和软件安全性相关的特定信息进行的系统而有序的获取和评价过程。

3.1.19 软件安全完整性 software safety integrity

软件在规定条件下和规定时间中，在系统中成功完成其安全功能的可能性的度量。

3.1.20 软件安全完整性级别 software safety integrity level

一种离散的级别(四种可能级别之一)，用于规定在安全相关系统中软件的安全完整性。

3.1.21 系统性失效 systematic failure

与确定的原因和确定的方式有关的失效。只有对设计或制造过程、操作程序、文档或其它相关因素进行调整后，才有可能排除这种失效。

3.1.22 目标失效测度 target failure measure

根据安全完整性的要求，需达到的危险失效的概率。

目标失效测度按照下列两种形式之一进行定义：

a) 平均每次执行其设计功能时发生失效的概率(对于低频率运行模式)，或

b) 危险失效平均每小时发生的概率(对于高频率或连续运行模式)。

注 1：虽然表 1 和表 2 对于安全完整性级别的规定是一种数值测度，但可能定量地预测软件所有方面的安全完整性级别数值并不可行，所以不排斥采用定性的技术、手段来预测。

注 2：较简单的系统可能超过表中给出的对于 SIL4 的要求，但是以目前的技术条件，表中 SIL4 的数据对于相对复杂的系统(比如 PE 安全相关系统)是一个极限。

表 1　低频率运行模式软件安全功能的目标失效测度要求

安全完整性级别	低频率运行模式(平均每次运行时失效概率)
4	$\geqslant 10^{-5}$ 至 $<10^{-4}$
3	$\geqslant 10^{-4}$ 至 $<10^{-3}$
2	$\geqslant 10^{-3}$ 至 $<10^{-2}$
1	$\geqslant 10^{-2}$ 至 $<10^{-1}$

表 2　高频率或连续运行模式软件安全功能的目标失效测度要求

安全完整性级别	高频率或连续运行模式(平均每小时危险失效发生的概率)
4	$\geqslant 10^{-9}$ 至 $<10^{-8}$
3	$\geqslant 10^{-8}$ 至 $<10^{-7}$
2	$\geqslant 10^{-7}$ 至 $<10^{-6}$
1	$\geqslant 10^{-6}$ 至 $<10^{-5}$

3.1.23　可容忍风险　tolerable risk

基于当前社会价值观，在给定的环境下可以接收的风险。

3.2　缩略语

下列缩略语适用于本指导性技术文件。

EUC(Equipment Under Control)——受控设备

PES(Programmable Electronic System)——可编程电子系统

PLC(Programmable Logic Controller)——可编程逻辑控制器

SIL(Safety Integrity Level)——安全完整性级别

4　概述

4.1　本指导性技术文件的结构

软件安全性分析的目的是通过获取和评估软件安全性相关信息，保证系统安全性质量。软件安全性分析主要包括如下任务：

a)　软件需求安全性分析；

b)　软件结构设计安全性分析；

c)　软件支持工具和编程语言安全性分析；

d)　软件详细设计安全性分析；

e)　软件编码安全性分析；

f)　软件测试安全性分析；

g)　软件变更安全性分析。

软件安全性分析任务包含于软件生存周期过程的若干活动中，是针对软件的安全性质量，对于这些活动的补充。

注：本指导性技术文件采用了 GB/T 8566－2001 给出的软件生存周期模型。

表 3 列出了本指导性技术文件的主要结构，同时给出了本指导性技术文件内容与 GB/T 8566－2001 中定义的软件生存周期过程和活动的对应关系。

表 3　本指导性技术文件的结构

章条	题目	安全性分析所属的软件生存周期活动	所属生存周期过程	过程分类
5	软件安全性分析的组织	—	—	生存周期组织过程
5.1	管理	—	管理	
5.2	基础设施	—	基础设施	
5.3	改进	—	改进	
5.4	培训	—	培训	
6	软件安全性分析的支持	—	—	生存周期支持过程
6.1	文档编制	—	文档编制	
6.2	配置管理	—	配置管理	
6.3	质量过程	—	质量保证； 验证； 确认； 联合评审； 审核	
6.4	问题解决	—	问题解决	
7	软件安全性分析准备	—	获取； 供应	生存周期基本过程
		系统需求分析； 系统结构设计	开发	
8	软件安全性分析任务	—	开发	
8.1	软件需求安全性分析	软件需求分析		
8.2	软件结构设计安全性分析	软件结构设计		
8.3	软件支持工具和编程语言安全性分析	—		
8.4	软件详细设计安全性分析	软件详细设计		
8.5	软件编码安全性分析	软件编码和测试		
8.6	软件测试安全性分析	软件编码和测试； 软件集成； 软件合格性测试； 系统集成； 系统合格性测试		
8.7	软件变更安全性分析	—	开发； 运行； 维护	
9	软件安全性分析报告	评审和评价	供应	

4.2　标准的应用原则

4.2.1　软件安全性分析是系统安全性分析的组成部分，应与系统安全性分析密切结合，协调一致。

注：安全相关软件存在于 PE 安全相关系统中。图 1 表示本指导性技术文件中安全相关系统和软件的开发过程包含的活动。在开发过程中有两类活动，分别是系统开发活动和软件开发活动。图 1 还显示了系统开发活动和软件开发活动的对称及相关性，设计活动与测试活动的对称及相关性。

4.2.2　本指导性技术文件中的软件安全性分析任务主要面向开发过程提出，但是软件安全性分析任务可被软件生存周期中涉及的多方执行。当需方、供方、维护者和独立的评价者在其过程中开展软件安全性分析时，其角色转换为开发者，宜采用开发过程中的软件安全性分析任务。

注：例如，在获取过程和供应过程中开发系统需求，维护过程中进行问题和修改分析，以及验证和确认等质量过程中，都可能采用软件安全性分析。

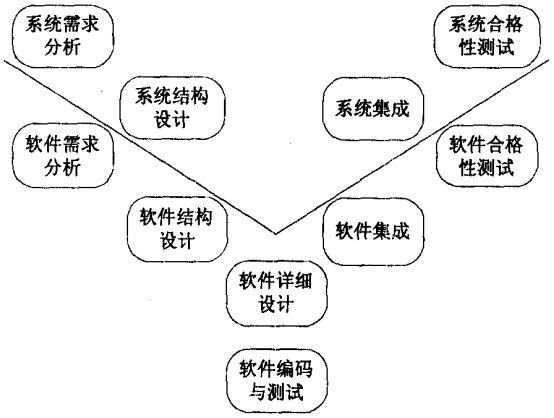


图1　开发过程的活动示意图

4.2.3　实施软件安全性分析的组织应采用相应的组织过程和支持过程来保障软件安全性分析。

4.2.4　维护危险和风险分析的信息是后续软件安全性分析的重要基础。在初步危险和风险分析之后的系统和软件生存周期过程中，应保存并在必要的情况下更新危险和风险信息。

4.2.5　安全相关软件在软件生存周期过程中应尽早地开展相应层次的软件安全性分析，发现安全性相关问题，以利于及时地并以较少的代价解决问题。

4.2.6　软件安全性分析在软件开发过程中是迭代进行的。随着项目开发的细化，软件安全性分析也应不断地深入完善。如果出现与较早生存周期活动有关的变化，则应对对该变化对安全性的影响进行评估，必要时重复与较早的生存周期活动和随后的活动相关的安全性分析。

4.2.7　如果使用已开发软件，包括已经装备的软件、外购的现货软件和重用的软件作为交付的安全相关软件，对于这些软件也应进行软件安全性分析。如果由于软件和文档的特性一些分析无法进行，应获得需方认可。

4.2.8　软件安全性分析中应使用适合的技术和措施，以提高软件安全性分析的质量和效率。根据安全完整性级别推荐的安全性技术和措施见附录C。

4.2.9　软件安全性分析应和软件与系统的其它方面的工作(包括但不限于可靠性、维护性、保密安全性)协调一致，一起策划与执行。

4.3　标准的剪裁

4.3.1　本指导性技术文件给出了软件的安全性分析的一般方法，并以此满足不同复杂程度软件的需要。在应用本指导性技术文件时，可对本指导性技术文件进行剪裁。经过剪裁的要求和任务及其要点，应列入合同或任务书等文档中。

4.3.2　安全完整性级别应作为剪裁的主要依据，对于高安全完整性级别的软件应依本指导性技术文件进行充分的安全性分析。而低安全完整性级别的软件，在得到需方认可的情况下，可省略或合并部分安全性分析任务。

4.3.3　本指导性技术文件中有若干任务和工作项目列表，它们并不是完整无遗漏的。各方可根据某项目或组织的特殊需要增加专门的软件安全性分析任务和工作项目。

4.3.4　对于简单的、技术新颖程度低的或重用程度大的软件，若其现有的可信应用经验能够为达到要求的安全完整性级别提供必需的置信度，并且得到需方认可的情况下，本指导性技术文件中的一些任务或工作项目可不执行。

5　软件安全性分析的组织

5.1　管理

5.1.1　启动和范围确定

在安全相关软件的合同或任务书中应提出软件安全性分析的范围和要求。

实施软件安全性分析的组织应确定安全性分析的管理者，明确其责任并给予必要的权限。

管理者应检查软件安全性分析所需的资源(包括人员、技术、基础设施和时间安排)，以保证软件安全性分析的开展。

注：需方提出安全性要求和供方制定安全性计划可依据 GJB 900。

5.1.2 策划

软件安全性分析管理者应制定安全性分析计划，该计划可作为所属软件过程或活动的计划的一部分。软件安全性分析计划中宜包括以下几个方面的内容：

a) 软件安全性分析的任务划分和进度安排；
b) 确定对安全性分析负责的组织、部门和个人，以及组织内部安全性分析工作的沟通方法；
c) 系统安全性分析、软件安全性分析和软件其它工作之间的关系；
d) 软件安全性分析的成果评估方法；
e) 保持危险和风险分析方面信息的规定；
f) 软件安全性分析将产生的文档和其它输出；
g) 对软件安全性分析的质量保证活动，包括验证、确认、联合评审、审核；
h) 跟踪并解决在安全性分析中发现的问题的程序；
i) 适合的分析技术和措施的选择；
j) 安全性分析所需的人员、资源和设施。

5.1.3 执行和控制

管理者应监控由软件安全性分析计划规定的任务的执行。管理者应控制安全性分析进展并对发现的问题进行调查、分析和解决(解决方案有可能导致计划变更)。

注：问题解决过程见 6.4。

5.1.4 评审和评价

管理者应对安全性分析及其输出的软件产品进行评价，以便使软件安全性分析达到目标，完成计划。

5.1.5 结束

管理者应根据合同或任务书中的准则，确定软件安全性分析任务是否完成，并应核查软件安全性分析中产生的软件产品和记录是否完整。

5.2 基础设施

安全相关软件项目中应该保证软件安全性分析人员获得所需的软硬件设施等资源。在进行安全性分析时，应对基础设施进行必要的维护和改进。

5.3 改进

应通过适当时间间隔的评审进行过程评估。根据评估的结果，必要时应对安全性分析的组织管理和技术措施进行改进。

5.4 培训

软件安全性分析是否成功很大程度上依赖于技能水平高、知识面宽的实施人员。为了完成安全性分析，应确保拥有搭配合理、类别齐全、经过培训的合格人员。应建立程序保证安全性分析涉及的人员能胜任其任务，包括开展安全性分析的相关培训。

培训过程应注意：

a) 培训的内容和资料对相应项目是否完整，不应忽略系统、硬件以及环境方面的培训内容，特别是安全性技术的培训；
b) 培训的类型和水平是否适合相应项目；
c) 培训的对象是否包含了所有需要相关技能的人员；
d) 是否规定了对各类人员的周期性再培训。

注：对软件安全性分析人员的资格要求见附录 E。

6 软件安全性分析的支持

6.1 文档编制

6.1.1 文档的内容

安全相关软件生存周期中，涉及安全性分析任务的内容应文档化。本指导性技术文件关注的是文档所包含的信息，除非进行了明确的要求，这些信息不要求必须形成独立的文档，可以与软件的其它文档相结合。文档的格式和内容应与软件项目的合同或任务书中的规定一致。

通常，文档化的内容包括：

a) 有效执行标准规定的任务所要求的输入信息；

b) 安全性分析任务产生的信息；

c) 组织和管理安全性分析所要求的足够信息；

d) 安全性报告所需的足够信息，也包括从任何评价过程得到的结果和信息。

注 1：信息是否充分取决于多个因素，包括系统的复杂程度、规模以及应用领域的要求。

注 2：软件文档方面的标准有 GJB 438A－1997。

6.1.2 文档的形式

文档的形式应做到:

a) 准确简明；

b) 让使用者容易理解；

c) 适合达到文档的预期目的；

d) 易于访问和维护：

1) 文档的信息应有标题或名称，指出信息的内容和目的，应具有某种形式的索引排列，以便于对照访问本指导性技术文件要求的信息；

2) 文档的格式结构可根据组织规程和应用领域的工作实践来确定；

3) 文档应有修订目录(修订次数)以区分不同版本；

4) 文档的结构应易于查找相关信息，应易于识别文档或内容的最后修订版本；

5) 文档的实际结构主要取决于系统规模、复杂程度和安全性要求等因素。

6.2 配置管理

应按照软件配置管理的要求，将软件安全性分析产生的各种工作产品纳入配置管理。

软件项的配置标识中应明确标明是否涉及安全性。可能的话，标注从安全性分析得到的软件项安全完整性级别。

应标明安全性相关的软件项的追溯关系。可使用检查单和交叉参照维护这种从安全性需求开始的追溯关系。根据软件安全性需求拟定并维护关于所需危险控制的检查单，开发需求向下传递矩阵，以便适当地向下传递并映射到规格说明、设计、代码、测试和文档。

配置管理规程中应规定对受控安全相关软件项的修改(包括改进和修补)需进行评价和审核，确定修改对安全性的影响。高安全完整性级别软件项的配置管理规程应更为严格。

注 1：对于配置管理的更详细的规定，见 GJB 5235。

注 2：变更安全性分析见 8.7。

6.3 质量过程

6.3.1 质量过程的配置

软件项目的一个关键方面是规定质量过程来适合项目的要求，软件安全性分析也需要质量过程的保障，GB/T 8566－2001 包括一系列的质量过程：

a) 质量保证；

b) 验证；
c) 确认；
d) 联合评审；
e) 审核。

并不是每个软件项目都需要所有的质量过程和活动，安全性要求是影响软件项目对质量过程规定的一个重要因素。安全完整性级别高的软件需要比较完整的质量过程、活动和任务，而安全完整性级别低的或非安全相关软件则对质量过程的要求较低。

6.3.2 质量过程的独立性

执行项目的质量过程的人员应相对独立于开发者，软件失效造成的后果和安全完整性级别是决定这种独立性的一个重要因素。

质量保证过程应由组织内独立于软件开发人员的人员开展。

执行项目的验证和确认的人员的最低独立等级可按表 4 和表 5 的规定，表格中的推荐项含义如下：

a) HR：对于规定的后果(表 4)或安全完整性级别(表 5)，推荐该独立性等级作为最低限。如果使用更低的独立等级则应详细说明不使用 HR 等级的理由。
b) R：对于规定的后果(表 4)或安全完整性级别(表 5)，该独立性等级是不够的并且肯定是不推荐的。如果采用该独立性等级则应详细说明其理由。
c) —：表示不推荐或不使用规定的独立性等级。

在表 4 和表 5 中，使用 HR^1 还是 HR^2(不能两个都用)取决于特定应用领域的多个影响因素，如 HR^1 比 HR^2 更适用，HR^2 应理解为不做要求；如果 HR^2 适用，则 HR^1 应理解为不推荐(NR)。如应用领域没有相关标准，应详细说明选择 HR^1 或 HR^2 的理由。

造成选择 HR^2 比选择 HR^1 合适的一些因素是：

a) 缺乏相似设计的经验；
b) 复杂程度高；
c) 设计新颖度高；
d) 技术新颖度高；
e) 缺乏设计标准。

表 4 根据后果决定验证和确认的最低独立等级

最低独立等级	后果			
	A	B	C	D
独立的人	HR	HR^1	NR	NR
独立部门	—	HR^2	HR^1	NR
独立组织	—	—	HR^2	HR

表 5 根据安全完整性级别确定验证和确认的最低独立等级

最低独立等级	安全完整性级别			
	1	2	3	4
独立的人	HR	HR^1	NR	NR
独立部门	—	HR^2	HR^1	NR
独立组织	—	—	HR^2	HR

在表 5 中独立性的最低等级应基于由软件执行的安全功能中最高的安全完整性级别确定。

在组织内部，可依靠组织的机构和专家，如要求独立的人和部门则需借助外部的组织。另外，如果

组织内有在风险评估和安全相关系统应用方面能力的内部组织，并且该内部组织是独立的，与组织中的主要开发部门是分开的(用行政安排或用资金支持等方法)，则可使用该内部组织作为独立组织。

应用表 4 之前，有必要规定后果的种类，并考虑应用领域中现有的经验。所谓后果是指运行时软件如果失效会出现的情况。

注 1：后果示例后果 A——较小的伤害(如功能的暂时丧失)；后果 B——对一个或多个人的严重的、永久的伤害，致一人死亡；后果 C——致多人死亡；后果 D——致使大量人员死亡。

注 2：软件验证和确认方面见 GJB 5234。

6.4 问题解决

安全相关软件项目应建立问题解决程序，软件安全性分析中发现的问题应纳入该程序，采用问题解决程序来对软件产品的有关安全性的偏差、问题和失效进行闭环跟踪，并应评审偏差的安全性影响。

注：问题解决过程中可能要应用变更安全性分析，变更安全性分析见 8.7。

7 软件安全性分析准备

7.1 概述

整体系统中除 EUC 外，可能包含若干 PE 安全相关系统与和其它技术安全相关系统(比如减压阀)。在整体系统中，PES 或者是独立于 EUC 起控制等作用，或者是 EUC 的组成部分，软件总是运行于某 PES 之中。在整体系统之外，还可能有外部安全设施(比如排水沟、堤坝、防护墙)。对于危险的控制可由 PE 安全相关系统、其它技术安全相关系统和外部安全设施来完成。只有将软件与所在的系统结合起来考虑，分析软件安全性才有意义。图 2 表明了安全相关软件同其所在系统(包括整体系统和 PES 子系统)的关系。

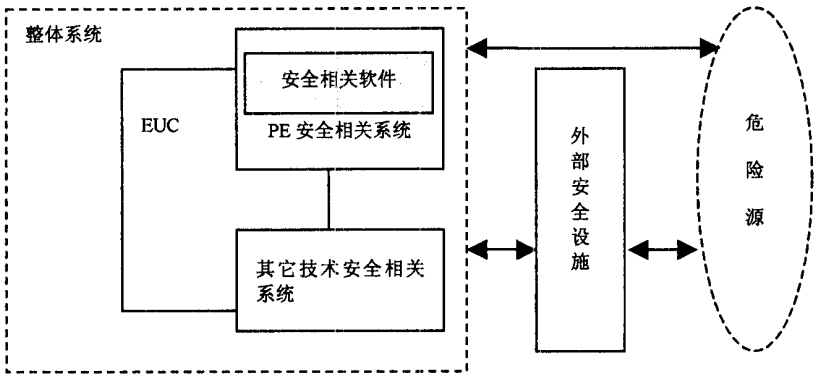


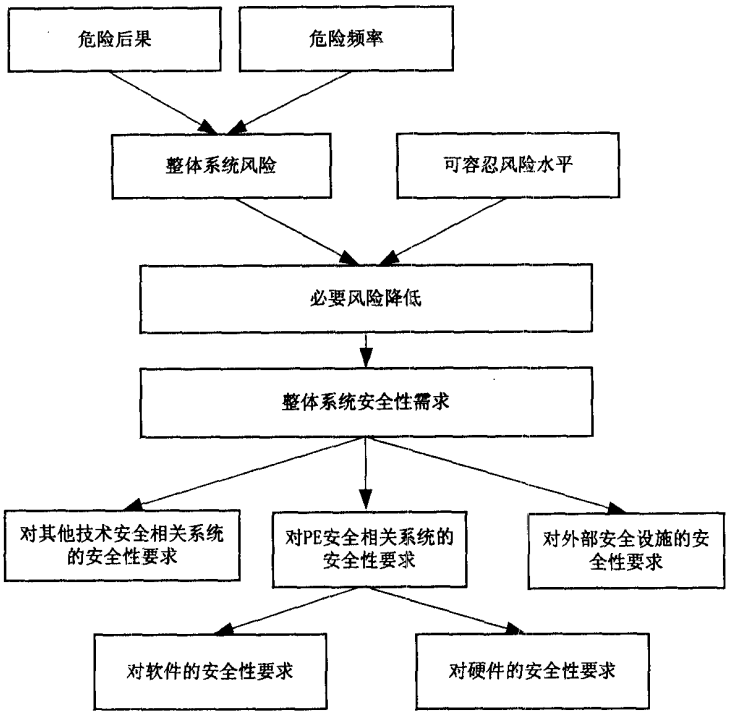
图 2 软件同系统的关系

在进行软件安全性分析前应对软件所在的系统进行初步的分析，作为软件安全性分析的准备，包含下列任务：

a) 概念分析和系统范围确定；
b) 初步危险和风险分析；
c) 系统需求安全性分析；
d) 系统安全性需求分配；
e) 软件的安全性需求分配。

这些任务应与获取过程中的启动活动和开发过程中的系统需求分析和系统结构设计活动结合开展。软件安全性分析准备中和本指导性技术文件有关的主要信息的处理流程见图 3。

注：风险和安全完整性的相关概念见附录 A。



注：图中的方框表示信息，箭头线表示信息的推导或者映射关系。

图 3 软件安全性分析准备信息处理流程示意图

7.2 概念分析和系统范围确定

7.2.1 概念分析

概念分析的目的是提高对整体系统及其环境的理解水平，确定整体系统的可容忍风险水平，以利于其它安全性分析活动能够顺利地进行。

在概念分析中，包括下列工作项目：

a) 对 EUC 及其需要的控制功能和所处物理环境进行全面的了解；

b) 确定可能的危险源；

c) 获取有关已确定危险源的信息(毒性、爆炸性，腐蚀性、反应性、易燃性等)；

d) 获取当前有关的安全法规；

e) 考虑相邻近的 EUC(包括已安装和将安装的)之间相互作用所产生的危险；

f) 确定整体系统的不可容忍的风险、可容忍风险、完全可接受风险的水平。

注：可容忍风险的概念见附录 A。初步危险表的例子附录 F。

7.2.2 系统范围确定

系统范围确定的首要目的是确定系统的边界，第二个目的是确定危险和危险分析的范围(如过程危险、环境危险等)。

在系统范围确定中，包括下列工作项目：

a) 确定在初步危险及风险分析范围内的实际物理设备，包括 EUC 和 EUC 控制系统；

b) 确定在初步危险及风险分析时应考虑的外部设备；

c) 确定与危险有关的子系统间以及它们与其它部分的接口；

d) 确定需要考虑的可引发事故的事件的类型(如零部件失效、程序错误、人员出错、可能引起事故的相应失效机制)；

e) 可采用初步危险表的方式列出可能需特别重视的危险或需做深入分析的危险部位。

7.3 初步危险和风险分析

初步危险和风险分析的目的是对所有应考虑的可预见情况，确定整体系统(在所有运行模式下)的危险、危险事件或事件序列，以及相应的风险和可容忍风险水平。

初步危险和风险分析包括下列工作项目：

a) 应考虑如何排除危险；

b) 在合理预见的情况下确定整体系统的危险和危险事件(包括出错条件和可预见到的误操作)，包括相关的人为因素，尤其应注意那些不常见的、异常的 EUC 运行模式；

c) 分析导致 b)确定的危险事件的事件序列；

d) 分析 b)确定的每种危险事件的可能性；

e) 分析 b)确定的每种危险事件所伴随的潜在后果；

f) 分析 b)所确定的每种危险事件对应的系统风险；

g) 分析 b)所确定的每种危险事件对应的可容忍风险水平。

对于危险和风险的分析中应考虑这些因素：每个确定的危险事件及影响该事件的因素；与危险事件相关的事件序列的因果关系和发生可能性；减少或消除危险的措施；危险分析中的假设，包括运行频率和设备失效率的评估，操作制约或人为介入的可靠性的细节；为有关安全分析给出关键的危险和风险信息。

注 1：可通过定性或定量的方法来实现初步危险和风险分析，有关这些技术的介绍见附录 B。

注 2：将确定的危险解决于源头是非常重要的，如应用固有的安全原理，以及提高工程质量等，这部分内容不包括在本指导性技术文件范围内。

注 3：一般应考虑用调整设计或所使用设备的方法排除事件序列。

7.4 系统需求安全性分析

系统需求安全性分析的目的是为达到需要的安全性，对整体系统中的 PE 安全相关系统，其它技术安全相关系统和外部安全设施提出总的安全性需求。

系统需求安全性分析应与系统需求分析活动结合开展。

系统需求安全性分析包括下列工作项目：

a) 针对确定的危险规定整体系统需具备的安全功能；

b) 根据每个危险对应的风险以及风险可容忍水平，确定危险的必要风险降低；

c) 根据必要的的风险降低，对每个安全功能提出安全完整性级别的需求；

d) 安全功能需求和安全完整性级别需求一起构成整体系统安全性需求，应以文档记录该需求。

注 1：在进行整体系统需求安全性分析时，由于尚不确定执行安全功能的技术和方法，因此要执行的安全功能并不以实现技术的特定术语描述。在安全性需求分配时，可能会修改安全功能的描述以反映特定的实现方法。

注 2：确定安全完整性级别的几个方法见附录 B。

7.5 系统安全性需求分配

7.5.1 概述

系统安全性需求分配的目的是将包含于整体系统安全性需求之中的安全功能分配给指定的安全相关系统和外部安全设施，并对每个安全功能分配安全完整性级别。

系统安全性需求分配应与系统结构设计活动结合开展。

整体系统安全性需求分配包括下列工作项目：

a) 确定安全相关系统，并将安全功能分配到安全相关系统；

b) 将安全完整性分配到安全相关系统；

7.5.2 确定安全相关系统和安全功能分配

应进行初步的系统结构设计，确定整体系统中的 PE 安全相关系统、其它技术安全相关系统以及整体系统以外的外部安全设施。

应将整体系统的安全功能分配到一个或多个 PE 安全相关系统、其它技术安全相关系统和外部安全

设施。安全功能应以最适宜的技术途径完成，PES 和软件安全性需求的分配应与其它技术安全相关系统和外部安全设施统一考虑。

对一个或多个安全相关系统分配规定的安全功能取决于许多因素，但主要取决于安全功能需达到的对风险的降低程度。风险降低要求目标越高，安全功能就越有可能分布于多于一个的安全相关系统。初始的分配是粗略的，可能需在系统和软件的开发过程中进一步调整。

如果某 PES 的失效可能整体系统风险提高，则该 PES 一般应被确定为安全相关系统。如果不将该 PES 作为安全相关系统，则该 PES 以外的安全相关系统应已经被设计为能够以适当的安全完整性提供安全功能，并且该 PES 还应满足下列各项条件：

a) PES 标称的危险失效率应经过下列来源之一获得的数据的证实：
   1) 在一个类似的应用中，该类 PES 的真实操作的经验；
   2) 根据一个公认的方法进行的可靠性分析；
   3) 类似设备可靠性的行业数据库。
b) 为该 PES 标称的危险失效率应不低于每小时 10^{-5} 个危险失效；
c) 在开发整体系统安全性需求时，应确定并考虑所有合理可预计的该 PES 的危险失效模式；
d) 该 PES 应与其它安全相关系统和外部安全设施分离并且是相互独立的。

注 1：要求 b)的根据是如果 PES 没被指定为安全相关的的系统，则 PES 标称的失效率应不低于安全完整性级别 1 对应的较高目标失效测度。(该危险失效率是每小时 10^{-5} 个危险失效，见表 2)；

注 2：本指导性技术文件没有对于其它技术安全相关系统和外部安全设施作出具体要求，本指导性技术文件涉及它们的目的是为了说明对 PES 和软件安全性需求的分配方法。

7.5.3 安全完整性分配

应确定分配到各个安全相关系统的安全功能的安全完整性。安全完整性的确定可以通过定性或定量的方法完成。

安全完整性分配之前的安全完整性是根据必要风险降低，针对整体系统的安全功能而规定的，而分配后中的安全完整性是针对由具体安全相关系统实现的安全功能规定的。

安全完整性分配应符合下列原则：

a) 所有安全性功能和相应的安全完整性需求分配完成后，其组合应能实现该安全功能的必要风险降低。如果必要风险降低没有达到，则可能需要修改整体系统结构设计，重新分配安全性需求。
b) 应确定安全相关系统是按照低频率运行模式还是高频率或连续运行模式运行，并据此按照表 1 或表 2 给出安全完整性需求的目标失效测度。
c) 如果安全相关系统执行不同安全完整性级别的安全功能时，应被认为是属于最高安全完整性级别，除非在设计中可表明不同安全完整性级别的安全功能之间的充分独立性。
d) 不应对单一的 PE 安全相关系统分配高于表 1 和表 2 中规定的 SIL4 级的目标失效测度。只有满足下列的条件 1)或同时满足 2)和 3)，才允许只有由单一的 SIL4 级的 PE 安全相关系统构成的结构存在。
   1) 通过综合运用适当的分析和测试手段，明确表明目标失效测度可以满足；
   2) 对于构成 PE 安全相关系统的部件有很广泛的使用经验，这些经验应该在类似的环境下或至少在一个复杂程度相当的系统中获得；
   3) 构成 PES 安全相关系统的硬件具有足够的失效数据，以便对于硬件的目标失效测度有足够的置信度，数据应与当前项目的环境、应用和复杂程度水平相关。
e) 分配中应考虑共因失效的可能性。如果 PES 被当作独立的安全相关系统，则它们应满足下列要求：
   1) 功能上是多样化的(也就是，使用完全不同方式实现同一目的)；
   2) 基于多样化的技术(也就是，使用不同类型的设备实现同一目的)，必须注意的是，即使采

用了多样化的技术，在高完整性级别，尤其是失效会导致严重后果的情况下，还需要特别警惕低概率的共因事件，比如飞行器坠毁、地震；

3) 不应共享如果失效会导致整体系统发生危险失效的通用的部件、服务和支撑系统(如电源)；

4) 不应共享通用的操作、维护和测试规程。

5) 在物理位置上分离，使可预料的失效不影响冗余的安全相关系统和外部安全设施。

6) 如果不能满足上述 1)～5)的要求，则除非进行了分析并且确认其在安全完整性方面是充分独立的，才能在分配中将该 PES 看作是独立的。充分独立是指外部相关性导致的失效概率与 PE 安全相关系统的安全完整性级别要求的失效概率相比足够低。

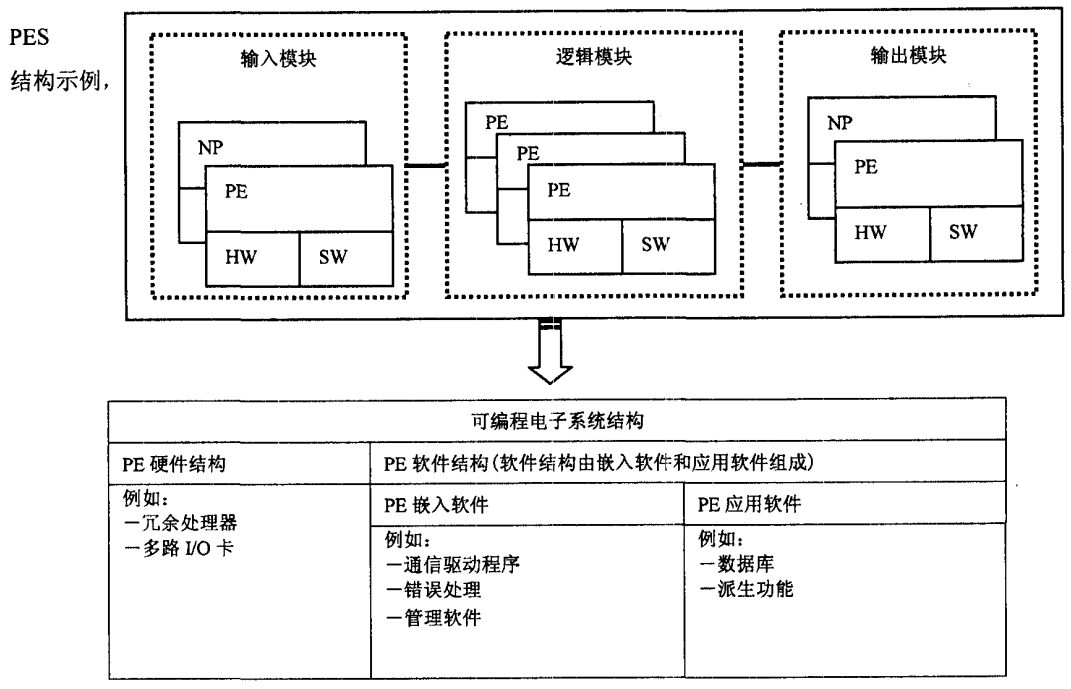
7.6 软件的安全性需求分配

应识别 PE 安全相关系统中的安全相关软件，并将分配于 PES 的安全性需求进一步分配到 PES 中的软件项，确定软件需要实现的安全功能以及这些安全功能的安全完整性。

软件的安全需求分配应与 PE 安全相关系统的结构设计结合开展。

PES 作为整体系统的子系统，在 PES 内部的安全需求分配中，整体系统安全性需求的方法和概念仍然适用。

图 4 表示 PES 中软件和硬件结构的关系，PES 一般由输入模块、输出模块和逻辑模块构成，PES 中的硬件提供软件运行的基础，由软件完成大部分 PES 逻辑功能。



PE—可编程电子　NP—非可编程电子　HW—硬件　SW—软件

图 4　可编程电子系统中硬件和软件结构的关系

典型的安全性相关软件一般是这样的软件：如果它不执行，或者执行不正确，或者不按规定顺序执行就可能导致危险发生或使危险状态存在。例如：

a) 用于直接指挥与控制具有潜在危险的功能和(或)硬件的软件；

b) 用于监视关键硬件部件的软件；

c) 对系统的关键条件和(或)状态进行监视的软件。

应确定安全相关软件的外部接口和设计约束，识别出对于软件安全性的潜在要求，例子包括：事件顺序(时序关系)、时间限制、性能要求、存储器限制、I/O 通道特性等。

8 软件安全性分析任务

8.1 软件需求安全性分析

8.1.1 概述

软件需求安全性分析的目的是要对分配给软件的系统级安全性需求进行分析，规定软件的安全性需求，保证规定必要的软件安全功能和软件安全完整性。

软件需求安全性分析的主要输入是软件安全性分析准备的结果和系统的初步结构设计文档，包括系统分配的软件需求、接口需求。

软件安全性需求分析开发出的软件安全性需求应成为软件需求规格说明的一部分。软件需求安全性分析可产生后续软件安全性分析的输入信息，和对后续的软件设计和测试的建议。应在软件需求评审中评审软件安全性需求，并将之反馈给进行中的系统安全性分析活动。

应采用适当的安全技术完成安全相关软件的软件需求分析，这些技术见表 C.1。

软件需求安全性分析既包括建立软件的安全性需求，又包括在安全性方面对形成中的软件需求进行评价。软件需求安全性分析包括下列任务：

a) 系统安全性需求映射；
b) 安全相关性分析；
c) 软件需求的安全性评价。

8.1.2 系统安全性需求映射

应分析分配给软件的系统安全性需求，并以将之转换为软件的安全性需求。系统安全性需求映射包括下列工作项目：

a) 检查软件安全性分析准备中产生的信息以确保充分满足要求。应特别考虑整体系统需求安全性分析和整体安全性需求分配的以下输出：
   1) 整体系统安全功能；
   2) 系统配置或结构；
   3) 硬件安全完整性要求(可编程控制器、传感器、执行器)；
   4) 软件安全完整性要求；
   5) 容量和响应时间性能要求；
   6) 设备和操作人员界面。
b) 为确保软件安全性，如果没有在系统的安全性需求中说明 EUC 的所有相关运行模式，应在软件安全性需求中进行说明。
c) 软件安全性需求应规定以下内容：
   1) 使 EUC 获得或维护安全状态的功能；
   2) 与检测、通告和管理控制部件中硬件错误有关的功能；
   3) 与检测、通告和管理传感器和执行器错误有关的功能；
   4) 与检测、通告和管理软件本身中的错误有关的功能；
   5) 与定期测试在线安全功能有关的功能；
   6) 与定期测试离线安全功能有关的功能；
   7) 与安全地修改 PES 有关的功能；
   8) 安全功能与非安全相关功能之间的界面；
   9) 容量和响应时间等性能；
   10) 软件与 PES 之间的接口。
d) 软件安全性需求描述应阐明所有与安全相关的软件和硬件之间的约束条件。软件安全性需求描述中需考虑下列与硬件结构设计有关的内容：
   1) 软件自检；

2) 监控可编程电子硬件、传感器和执行器；
3) 在系统运行时定期对安全功能进行检测；
4) 当EUC在操作时，能够对安全功能进行测试。

e) 分析拟定的软件的安全性需求，如果发现存在不可行的系统需求、接口需求和不适宜软件实现的安全功能，可以考虑进行系统安全性需求的重新分配，以便借助软件以外的风险降低手段(即外部安全设施或系统内其它的安全性技术手段)，实现系统总的安全性需求。

8.1.3 安全相关性分析

安全相关性分析的目的是分析所有的软件需求及其之间的关系，确定和安全相关的软件需求，并明确地标明它们。这些软件安全性需求可以是自顶向下即从系统需求传递下来的，也可以是自底向上分析推导出来的。安全相关性分析包括下列工作项目：

a) 将系统安全性需求直接映射的软件需求标识为软件安全性需求；
b) 进行自底向上的分析，识别与系统需求不一致，或系统需求所未阐述的安全性需求。自底向上的分析还可能揭示非期望的达到危险和不安全状态的路径(例如潜通路)。必要时应通过增加或修改系统需求加以解决所识别出的潜在危险，并在将它们再传递给软件需求；
c) 应分析系统分配给软件的外部设计约束，包括时序、吞吐量和规模等，确定它们是否是安全相关的；
d) 当要求安全相关软件执行非安全相关功能时，软件安全性需求应清楚地标明这些功能；
e) 以安全完整性级别的形式规定软件安全功能的安全完整性要求，并且标识出目标失效测度的运行模式。确定软件的安全完整性级别一般遵循下列原则：
   1) 软件失效一般是严重的系统性失效，系统中软件的安全完整性级别最初应与系统总的安全完整性级别相同。对系统和软件结构进行分析后，新确定的软件功能的安全完整性级别可能低于系统总的安全完整性级别；
   2) 如果软件功能单独失效导致产生危险或孤立地失效导致无法提供风险控制功能，则赋予该软件功能与其失效导致的最严重危险对应的安全完整性级别；
   3) 如果软件与其它系统部件，比如硬件部件相结合失效才导致危险或导致无法提供风险控制功能，则有可能通过分析依赖关系和失效频率，适当降低其安全完整性级别；
   4) 如果软件失效不导致危险，并且软件的功能与任何风险控制功能无关，则可确定这些软件不是安全相关的。应在软件和系统的体系结构方面采取足够的措施实施故障隔离以确保这些软件失效不导致危险；
   5) 如果通过失效处理机制达到了故障隔离，则与失效处理机制相关的软件应被赋予与系统整体安全完整性级别相同的软件安全完整性级别；
   6) 如果软件同时执行安全功能和非安全功能时，软件整体应被认为是与安全有关的；
   7) 如果软件执行不同安全完整性级别的安全功能时，所有的软件都被认为是属于最高安全完整性级别，除非在设计中可表明不同安全完整性级别的安全功能之间的充分独立性；
   8) 软件开发者应建立特定规程以解决软件安全完整性级别分配中的任何矛盾。

8.1.4 软件需求的安全性评价

应对形成中的软件需求在安全性方面进行评价，评价的工作项目包括：

a) 分析软件安全性需求是否能溯源到软件需求安全性分析的输入，也就是系统安全性需求和安全性需求分配，显示对于系统需求的依从性；
b) 分析软件需求规格说明，对应每种识别出的危险，保证已经完整地规定了分配给软件的系统安全性需求，并且已经转化为恰当的软件需求；
c) 分析软件安全性需求与系统安全性需求、软件安全性需求和其它软件需求的外部一致性，以及软件安全性需求的内部一致性；

d) 根据安全完整性级别的要求，软件安全性需求的规定要求应这样表示和构成：
   1) 清楚、一致、准确、可验证并与安全完整性级别相当；
   2) 不使用含糊的或文档受众所不理解的术语和描述。
e) 应分析安全性需求的可测试性，提出对后续测试需求的建议；
f) 应分析软件安全性需求设计和实现的可行性，确定软件安全性需求的规定是否足够细致以便软件的设计和实现达到要求的安全完整性，进而对软件设计提出建议；
g) 应提出与安全性相关的必要、提倡、不提倡和禁止使用的软件设计、编码和测试技术列表。安全性技术和措施的选取可以参考附录C。

8.1.5 软件需求安全性分析技术

下面列出了一些可用于软件需求安全分析的主要技术手段，其它的一些技术和措施见附录 C 和附录D。

a) 检查单和交叉参照：
   用于需求向下传递分析的工具和方法是检查单(见 D.1.3)和交叉参照。应根据软件安全性要求拟定并维护关于所需危险控制及其相应安全性要求的检查单，用来在整个开发期间确保适当地向下传递并映射到设计、代码和测试。
   应开发一个软件安全性需求和任何危险控制的系统化检查单，确保它们正确地充分地包括(并交叉参照)相应的规格说明、安全性分析、测试和设计文档。应开发一个危险要求向下传递矩阵，其中将安全性要求和危险控制映射到系统/软件功能，并映射到软件部件与单元。在部件和单元尚未定义的情况下，要与将来的向下传递结合才可能传递到最低级。
b) 层次分析：
   层次分析建立需求的相互依赖关系。
c) 控制流分析：
   检查软件功能的执行次序。控制流分析识别遗漏的和规定得不一致的功能；分析哪些进程串行执行，哪些进程并行执行(例如多任务)，以及哪些任务是前提或者依赖于其它任务。
d) 信息流分析：
   检查功能和数据之间的关系。识别不正确、遗漏和不一致的输入/输出规格说明。
   一般用数据流图(见 D.5.2)来报告这个活动的结果，因此，这项技术最好在结构设计期间使用。但是，如果用快速原型和/或螺旋型生存周期模型，则在早期对基本数据和命令流也可能是有效的。
e) 功能模拟：
   模拟器是有用的开发工具(见表 C.15)，用来评价系统性能和人机交互作用，检查软件部件的特性，以预计性能，检查人对系统特性的理解，以及评价可行性。模拟器有局限性，它们是描述性模型，有时并不准确反应实际的设计，或者所作的环境假设可能与现场条件不同。
f) 约束分析：
   对安全相关功能的约束分析将评价与执行时间和存储器分配相关的软件需求(见 D.8.22)。约束分析的焦点在程序的限制上。典型的限制要求是最大执行时间和最大存储器使用。应评价安全相关的计时和规模要求的恰当性和可行性，还要评价是否在每一种情况中最坏情景下都已经分配恰当的资源。例如，I/O 通道是否被许多差错消息过载，因而妨碍安全相关特征运行。量化计时/规模资源要求可能非常困难，可用现有类似系统的实际数据作为估计的基础。要考虑的项目包括：
   1) 存储器使用与可用性：
      可以根据以前的软件开发经验来对存储器使用进行评估。更详细的估计应评价存储在存储器中的代码规模，以及存储数据所需的附加空间和用于存储中间和最后计算结果的高速暂

存空间。在早期阶段进行的存储器估计可能不准确，在估计变成实际之前应在原型代码和仿真的基础上对估计进行修改。对于保证关键数据的适当计时和使用来说，动态存储器分配既是一种切实可行的存储器运行时间的解决方案，也是一种令人担忧的根源。对动态存储器分配，应很仔细地加以分析；即使在“非安全相关的”功能模块中也应如此。

2) I/O 通道使用(载荷)与能力及可用性：

阐述用于科学数据收集、整理和控制的 I/O。评价科学数据收集和安全相关数据可用性之间的资源冲突。在失效事件期间，I/O 通道可能被差错消息过载，而使重要消息可能丢失或改写。可能的解决方法包括专门设计的附加模块，用来捕获低级差错消息，使这些信息相关并加以管理，或者将差错沿程序调用层次向上传递直到能够处理该问题的层次；从而，仅传递有关可以引起失效的关键故障或关键故障的组合的信息。

3) 执行时间与 CPU 载荷及可用性：

审查 CPU 载荷随时间的变化，确定峰值载荷情况和这种情况是否可接受。考虑多作业的影响。注意，过量的多作业能引起的系统不稳定性，进而导致“崩溃”。

4) 采样率与实际参数变化率：

考虑计时、规模和吞吐量方面的设计准则，分析所用的系统性能模型的有效性，如果可以得到仿真和测试数据，则可用这些数据来说明。

8.2 软件结构设计安全性分析

8.2.1 概述

软件结构设计进行软件需求的高层体系结构设计。作为这个过程的一部分，结构设计安全性分析的目的是将全部软件安全性需求综合到软件的体系结构设计中，确定结构中与安全性相关的部分，并评价结构设计的安全性，以保证软件安全功能的安全完整性。

软件结构设计安全性分析的主要输入是系统的初步结构设计、硬件的初步结构设计、软件需求和之前开展的软件安全性分析提供的信息。

软件结构设计安全性分析可产生后续软件安全性分析的输入信息，可提出对后续的软件设计和测试的建议。应在软件结构设计评审中评审软件结构设计安全性分析的结果，并将之提供给进行中的系统安全性分析活动。

应采用适当的安全技术完成安全相关软件的结构设计，这些技术见表 C.2。

软件结构设计安全性分析包括下列任务：

a) 软件安全性需求传递；

b) 安全相关性分析；

c) 结构设计的安全性评价；

d) 软件集成安全性测试需求分析；

e) 用户文档安全性分析。

注：在某些领域中，软件结构设计也可称作功能设计、概要设计、体系结构设计。

8.2.2 安全性需求映射

软件结构定义软件主要部件，以及它们如何交互，如何获得所要求的属性，特别是安全完整性。从安全性角度讲，软件结构设计是制订软件基本安全性策略的阶段。应将软件安全性需求映射到结构设计划分的软件部件和接口上。

在安全性需求映射中，可以考虑下列结构设计的原则：

a) 将软件中的安全相关部分最小化；

b) 包含容错(与硬件一致)、避错、冗余和多样性(适用时)的设计策略；

c) 实现校验测试和其它诊断测试的软件功能；

d) 选择用于维护所有数据安全完整性的设计特征。这种数据可包括系统输入/输出数据、通信数

据、操作界面数据、维护数据和内部数据库数据。软件设计应包括对控制流和数据流进行的，与要求的安全完整性级别相当的自检。应在检测到故障时，采取适当的控制方法；

e) 包含方便软件更改的特性，这些特性的例子有模块化、信息隐藏和封装；

f) 其它控制复杂性的特性。

注：软件和硬件结构设计不可避免地会有反复，因此有必要与硬件开发者讨论软件的结构设计。

8.2.3 安全相关性分析

安全相关性分析的目的是分析软件部件和接口，确定和安全相关的软件部件，并明确地标明它们。这些部分可以是自顶向下即从软件需求传递下来的，也可以是自底向上分析推导出来的。安全相关性分析包括下列工作项目：

a) 从软件的需求出发，分析其中的安全性需求与软件的部件之间的关系。将与这些安全性需求有关的软件部件确定为安全相关的软件部件；

b) 检查形成中的软件结构，确定软件部件之间的相互依赖和接口关系，对软件的控制流和数据流进行分析，确定其相关的程度。直接或间接影响安全相关的软件部件的软件部件也要标识为安全相关的软件部件；

c) 更新危险和风险分析，应分析软件的部件划分和部件接口间等可能引起的危险，应将识别出的新的危险反馈给软件和系统的安全性需求，并再次映射到软件结构设计；

d) 更新对软件的设计约束的分析，确定软件结构设计没有违反安全性相关的设计约束。

e) 给出软件部件的安全完整性级别。初步的安全相关性分析对每一个部件是否安全相关简单地赋予一个是或否，进一步的分析可确定其安全完整性级别。应考虑：

1) 软件部件的安全完整性级别应与其涉及的安全功能的最高的安全完整性级别相同；

2) 分析软件结构设计中的安全完整性级别的一致性，检查每一软件部件是否影响了具有更高安全完整性级别的部件，如果这种情况出现，则应为该部件分配同样的安全完整性级别；

3) 应检查先前规定的软件安全完整性级别，必要时可根据实现技术和连接方式进行调整。

如果将现货软件或以前开发的软件用作设计的一部分，则应清楚地标明其安全相关性，并应判断软件在满足软件安全性需求规格说明方面的适用性，包括对来自以往软件环境的约束(例如操作系统和编译器)进行评估。可通过在相似系统的使用中取得的证据或与新开发的软件相同的验证和确认程序来确立这种适用性。

8.2.4 结构设计的安全性评价

应对形成中的软件结构设计进行安全性方面的评价，评价的工作项目包括：

a) 分析软件结构设计是否能溯源到软件需求，以显示对于软件需求的依从性；

b) 分析软件结构设计，保证其已经覆盖了软件安全性需求的要求，软件安全性需求已经完整地分派到相应的软件部件；

c) 分析软件结构设计与软件安全性需求和系统安全性需求的外部一致性，以及软件结构设计的内部一致性；

d) 结构描述应基于或限制于定义清晰的符号，结构设计的描述应有利于实现以下特性：

1) 利于表达结构信息，包括：功能、部件间的信息流、与信息有关的次序和时间、定时约束、并发性、数据结构及其特性；

2) 设计的易理解性；

3) 易验证和确认。

e) 分析软件结构设计中的安全功能详细设计可行性，软件安全性部件的规定应足够细致，以能使软件的详细设计和实现达到要求的安全完整性。对进一步的软件设计在安全性方面提出建议。

f) 修改与安全性相关的必要、提倡、不提倡和禁止使用的设计、编码和测试技术列表。安全性技术和措施的选取可以参考附录 C。

8.2.5 软件集成安全性测试需求分析

在软件结构设计中应规定软件集成的初步测试需求。应分析该测试需求是否覆盖了软件安全性相关部件，是否包含安全性方面的测试项目，以保证软件结构满足软件安全性功能需求和软件安全完整性需求。

应对结构设计进行分析以确认危险与测试过程和用例的对应性。分析软件结构和软件部件的可测试性，在可测试性方面应提出对结构设计的修改建议，也可对后续测试提出建议。

8.2.6 用户文档安全性分析

在软件结构设计中，应编制用户文档的最初版本。应对用户文档进行分析，提供保证软件运行期间安全性所需的信息和规程。这个分析在其后开发过程的软件安全性分析中有可能需要得到更新。

应分析用户文档中是否包括下列内容：

a) 软件运行中会出现的警告、报警以及可能发生的危险等；
b) 为保持软件运行的安全性，所需执行的日常运行活动；
c) 为保证特殊情况下的安全性和降低危险事件造成的损失，所需的必要活动和约束(例如，在安装、启动、正常操作、日常测试、可预见的干扰、故障和失效，以及断电)；
d) 对校验测试的规定：定期或者需要时进行安全相关软件的校验测试，并分析校验测试的结果，判断软件的安全完整性级别是否还符合要求，作出是否启动软件变更程序的决定；
e) 当在安全相关软件出现故障或失效时，接下来的维护活动；
f) 维护执行情况的报告，特别是对失效的报告和分析；
g) 维护和重新确认所需的必要工具和设备；
h) 运行过程中产生的需要保存的文档。

注 1：出于安全和经济的考虑，将软件的操作和维护规程与系统的操作和维护规程集成可能是有益的。

注 2：检验频率，诊断测试间隔和随后修改所需的时间会依赖以下几个因素：

a) 与安全完整性级别相关的失效测度目标；
b) 结构；
c) 诊断测试的诊断覆盖；
d) 预期使用率。

8.2.7 软件结构设计安全性分析技术

下面列出了一些可用于详细设计安全分析的主要技术手段，其它的一些技术和措施见附录 C 和附录 D。

a) *动画/仿真*：

可开发仿真器、原型(或设计规定的所需功能性的其它动态表示)、以及运用安全功能所需的测试用例(另见 D.8.17)。运行这些测试并观察系统的响应。必要时可修改需求。

记入文档的测试结果能证实期望的行为或揭示非期望的行为。用危险报告收集安全性验证的状态。

b) 接口分析(另见 D.8.3)：

1) 相互依赖关系分析:

检查软件，确定软件部件、表、变量等之间的相互依赖关系。直接或间接影响安全相关的软件部件的软件元素也要标识为安全相关的软件部件，并且应分析它们的不希望的影响。例如，两个或更多安全相关的软件部件使用的共享存储器块。对每一个安全相关的软件部件的输入和输出进行审查，并追溯到它们的起源和目的地。

2) 独立性分析:

安全性活动评价可用的设计文档，确定安全相关的软件部件对安全关键和非安全关键的部件的独立性/依赖性和安全相关的软件部件的相互依赖性。影响安全相关的软件部件输出

的部件都要指定为安全相关的软件部件。识别故障隔离区域完整性被损害的范围。

方法：将安全功能映射到软件模块，并将这些软件模块映射到宿主硬件和故障隔离区域。审查每一个安全相关的软件部件的每一个输入和输出。

c) 检查单和交叉参照：

见 8.1.5。

d) 失效模式、影响和关键性分析：

见 D.4.3.3。

8.3 软件支持工具和编程语言安全性分析

8.3.1 概述

支持工具和编程语言安全性分析的目的是在安全相关软件的开发过程中，选择合适的设计、编码、评价和修改的软件支持工具，包括语言和编译器。

支持工具和编程语言安全性分析可以独立于某软件项目进行，也可以根据某软件项目的需要开展，但是一般应该在软件详细设计前完成此项工作，最迟在软件编码之前完成。

支持工具和编程语言安全性分析另见表 C.3。

支持工具和编程语言安全性分析包括下列工作项目：

a) 支持工具分析；

b) 编程语言分析；

c) 编码标准分析。

注：开发工具的选择将依据软件开发活动的特性和软件结构：

a) 对于只具有有限可变性的用户应用程序，在低安全完整性级别下，要求的工具和编程语言可被局限为标准 PLC 语言、编辑器、加载器。支持工具和编程语言的适用性的责任主要由 PLC 供方承担；

b) 在较高的安全完整性级别上，可能有必要限制采用 PLC 语言的子集，需要评价工具如代码分析器和模拟器等。在这种情况下责任由 PLC 供方和 PLC 用户共同承担；

c) 即便是在低的安全完整性级别下，使用完全可变性语言的嵌入式应用也应尽量提高可理解性。支持工具和编程语言的适用性的责任主要由软件开发者来承担。这包括使用完全可变性语言为 PLC 用户应用程序提供有限可变语言的 PLC 供方。

8.3.2 支持工具分析

应根据要求的安全完整性级别选择一套合适的工具集，包括语言、编译器、配置管理工具、自动测试工具(如果必要)等。

应考虑合适的工具在软件整个生存周期中的可用性，以便提供相应服务(不应局限于那些在系统开发过程中使用的工具)。

工具应与应用的特点匹配，应具有支持相应的设计方法的特性。

8.3.3 编程语言分析

安全相关的软件应对使用的编程语言进行评价，根据安全完整性级别和应用的特点，选择合适的编程语言，在评价一种语言时，可提出下列问题：

a) 程序语言是否完整并清楚地定义特性或仅使用定义清楚的特性？

b) 能否限制使用不能静态预测的语言特性以及与特定编译器相关的语言特性？

c) 程序语言是否具有方便程序进行错误探测的特性，具有强类型检查机制？

d) 是否能表明该程序不可能跳转到任意的位置？

e) 是否有防止任意的存储器位置被改写的语言特征？

f) 对标准范围内的整数计算和浮点计算是否都有严格的模型？

g) 在目标处理器上运行时，是否有检查运行程序遵守计算模型的程序？

h) 强类型手段是否足够强，能防止误用变量？

i) 语言中是否有防止在运行时存储器被用完的措施？

j) 语言是否提供对具有跨模块类型检查的模块进行分开编译的设施？

k) 设计人员和编程人员是否已良好地理解语言，因而能编写安全相关软件？

l) 语言中是否存在具有安全语言特性的子集？

如果某种语言有精确的定义(也有完备的功能性)，是逻辑上清晰的，有易管理的规模和复杂度，那么就认为这个语言适用于安全相关软件。当选择编程语言时，应阐明选择该语言的理由，理由应足够详细，宜包括语言对目的的适用性的说明，以及任何附加的克服语言缺点的措施说明。

必要的情况下，应开发编程语言的安全子集，要求将语言定义限制到一个子集有两条基本原因：语言的某些特征是以一种含糊的方式定义的，即语言的逻辑不完善；或者语言的某些特征太复杂。

注：所有编程语言无论在其定义还是在其实现中都有不安全性。计算机语言的发展趋势表明，较新的语言试图纠正老一代语言的不足之处。语言中常见的几个缺点举例如下：

a) 未初始化的变量。这种误用很难捕获，因为除非进行特别的检查，否则单元测试不能标记它们。这种差错的典型表现是，一个一直成功地工作的程序在不同的环境条件下运行时结果不是期望的；

b) 要求重新分配存储器的调用应予以检查，以确保不仅释放指针而且释放该结构所用的存储器；

c) 当函数调用的副作用修改操作数时，通常将操作数计值次序仅作为不良编程实践，但实际上这是一个定义得不好(至今没有定义任何类型的标准)并且由语言编译程序实现者随意解决的问题。

8.3.4 编码标准分析

应考虑制定安全性专用编码标准，编码标准应推荐好的编程经验，说明安全关键代码的注释要求，描述不安全语言特性(如未定义的语言特性、非结构化的语言特性等) 和对这些语言特征的使用限制。编码标准应经过评审，确定其对目的的适用性，并用于所有安全相关软件的开发(另见表 C.11)。

下列是一些可参考的编码标准：

a) 数据规则：

1) 最大程度地使用语言提供的类型检查机制；

2) 变量和常数的助忆符应代表它们的物理和功能特性；

3) 语言关键字和保留字绝不能用作数据名；

4) 一个变量必须有且只有一个名字；

5) 一致性。所有数据必须在一说明块中给出一显式类型(除循环变量外)。按一特定类型所定义的数据，每次使用时均应以此类型出现；

6) 所有变量均须在其说明处加注解；

7) 所有变量均应显式初始化；此初始化应在其首次使用前完成。如果变量在函数执行内部初始化或者在一迭代处理中重复初始化就会降低效率；

8) 缺省值必须定义；

9) 限制使用全局变量或数据；软件单元之间的数据通信尽可能使用参数方法；

10) 同一表达式中所使用的变量类型不应混用。在表达式求值前应显式完成转换；

11) 用文件终或标志，别用计算来结束输入；

12) 用统一的方式处理文件结束；

13) 保证输入数据不违背程序的限制；

14) 使输入数据易准备，输出数据易理解；

15) 使用统一的输入格式；

16) 使输入数据易校对；

17) 将输入输出局限于一子例程中；

18) 避免使用临时变量。

b) 调用规则：

1) 使单元间的联系一目了然；

2） 每个单元都应做好一件事；
3） 提倡有利于可理解性、维护性和完整性的代码重用；
4） 在一单元中有固定值的量不要作为可变参数传送给被调用的单元；
5） 调用序列接口应是显式参数；
6） 入／出参数应按如次顺序存放：输入、输出、状态码。

c） 异常处理规则：
1） 当一单元的输入数据会影响处理的进展(选择变量或循环变量)，或者有发生溢出(表的长度、除法)危险的情况下，应对这些数据的定义域进行检查；
2） 在开始处理前要检查每个输入。为了避免在发现不完整的或不正确的输入数据之前走过了若干处理步，所以在处理开始前要完成对所有输入数据的检查；
3） 在处理期间要检查“关键”输出参数的合理性；
4） 有一错误时，要把一错误码返回给调用者；
5） 检验输入数据的合法性和合理性。

d） 表示法规则：
1） 遵循命名规则，选用不易混淆的名字；
2） 避免使用难用、复杂和晦涩的语言特性，强制使用简洁的句法；
3） 适当使用大小写字母、有意义的标识符、注解、空行和缩进以提高代码可读性；
4） 代码模块不能过大，以降低模块间数据依赖关系；
5） 代码应尽可能按线性方式组织。在这种情况下，代码是自顶向下阅读的。共用块应放在汇编或编译单位之尾；
6） 页面设置应使代码易读。必须指出的是在处理块中基本的结构应是：顺序、选择、迭代(缩进格式)。这些基本结构应以可读、易识别的方式从语句中分离出来。(如，标识监视结构的的关键字用下划线，代码语句用大写字母，注解用小写字母)；
7） 代码愈结构化，其自描述性就愈好；
8） 每行至多一条语句。

e） 注解规则：
1） 软件单元的注解。它标识此单元的用途、所处理的数据或服务、它所管理的主要错误情况、它给出关于此单元内容的信息。每个单元首部的注解。可包括：名字、用途和功能、输入、输出、返回值、调用者、被调者、编程语言、版本号、生成者和日期、修改者和日期；
2） 单元内部代码段的注解解释特定的处理；
3） 单个语句的注解。用以加深对该语句细节的理解；
4） 应广泛使用注解。其目的是说明代码逻辑。构成顺序代码段的任何语句组都应引入注解，描述该段之功能和逻辑(其命名则必须与详细设计文档中相同)。此注解并不排除对单个语句的注解，如果必要的话；
5） 注解应按功能观点书写。它应清楚地解释该代码，主要指出它是干什么的、如何干的，以及何时干的。所用的词汇应面向用户。应避免使用编码符号和缩写，除非是通常能识别的缩写(如，用 kg 表示千克)；
6） 数据注解。在每个单元中，对参数、常数和变量的说明都必须加注解；
7） 控制结构的注解必须按可识别的、易读的方式用准确的形式表述之；
8） 别注释太少或太过多(约1/4)；
9） 确保注释与代码一致；
10） 注释不能重复代码，使每个注释都有用；
11） 不要为糟糕的代码注释，干脆重编。

f) 编程风格：

1) 限制使用不能静态预测的语言特性，或者和特定编译器相关的语言特性；
2) 应使用结构化编码技术；
3) 避免不必要的分支转移；
4) 在一单元中路径或分枝愈多，其复杂性也就愈大。所以必须设法减少单元中的决策或循环个数。决策语句的编码应高效率；
5) 控制结构必须完整。对每个决策点的所有条件和处理均应定义。当一条件测试成功时，应预见一下把所有假定不可能的情况重新合成为“其它情况”；
6) 循环变量必须不为循环体内所执行的处理所修改(修改控制变量不仅是模块的逻辑变得复杂，而且会给测试带来严重问题)；
7) 与循环无关的计算要放在循环体外；
8) 不应在循环体内对常数求值；
9) 不要用条件转移代替逻辑表达式；
10) 应对难以理解的逻辑表达式做适当变换；
11) 使用括号避免二义性；
12) 采用基本的控制流结构；
13) 尽可能近地在决策语句后写出相应处理；
14) 分块编写和测试大的程序；
15) 使用带调试功能的编译程序；
16) 分支时，注意等号与不等号的使用；
17) 禁止循环多重出口；
18) 不要对浮点数作相等比较；
19) 欲使程序提高运行速度，应先令其正确、清晰；
20) 让编译程序作简单的优化；
21) 欲使程序提高运行速度，最好寻找更好的算法；
22) 在改进效率前，先测试代码；
23) 禁止对代码作动态修改；
24) 每个软件单元只能有一个入口和一个出口；
25) 在算术表达式中应尽量使用括号；
26) 不完成有关功能的可执行代码段显然是无效的，且耗费存储空间，有时可能会招致意外的、不确定的后果，故必须消除此类代码。

8.4 软件详细设计安全性分析

8.4.1 概述

软件详细设计进一步细化高层的体系结构设计，将软件结构中的主要部件划分成能独立编码、编译和测试的软件单元，并进行软件单元的设计。在小型应用中，软件详细设计和结构设计可结合起来。

详细设计安全性分析的目的是分析软件的设计和实现是否能以相应的软件安全完整性级别满足软件安全性需求，设计和实现应是可分析和验证的，并能安全地修改。

软件详细设计安全性分析的主要输入信息包括：软件需求，结构设计描述，软件集成测试计划和之前开展的软件安全性分析提供的信息。

软件详细设计安全性分析的输出可用作后续软件安全性分析的输入信息。软件详细设计安全性分析可提出对后续的软件编码和测试的建议，必要的情况下还应建议更新用户文档。在软件详细设计评审时应提交软件详细设计安全性分析的结果，并将之反馈给进行中的系统安全性分析活动。

应采用适当的安全技术完成安全相关软件的详细设计，这些技术见表 C.4。

软件详细设计安全性分析包括下列工作项目：

a) 软件安全性需求向下映射；

b) 安全相关性分析；

c) 详细设计的安全性评价；

d) 软件单元安全性测试需求分析。

注：详细设计和开发的性质根据软件开发活动和软件结构的性质变化。对于使用有限可变语言的用户，详细设计可被认为是配置而不是编程。

8.4.2 安全性需求映射

软件详细设计中应将软件安全性需求从软件部件映射到低层的软件单元，这些需求包括安全功能、安全完整性和安全性相关的设计约束。

应详细描述每个安全相关部件中各个软件单元的组成关系和控制关系；必须详细定义每个软件单元的处理逻辑和单元间的所有接口信息以及每个单元内部所用的每个数据。安全相关的软件单元应进一步细化设计以便于编码。

在安全性需求映射中，可以考虑下列详细设计的原则：

a) 结构化：

1) 设计应使用层次式的逻辑控制结构。

2) 每个软件单元只应有一个入口和一个出口；

3) 安全性关键的功能应尽量集中，可能的话，应使这类程序单元的数量尽量少。

b) 模块化：

1) 应采用模块化的机制；

2) 设计应使软部件由一些较小的、以层次结构相互联系的软件单元组成；

3) 设计应使用特殊的规则来限制软件单元的规模(模块度)。

c) 可预测性：

1) 应为软件单元提供所标识的出错情况下所需的反应；

2) 所设计的计算机资源调度方式应是确定的和可预测的；

3) 应使用尽可能少的中断和事件驱动，并应对此进行充分的论证；

4) 在程序运行过程中进行正确性检查，以发现运行错误和违反运行许可的情况。

d) 健壮性：

1) 设计应覆盖需求定义中所要求的容错和故障弱化的需求；

2) 必须判定所有错误情况，并告警、记录。

e) 易修改性：

1) 使用信息隐蔽技术；

2) 每一个软件单元应都只实现一个特定的子功能。

8.4.3 安全相关性分析

全相关性分析的目的是分析软件单元和接口，确定和安全相关的软件单元，并明确地标明它们。这些部分可以是自顶向下即从软件需求传递下来的，也可以是自底向上分析推导出来的。安全相关性分析包括下列工作项目：

a) 从软件的结构设计出发，分析其中的安全相关软件部件与软件的单元之间的映射关系，将这些直接实现安全功能的单元确定为安全相关的软件单元。这些安全相关的软件单元的例子包括：

1) 系统的故障检测和安全改正的逻辑是与安全直接相关的，凡是用于履行上述功能的软件单元均属于安全相关的软件单元；

2) 决定是否中断系统运行的单元和中断处理单元属于安全相关的软件单元；

3) 能产生对硬件实施主动控制的信号单元属于安全相关的软件单元；

4) 通过详细分析(如故障树分析)表明能发出直接影响硬件部件移动或启动安全性相关活动(如发动机马达臂命令)的信号的软件单元属于安全相关的软件单元；

5) 用于显示硬件安全性关键部位的工作状态的软件单元属于安全相关的软件单元。

b) 对软件的控制流和数据流进行分析，确定软件单元的相互作用，评估及细化它们的重要性，确定哪些软件单元影响已经确定的安全相关软件单元，这些单元也要标识为安全相关的软件单元；

c) 应对安全功能的设计约束进行分析，确定详细设计中的安全功能没有违反定时、容量等设计约束；

d) 更新危险和风险分析，应分析软件的单元划分和单元接口可能引起的危险，应将识别出的新的危险反馈给软件和系统的安全性需求，并再次映射到软件详细设计。

e) 分析软件单元的安全完整性级别。由于软件连接和实现的复杂性，确定每一个软件单元的确切安全完整性级别可能并不现实，但是至少应标识其是否与安全性相关。确定安全完整性级别时需考虑：

1) 软件单元的安全完整性级别应与其涉及的安全功能的最高的安全完整性级别相同。

2) 分析分配给各个软件单元的安全完整性级别的一致性，检查每一软件单元是否影响了具有更高安全完整性级别的单元，如果这种情况出现，则应为该单元分配同样的安全完整性级别。

3) 应检查先前规定的软件安全完整性级别，必要时可根据实现技术和连接方式进行调整。

8.4.4 详细设计的安全性评价

应对形成中的软件详细设计进行安全性方面的评价，评价的工作项目包括：

a) 分析软件详细设计是否能溯源到软件需求，以显示对于软件需求的依从性；

b) 分析软件详细设计，保证其已经覆盖了软件安全性需求和软件结构设计的要求，软件安全性需求已经完整地分派到相应的软件单元；

c) 分析软件详细设计与软件结构设计、软件安全性需求和系统安全性需求的外部一致性，以及软件详细设计的内部一致性；

d) 应分析软件的详细设计是否满足模块化、可验证、易安全修改的要求；

e) 应基于或限于定义清晰的符号描述详细设计；

g) 应分析软件详细设计中的安全功能的编码可行性，如果存在不可行的安全功能，可以考虑进行重新设计。软件安全性单元的规定应足够细致，以能使软件的编码实现达到要求的安全完整性。应对软件编码在安全性方面提出建议；

h) 应修改与安全性相关的必要、提倡、不提倡和禁止使用的设计、编码和测试技术列表。安全性技术和措施的选取可以参考附录 C。

8.4.5 软件单元安全性测试需求分析

在软件详细设计中应规定软件单元的测试需求。应分析该测试需求是否覆盖了软件安全性相关单元，是否包含安全性方面的测试项目，以保证软件详细设计满足软件安全性功能需求和软件安全完整性需求。

应对详细设计进行分析以确认潜在危险与测试过程和用例的对应性。分析软件单元的可测试性，在可测试性方面应提出对详细设计的修改建议，也可对后续测试提出建议。

8.4.6 软件详细设计安全性分析技术

下面列出了一些可用于详细设计安全分析的主要技术手段，其它的一些技术和措施见附录 C 和附录 D。

a) 设计逻辑分析:

设计逻辑分析评价软件设计的方程式、算法和控制逻辑(另见 D.8.9)。逻辑分析检查软件部件

的安全相关区域。识别安全相关区域的一种方法是检查软件部件执行的每一个功能。如果它响应或有可能违反安全需求之一，那么就应把它视为安全相关的并对它进行逻辑分析。进行逻辑分析的一种方法是分析设计说明和逻辑流并记下差异。

充分的设计逻辑分析可以采用形式化方法(见 D.5.4)。形式化程度较低的设计逻辑分析由审查员，对较小量的安全性相关软件产品(例如 PDL、原型代码)进行评审并人工跟踪逻辑。待审查的安全相关逻辑可能包括失效检测/诊断、冗余管理、变量报警范围和禁止命令逻辑的先决条件。

b) 设计数据分析:

设计数据分析评价软件设计中每一个数据项的说明和预期的用途(另见 D.8.10)。数据分析确保数据的结构和预期用途不违反安全性要求。进行设计数据分析时使用的技术是将设计逻辑中每一个数据项的说明与用途进行比较。

在安全相关区域必须特别注意中断及其对数据的影响。分析应验证中断和中断处理例程不改变其它例程所使用的关键数据项。

应评价每一个数据项相对于其环境和宿主的完整性。共享存储器和动态存储器分配能影响数据完整性。还应保护数据项使它避免被未授权的应用程序改写。应和系统安全性分析一起对影响存储器的电磁干扰进行检查。

c) 设计接口分析:

设计接口分析验证软件部件与系统中其它部件的接口的设计是否适当(另见 D.8.3)。这个分析将验证软件部件的接口已适当地设计，验证接口部件之间的控制和数据连接已适当地设计。接口需求规格说明是评价接口时对照的原始依据。

阐述的接口特性应包括数据编码、差错检查和同步。

这个分析应考虑检查和以及循环冗余检验码的真实性和有效性。所实现的差错检查的完善程度应适合预计的该接口的位差错率。应定义整个系统的差错率，并分配到每一个接口。

接口问题的例子包括：

1) 发送器发送八位字，其第七位是奇偶校验位，而接收器认为第零位是校验位；
2) 发送器以 10Hz 传送更新信息，而接收器仅以 1Hz 更新；
3) 发送器对字编码从领先位开始，而接收器解码从末尾位开始；
4) 接口死锁阻碍数据传送(例如，接收器忽视或不能识别“发送准备就绪”)；
5) 用户从差错地址读数据；
6) 发送器将数据存入差错地址；
7) 在像 C 或 C＋＋这样的语言中，数据类型不严格，发送器可能使用不同于期望的数据类型(如果有强数据类型机制，编译程序就会捕获这个问题)。

d) 设计约束分析:

设计约束分析评价需求强加的约束、现实世界和环境的约束、以及设计方案要求的约束。设计资料应说明所有已知或预料的对软件部件的约束。这些约束可以包括：

1) 系统分配给软件的计时和规模约束；
2) 方程式和算法约束；
3) 输入和输出数据约束；
4) 设计方案约束；
5) 传感器/执行机构准确性和校准；
6) 数字字长(量化/舍入噪声/误差)；
7) 人的因素，人的能力和局限性；
8) 物理的时间约束和响应时间；

9） 模型的有效性和控制规律与实际系统行为的符合性。

设计约束分析评价软件在这些约束下运行的能力。

e） 复杂性度量：

高度复杂的数据结构和命令结构难于彻底测试，并容易导致初始构件中或其随后的修改中有逻辑差错。通常不可能对所有路径都考虑周到或测试到，而这就可能使软件以一种非期望的方式运行。

可采用诸如 McCabe 或 Halstead 等这样一些复杂性估计技术(另见 D.8.14)。如果可以用某种自动化工具，软件设计和/或代码就可以通过该工具加以控制。如果没有自动化工具可用，就要检查详细设计的关键区域和任何初步代码的深层嵌套，检查待传送的大量参数，检查为数众多的紧张的通信路径等。

详细设计、高级语言说明、源代码和自动化复杂性测量工具都是有用的资源。

输出产品是复杂性度量、预计的差错估计、以及标识为需要进一步分析或需要考虑简化的高复杂性区域。使用这些度量时要注意，因为它们可能指示像 CASE 语句这样的结构为高度复杂的，而实际上该复杂性导致更简单更易懂的编程和维护方法，因而减少差错风险。

语言学的测量是要度量某些正文特性而不管其内容如何(例如，代码行、语句数、操作符数目和类型、标记类型和总数等)。Halstead 度量是几个这种测量方法中比较为人熟知的一个。

结构度量集中在软件内部的控制流和数据流，通常可将它们映射成一种图形表示。对连接数和/或调用数、节点数、嵌套深度等这样一些结构关系进行检查，以获得复杂性的度量。McCabe 的圈复杂性度量是用于这类复杂性评价的最有名也是使用最多的一种度量。

f） 故障树分析：

见表 C.14。

g） Petri 网：

见 D.1.1.3。

h） 马尔可夫模型：

见 D.9.4。

i） 半形式化和形式化方法：

见表 C.17 和 D.5.4。

j） 有限状态机：

见 D.1.1.2。

8.5 软件编码安全性分析

8.5.1 概述

软件编码完成软件详细设计的实现。代码应实现设计过程中开发的安全性设计特征和方法，遵循设计中提出的各种约束以及编码标准。编码安全性分析的目的是分析软件的实现是否能以相应的软件安全完整性级别满足软件安全性需求，实现代码应是可分析和可验证的，并能安全地修改。

软件编码安全性分析的输入主要包括：软件需求，结构设计的描述，详细设计描述，软件集成测试计划，软件单元测试计划和之前开展的软件安全性分析提供的信息。

软件编码安全性分析的输出可用作后续软件安全性分析的输入信息。软件编码安全性分析可提出对于后续的软件测试的建议，必要的情况下还应建议更新用户文档。在代码评审中应提交软件编码安全性分析结果，并将之反馈给进行中的系统安全性分析活动。

应采用适当的安全技术完成安全相关软件的软件编码，这些技术见表 C.4。

软件编码安全性分析包括下列任务：

a） 安全相关性分析；

b） 软件编码的安全性评价。

8.5.2 安全相关性分析

安全相关性分析的目的是分析软件代码，以便发现尚未被发现的软件危险。安全相关性分析可能会更新危险和风险分析结果，安全相关性分析确定：

a) 软件编码中是否引入了新的危险；

b) 原有的识别出的危险是否都得到了相应的控制。

8.5.3 软件编码的安全性评价

源代码应通过静态方法评价以确保与软件模块的设计规格、要求的编码标准和安全性计划要求之间的一致性，评价的工作项目包括：

a) 分析软件代码是否能溯源到软件需求，以显示对于软件需求的依从性；

b) 分析软件代码，保证其已经覆盖了软件安全性需求和软件设计的要求，软件安全性需求已经完整地由相应的代码实现；

c) 分析软件代码与软件详细设计、软件结构设计、软件安全性需求和系统安全性需求的外部一致性，以及软件详细设计的内部一致性；

d) 分析软件代码是否满足模块化、可验证、易安全修改的要求；

e) 分析软件代码是否符合支持工具和编程语言分析以及设计活动中提出的对于语言和编码标准的要求；

f) 编码应基于或限于定义清晰的符号，源代码应可读、易理解和标准化；

g) 确保充分而显著地注解安全相关的代码；应对安全性相关代码加以适当注解，以降低未来因代码变更而引起危险状态的可能性；

h) 分析软件编码中使用的技术和方法的合理性，修改与安全性相关的必要、提倡、不提倡和禁止使用的设计、编码和测试技术列表。安全性技术和措施的选取可以参考附录 C。

8.5.4 软件编码安全性分析技术

下面列出了一些可用于代码安全相关性分析的主要技术手段，其中一些是对高层的安全性分析的细化。其它的一些技术和措施见附录 C 和附录 D。

a) 代码逻辑分析:

代码逻辑分析评价已编码程序所提供的运行顺序，检测已编码软件中的逻辑差错(另见 D.8.9)。这种分析通过实施逻辑重构、方程式重构和存储器解码来进行。

逻辑重构要求根据代码编制流程图，并将它们与设计材料说明和流程图进行比较。

方程式重构通过将代码中的方程式与设计材料所提供的方程式进行比较来完成。

存储器解码识别安全相关指令序列，甚至当这些指令序列可能被伪装成数据时也要加以识别。分析员应确定是否每一条指令都有效且这些指令的执行条件也有效。存储器解码应在最后的未加工具的代码上进行。

b) 代码数据分析:

代码数据分析集中对数据结构和数据在已编码软件中的用途进行分析(另见 D.8.10)。数据分析关注如何定义和组织数据项。目的是确保恰当地定义和组织这些数据项。这是通过将代码中所有数据项的用途和值与设计材料所提供的说明进行比较来完成的。

与安全性特别相关的是确保安全相关数据的完整性，即确保安全相关数据不被无意地改变或改写。例如，检查中断处理是否干扰安全相关数据。还有，检查对申明的安全相关变量的“类型说明”。

c) 代码接口分析:

代码接口分析验证软件部件的内部和外部接口的兼容性(另见 D.8.3)。一个软件部件有许多代码段组成，这些代码段共同完成要求的作业。这些代码段必须相互通信，并与硬件、其它软件部件及操作人员通信，以便完成它们的作业。要检查是否正确地将参数传递通过接口。

所有这些接口都是潜在问题源。代码接口分析就是要验证这些接口都已得到正确实现。

d) 复杂性度量:

作为一个目标，应使软件复杂度最小，以减少差错可能性。复杂软件也更可能不稳定，或更可能经受不可预计的行为。

模块性是降低复杂度的有用技术。可以用McCabe度量和类似技术来测量复杂度(另见D.8.14)。

e) 更新设计约束分析:

应用于详细设计的设计约束分析准则可利用最后的代码加以更新。在编码阶段可能进行实际测试，以便除分析之外还描述实际软件的行为和性能。

应阐述硬件平台进行处理时的物理约束。作为这个过程的一部分还应重复进行计时、规模和吞吐量分析，以确保可用的计算资源和存储器足够安全相关功能和过程使用。

识别出由输入/输出计时、多重事件、事件顺序不对、事件失效、有害环境、死锁、错误事件、不合适的量、不正确的极性，以及硬件失效等原因可能引起的潜在的不安全状态;

某些语言中下溢/上溢导致软件产生“异常”或差错信息。如果可能，应通过设计消除这些状态;如果不能预防，那么应用中的差错处理例程就必须提供适当的响应，例如再试、重新启动等。

f) 未用代码分析:

普遍的一个实际编码差错是生成的代码逻辑上不参与执行;就是说这些代码执行的先决条件永远不会满足。这种代码是不希望的，原因有三个:

1) 它是实现软件设计时重大差错的潜在征兆;

2) 它引入不必要的复杂性并占据存储器，而这些往往是有限的资源;

3) 未用代码可能包含一些程序，如果无意中执行这些程序，就会造成危险。

没有用于识别未用代码的特殊技术;不过，未用代码常在进行其它类型代码分析的过程中被发现。在用 COTS 覆盖分析器工具进行单元测试期间可能发现未用代码。

在代码逻辑分析期间应注意确保代码的每一部分都在系统所有可能操作模式期间的某个时刻最终被检查到。

g) 故障树分析:

见表 C.14。

h) Petri 网:

见 D.1.1.3。

i) 半形式化和形式化方法:

见表 C.17 和 D.5.4。

j) 有限状态机:

见 D.1.1.2。

8.6 软件测试安全性分析

8.6.1 概述

进行安全性测试是保证软件安全性的重要手段。安全性测试不应只依靠一次总的测试，而应分层次地进行。

注 1：通常，测试包括下列层次:

a) 软件单元测试;

b) 软件集成测试;

c) 软件合格性测试;

d) 系统集成测试;

e) 系统合格性测试。

注 2：可能需要与硬件开发者紧密合作以开展系统层次的测试。

对非开发软件(现货软件、标准软件)也应进行安全性方面的测试。

测试安全性分析的目的是保证通过测试检验达到的软件安全性需求(根据要求的软件安全功能和软件安全完整性级别)。

软件测试安全性分析的输入主要包括：软件需求，结构设计的描述，详细设计描述，代码，对应测试层次的测试计划、测试集和之前开展的软件安全性分析提供的信息。

软件测试安全性分析的输出可用作后续软件安全性分析的输入信息。软件测试安全性分析可提出对于后续的软件测试和变更的建议，必要的情况下还应建议更新用户文档。在对应层次测试评审中应提交软件测试安全性分析结果，并将之反馈给进行中的系统安全性分析活动。

软件测试安全性分析既包括事前分析以保证测试有效性，又包括对测试结果的评价。软件测试安全性分析包括下列任务：

a) 测试集分析；
b) 测试覆盖分析；
c) 测试结果分析。

注：关于软件测试的方法见 GJB/Z 141。

8.6.2 测试集分析

分析测试集中的测试用例和测试通过准则，分析项目包括：

a) 分析测试用例是否覆盖了所有的安全相关软件项，包括安全相关软件部件和单元；
b) 分析测试用例是否能用来检验相应层次软件设计、编码的安全性，分析测试用例对安全性需求的追溯关系；
c) 分析测试设计是否合理可行；
d) 分析描述测试过程(步骤)是否详细，包括每条命令、每个操作；
e) 分析测试用例输入输出是否规定完整，输入输出应定量、确切地逐个列出；
f) 分析是否包含了所需的测试类型。除正常情况下的测试外，应特别考虑异常情况下的测试用例。并非在每个测试级上都要完成所有的测试，也不是对任何软件都要完成所有的测试。究竟必须执行哪些测试，应根据软件的复杂性、安全完整性级别和当前的测试级选定。一些可以考虑的测试类型如下：
   1) 功能测试；
   2) 性能测试；
   3) 强度测试；
   4) 接口测试；
   5) 可靠性测试；
   6) 安装性测试；
   7) 恢复性测试；
   8) 外部约束测试，测量此软件项的存储空间和响应时间；
   9) 边界和异常输入测试，包括使用虚假值、最大值和错误值对此软件项进行测试；
   10) 错误检测和错误恢复的测试，失效发生时能否适度降级处理；
   11) 功能多余物测试；
   12) 针对问题实际所必需的专门测试。
g) 分析测试通过准则。测试通过准则应能反映软件能正确执行所有安全性相关的功能并且不执行非预定的功能，软件测试的通过/失败准则应包括：
   1) 要求的输入信号的次序和值；
   2) 预期的输出信号的次序和值；
   3) 其它通过准则，如存储器使用、定时、值的偏差。

注 1：这不意味测试所有输入组合或输出组合。进行所有的等效类测试或基于结构的测试就可能满足要求。 边界值分析、控制流分析或潜通路分析(见附录 D)可将测试工作量减少到一个可接受的范围。如果开发过程产生可分析的程序代码，则测试要求更容易被满足。

注 2：当使用形式化方法，形式化证明或断言开发时，可适当减少测试量。也可以使用统计方法证实安全性。

8.6.3 测试覆盖分析

对于小块代码有时可能达到 100%测试覆盖(即检查代码的每一个可能的状态和路径)。但一般复杂软件无法达到 100%的测试覆盖率，因为计算机程序执行中状态变更数量太大，检查所有这些可能状态所需的时间太长。此外，常常还有大量不确定数目的环境变量，要充分测试是不可能的。

测试覆盖的要求应与安全完整性级别对应。测试覆盖分析包括下列工作项目：

a) 确定能够验证安全完整性级别要求的最佳测试覆盖率；

b) 分析是否达到了相应的测试覆盖。

8.6.4 测试结果分析

应分析测试结果以验证所有的安全性需求是否均已满足。该分析还应验证所有识别出的危险是否均已消除或已被控制到可接受的风险水平。测试结果分析包括下列工作项目：

a) 检查是否按照相应测试计划进行了对应的测试，高安全性完整性等级的软件测试应说明测试见证的情况；

b) 标识偏离“测试计划”和“测试说明”的所有测试，即指：与“测试计划”和“测试说明”相对照，其测试环境不一致的，以及额外增加的、删除的、更动的所有测试；

c) 分析测试结果的一致性、完整性；

d) 分析测试结果以验证与预期结果的符合程度，说明是否满足测试目的和测试通过准则，标识出不满足测试准则的结果；

e) 根据实测结果对该软件配置项的能力和设计质量给出全面的分析，指明其遗留的缺陷、局限性和约束情况。对于所遗留的缺陷、局限性和约束应进一步指明它们对系统安全性的影响；

f) 根据软件测试发现的问题提出对软件的修改或改进建议；

g) 对系统和软件的进一步测试提出改进建议。

8.6.5 软件测试安全性分析技术

应采用适当的技术完成安全相关软件的测试安全性分析，这些技术见表 C.5 和表 C.6。

8.7 软件变更安全性分析

8.7.1 概述

在执行任何软件变更之前，应建立软件变更规程。除非变更的性质表明明显不必要进行变更分析外，否则对于安全相关软件的已经受控的规格说明、需求、设计、代码、计划、规程、系统、环境、用户文档的任何变更都应进行安全性分析。

软件变更安全性分析的目的是分析对软件进行改正、增强或改写时是否能保持要求的软件安全完整性级别。

软件变更安全性分析包括下列任务：

a) 变更原因分析；

b) 变更影响分析；

c) 变更结果分析。

8.7.2 变更原因分析

应进行变更原因分析，确定变更是否必要。应分析错误可能招致的危害及其严重程度，并确定在目前现实的(软/硬件资源和经费)条件下，是否有可能进行此项变更。

变更的原因可能有：

a) 安全性低于规定；

b) 有系统性错误发生；
c) 新制定或修订的安全法规；
d) 对 EUC 或其使用方式的改变；
e) 系统安全要求的修改；
f) 经过对运行的性能分析，表明性能低于目标值；
g) 日常的安全性审核和校验测试发现的问题。

8.7.3 变更影响分析

应分析对软件的任何变更对安全性的影响，以确定：
a) 是否可能导致系统或软件的质量特性退化；
b) 对其它系统部件/单元、文档的影响，包括：
   1) 对安全功能的完成有不利影响；
   2) 改变软件项的安全完整性级别；
   3) 若作此项变更，则哪些软件产品需要作相应的变更。
c) 是否需要进行危险和风险分析，包括：
   1) 变更是否会引起危险状态；
   2) 变更是否会影响危险控制；
   3) 变更是否会增加危险状态发生的可能性。
d) 是否需要重复某些软件生存周期活动，包括重复设计和重复验证活动；
e) 对工程进度和经费可能会有多大影响。

所有对软件的安全性有影响的变更应重复软件生存周期中的适当活动，活动中的安全性分析也应根据本指导性技术文件对相应活动的规定规程执行。

8.7.4 变更结果分析

应分析变更结果，保证变更按照预期目标完成，包括：
a) 所有受影响软件的原定错误均已被纠正；
b) 所有受影响软件的功能和性能均未退化；
c) 系统的功能和性能亦未退化；
d) 变更没有引入新的错误；
e) 分析所有受变更活动影响的文档信息，确保任何受到影响的文档均被相应地更新，以正确反映已进行的变更；
f) 变更是否根据影响分析的结果和软件安全完整性级别进行；
g) 是否对变更后的软件进行了充分的回归测试；
h) 未作任何未授权的变更。

8.7.5 软件变更安全性分析技术

应采用适当的技术完成安全相关软件的变更安全性分析，这些技术见表 C.8。

9 软件安全性分析报告

9.1 概述

软件安全性分析报告应由供方提供，它是对于预定的运行环境下软件产品安全性质量的合格性保证文件。在软件交付前，供方应向需方提交软件安全性分析报告，需方应做出接受或不接受该分析报告的决定。软件安全性分析报告也可做为其它报告的一部分。

软件安全性分析报告一般包括如下方面的内容：
a) 安全性环境；
b) 软件安全性保证；

c) 软件安全性证据；
d) 软件残留风险及控制。

9.2 安全性环境

软件安全性环境宜包括下列内容：

a) 对软件所在系统的简介；
b) 软件所在系统预计的运行环境；
c) 软件所在系统可能遇到的危险列表；
d) 软件在系统中的作用，软件功能的介绍；
e) 软件失效可能产生的危险；
f) 可容忍风险水平的说明。

9.3 安全性保证

供方对软件消除和控制危险的措施的说明和对软件安全性质量特性的保证。包括：

a) 软件的安全功能；
b) 软件安全功能的安全完整性等级；
c) 其它利于安全性的软件质量特性。

9.4 安全性证据

供方应提出证据证明安全保证的可信性、合理性、充分性。安全性证据可分为四类：

a) 过程证据；
b) 历史记录证据；
c) 测试证据；
d) 分析证据。

9.4.1 过程证据

可提供过程证据证明软件开发过程利于保证软件安全性，过程证据可包括：

a) 软件开发过程规范，符合相关的标准，列举进行的软件质量保证活动；
b) 开发机构通过相关认证，软件开发人员专业技能合格；
c) 软件开发过程中使用的开发方法与开发工具可靠、适用；
d) 在软件开发过程中保证了软件安全性需求的落实，列举软件开发过程中完成的软件安全性工作活动。

9.4.2 历史记录证据

历史记录证据包括软件的失效记录，维护记录。在软件已存在和运行了一段时间后，可用历史记录来获取安全性置信度。如果用于证明交付的软件中包含的非开发软件的安全性，则只有在和当前的运行环境类似的环境中获取的软件失效记录和维护记录才有效。非开发软件在其它安全相关系统中的使用记录也可用作证据。

9.4.3 测试证据

测试证据是指与安全相关的软件测试文档，包括测试记录、测试报告等的总结。测试证据宜说明：

a) 安全性测试对于软件安全性需求的覆盖情况；
b) 安全性测试的通过情况；
c) 测试见证的情况。

9.4.4 分析证据

分析证据是根据软件开发过程中所进行的各项安全性分析任务的结果而得到的安全性证据。这个证据不是所有分析结果的简单罗列，而是这些分析的结果的一个组织良好的、易理解的归纳总结。分析证据宜说明：

a) 软件开发过程中依据本指南实施的软件安全性分析工作项目；

b) 安全性分析中采用的分析技术和方法、措施；

c) 安全性分析结果的总结及这些结果对于安全性质量的证明。

9.5 软件残留风险及控制

软件供方应说明软件安全性分析中识别出的，但是没有完全消除的风险。说明残留风险对系统预期作用的影响，提出在使用和维护中跟踪、控制残余风险的方法建议。

软件供方应做出对软件安全性分析过程中未识别风险的估计。

附 录 A
（资料性附录）
风险和安全完整性的基本概念

A.1 风险和安全完整性

A.1.1 必要风险降低

风险被定义为危险的可能性和严重性的组合。必要风险降低是指为了保证在特定情况下不超过可容忍风险(可能用定性或定量的方式说明) 必须要降低的风险。必要风险降低的概念在开发 PE 安全相关系统的安全性需求方面非常重要(特别是安全性需求中的安全完整性部分)。确定特定危险事件的可容忍风险的目的是分别说明危险事件的频率(或可能性)和其特定后果确实是合理的。安全相关系统应设计为减小危险事件的频率(或可能性)和/或危险事件的后果。

注 1：为了达到可容忍风险，需要确定必要风险降低。在 B.2 和 B.3 中描述了两种定性的方法。

注 2：定量的说明方式的例子：导致一个特定后果的危险事件应该不会在 10^8 小时内出现 1 次以上。

可容忍风险应根据多种因素(如伤害的严重程度、暴露在危险中的人数、一人/多人暴露在危险中的频率和持续时间)确定。暴露在危险中的人的感觉也是一个因素。在分析一个特定应用的可容忍风险时，应考虑下列信息：

a) 相应安全管理部门的规定；
b) 参与应用项目的不同团体的讨论与协议；
c) 国家标准、军用标准、行业标准和指南；
d) 国际协议和标准；
e) 咨询机构提供的专门建议；
f) 法律要求，包括公共的和直接与特定应用相关的法律。

A.1.2 PE 安全相关系统

PE 安全相关系统通过满足必要风险降低来达到可容忍风险的要求。

安全相关系统：

a) 实现使 EUC 达到或维持安全状态所必需的安全功能；
b) 依靠其自身或其它 PE 安全相关系统、其它技术安全相关系统或外部安全设施获得所要求的安全功能的必需的安全完整性。

注 1：a)描述了安全相关系统必须实现安全功能需求中规定的安全功能。 例如，安全功能需求中可能指出当温度达到 x，阀 y 应打开允许水流入管道中。

注 2：b)描述了安全相关系统应以与应用相适应的置信度的来实现安全功能，以获得可容忍风险。

人可能会是 PE 安全相关系统的一个不可分割的部分，比如，一个人可通过屏幕显示来获取 EUC 状态的信息，并根据该信息完成安全操作。

PE 安全相关系统可分为在低频率运行模式下和高频率或连续运行模式下运行，见 3.1.22。

A.1.3 安全完整性

安全完整性定义为在规定的条件下、规定的时间内，安全相关系统成功实现所要求的安全功能的可能性。安全完整性与安全相关系统执行安全功能的效能有关(执行的安全功能列在安全功能需求规格说明中)。

安全完整性可认为由硬件安全完整性和系统性的安全完整性两个因素组成。

硬件安全完整性与在危险失效模式下的随机硬件失效有关。硬件安全完整性级别能够被估计到一个合理的精确程度，因此安全完整性需求可以使用概率组合的常用方法在子系统之间分配。可能需要使用

冗余结构来获得足够的硬件安全完整性。

系统性的安全完整性与在危险失效模式下的系统性失效有关。尽管可以估计与系统性失效有关的平均失效率，但是，从设计缺陷和共因失效获得的失效数据表明失效分布难以预测。这样便增加了特定情况下失效概率计算的不确定性(例如，安全相关防护系统的失效概率)。因此需做出选择，以便用最佳技术来将不确定性最小化。注意，不要认为减少硬件随机失效的措施总是能对系统性失效产生同样的效果。用同样硬件实现冗余通道之类的技术在控制随机硬件失效方面非常有效，但在减少系统性失效方面的作用则非常有限。

PE 安全相关系统、其它技术安全相关系统和外部安全设施所要求的安全完整性必须具有相应等级以保证安全相关系统的失效频率足够低，从而防止危险事件频率超过可容忍风险的要求，和／或安全相关系统将失效后果控制在可容忍风险要求的范围内。

图 A.1 说明风险降低的一般概念。这个模型假定：

a) 有一个 EUC 和一个 EUC 控制系统；

b) 有相关人员因素；

c) 安全防护特性包括：

1) 外部安全设施；

2) PE 安全相关系统；

3) 其它技术安全相关系统。

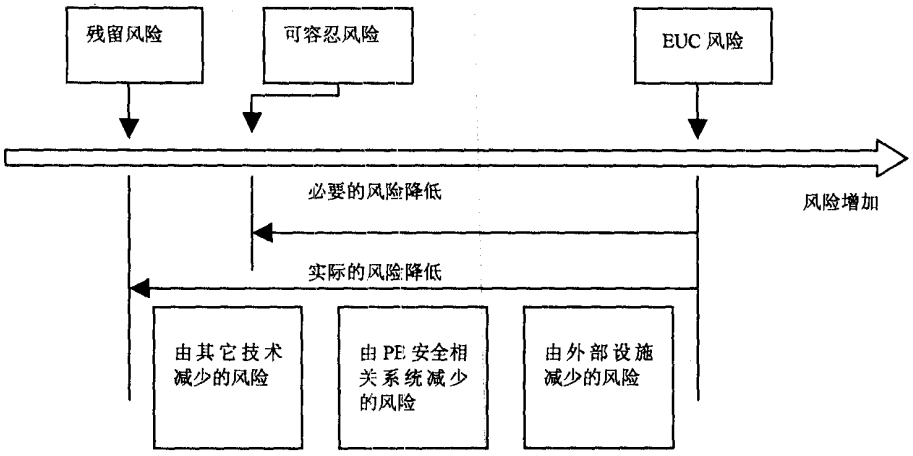


图 A.1 关于风险降低的概念示意图

注：图 A.1 是说明一般原理的风险模型。为特定应用开发的风险模型，应考虑 PE 安全相关系统和／或其它技术安全相关系统和／或外部安全设施实际获得必要风险降低采用的特定方式。因此，最终的风险模型可能与图 A.1 所示的不同。

图 A.1 中表示的各种风险为：

EUC 风险：和 EUC、EUC 控制系统及人员因素特定危险事件相关的风险——在确定该风险时未考虑特定安全防护特性，EUC 风险是一种与 EUC 本身有关的风险，同时要考虑 EUC 控制系统的风险降低。为防止对 EUC 控制系统不合理的安全完整性要求，本指导性技术文件对这种要求进行了限制；

可容忍风险：根据当今社会价值水平所能接受的风险；

残留风险：在本指导性技术文件中，残留风险是采用外部安全设施、PE 安全相关系统和其它技术安全相关系统后，仍存在的和 EUC、EUC 控制系统、人员因素特定危险事件相关的风险。

必要风险降低是通过所有安全防护特性的组合实现的。图 A.1 表示了从 EUC 风险的起始点达到特定可容忍风险目标的必要风险降低。

A.1.4 风险和安全完整性的关系

正确区分风险和安全完整性是非常重要的。风险是对一个特定危险事件出现的可能性和频率的测度，可以对不同情形的风险进行评价(EUC 风险、可容忍风险目标的风险、实际风险(见图 A.1))。可容忍风险要根据社会基础和有关社会和政治因素确定。安全完整性只适用于 PE 安全相关系统、其它技术安全相关系统和外部安全设施，并作为这些系统/设施在特定安全功能方面取得必要风险降低的可能性的测度。一旦设定可容忍风险和估算出必要风险降低，就可分配安全相关系统的安全完整性要求。

注：分配有必要迭代进行，以使设计最优化来满足各种要求。

安全相关系统在获取必要风险降低方面所起的作用由图 A.1 和 A.2 来说明。

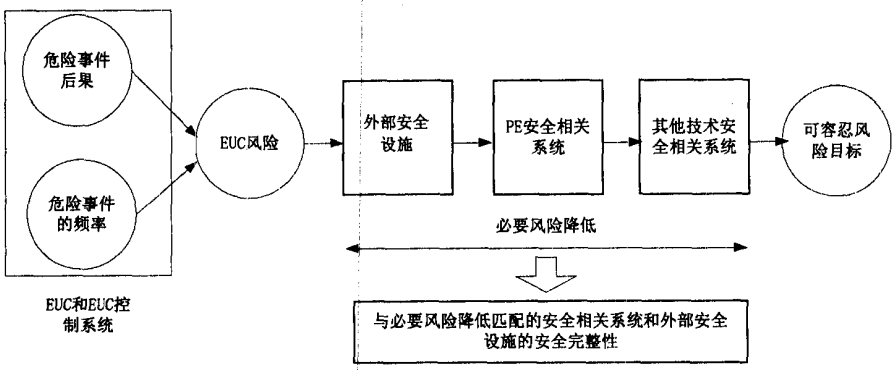


图 A.2　风险和安全完整性概念

A.1.5　安全完整性级别

为适应安全相关系统需获得的范围广泛的必要风险降低，需要定义完整性级别作为描述分配到安全相关系统的安全功能的安全完整性要求的途径。安全完整性要求应规定 PE 安全相关系统的安全完整性级别。软件安全完整性级别是规定安全相关软件安全功能的安全完整性要求的基础。

本指导性技术文件中，规定了四级安全完整性级别，安全完整性级别 4 为最高，安全完整性级别 1 为最低。

正文中的表 1 和表 2 给出了四级安全完整性级别的目标失效测度。本指导性技术文件规定了两组参数，一个用于低频率运行模式的安全相关系统，另一个用于高频率运行模式或连续运行模式的安全相关系统。

注：对于运行在低频率运行模式下的安全相关系统，感兴趣的安全完整性度量是按要求执行其设计功能的失效概率。对于运行在高频率运行模式或连续运行模式下的安全相关系统，感兴趣的安全完整性度量是每小时危险失效的平均概率。

A.1.6　安全性需求分配

在 PE 安全相关系统、其它技术安全相关系统和外部安全设施之间的安全性需求分配(安全功能和安全完整性需求)，在 7.5 中已经进行了描述。

为 PE 安全相关系统、其它技术安全相关系统和外部安全设施分配安全完整性需求的方法，根据对必要风险降低是以数字形式精确规定，还是以定性的方式提出要求，大体分为两类，分别称为定量方法和定性方法(见 B.1、B.2 和 B.3)。

A.2　ALARP 和可容忍风险

A.2.1　ALARP 模型

ALARP(即 As Low As Reasonably Practicable，合理可行前提下尽可能低)原理概述了可被应用于调节风险的主要方法，并且明确了与确定以下内容有关的行为：

a)　风险非常大，必须完全排除；或

b)　风险(或已经被控制的)非常小，可以认为无关紧要；或

c)　风险介于上述 a)和 b)之间，并已被降低到合理可行的水平，应考虑接受风险带来的利益和任

何进一步减小风险所需的成本。

对于 c)，ALARP 原理要求任何风险必须在合理的可行前提下尽可能的降低。如果风险介于两个极限值之间(即不可接受的区域和广泛可接受的区域)并且应用了 ALARP 原理，那么对该特定应用该风险是可容忍风险。图 A.3 表示了这三种区域。

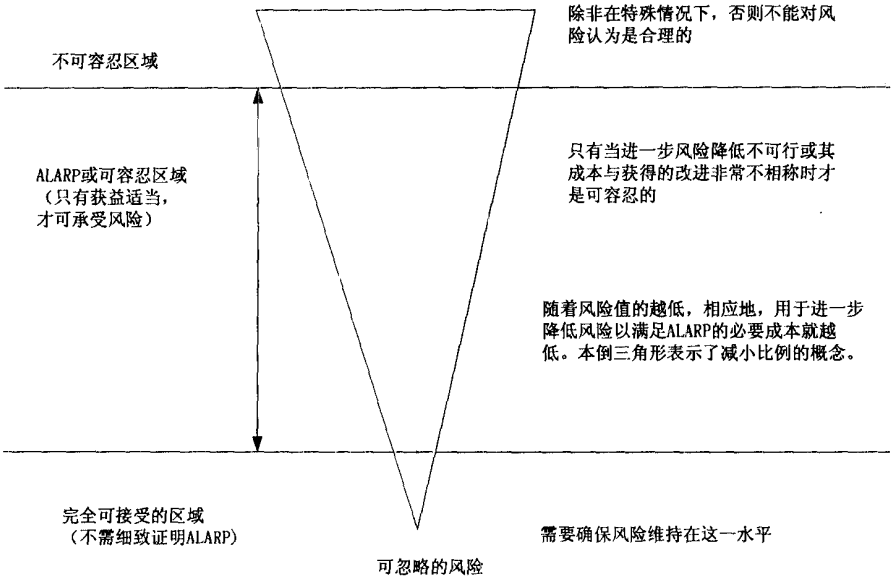


图 A.3　可容忍风险和 ALARP

超过特定风险水平，则风险被认为是不可容忍的，并且不能在任何通常情况下被认可。

低于该水平，存在一个可容忍区域，在此范围内可采取有关措施使风险降低到合理的可行水平。这里的可容忍不同于可接受：它表明这样一种愿望，即可承受某风险以节约成本，同时又期望对它保持检查并在可能时减小它。在此，为了权衡成本和需求(或为了其它附加安全措施)，进行成本－利益评估是必要的。风险越高，则用于减少风险的相应期望成本就越大。在可容忍值的极限，应论证与利益不成比例的总成本。

当风险较不明显时，为减小风险只需承担对应的较小代价，在可容忍区域的低端，应该存在成本与利益的平衡点。

低于可容忍区域，风险的水平被认为是非常不明显，不必要求进一步改善。这是一个广泛可接受区域，在此区域内风险小于我们每天会实际经历的风险，不需要细致工作来证明 ALARP。但需保持警惕将风险维持在该水平。

ALARP 的概念可以用于定性或定量的风险目标。A.2.2 简述了基于定量风险目标的方法(B.1 简述了一种定量方法，B.2 和 B.3 简述了针对特定危险确定必要风险降低的定性方法。在决策时，可以结合 ALARP 概念选择相应的方法)。

A.2.2　可容忍风险目标

获得可容忍风险目标的一个方法是对于一系列要被确定的后果，分配可容忍频率。后果与可容忍频率的匹配应由各方(如安全管理机构、可能造成风险的人员和承受风险的人员)讨论并达成一致。

基于 ALARP 概念，可以通过后果与可容忍频率的匹配来定义风险等级。表 A.1 是基于一系列后果和频率，确定四级风险(1、2、3、4)等级的示例。表 A.2 使用 ALARP 概念解释了每一风险等级。四级等级根据图 A.3 规定。这些风险等级定义中的风险是指采取风险降低措施后仍存在的风险。根据图 A.3，风险等级如下：

a)　等级 1 处于不可容忍区域；

b)　等级 2 和 3 处于 ALARP 区域，等级 2 刚好在 ALARP 区域内；

c） 等级 4 处于完全可接受区域。

对每一特殊情况或相应工业领域，考虑社会、政治和经济等综合因素，应开发类似于表 A.l 的一个表。分析风险的后果和频率，与表中的相应风险等级匹配。如表 A.1 中频繁发生的频率可能指事件频率大于每年 10 次。其严重后果可能是个体死亡，或大量严重伤害，或严重职业病。

表 A.1 关于意外风险等级的示例

频 率	后 果			
	大灾难	严重	不严重	可忽略
频繁发生	1	1	1	2
很可能发生	1	1	2	2
偶尔发生	1	2	3	3
极小可能发生	2	3	3	4
不可能发生	3	3	4	4
难以相信会发生	4	4	4	4

注 1：风险等级 1、2、3、4 的实际情况将依赖于图 A.3 扇形区域，还依赖于实际频率是频繁、可能或其它。因此应将本表看作是一个说明如何计算此类表的示例而不是对未来应用的规范。

注 2：从本表中的频率确定安全完整性级别在 B.1 中有说明。

表 A.2 风险等级解释

风险等级	解 释
等级 1	不可容忍风险
等级 2	不期望风险，只有当风险降低不可行或成本与取得的改善严重不相称时才为可容忍的风险
等级 3	可容忍风险，在风险降低的成本超过取得的改善的条件时
等级 4	可忽略风险

附 录 B
（资料性附录）
确定安全完整性级别的方法示例

B.1 确定安全完整性级别的一种定量方法

B.1.1 概述

本条描述了如何通过采用一种定量方法来确定安全完整性级别，并说明如何使用表 A.1 中的信息。定量方法具有两类特定的数字表示的数值：

a) 以数字表示的形式规定了可容忍风险（例如，一个特定的后果不能以大于 10^4 年 1 次的频率出现）；

b) 对安全相关系统的安全完整性级别详细规定了数字表示的目标（见表 1 和表 2）。

如果风险模型为图 A.1 和 A.2 中说明的模式，则更适用。

B.1.2 一般方法

说明一般原理的模型见图 A.1。方法中的关键步骤如下，这些步骤需要针对由 PE 安全相关系统实现的每一安全功能来实施：

a) 用类似表 A.1 的表来确定可容忍风险；

b) 确定 EUC 风险；

c) 确定满足可容忍风险的必要风险降低；

d) 将必要风险降低分配到 PE 安全相关系统、其它技术安全相关系统和外部安全设施。

表 A.1 中给出了风险频率，并可用数字表示可容忍风险目标（F_t）。

与 EUC 存在的风险（包括 EUC 控制系统和人员因素，并且没有任何防护特性）有关的频率可使用定量风险评估方法进行估计。没有防护特性存在时危险事件可能出现的频率 F_{np} 是 EUC 风险的两个构成部分之一；另一构成部分是危险事件的后果。

注：F_{np} 可通过三种途径确定：由分析可比较情形的失效率得到；由有关数据库的数据得到；使用适当的预估方法计算得到。

本指导性技术文件对可由 EUC 控制系统声明的最小失效率设置了约束。如果声明 EUC 控制系统应具有与最小失效率相比更小的失效率，则 EUC 控制系统可认为是安全相关系统，并应当满足本指导性技术文件中对安全相关系统的所有要求。

B.1.3 计算示例

图 B.1 提供一个计算单一安全防护系统的目标安全完整性的示例。对该情况，见式（B.1）：

$$PFD_{avg} \leqslant F_t / F_{np} \quad \cdots\cdots (B.1)$$

式中：

PFD_{avg}（Average of Probability of Failure on Demand）——每次运行时的安全相关防护系统的平均失效概率，即针对在低频率运行模式下运行的安全相关防护系统的安全完整性失效的测度；

F_t——可容忍风险频率；

F_{np}——没有安全相关防护系统的风险频率。

由于 F_{np} 与 PFD_{avg} 的关系，从而影响到安全防护系统的安全完整性级别的确定，可以看出确定 EUC 的 F_{np} 是重要的。在全部必要风险降低是通过单一安全防护系统（该安全防护系统必须将最小危险率从 F_{np} 减小至 F_t）获得的情况下，获取安全完整性级别（当后果 C 保持不变时）的必要步骤如下（图 B.1 所示）：

a) 确定不考虑任何防护特性的 EUC 风险的频率因素 F_{np}；

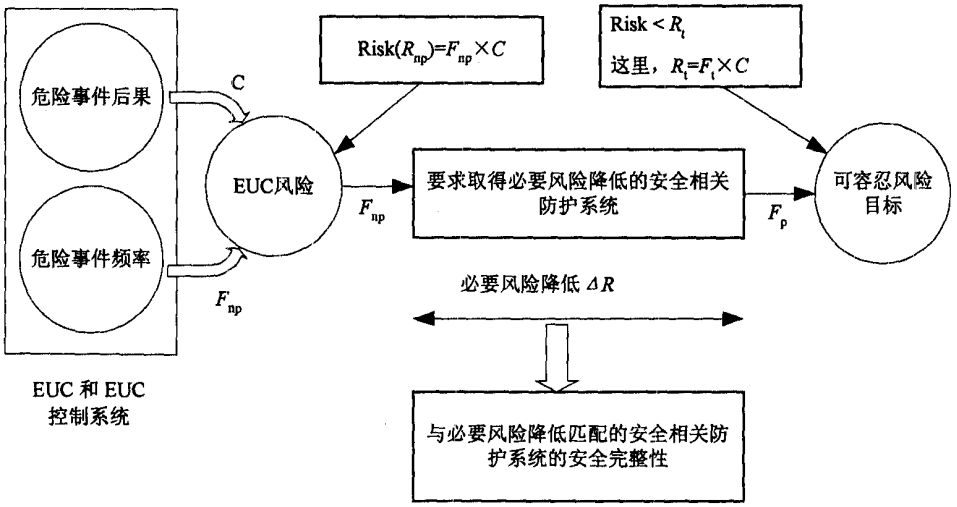
b) 确定不考虑任何防护因素的后果 C；

c) 通过表 C.1，确定根据频率 F_{np} 和后果 C 能否获得可容忍风险等级。如通过使用表 A.1 得到风险等级 1，则要求进一步的风险降低。风险等级 4 或者 3 是可容忍风险。风险等级 2 则需进一步调查；

注：表 A.1 用于检查是否需要进一步的风险降低措施，可能不采用附加防护因素也能获得可容忍风险。

d) 确定满足必要风险降低（ΔR）安全防护系统每次运行时的失效概率（PFD_{avg}）。对于所描述的特定情况中的后果 C 为常量的情形，$PFD_{avg}=(F_t / F_{np})=\Delta R$；

e) 对于 $PFD_{avg}=(F_t / F_{np})$，可通过表 1 确定安全完整性级别（如对于 $PFD_{avg}=10^{-2}$ 至 10^{-3}，安全完整性级别为 2）。



C—危险事件的后果；F_p—具有防护特性的风险频率

图 B.1 安全完整性分配：安全相关防护系统示例

下面给出一个载人航天宇航员逃逸系统的例子：

基于社会、政治和经济等综合因素，开发类似于表 A.1 的一个表。

载人航天系统属于低频率运行模式，考虑其投资规模、政治和社会因素，可以确定后果 C 是“大灾难”；

由于全球范围的载人航天的历史很短，风险数据非常少，但我们可以得出载人火箭在上升阶段爆炸会“偶尔发生”的印象。可以假定，没有特殊防护特性的风险 F_{np} 为 2%。

通过使用表 A.1，我们可以得到没有特殊防护特性的风险等级为 1。由此可见，我们必须进行进一步的风险降低，于是产生了对“宇航员逃逸系统”安全相关系统的需求。

对于具有“宇航员逃逸系统”的载人航天整体系统：

根据对于运行模式的分析，“宇航员逃逸系统”属于低频率运行模式；

如果系统失效，则宇航员会丧生，后果 C 可视为常量；

设定可容忍风险频率 F_t 为 10^{-5}，即要达到 99.99999%的成功率，也就是 10 万次只允许失败 1 次；

前面已经确定没有防护特性的风险频率 F_{np} 为 2%。

那么，根据公式

$PFD_{avg}=F_t / F_{np}$

可以计算出：$PFD_{avg}=F_t / F_{np}=5*10^{-4}$

由此，根据表 1（低频率运行模式），可以得出如下结论：

“宇航员逃逸系统”的安全完整性级别为3。

B.2 确定安全完整性级别的一种定性方法：风险图

B.2.1 概述

本条描述了一种风险图方法，这种方法可以通过对与EUC和EUC控制系统有关的风险因素的了解来确定安全相关系统的安全完整性级别。当风险模型为如图A.1和A.2所描述的那样时，这种方法特别有效。

当采用定性方法时，为了简化问题，采用一些参数来描述当安全相关系统失效或不可用时危险情况的性质。从四组风险参数中每组确定一个参数，然后，四个参数组合起来确定分配到安全相关系统的安全完整性级别。这些参数：

a) 允许对产生的风险进行有意义的分级；

b) 包括风险评估的关键因素。

B.2.2 风险图合成

以下简化程序根据式(B.2)：

$$R=f\times C \cdots\cdots (B.2)$$

式中：

R——没有适当安全相关系统时的风险；

F——没有适当安全相关系统时的危险事件的频率；

C——危险事件的后果(后果可能与健康和安全相关的伤害或环境破坏带来的伤害有关)。

在这种情况下，危险事件的频率f可认为由三种相关因素组成：暴露在危险区域的时间和频率；避免危险事件的可能性；没有任何附加安全相关系统时的危险事件发生的概率(但有适当的外部安全设施)，它被定义为不期望情况的概率。

这样，产生了下列四个风险参数：

a) 危险事件的后果(C)；

b) 暴露在危险区域的时间和频率(F)；

c) 未能避开危险事件的可能性(P)；

d) 不期望情况的概率(W)。

B.2.3 其它可能风险参数

以上规定的风险参数被认为一般已足够处理各种应用情况。但可能还有要求在某方面引入附加风险参数的应用。例如，EUC或EUC控制系统中新技术的使用。附加参数的目的是更加精确地估计必要风险降低(见图A.1)

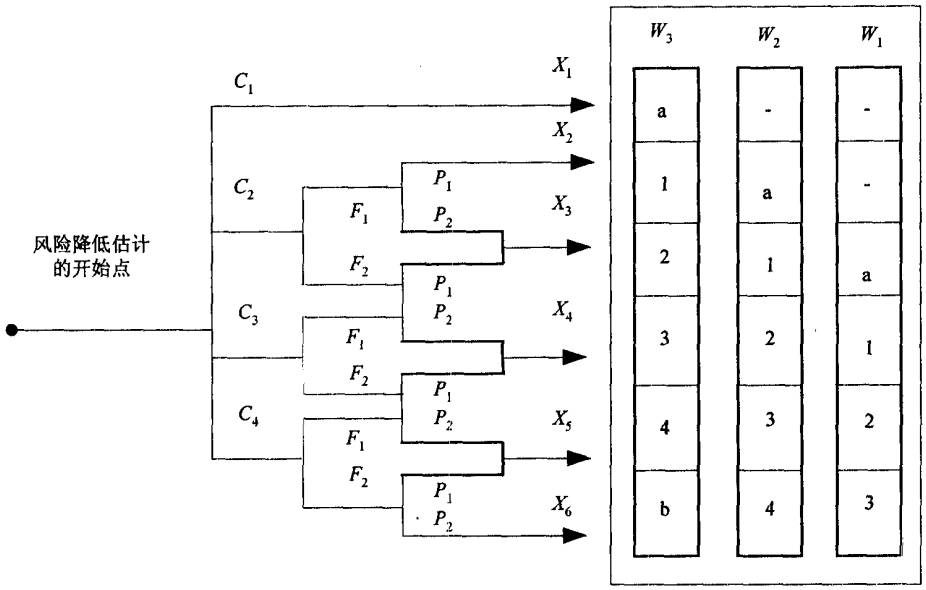
B.2.4 风险图实现：一般框图

以上描述的风险参数的组合使得可以开发如图B.2中所示的风险图。在图B.2中，$C_1<C_2<C_3<C_4$；$F_1<F_2$；$P_1<P_2$；$W_1<W_2<W_3$。对于此风险图的解释如下：

a) 使用风险参数C、F和P产生一系列数据输出X_1，X_2...X_n(准确的数据依赖于风险图使用的特定应用领域)。图B.2表示的是没有强调考虑更加严重后果的情况。每一个输出都映射为三种刻度之一(W_1、W_2、W_3)。这些刻度上的每一点都表示必需的安全完整性，由相应PE安全相关系统满足。实际应用中，会出现单一PE安全相关系统不足以提供必要风险降低的情况。

b) 进行到W_1、W_2和W_3的映射时，允许考虑其它风险降低措施的作用。W_1、W_2和W_3的刻度的偏移特征可表示源自其它措施的三种不同水平的风险降低。W_3表示由其它措施带来的最小风险降低(即不期望发生情况的最高概率)，W_2表示中等风险降低，W_1表示最大风险降低。对于风险图的特定中间输出量(即X_1，X_2...或X_6)和特定W刻度(即W_1、W_2或W_3)，风险图的最后输出给出PE安全相关系统的安全完整性级别(即1、2、3或4)，并且是对该系统要求的风险

降低的测度。该风险降低，与由 W 刻度机制所代表的其它措施(如其它技术安全相关系统和外部安全设施)获得的风险降低一起，共同给出了特定情况下的必要风险降低。



图中：

C—后果（风险参数）；F—暴露时间和频率（风险参数）；

P—未能避开危险的概率（风险参数）；W—不期望情况的概率；

-—无安全性要求；a—没有特别安全性要求；

b—一个安全相关系统不足以满足安全要求；

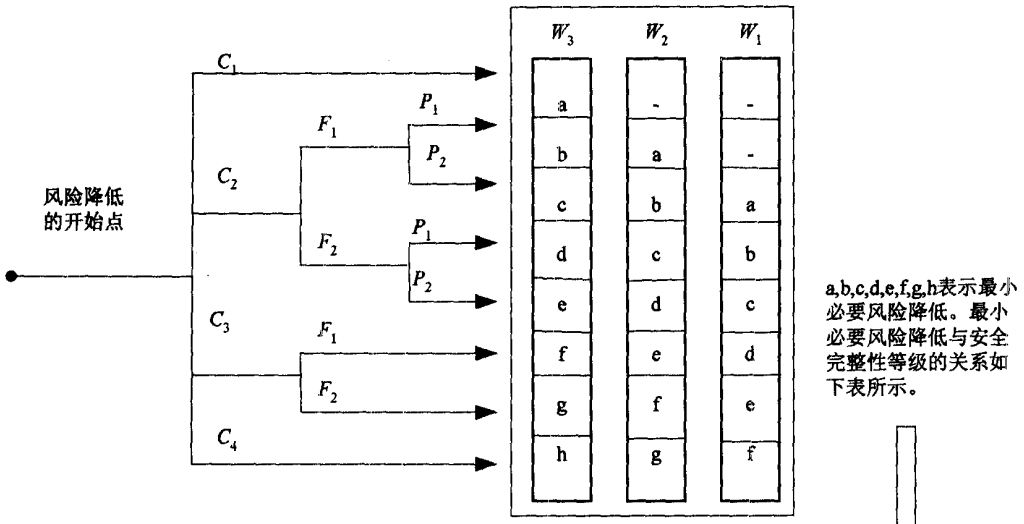
1、2、3、4——安全完整性级别

图 B.2　风险图：一般框图

图 B.2 中所示的参数(C_1、C_2、C_3、C_4、F_1、F_2、P_1、P_2、W_1、W_2、W_3)和其权重，在特定情况下或特定工业领域中需要被准确定义；在实际实现中，参数值在图中的排列是特定于风险图所适用的应用的。

B.2.5　风险图示例

图 B.3 表示了根据图 B.2 中的数据的风险图示例，示例中的有关数据示例见表 B.1。使用风险参数 C、F 和 P 产生八种输出之一。每一输出变换到三个刻度(W_1、W_2、W_3)上。这些刻度上的每一点(a，b，c，d，e，f，g，h)都是安全相关系统需满足的必要风险降低的指示。



最小必要风险降低	安全完整性等级
—	无安全性要求
a	没有特别安全性要求
b,c	1
d	2
e,f	3
g	4
h	一个安全相关系统不足以满足安全要求

C_1～C_4—后果(风险参数)； F_1、F_2—暴露时间和频率(风险参数)； P_1、P_2—未能避开危险的概率(风险参数)；
W_1～W_3—不期望情况的概率； a, b, c, d, e, f, g, h—针对安全相关系统的必须风险降低的估计

图 B.3 风险图：示例(只说明一般原理)

表 B.1 风险图示例中的有关数据示例(图 B.3)

风险参数		分类	备注
后果(C)	C_1	微小伤害	a) 此处的分类基于对人的伤害或死亡进行。需开发其它分类方法表示对环境的破坏或物质破坏。 b) 对于 C1、C2、C3、C4 的解释，应考虑意外的后果和一般的康复情况。
	C_2	对一人或多人的严重永久伤害；一人死亡	
	C_3	几人死亡	
	C_4	大量人员死亡	
在危险区域中的频率和暴露时间(F)	F_1	很少至较多暴露在危险区域	此处的分类基于对人的伤害或死亡进行。需开发其它分类方法表示对环境的破坏或物质破坏。
	F_2	经常至永久暴露在危险区域	
避开危险事件的可能性(P)	P_1	在一定条件下可能	这些参数考虑了： a) 过程的操作(被监控(即由专业或非专业人员操作)或不被监控)； b) 危险事件的发展速度(如突然、快速或缓慢) c) 识别危险的简易性(如立即看到、通过技术方法探测或不需通过技术方法探测)； d) 危险事件的避免(如可能的逃生路线、在一定条件下可能或不可能)； e) 实际安全经验(这类经验存在于同一 EUC 或相似 EUC，或可能不存在)。
	P_2	几乎不可能	

表 B.1(续)

风险参数		分类	备注
不期望情况的概率(W)	W_1	遇到不期望情况的概率非常小并且只有很少不期望情况有可能出现	a) W 的目的是估计在没有任何附加安全相关系统(PE 或其它技术系统)，但包括任何外部安全设施的情况下，不期望事件发生的频率 b) 如果对于 EUC 或 EUC 控制系统、或类似 EUC 和 EUC 控制系统只有少量经验或没有经验，W 的估计可能通过计算得出。在这种情况下应作出最坏的预测。
	W_2	遇到不期望情况的概率小并且只有少量不期望情况有可能出现	
	W_3	遇到不期望情况的概率相对高并且不期望情况有可能频繁出现	

B.3 确定安全完整性级别的一种定性方法：危险事件严重性矩阵

B.3.1 危险事件严重性矩阵

当风险(或风险的频率)不能定量时，B.1 中描述的数学方法则不适用。危险事件严重性矩阵方法是一种定性方法，使得根据对 EUC 或 EUC 控制系统有关的风险因素的了解就能确定 PE 安全相关系统的安全完整性级别。当风险模型为图 A.1 和 A.2 所示时，这种方法特别适用。

以下条件是危险事件严重性矩阵的基础，是方法的有效性的必要条件：

a) 安全相关系统(PE 和其它技术系统)与外部安全设施是独立的；

b) 每一安全相关系统(PE 和其它技术系统)和外部安全设施都认为是提供图 A.1 中所示的部分风险降低的保护层；

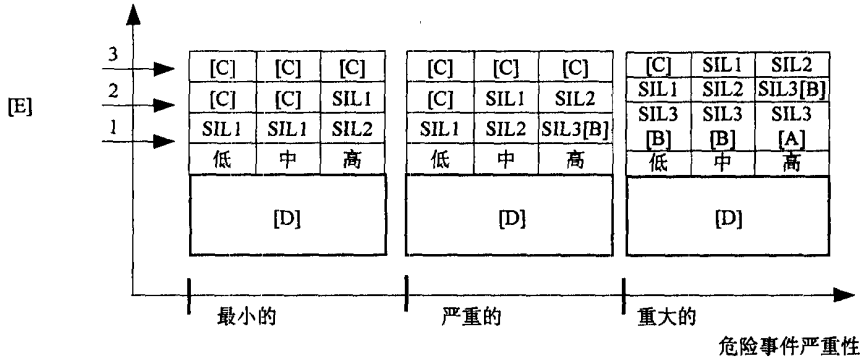
注 1：只有保护层的定期校验测试被执行时，该假定才有效。

c) 当增加一个保护层时(见上一要求)，安全完整性则获得一个数量级的提高；

注 2：只有当安全相关系统和外部安全设施具有足够的独立水平时，该假定才有效。

d) 在该方法建立必需的安全完整性级别时，只使用唯一一个 PE 安全相关系统(但也可以与另一技术安全相关系统和 / 或外部安全设施结合)。

基于上述考虑，可产生图 B.4 所示危险事件严重性矩阵。应注意该矩阵只是通过计算示例数据来说明一般原理。对每一特定情况，或类似工业领域，应开发类似于图 B.4 的矩阵。



注：[A] 表示一个 SIL3 PE 安全相关系统不能在该水平上提供足够的风险降低，需要附加风险降低措施；

[B] 表示一个 SIL3 PE 安全相关系统不能在该水平上提供足够的风险降低，要求进行危险和风险分析以确定是否需要附加的风险降低措施；

[C] 表示可能不需要一个独立的 PE 安全相关系统；

[D] 表示事件概率，是在没有任何安全相关系统或外部安全设施下危险事件出现的可能性；

[E] 表示独立安全相关系统和外部安全设施的数目，事件概率和独立保护层的数量是由与特定应用领域来定义的。

图 B.4 危险事件严重性矩阵示例(只说明一般原理)

附 录 C
（资料性附录）
技术和措施选择导则

本附录中的一些表格与正文对应，如 8.1(软件需求安全性分析)与表 C.1 有关；一些表格则被另外的表格所引用，如表 C.12 被表 C.5 引用。

本附录提及的技术和措施见附录 D。

表 C.1～表 C.19 中的每一技术和措施都有针对安全完整性级别 1 到 4 的建议。各种建议的含义如下：

a) HR：对于该安全完整性级别极力建议的技术或措施。如未采用这种技术或措施则应在安全计划中详细记录未使用其的原因并须经评估者同意；

b) R：对于该安全完整性级别建议的技术或措施(建议程度低于 HR)；

c) ——：对于该安全完整性级别不建议或不能使用的技术或措施；

d) NR：在该安全完整性级别下绝对不推荐的技术或措施。如果采用这类技术或措施应在安全计划中详细记录使用其的原因并须经评估者同意。

应根据安全完整性级别选择适当的技术/措施；确定附录 C 中提及的完整性级别的方法，见附录 B。本附录的表中，通过相同的数字编号后跟英文字母来标识替代或等价的技术/措施，一般选用一种替代或等价的技术/措施即可。

在其它因素相同的前提下，HR 类的技术不论是在软件开发中防止系统性错误(针对软件结构的情况)还是在软件执行中控制残留错误方面都比 R 类技术更加有效。

由于影响软件安全完整性的因素很多，所以不太可能给出一种对于任何特定应用都适合的技术和措施的组合。如果某项特定的技术没有在表中列出，不应认为其已经被排除在考虑之外。

对于特殊的应用，除非在表中的注释作出了其它要求，在安全计划中应陈述适当的技术或措施的组合，以及选择的适当的技术或措施。

表 C.1 软件需求分析

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 计算机辅助规格说明工具	D.1.2	R	R	HR	HR
2a 半形式化方法	表 C.17	R	R	HR	HR
2b 形式化方法，例如：CCS、CSP、LOTOS、OBJ、时态逻辑、VDM 和 Z	D.5.4	—	R	R	HR
注 1：软件需求安全性分析技术另见 8.1.5。 注 2：软件安全性需求规格说明需使用自然语言和任何能反映应用的必要的数学符号对问题进行描述。 注 3：本表反映了应清楚和精确地规定软件安全性需求规格说明的额外要求。					

表 C.2 软件结构设计

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 错误探测和诊断	D.6.1	—	R	HR	HR
2 错误检测和纠错编码	D.6.2	R	R	R	HR
3a 失效断言程序设计	D.6.3	R	R	R	HR
3b 安全袋技术	D.6.4	—	R	R	R
3c 软件多样性	D.6.5	R	R	R	HR
3d 恢复块	D.6.6	R	R	R	R

表 C.2(续)

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
3e 向后恢复	D.6.7	R	R	R	R
3f 向前恢复	D.6.8	R	R	R	R
3g 重试故障恢复机制	D.6.9	R	R	R	HR
3h 记录执行情况	D.6.10	—	R	R	HR
4 适度降级	D.6.11	R	R	HR	HR
5 人工智能故障纠正	D.6.12	—	NR	NR	NR
6 动态重配置	D.6.13	—	NR	NR	NR
7a 结构化方法，包括如 JSD、MASCOT、SADT 和 Yourdon	D.5.1	HR	HR	HR	HR
7b 半形式化方法	表 C.17	R	R	HR	HR
7c 形式化方法，包括如 CCS、CSP、LOTOS、OBJ、时态逻辑、VDM 和 Z	D.5.4	—	R	R	HR
8 计算机辅助规格说明工具	D.1.2	R	R	HR	HR
注 1：软件结构设计安全性分析技术另见 8.2.7。 注 2：表中有关容错(错误控制)的措施应同硬件中的结构要求和错误控制结合起来考虑。					

表 C.3 软件支持工具和编程语言

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 适合的编程语言	D.7.6	HR	HR	HR	HR
2 强类型编程语言	D.7.1	HR	HR	HR	HR
3 语言子集	D.7.2	—	—	HR	HR
4a 经过认证的工具	D.7.3	R	HR	HR	HR
4b 工具：通过使用提高置信度	D.7.4	HR	HR	HR	HR
5a 经过认证的翻译程序	D.7.3	R	HR	HR	HR
5b 翻译程序：通过使用提高置信度	D.7.4	HR	HR	HR	HR
6 可信的/经验证的软件模块和部件库	D.7.5	R	HR	HR	HR

表 C.4 软件详细设计和编码

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1a 结构化方法，包括如 JSD、MASCOT、SADT 和 Yourdon	D.5.1	HR	HR	HR	HR
1b 半形式化方法	表 C.17	R	HR	HR	HR
1c 形式化方法，包括例如 CCS、CSP、LOTOS、OBJ、时态逻辑、VDM 和 Z	D.5.4	—	R	R	HR
2 计算机辅助设计工具	D.2	R	R	HR	HR
3 防卫性编程	D.5.5	—	R	HR	HR
4 模块化方法	表 C.19	HR	HR	HR	HR
5 设计和编码标准	表 C.11	R	HR	HR	HR
6 结构化程序设计	D.5.7	HR	HR	HR	HR
7 使用可信的/经验证的软件模块和部件	D.5.10 D.7.5	R	HR	HR	HR
注 1：软件详细设计安全性分析技术另见 8.4.6。 注 2：软件编码安全性分析技术另见 8.5.4。					

表 C.5 软件测试和软件集成

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 概率测试	D.8.1	HR	HR	HR	HR
2 动态分析和测试	D.4.2 表 C.12	R	HR	HR	HR
3 数据记录和分析	D.8.2	—	R	R	HR
4 功能和黑盒测试	D.3.1 D.3.2 表 C.13	R	R	HR	HR
5 性能测试	D.8.20 表 C.16	—	R	HR	HR
6 接口测试	D.8.3	HR	HR	HR	HR
注：应根据安全完整性级别选择适当的技术/措施。					

表 C.6 系统集成

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 功能和黑盒测试	D.3.1 D.3.2 表 C.13	HR	HR	HR	HR
2 性能测试	D.8.20 表 C.16	R	R	HR	HR

表 C.7 软件确认

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 概率测试	D.8.1	—	R	R	HR
2 模拟 / 建模	表 C.15	R	R	HR	HR
3 功能和黑盒测试	D.3.1 D.3.2 表 C.13	—	R	R	HR

表 C.8 软件变更

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 影响分析	D.8.23	HR	HR	HR	HR
2 模拟 / 建模	D.8.23	HR	HR	HR	HR
3 重新验证变更的软件模块	D.8.23	R	HR	HR	HR
4 重新验证受影响的软件模块	D.8.23	—	R	HR	HR
5 软件配置管理	D.8.24	HR	HR	HR	HR
6 数据记录和分析	D.8.2	HR	HR	HR	HR

表 C.9 软件验证

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 形式化证明	D.8.13	—	R	R	HR
2 概率测试	D.8.1	—	R	R	HR
3 静态分析	D.4.1 表 C.18	R	HR	HR	HR
4 动态分析和测试	D.4.2 表 C.12	R	R	R	R
5 软件复杂性度量	D.8.14	R	R	R	R
注：在软件安全生命周期的早期阶段，验证是静态的，例如通过审查、评审、形式化证明。当代码产生后，动态测试变得可行。验证需要两类信息的综合：静态方法对软件模块的代码验证包括软件审查、走查、静态分析、形式化证明等技术；动态方法的代码验证包括功能测试、白盒测试、统计测试。通过两类证据的结合提供每一软件模块满足相应规定的保证。					

表 C.10　安全性评估

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　检查单	D.1.3	R	R	R	R
2　判定 / 真值表	D.9.1	R	R	R	R
3　软件复杂性度量	D.8.14	R	R	R	R
4　失效分析	表 C.14	R	R	HR	HR
5　不同软件的共同原因失效分析	D.9.3	—	R	HR	HR
6　可靠性方框图	D.9.5	R	R	R	R

表 C.11　设计和编码标准

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　编码标准使用	D.5.6.2	HR	HR	HR	HR
2　禁止使用动态对象	D.5.6.3	R	HR	HR	HR
3a　禁止使用动态变量	D.5.6.3	—	R	HR	HR
3b　在线检查动态变量的建立	D.5.6.4	—	R	HR	HR
4　限制使用中断	D.5.6.5	R	R	HR	HR
5　限制使用指针	D.5.6.6	—	R	HR	HR
6　限制使用递归	D.5.6.7	—	R	HR	HR
7　高级语言程序中不使用无条件跳转	D.5.6.2	R	HR	HR	HR
注：如果使用的编译器保证所有动态变量和对象在运行时前分配足够内存或插入运行时检查以便正确地在线分配内存，则不需应用措施 2 和 3a。					

表 C.12　动态分析和测试

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　边界值分析及其测试用例执行	D.8.4	R	HR	HR	HR
2　错误推测及其测试用例执行	D.8.5	R	R	R	R
3　错误散播及其测试用例执行	D.8.6	—	R	R	R
4　性能建模	D.8.20	R	R	R	HR
5　等价类别和输入分区测试	D.8.7	R	R	R	HR
6　基于结构的测试	D.8.8	R	R	HR	HR
注：针对测试用例的分析应在子系统级别上进行，并基于规格说明和/或代码来进行。					

表 C.13　功能测试和黑盒测试

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　因果图及其测试用例执行	D.4.3.1	—	—	R	R
2　原型设计 / 动画	D.8.17	—	—	R	R
3　边界值分析	D.8.4	R	HR	HR	HR
4　等价类别和输入分区测试	D.8.7	R	HR	HR	HR
5　过程模拟	D.8.18	R	R	R	R
注 1：针对测试用例的分析应在软件系统级别上进行，并只根据规格说明来进行。 注 2：模拟的完整性将依赖于安全完整性级别、复杂性和应用。					

表 C.14　失效分析

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1a　因果图	D.4.3.1	R	R	R	R
1b　事件树分析	D.4.3.2	R	R	R	R
2　故障树分析	D.4.3.4	R	R	HR	HR
3　失效模式、影响和危险程度分析	D.4.3.3	R	R	HR	HR
4　蒙特－卡洛模拟	D.9.6	R	R	R	R
注：为了确定软件最合适的软件安全完整性级别，应进行初步危险分析。					

表 C.15　建模

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　数据流图	D.5.2	R	R	R	R
2　有限状态机	D.1.1.2	—	R	HR	HR
3　形式化方法	D.5.4	—	R	R	HR
4　性能模型	D.8.20	R	HR	HR	HR
5　时间 Petri 网	D.1.1.3	—	R	HR	HR
6　原型设计/动画	D.8.17	R	R	R	R
7　结构图	D.5.3	R	R	R	HR
注：如果某项特定的技术没有在表中列出，不应认为其已经被排除在考虑之外。					

表 C.16　性能测试

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　强度测试	D.8.21	R	R	HR	HR
2　响应时间和存储限制	D.8.22	HR	HR	HR	HR
3　性能要求	D.8.19	HR	HR	HR	HR

表 C.17　半形式化方法

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1　逻辑/功能块图	见下注	R	R	HR	HR
2　顺序框图	见下注	R	R	HR	HR
3　数据流图	D.5.2	R	R	R	R
4　有限状态机/状态变换图	D.1.1.2	R	R	HR	HR
5　时间 Petri 网	D.1.1.3	R	R	HR	HR
6　判定/真值表	D.9.1	R	R	HR	HR
注：GB/T 1526－1989 和 IEC61131－3 中有逻辑/功能块图和顺序框图的技术描述。					

表 C.18 静态分析

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 边界值分析	D.8.4	HR	HR	HR	HR
2 检查单	D.1.3	HR	HR	HR	HR
3 控制流分析	D.8.9	R	HR	HR	HR
4 数据流分析	D.8.10	—	R	HR	HR
5 错误推测	D.8.5	HR	HR	HR	HR
6 菲根(Fagan)检查法	D.8.15	HR	HR	HR	HR
7 潜通路分析	D.8.11	—	—	R	R
8 符号执行	D.8.12	R	R	HR	HR
9 走查／设计评审	D.8.16	HR	HR	HR	HR

表 C.19 模块方法

技术/措施	参见	SIL1	SIL2	SIL3	SIL4
1 软件模块规模限制	D.5.9	HR	HR	HR	HR
2 信息隐藏/封装	D.5.8	R	HR	HR	HR
3 参数数量限制	D.5.9	R	R	R	R
4 子程序和函数的单入口/单出口	D.5.9	HR	HR	HR	HR
5 完全定义所有接口	D.5.9	HR	HR	HR	HR
注：可能使用任何一项单独的技术不能满足要求，应考虑所有合适的技术。					

附 录 D
（资料性附录）
技术和措施概述

D.1 PES需求安全性分析

D.1.1 半形式化方法

D.1.1.1 概述

目的：为了清楚地和一致地表达规格说明的各部分，以利于检测出某些错误和遗漏，证明设计满足其需求规格说明。

描述：半形式化方法为在系统开发的某个阶段(例如，规格说明、设计或者编码)编制该系统的描述文档提供了一种方法。在某些情况下，可用机器分析该描述，或者为了显示系统各方面的行为，可把该描述制作成动画。此动画可对系统满足实际要求以及规定的要求提供额外的置信度。

D.1.1.2 有限状态机/状态转换图

目的：为了模型化、详细设计或者实现系统的控制结构。

描述：许多系统能够用状态、输入和动作来描述。例如，某系统处于状态 s1 时，在接受到输入 1 后，系统将会执行动作 A 并转换到状态 s2。通过描述一个系统处于每个状态、每个输入导致该系统的动作，就能彻底地描述一个系统。产生的系统模型叫做有限状态机。常常是把它画成一个所谓的状态转换图，此图显示了系统怎样从一个状态转移到另一个状态；或者把它画成一个矩阵，在矩阵中，它的两维是状态和输入，矩阵单元包含当系统处于给定状态中时，由接受输入导致的动作和新状态。

对于一个复杂系统，或者具有一个自然结构的系统，可以用一个分层的有限状态机反映出来。

被采用有限状态机来表达的规格说明或者设计，应检查分析如下三个方面：

a) 完整性(在每个状态，对每个输入系统必须有一个动作和新状态)；

b) 一致性(每个状态/输入对只描述一次状态改变)；

c) 可达性(是否能通过任何输入序列从一个状态到达另一状态)。

这些方面是关键系统的一些重要属性。很容易开发检查它们的支持工具。也存在用于验证一个有限状态机实现或者制作有限状态机模型的动画的算法，这种算法可自动生成测试用例。

D.1.1.3 时间 Petri 网

目的：给系统行为的有关特征建模；通过分析和重新设计来评价或可能的情况下提高安全及操作上的要求。

描述：Petri 网属于图形理论模型一类，它适用于在呈现并发性和有异步行为的系统中表示信息和控制流。

Petri 网是一个库所(place)和变迁(transition)网。库所可被“标记”或“未标记”。当一个变迁的所有输入库所被标记时，就“使能”该变迁。当变迁被使能时，就被允许(而不是强迫)“启动”。如果它启动，引起变迁的输入库所就变成未标记了，并且代之以变迁产生的每个输出库所被标记。

在模型中可把潜在的危险表示成特定的状态(标记)。Petri 网模型可以被扩展以使系统具有时间特征。“传统的”Petri 网技术集中关注控制流方面的模型，但把数据流包括到模型中去是可行的。

D.1.2 计算机辅助的规格说明工具

D.1.2.1 概述

目的：使用形式化的规格说明技术以便自动检测歧义性和完备性。

描述：本技术可产生一个数据库形式的规格说明，这种形式可被自动检查从而评估一致性和完备性。规格说明工具能把为用户指定的系统的各种特征制作成动画。可根据以下条目对规格说明工具进行分

类。

D.1.2.2 普通规格说明工具

目的：通过提供提示和各相关部分间的连接，帮助用户编写一份好的规格说明。

描述：规格说明工具可从用户手中接管一些日常工作并支持项目管理。它不强求任何特殊的方法。在方法方面的相对独立性允许用户在建立规格说明时有很大的自由度，但同时只能为用户提供少量的所需专门支持。这使得熟悉系统更为困难。

D.1.2.3 带层次分析的面向模型方法

目的：通过保证各抽象层次的行为和数据的描述之间的一致性，帮助用户编写一份好的规格说明。

描述：本方法以各种抽象层次(精确程度)给出了期望的系统(结构化的分析)的功能描述。在各个层次下的分析对行为和数据两者都起作用。在层次之间和同一层次的两个功能单元(模块)之间评价歧义性和完备性是可能的。

D.1.2.4 实体模型

目的：通过集中在系统中的各实体以及它们之间的关系上，帮助用户编写一份好的规格说明。

描述：把期望的系统描述成一些对象和它们之间的关系的一个集合。工具使我们能确定系统能解释哪些关系。通常，这些关系允许描述对象、数据流的层次结构，数据之间的关系，以及哪些数据须隶属于一些生产过程。为了用于过程控制，已对传统的程序进行了扩充。检查能力和对用户的支持与所说明的各种关系有关。另一方面，大量可能的表达方式可能使本技术的应用变得很复杂。

D.1.2.5 激励和响应

目的：通过识别激励－响应关系，帮助用户编写一份好的规格说明。

描述：系统的各对象之间的关系用“激励”和“响应”符号表示方式来描述。使用了一种简单和容易的扩充语言，这种语言包含有代表对象、关系、特性和结构的语言元素。

D.1.3 检查单

目的：为了引起对系统各重要问题的注意并管理系统各重要方面的认证或者评价，同时确保生命周期阶段全面的覆盖而不用放弃精确的需求。

描述：由执行检查单的人来回答一系列问题。许多问题具有普遍性，评估者对这些问题应以最适合于正在评估的特定系统来解释它们。检查单可用于整个软件安全生命周期的各阶段并特别适合作为帮助功能安全评估的一种工具。

为了适应正在确认的系统的变化，大多数检查单包含了能适用于许多类型的系统的一些问题。如果使用的检查单中的问题和正在讨论的系统无关，那么，对于一个特定的系统就需要补充一个标准的检查单，这个检查单包含的问题是专门针对正在讨论的系统的。

在任何情况下，检查单的使用关键取决于工程师选择和应用检查单的专门知识和判断能力。工程师针对选择的检查单采取的决定和任何附加的或者多余的问题都应全部编制成文档并证明是合理的。目的是要保证能评审检查单的应用并且保证使用相同的准则都应能达到同样的结果。

在完成一份检查单时，目标要尽可能简明。当有必要进一步证明时，应通过引用附加文件来进行这种证明。编写每个问题的结果，应使用通过、失败和不确定或者一些类似的受限的应答集合。这种简洁方式大大简化了获得有关检查单评价结果的全部结论的过程。

D.2 PES 设计和开发

对于计算机辅助设计工具：

目的：更系统地执行设计过程；包括已经可用的和经测试过的合适的自动的构造要素。

描述：在硬件和软件设计过程中，当依据系统的复杂性认为使用计算机辅助设计工具是合理的并且工具可用的时候，应使用计算机辅助设计工具(CAD)。应通过专门测试、广泛和长时间的成功应用历史或者单独认证来证明这种工具应用于设计安全相关系统的正确性。

D.3 PES 集成

D.3.1 功能测试

目的：揭示在规格说明和设计阶段产生的失效。避免在软件和硬件的实现和集成阶段产生失效。

描述：在功能测试过程中，应执行评审以判断系统是否已达到规定的特性。提供给系统的输入数据应足以刻画正常预期的工作特性。观察输出并把它们的响应同规格说明给出的响应作对比。同规格说明的偏差和关于不完整的规格说明的指示说明应被编制成文档。

D.3.2 黑盒测试

目的：检验实际功能条件下的动态行为。揭示失效以便满足功能规格说明和评价实用性和健壮性。

描述：在一个规定的环境中，利用特定的测试数据(这些数据是系统性地根据制定的准则从规格说明中导出的)执行一个系统或者程序的功能。这将揭露出系统的行为并允许同规格说明进行比较。不需使用系统内部结构的知识来指导测试。测试的目的是要确定功能单元是否能正确执行规格说明要求的所有功能。形成等价类别的技术是黑盒测试数据准则的一个例子。借助规格说明，可把输入数据空间分成特定的输入值范围(等价类)。于是由下列情况构成各种测试用例：

a) 来自允许范围的数据；
b) 来自不允许范围的数据；
c) 来自范围边界的数据；
d) 极值；
e) 以上各类数据的组合。

使用其它准则选择各种测试活动(模块测试、集成测试和系统测试)中的测试用例也是可能的。

D.4 PES 安全性确认

D.4.1 静态分析

目的：为了避免系统性故障，此故障可导致受试系统在工作多年后迟早会发生事故。

描述：这种系统性的和可能的计算机辅助的方法可检查原型系统的特定静态特性，从而可保证所讨论的需求的完备性、一致性和无歧义性(例如构造准则、系统规格说明)。静态分析是可再现的，它适用于已达到良好定义的完成阶段的原型。对于硬件和软件的静态分析的一些例子有：

a) 数据流的一致性分析(比如测试一个数据目标是否在任何地方都被解释成同一值)；
b) 控制流分析(比如确定通路，确定不可访问代码)；
c) 接口分析(比如检查各种软件模块之间的变量传递)；
d) 用于检测创建、引用和删除变量的可疑序列而进行的数据分析。

D.4.2 动态分析

目的：通过检查处于高度完成阶段的原型的动态行为来检测规格说明中的问题。

描述：通过把拟订的工作环境下的一些典型的输入数据加给安全相关系统的一个接近正式运行版本的原型，从而可以进行安全相关系统的动态分析。当观察到的安全相关系统的行为同要求的行为一致时，分析获得满意的结果。应纠正安全相关系统的任何失效，然后重新分析新的运行版本。

D.4.3 失效分析

D.4.3.1 因果图

目的：以一种图形方式建立一个系统中可能形成的事件序列的模型，此事件序列是基本事件组合的结果。

描述：此技术可看成是故障树和事件树的一种组合。从一个关键事件开始，一幅因果图被向后和向前描绘。在后向描绘中，因果图相当于一个具有关键事件作为顶事件的事件树。在正向描绘中，由一个事件产生的可能后果被确定。图形包含顶点符号，这些符号描述了从顶点开始沿各分支传播的条件；也能包括时间延时；还可用故障树来描述这些条件。传播路线能够与逻辑符号结合在一起，从而使图形更

简洁。应该定义一个标准的符号集以供因果图使用。可用这些图来计算发生某些关键后果的概率。

D.4.3.2 事件树分析

目的：用图形方式建立事件序列的模型，这种事件序列是在一个初始化事件之后，在一个系统中开发出来的；并因此也为了指示可能产生的严重后果。

描述：在图顶部写上紧随启动事件之后的相继事件有关的序列条件。在序列中从初始化事件下面开始(它是分析的目标)到第1个条件，画一条线。在那里，图分成“Yes(是)”和“No(否)”两个分支，它们描述了后来事件对条件的依从情况。对这两个分支来说，每个都以类似的方式继续到下一条件。但不是所有的条件都和所有分支有关。分支继续到序列的末端，并且按这种方式构造的树的每个分支代表一个可能的后果。根据序列中的条件的概率和数目，可用事件树来计算各种后果的概率。

D.4.3.3 失效模式、影响和关键性分析

目的：确定在设计或者运行过程中需要特别关注哪些部件和采取哪些必要的措施，评定部件的关键性等级。因为某些部件的单点失效可能危害、损坏系统或者使系统功能下降。

描述：有许多评定关键性等级的方法。确定关键性的一种很简单的方法是部件失效的概率乘以可能出现的损坏；这种方法类似于简单的风险因素评价法。

D.4.3.4 故障树分析

目的：帮助分析将会导致危险的严重结果的事件或事件组合。

描述：从一个可能引起危险或者严重后果的直接原因的事件(“顶事件”)开始，沿一条树路径执行分析。使用逻辑操作符(与，或等)描述原因的组合。按同样的方法分析中间原因等，在分析停止处返回到基本事件。

本方法是基于图形的，使用了一组标准化的符号来画故障树。本技术原来主要用来分析硬件系统，但也可用于软件失效分析。

D.5 软件需求和详细设计

D.5.1 结构化方法

D.5.1.1 概述

目的：结构化方法的主要目的是通过把注意力集中到生命周期的早期阶段来提高软件开发的质量。本方法旨在通过精确的和直观的过程和符号表示(计算机辅助的)来达到该目的，从而可用逻辑顺序和结构化方式确定需求和实现特征并把它们编制成文档。

描述：存在许多结构化方法。一些方法被设计用于传统的数据处理和事务处理功能，而另一些方法(MASCOT、JSD、实时 Yourdon)更适于过程控制和实时应用(趋向于更关键的安全性)。

结构化方法实质上是系统地考察和分类问题或系统的一些“思维工具”。它们的主要特征如下：

a) 逻辑思维的次序，把一个大的问题分解成可管理的一些阶段；

b) 总系统(包括环境及要求的系统)的分析和文件编制；

c) 所要求系统数据和功能的分解；

d) 检查单，例如，需要分析的项目的已排序清单；

e) 低的智力负担:简单、直观、实用。

分析和文档化问题与系统实体(例如过程和数据流)的支持符号趋向于准确，而表示由这些实体执行的处理功能的符号则趋向于更加非形式化。但某些方法部分地使用了(数学上的)形式表示法(例如，JSD使用了正则表达式；Yourdon 使用了有限状态机器)。精确性的提高不仅缩小了错误理解的范围，还提供了自动处理的范围。

结构化表示法的另一好处是它们的可视性，它使用户能够根据自己的丰富知识直观检验一个规格说明或者设计。

这里描述了5种结构化方法：受控的需求表示、杰克逊系统开发(JSD)、MASCOT、实时 Yourdon

及结构化分析和设计技术(SADT)。

D.5.1.2 受控的需求表示(CORE)

目的：保证能确定和表示所有的需求。

描述：本方法用来在需方／最终用户和分析员之间连接一座桥梁。它在数学上并不严密但有助于交互——CORE(Controlled Requirements Expression)是为需求表达而不是规格说明而设计的。本方法被构建完成时，需求表示要经历各个求精级别。CORE 法鼓励对问题的各种看法，引入系统使用的环境知识以及各种类型用户的不同观点。CORE 包括识别背离“主要设计”的原则和战术。可以校正或者明显地标识这些背离并把它们归档。因而，规格说明可能不完全，但可检测未辨别出的问题和高风险区域，它们是在其后的设计中必须要考虑的。

D.5.1.3 杰克逊系统开发(JSD)

目的：包括从需求一直到编码的一种软件系统(特别着重于实时系统)开发方法。

描述：JSD(Jackson System Development)是一种分阶段的开发过程，在这个过程中开发者将构建真实世界行为的模型，系统功能则以这些行为为基础，并且确定所需的功能并把它们插入模型，以及把产生的规格说明变换成目标环境中可实现的规格说明。因此它包括规格说明和设计及开发的传统阶段，但采用了有些与传统方法不同的观点，即它不是自动向下的。

而且，它特别着重于在初始阶段发现真实世界中的实体，该真实世界与正在建立的系统有关并且关系到构造它们的模型以及它们可能发生什么情况。一旦完成对“真实世界”的这种分析并建立起一个模型，就能分析系统所需功能，从而确定怎样把它们纳入到这个真实世界模型中。产生的系统模型随描述模型中的所有过程而不断增大，并且整体被变换成将在目标软件和硬件环境中运行的一些程序。

D.5.1.4 软件构建、运行和测试的模块化方法(MASCOT)

目的：实时系统的设计和实现。

描述：MASCOT(Modular Approach to Software Construction,Operation and Test)是一种由一个程序设计系统支持的设计方法。它是表示实时系统结构的一种系统方法，其表示方法与目标硬件或实现语言无关。它强调把一种规定的方法运用于设计，以产生高度模块化的结构，从而保证出现在系统集成中的构造部件和设计中的功能部件之间的严密一致性。通过并发进程网络来设计系统，该网络可通过一些通道进行通信；这些通道既可以是固定数据池，也可是队列(数据流水线)。对通道访问的控制基于访问机制的方式描述，独立于进程，同时为进程施加了调度规则。

MASCOT 支持的接受策略基于单软件模块和与软件模块有关的较大功能集合的测试和验证。MASCOT 实现通常建立在一个 MASCOT 内核之上，该内核具有支持实现和存取机制的一组调度原语。

D.5.1.5 实时 Yourdon(Real-time Yourdon)

目的：实时系统的规格说明和设计。

描述：作为本技术基础的开发方案假定采用三阶段进化来开发系统。第一阶段包含一个“基本模型”的建立，此模型描述了系统必需的行为。第二阶段包含一个描述结构和机制(即当实现时，具体化必需的行为)的实现模型的建立。第三阶段包括以硬件和软件组成的系统的实际建立。这三个阶段大体相当于传统的规格说明和设计及实现阶段，不过，鼓励开发者更侧重于建模活动。

基本模型有两部分：

a) 环境模型：包含系统和它的环境之间的边界的描述和系统必须作出响应的外部事件的描述；

b) 行为模型：包含描述系统响应事件进行的变换的安排以及为了响应系统必须保持的数据的描述。

实现模型还分成一些子模型，它们包含分配给处理机的各个进程以及这些进程分解成的软件模块。

为了捕获这些模型，本技术结合了许多其它众所周知的技术：数据流图、变换图形、结构化英语、状态转换图和 Petri 网。

D.5.1.6 结构化分析和设计技术(SADT)

目的：用一种图形化方式使用信息流，对与一个复杂系统相关的管理任务和决策过程建模和分析。

描述：在 SADT(structured Analysis and Design Technique)中，活动因素(activity-factor，A/F)图的概念起一种核心作用。一幅 A/F 框图由所谓"动作框"组织的一些活动构成。每个动作框有一个唯一的名称，并通过因素关系(画成箭头)与其它动作框相连接，每个因素关系还有一个唯一的名字。可把每个动作框按层次分解成一些子动作框和子关系。有 4 种类型的因素：输入、控制、机构和输出：

a) 输入：用从左手边进入动作框的箭头表示。输入可以是具体的或者抽象的事物，它们适合于被动作框中的一个或几个活动来操纵；

b) 控制：典型地是一条指令、一个操作规程、一个选择准则等。一个控制可指导一个活动的执行，它用进入一个动作框上部的箭头来表示；

c) 机构：为执行任务的活动所必需的资源，比如人员、组织单元或设备等；

d) 输出：一个活动产生的任何事物，它用从右手边离开动作框的箭头来表示。

当这些活动相互间由许多因素关系强相关时，也许最好把这些活动考虑成包含在一个动作框内的一个不可分的组中，其内容不用太细化。把活动组织到动作框的指导性原则是，最终的动作框只由很少的因素成对地连结。

A/F 框图的模型分层将继续进行下去，直到动作框的进一步细化变得毫无意义为止。当框内的活动不可再分，或者当动作框的进一步细化超出系统分析范围之外时，就达到了本技术的目标。

D.5.2 数据流图

目的：用一个图形方式描述一个程序的数据流。

描述：数据流图记录了数据输入是怎样变换成输出的，图中的每一步表示一个不同的变换。

数据流图由三种组成元素构成：

a) 带注释的箭头：代表进出变换中心的数据流，注释记录了是什么数据；

b) 带注释的圆圈：代表变换中心，注释记录了变换；

c) 操作符(and、xor)：这些操作符被用来连接带注释的箭头。

数据流图中的每个圆圈可看作是一个独立的黑盒，只要它的输入可得到，它就可把这些输入变换成它的输出。主要的优点之一就是它们在显示出变换时不用对怎样实现这些变换作任何假定。一幅纯数据流图并不包括控制信息或者顺序信息，但是这样的要求也可以通过扩充符号得到满足，例如，像实时 Yourdon 里的实时符号。

考查系统输入并朝系统输出方向进展是制作数据流图的最佳解决办法。每个圆圈必须代表一个可区分的变换：它的输出应与它的输入在某些方面有所不同。不存在确定框图的总体结构的规则，构建一幅数据流图是系统设计中最需要创造力的任务之一。同所有设计一样，它是一个迭代过程，通过对初期尝试逐步求精产生最后框图的方式进行。

D.5.3 结构图

目的：用图形显示程序的结构。

描述：结构图是一种符号表示法，此法可补充数据流图。结构图描述了程序设计系统和各部分的一个分层结构，并用树型图来显示分层。结果图能够记录如何把数据流图的组成元素按程序单元的一种层次结构来实现。

结构图显示了程序模块之间的关系而不包含激活这些单元的次序的任何信息。结构图使用了以下 4 种符号：

a) 矩形：带有模块名的注释；

b) 线条：用线条来连接创建这些矩形从而建立结构的；

c) 带圆圈的箭头(圆圈是空心的)：带有输入和输出结构图中各组成元素的数据名称的注释(通常带圆圈的箭头平行于连接图中各矩形的线)。

d) 带圆圈的箭头(圆圈是实心的)：带有从结构图中的一个模块到另一模块的控制信号名称的注

释，箭头也是平行于连接图中的各矩形的线。

从不显而易见的数据流图，有可能导出许多不同的结构图。

数据流图描绘了系统中的信息和功能之间的关系。结构图描绘了元素组成系统的方式。从系统的观点来看，虽然并不相同，但两种技术都是有效的。

D.5.4 形式化方法

D.5.4.1 概述

目的：采用基于数学的一种方法来开发软件。它包括形式化设计和形式化编码技术。

描述：形式化方法提供了一种在系统规格说明、设计或实现的某个阶段开发一个系统的描述的方法。产生的描述采用一种严密的符号表示法，它可通过数学分析检测各种类型的不一致性或者不正确性。此外，在某些情况下可用机器分析该描述，其精确性类似于使用一台编译器对一个原程序进行语法检查的精度，或者可把该描述制作成动画从而显示所描述的系统各方面的行为，动画可给出对系统满足真实要求及形式上规定的要求的额外置信度，这是因为它提高了人们对规定行为的识别能力。

一种形式化方法通常将提供一种表示法(通常使用离散数学的某种形式)和一种用于该表示法的描述技术，以及用来检查不同正确性属性的描述的各种形式的分析。

常用的形式化方法有：CCS、CSP、LOTOS、OBJ、时态逻辑、VDM 和 Z。注意，其它的技术，比如有限状态机和 Petri 网，根据该技术符合严格数学基础的程度，也可以认为是形式化方法。

D.5.4.2 通信系统微分(CCS)

目的：CCS(Calculus of Communicating Systems)是描述和揭示并行的通信进程系统行为的一种方法。

描述：CCS 是关于系统行为的一种数学微分。系统设计被建模为一个这样的网络——由顺序运行或者并行运行的独立进程构成的网络。这些进程可通过端口(类似于 CSP 的通道)进行通信，只有在两个进程准备就绪时才能进行通信。对于非确定性也能建模。从整个系统的一个高层抽象描述开始，可能进行系统的一次逐步求精，该步求精可成为通信进程的一部分，所有通信进程的总行为就是整个系统所必需的行为。同样地，有可能按自下而上的方式进行，组合进程并使用与合成规则有关的推理规则演绎最后得到的系统的属性。

D.5.4.3 通信顺序进程(CSP)

目的：CSP(Communicating Sequential Processes)是用于并行软件系统(例如，并行运行的通信进程的系统)的规格说明的一种技术。

描述：CSP 为进程系统的规格说明提供了一种语言，并且为检验进程的实现满足它们的规格说明提供了证明。

系统被建模为由顺序运行或者并行运行的独立进程构成的网络。以每个进程可能的所有行为来描述该进程。这些进程可通过通道进行通信(同步或交换数据)，这里的通信只有在两个进程准备就绪时才能进行。事件的相关时序也能够被建模。

D.5.4.4 时序规格说明语言(LOTOS)

目的：LOTOS(1anguage for temporal ordering specification)是用于描述和揭示并行通信进程系统行为的一种方法。

描述：LOTOS 是以 CCS 为基础并附有 CSP 等其它特征的技术。它克服了 CCS 在处理数据结构和数值表示方面的弱点。LOTOS 的进程描述部分仍能与其它关于抽象数据类型的形式化方法一起使用。

D.5.4.5 OBJ

目的：为了在实现之前提供具有用户反馈和系统确认的精确系统规格说明。

描述：OBJ 是一种代数规格说明语言。通过代数方程，用户描述需求。通过按照抽象数据类型(ADT)运行的操作，系统行为特征和结构特征得到详细描述。

OBJ 规格说明和其它的分步方法，也同其它形式化方法一样采用形式化证明技术。此外，既然 OBJ

规格说明的结构特征是可用机器执行的，那么从规格说明本身可直接实现系统确认。执行过程实质上是通过持续的方程置换(重写)一直到达规定的输出值为止来评估功能的。这种可执行性允许设想的系统的最终用户在系统规格说明阶段获得最终系统的一个“视图”，而不必精通基础的形式化规格说明技术。

正如其它所有的 ADT 技术一样，OBJ 只适用于顺序系统或者并行系统的顺序特征方面。OBJ 已被用于小型和大型工业应用的规格说明。

D.5.4.6 时态逻辑

目的：安全性和运行需求的直接表达和这些属性在其后的开发步骤中得到保持的形式化证明。

描述：标准的一阶谓词逻辑不包含时间概念，通过增加模态操作符(例如，“此后”和“最后”)，时态逻辑扩展了一阶逻辑。这些操作符可用来限定有关系统的一些断言。例如，安全属性可能需要包含“此后”，而期望的系统的其它状态可能需要借助某些另外的启动状态达到“最后”。将根据状态(行为)序列来解释时态规则。构成“状态”的内容依赖于被选择的描述级别。它能够指向整个系统、系统部件或者计算机程序。

在时态逻辑中并不明确处理量化的时间间隔和约束。绝对定时不得不被通过建立附加的时间状态为状态描述的一部分来处理。

D.5.4.7 维也纳开发方法(VDM/VDM++)

目的：顺序(VDM)和并行实时(VDM++)程序的系统性规格说明和实现。

描述：VDM(Vienna Development Method)是基于数学的一种规格说明技术，它是用可被验证正确性的规格说明的方式，对实现逐步求精的技术。

规格说明技术是以模型为基础的，在这个模型中将根据集合理论结构(根据此集合理论结构来描述不变式(谓词))和运算(对某个状态进行运算的那个状态的模型是借助系统状态并通过规定运算的先决条件和后续条件来建立的)建立系统状态模型。为了保留系统不变式，可验证这些运算。

通过借助目标语言的数据结构具体化系统状态和借助目标语言的程序求精运算可完成规格说明的实现。具体化和求精步骤将引出验证它们的正确性的义务。设计者应决定是否要履行这些义务。

原则上讲，VDM 是用在规格说明阶段，但也可在产生源码的设计和实现阶段中使用。它仅适用于并行系统中顺序结构程序或者顺序过程。

VDM++是面向对象的和并行实时扩充的 VDM，它是基于 ISO 语言 VDM-SL 和面向对象的语言 Smalltalk 的一种形式化规格说明语言。

VDM++提供了一个很宽的结构范围，因而一个用户可在形式上规定并行实时系统的一种面向对象的风格。在 VDM++中，一个完整的形式规格说明由类规格说明的一个集合和任选的一个工作区组成。

VDM++的实时机制是：

a) 提供时序表达式以表示一个方法体中的当前时刻和方法激活时刻；
b) 给方法增加一个时间相关的后续表达式以便为正确实现规定执行时间的上(或下)限；
c) 时间连续变量已经被引入。利用假设子句和效果子句，可以规定这些时间函数之间的关系(例如微分方程)。这个特征已证明对工作在一个时间连续环境中的系统要求的规格说明是很有用的。求精步骤可产生这类系统的离散软件方案。

D.5.4.8 Z

目的：Z 是用于顺序系统的一种规格说明语言表示符号和设计技术，该技术允许开发者从一个 Z 规格说明开始得到可执行算法，并可验证规格说明的正确性。

原则上 Z 可用于规格说明阶段，但已发明出一种也可用于设计和实现的方法。它最适合于开发面向数据的顺序系统。

描述：像 VDM 一样，规格说明技术也是以模型为基础的，在这个模型中将根据集合理论结构(根据此集合理论结构来描述不变式(谓词))和运算(对某个状态进行运算的那个状态的模型是借助系统状态并通过规定运算的先决条件和后续条件来建立的)建立系统状态模型。为了保留系统不变式，从而演

示它们的一致性可验证这些运算，将把一个规格说明的形式部份分成一些模式，这些模式允许通过求精构成规格说明。

典型地，一个 Z 规格说明是形式化 Z 和用自然语言的非形式说明文本的混合物。形式文本本身对易读而言是太简洁了，并且常常需要解释它的目的，而非形式自然语言容易变得模糊和不准确。

与 VDM 不同，Z 是一种表示符号而不是一个完整的方法，然而已开发出一种辅助方法(称为 B)，它可同 Z 一起使用。B 方法依据的是逐步求精的原理。

D.5.5 防卫性编程

目的：开发出能够在执行中检测到异常控制流、数据流或数据值并且能够以一种预定的和可接受的方式对这些异常作出反应的程序。

描述：在程序设计过程中可以使用许多技术来检查控制流或者数据流的异常。为了减少错误的数据处理的可能性，在一个系统的整个程序设计过程中，都应该系统地使用这些技术。

防卫性编程技术有两个重叠的区域。

a) 内部能够处理安全错误的软件。该类软件可以克服本身设计上的一些缺点，这些缺点可能是设计、编码中的错误或者不正确的需求造成的。下面列出了一些防卫性编程技术：
   1) 检查变量的范围；
   2) 尽可能检查数据值的合理性；
   3) 在程序入口处检查程序参数的类型、大小和范围。

   从程序功能和变量的实际意义这两方面来看，以上三条建议有助于保证程序处理的数据是合理的。此外，只读和可读/写的参数应区分并检查对它们的存取。函数中应把所有参数处理成只读参数。文字常量应该是不可写的，这有助于检测意外的重写或者误用变量。

b) 容错软件。不论在正常或者不正常的条件下，容错软件都能“预见”失效的发生，按照预先规定的方式来运行，下面列出了一些容错技术：
   1) 检查具有实际意义的输入变量和中间变量的合理性；
   2) 检查输出变量的效果，最好直接观察相关的系统状态的改变；
   3) 检查软件的配置，其中还检查预期硬件是否存在及其可访问性，另外检查软件本身是否完整，这对于在维护过程之后保持完整性是特别重要的。

   有些防卫性编程技术，比如控制流序列检查，也能处理外部失效。

D.5.6 设计和编码标准

D.5.6.1 概述

目的：提高可验证性，促进以团队为中心、客观的开发方法和实施一种标准设计方法。

描述：在项目一开始参加者要就共同遵守的规则协商一致，这些规则包含要遵守的设计和开发方法(例如：JSD、MASCOT、Petri 网等)以及相关的编码标准(见 D.5.6.2)。

制定这些规则是为了更容易地进行开发、验证、评价和维护。因此，他们应考虑运用可利用的工具，特别是分析器和反向工程方面的工具。

D.5.6.2 编码标准

目的：提高生产代码的可验证性。

描述：在编码之前应对要遵守的详细规则完全达成一致，典型规则包括：

a) 模块化细节，例如接口形式、软件模块大小；
b) 在面向对象语言中使用封装、继承（受深度的限制)和多态；
c) 限制使用或者尽量避免某些语言结构，例如“goto”(跳转)和“equivalence”语句，动态对象，动态数据，动态数据结构，递归，指针，出口等；
d) 在执行安全关键代码期间限制使用中断；
e) 规范代码的布局(列表)；

f) 在高级语言程序中禁止使用非条件转移,例如“goto”语句。

制订这些规则是为了更容易地进行软件模块的测试、验证、评估和维护。因此，它们应考虑可利用的工具，特别是分析器。

D.5.6.3 禁止使用动态变量和动态对象

目的：为了排除以下情况：

a) 不需要的或未发现的内存覆盖；

b) 在安全相关的运行时间内资源的分配瓶颈。

描述：在本方法中，动态变量和动态对象是指那些在运行时分配内存并且在运行时才确定绝对地址的变量和对象。所分配内存的大小及其地址取决于分配时系统的状态，因此不能用编译程序或者任何其它的脱机工具来检查。

因为动态变量和对象的数目以及能分配给新动态变量或对象的现存空闲内存空间与分配时的系统状态有关，所以当分配或使用变量或对象时就有可能发生错误。例如，当系统分配内存时，闲置内存空间不足，这时必然导致某一变量的存储内容就可能无意中被重写。如果禁止使用动态变量和对象就可避免这些错误。

D.5.6.4 在线检查动态变量或对象的建立

目的：在给动态变量和对象分配内存空间之前，确认这些即将分配的空间是空闲的，从而保证在运行时动态变量和对象的内存分配不会影响现存的变量、数据或代码。

描述：在本方法中，动态变量是指那些在运行时分配内存并且在运行时才确定绝对地址的变量(从这个意义上讲，这种变量也是对象实例的属性)。

在给动态变量或对象分配内存之前，借助硬件或软件来检查这些内存空间以便保证它是空闲的(例如用来避免堆栈溢出)。如果不允许分配(例如当内存空间不足，不能提供所确定的地址)，必须采取适当的措施。在已经使用完了一个动态变量或对象之后(例如退出一个子程序之后)，曾经分配给它的内存空间必须释放。

注：另一种替代方法是静态证明所有情况下内存空间都足够大。

D.5.6.5 限制使用中断

目的：为了保证软件可验证和可测试。

描述：应限制使用中断。当能简化系统时可以考虑使用中断。在被执行的软件功能的关键部分(例如，时间上的临界点、数据改变的临界点)禁用中断处理软件。如果使用了中断，那么不可中断部分应指定一个最大估计时间，使之能计算禁止一次中断的最大时间，中断使用和屏蔽应全部编制成文档。

D.5.6.6 限制使用指针

目的:避免没有首先检查指针的范围和类型而去存取数据引起的问题,支持软件的模块测试和验证,减弱失效的后果。

描述：在应用软件中，只有在存取之前检查指针数据类型和值的范围(为了保证引用的指针是在正确的地址空间内)时，才能在源码级使用指针运算。在任务间直接引用不能实现应用软件任务间的通信，应当通过操作系统来进行数据交换。

D.5.6.7 限制使用递归

目的：为了避免不可验证和不可测试的子程序调用。

描述：当使用递归时，必须要有一个明确的准则，此准则可预测递归深度。

D.5.7 结构化程序设计

目的：为了设计和实现不用运行就能分析的程序，该程序可能只包含一个统计学上不可测试状态的绝对极小值。

描述：应运用下面的原则来尽可能地降低结构的复杂性：

a) 把软件分成适当的小模块，从而保证尽可能地使它们分离并且所有的交互都是清晰的；

b) 使用结构化方法来组织软件模块控制流，例如顺序、循环和分支；

c) 保持一个软件模块的可能通路要少，输入和输出参数之间的关系要尽量简单；

d) 避免复杂的分支，特别要避免高级语言中的无条件转移(如 goto 语句)；

e) 尽可能通过输入参数来控制循环和进入分支的条件；

f) 避免使用复杂计算来决定分支和循环的依据。

除考虑效率绝对优先的情况(例如某些安全关键系统)之外，应使用有利于实施上述方法的程序设计语言的特征，而不去使用一些即使效率更高的其它特征。

D.5.8 信息隐藏/封装

目的：为了防止无意识的访问数据或过程，并从而支持一个好的程序设计结构。

描述：所有软件部件都可访问的全局数据可能会被其中任何部件意外地或不正确地修改，任何这些数据结构的改变都可能需要详细地检查代码和进行广泛的修正。

信息隐藏是解决这些问题的一种通用方法。关键数据结构被“隐藏”，只有通过一个定义好的访问过程集合才能操纵它。这种方法允许修改内部结构或者增加另外的程序，而且不会影响软件其它部分的功能。例如，一个姓名目录可能有“插入”、“删除”和“查找”访问过程，可以重写访问过程和内部数据结构(例如，使用不同的查找方法或者把名字存放在硬盘上)，这样也不会影响软件其它部分使用这些过程的逻辑特性。

关于这一点，应使用抽象数据类型的概念。如果不能直接支持这种类型，则有必要检查是否在无意中破坏了这种抽象。

D.5.9 模块方法

目的：把软件系统分解成一些小的部分，从而降低系统的复杂性。

描述：模块方法或者模块化包含一个软件项目的设计、编码和维护阶段的各种规则。随着设计过程中使用的不同方法，这些规则也会变化。大多数方法都包含以下规则：

a) 应该明确定义一个软件模块应该完成的任务(功能)；

b) 应限制和严格定义软件模块之间的关系，在同一软件模块中应具有很强的一致性；

c) 应建立不同的子程序集合来提供几级的软件模块；

d) 应限定子程序的大小为某个规定值，典型地为 2－4 屏幕大小；

e) 应限定子程序只有唯一入口和唯一出口；

f) 软件模块之间应该通过接口通信——在使用全局变量的情况下，模块应具有良好的结构，应控制对这些变量的访问，并总是在证明是合适的情况下才使用它们；

g) 应把所有的软件模块接口写入文档；

h) 任何软件模块的接口都应该只包含功能所需的那些参数。

D.5.10 使用可信的/经验证的软件模块和部件

目的：避免对每一项新应用都需彻底重新确认或者重新设计软件模块和硬件部件。尽管有的设计还没有经过形式化或严格的验证，但是可以充分利用它重要的运行记录。

描述：本方法可证明使用这种可信的或经验证软件模块和软件部件可以不会产生系统设计故障以及运行失效。只在极少见的情况下，使用这种模块和部件(即在使用中被验证过的)才是足以作为保证达到必需的安全完整性的唯一方法。对具有许多功能的复杂软件部件(如操作系统)而言，每项功能都要在使用中经过充分验证。例如，有一个为检测硬件故障的自测试程序，如果在运行期内没有出现硬件故障，就不能认为检测故障的自测试程序是经过使用验证过的。

当一个软件部件或者软件模块可以认为是充分可信时，它必须具备的条件是：根据所要求的安全完整性级别已经验证过，或者它满足下列准则：

a) 未改变的规格说明；

b) 在不同应用中使用的系统；

c) 至少运行一年；

d) 符合安全完整性水平的运行时间或者适当的运行次数；证明非安全相关的失效率时要满足下列条件：

1) 在每次运行(或每年)失效率小于 10^{-2}(置信度 95%)的情况下，必须经历 300 次(或年)运行；

2) 在每次运行(或每年)失效率小于 10^{-5}(置信度 99.9%)的情况下，必须经历 690000 次(或年)运行。

e) 所有的运行历史必须记录在一个已知的软件模块功能的要求简要表，以确保增加的运行历史能真正增加对该要求简要表相关的软件模块特性的认识；

f) 无安全关键的失效。

注 1：对于 f)，在某种情况下一个并非是安全关键的失效而在另一种情况下可能是安全关键的，并且反之亦然。

注 2：为了能够验证一个软件部件或模块是否满足以上准则，应该文档化下列几项内容：

a) 每个系统和它的组成部件的准确标识，包括其(软件和硬件的)版本号；

b) 用户标识和应用时间；

c) 运行时间；

d) 针对选择用户应用系统和应用实例的过程；

e) 针对检测和记录失效以及消除故障的过程。

D.6 结构设计

D.6.1 故障检测和诊断

目的：为了检测一个系统中的故障(这些故障可能导致失效)，从而为减少失效影响的对应措施提供依据。

描述：故障检测是检查一个系统的错误状态的活动(这些错误状态是被检查的(子)系统内的故障引起的)。故障检测的主要目的是阻止错误结果的影响。当一个系统与并行部件共同起作用并检测到自己的结果有错误时放弃控制，则被称为自检。

故障检测基于冗余和多样性(软件故障)原理。需要用某种表决来确定结果的正确性。可用的特殊方法是：失效断言程序设计，多样性编程和安全袋技术。

可以通过检查不同方面的值域或时间域，特别是物理的(温度、电压等)、逻辑的(差错检测码)、功能的(断言)或者外部的(合理性检查)来实现故障检测。为了能进行失效跟踪，检验的结果可被存储，并与影响允许失效跟踪的数据关联。

复杂系统是由子系统构成的，故障检测、诊断和故障校正的效果与各子系统中相互作用的复杂程度有关，这种相互作用的复杂性可影响故障的传播。

应在最小的子系统级使用故障诊断，因为较小的子系统可以更详细地诊断故障(错误状态检测)。

整个组织范围的集成信息系统能够例行地把安全相关系统的状态包括诊断测试信息传递给其它监管系统。当检测到异常时，就会关注它并在它发展成一种危险情况之前就用它来触发纠正行动。最后，当发生事故时，这种异常记录文件可帮助事后的调查。

D.6.2 错误检测和纠错编码

目的：为了探测和校正有关信息中的错误。

描述：以对一条 n 位信息而言，生成一个 K 位编码块，此块使之能检测和校正 r 个错误。编码的两种例子是汉明(Hamming 码)和多项式代码。

应注意，在安全相关系统中，通常需要丢弃故障数据而不是试图纠正它，因为只有一部分预定的错误才能得以正确纠正。

D.6.3 失效断言程序设计

目的：为了在执行一个程序的过程中检测残留软件设计故障，从而防止系统安全关键失效并继续高可靠地运行。

描述：失效断言程序设计方法遵循的观念是检验一个先决条件(在执行一个语句序列之前，对初始条件进行有效性检验)和一个后续条件(在执行一个语句序列之后检查结果)。如不满足先决条件或者后续条件，处理将报告出了错误。

例如：

断言<先决条件>；

动作 1；

：

：

动作 x；

断言<后续条件>。

D.6.4 安全袋

目的：为了防止软件中残留的规格说明和实现故障，这些故障对安全有不利的影响。

描述：安全袋是按照不同的规格说明实现的，在一台独立的计算机上执行的外部监视程序。安全袋只涉及保证主计算机执行安全的(不必正确的)动作。安全袋连续地监视主计算机。安全袋防止系统进入一个不安全的状态，此外，当它检测到主计算机进入一种潜在的危险状态时，安全袋或者主计算机其中之一必须使系统恢复到一个安全状态。

安全袋的软件(有时还包括硬件)应按适当的 SIL(安全完整性级)分类和标识。

D.6.5 软件多样性(多样性编程)

目的：为了防止系统的安全关键失效和持续高可靠地运行，在执一个程序过程中检测和屏蔽残留的软件设计和实现故障。

描述：在多样性编程中，以不同的方式 N 次设计和实现一个给定的程序规格说明。给 N 个版本都提供同样的输入值，并比较 N 个版本产生的结果，如认为结果有效，将把结果传送到计算机输出。

N 个版本能在单独的计算机上并行运行，或者所有版本都能在同一台计算机上运行，结果须经一次内部表决，对 N 个版本可使用不同的表决策略，这要根据下列应用需求而定。

a) 如果系统是有安全状态的，则要求完全一致性(N 的所有版本产生的结果一致)是可行的，否则将采用一个使系统达到安全状态的输出值。对简单的跳闸系统来说，表决能够朝安全方向偏倚。在这种情况下，只要任一版本要求一次跳闸，那么安全动作就将是跳闸。这种方式在典型情况下只使用两种版本(N=2)；

b) 对没有安全状态的系统而言，可使用多数表决策略。对于不存在集合一致性的情况，为了使选择校正值的机会最大化(例如取中间值，在恢复一致性之前，暂时冻结输出等)可使用概率方法。

本技术不能消除残留软件设计故障，也不能避免解释规格说明时的错误，但它提供了在这些故障或错误影响安全性之前检测和屏蔽的一种方法。

D.6.6 恢复块

目的：为了尽可能增大最终执行程序预期的功能。

描述：常常是独立地写几个不同的程序段，打算每个段都执行同一期望的功能。从这些程序段来构造最后的程序，首先执行叫做主程序段的第一段，随后是对它计算的结果的一个验收测试。如果测试获得通过，则结果被接受并且被传递给系统的后续部份。如果测试不合格，则第一段的任何副作用将被复位，然后执行被称为第一替补的第二段。它后面也跟随一个验收测试，并且也按第一段的情况一样地处理，需要时可提供第二、第三甚至更多的替补。

D.6.7 向后恢复

目的：为了在出现了一个或多个故障时提供正确的功能操作。

描述：当检测到一个故障时，系统就复位到一个较早的内部状态，该状态的一致性已在过去被证明过。这种方法意味着内部状态被频繁保存在所谓良定义检验点上。可以全局性地(对整个数据库)或者增量地(只在检验点之间改变)进行状态保存。然后，系统必须补偿这些改变，这些改变是在通过使用日志(动作的审核跟踪)、补偿(消除这些改变的所有影响)或者外部(人工)交互期间而发生的。

D.6.8 向前恢复

目的：为了在出现一个或几个故障时提供正确的功能操作。

描述：如果探测到一个故障，那么，系统的当前状态将被处理以得到一个状态，这个状态将和稍后某时的状态一致，这种概念特别适用于具有一个小型数据库和内部状态变化速度快的实时系统。这里假定了至少部子系统状态可能影响到环境，并且只有部分状态受到环境的影响(强制)。

D.6.9 重试故障恢复机制

目的：利用重试机制从一个已发现的故障状况尝试功能的恢复。

描述：在已发现故障或错误状况的事件中，通过重新执行同样的代码尝试恢复到正常情况。重试恢复可能和一次软件超时或者一次任务监视动作之后的一次重新引导、或一次重新启动过程、或一次小的重新调度、或重新启动任务一样完成。重试技术常用于通信故障或者错误恢复中，并且利用通信协议错误(检验和等)或者通信确认响应超时标记重试条件。

D.6.10 记录执行情况

目的：当软件尝试执行一条不允许的路径时，强迫软件安全地失效。

描述：每个程序执行的所有有关详情都被归档。在正常运行过程中，每个程序的执行都同先前记录的详情作比较。如果有差异，就采取一个安全动作。

执行文档包含单个判定－判定路径(DD路径)的序列或者单个访问数组、记录或者卷的序列，或者两者。

可能采用不同的方法存储执行路径。能够使用散列编码方法把执行序列映射成单个大数或数字序列。在正常运行过程中，在任何输出操作发生之前，必须对照存储的情况进行检验。

既然在一个程序执行过程中判定－判定路径的可能组合是很大的，那么把这些程序作为整体来处理是可能行不通的。在这种情况下，在软件模块级使用本技术。

D.6.11 适度降级

目的：尽管出现了失效，但通过终止比较不太关键的功能，而使更重要的关键功能可用。

描述：本技术给出将被系统执行的各种功能的优先级，设计保证了当执行所有系统功能的资源不足时，将优先执行高优先级的功能。例如，错误和事件记录功能的优先级比系统控制功能的优先级低，在此情况下，当与错误日志有关的部件发生故障时，系统控制将继续进行。另外，当系统控制部件发生故障，而错误记录部件并未出故障时，错误记录部件将接管控制功能。

它主要适用于硬件但也适用于总系统和软件。从最上层设计阶段开始就必须考虑本技术。

D.6.12 人工智能故障纠正

目的：借助引入方法和过程模型的组合以及某种在线安全性和可靠性分析，以便能用一种很灵活的方式应对可能的危险。

描述：既然规则可以从规格说明直接导出并可和规格说明对照来检验规则，那么，基于人工智能(AI)的系统将以一种很有效的方式在系统的不同通道中支持故障预测(计算趋势)、故障纠正、维护和监控动作。特别是当应用功能或者描述形式的模型和方法的组合时，本方法可有效地避免某些已被隐含根植于头脑中的某些设计和实现规则而引入规格说明中的共因故障。

为了满足要求的安全完整性，应这样选择这些方法：既可使故障得到纠正，又可使失效的影响减到最小。

D.6.13 动态再配置

目的：为保持系统功能性而不管某个内部故障。

描述：系统的逻辑结构必须要能把它映射成系统可用资源的一个子集，结构要求能检测物理资源的失效，然后重新把逻辑结构映射回剩余的起作用的有限资源。虽然本概念传统上更多地限于用来恢复有故障的部件(尤其是硬件)，但如果有足够的“运行时间冗余”来允许软件重试或者如有足够的冗余数据使得单独的和隔离开的失效不重要的话，它适用于有毛病的软件单元。

在系统设计的第一阶段就应考虑本技术。

D.7 开发工具和程序设计语言

D.7.1 强类型程序设计语言

目的：通过使用某种语言来降低故障概率，这种语言允许使用编译程序进行高级检验。

描述：当编译一个强类型程序设计语言时，对使用的变量类型进行许多检验，例如，在过程调用和外部数据存取中，对任何不符合预定规则的应用，编译都将会失败并产生一条出错消息。

通常，这样的一些语言允许根据基本的语言类型(比如整型、实型)定义用户自己的数据类型，然后按基本类型严格相同的办法使用这些类型。为了保证使用的类型正确，应实施严格检验。即使程序是由分离的编译单元构造的，整个程序都要施行这些检查，甚至引用来自独立编译的软件模块时，这些检查也能保证过程变元的数目和类型相匹配。

强类型语言一般支持良好的软件工程实践的其它特征，比如容易分析的控制结构(例如 if..then..else，do...while 等)，这些结构可产生良构程序。

强类型语言的典型例子是 Pascal，Ada 和 Modula 2。

D.7.2 语言子集

目的：为了减少引入程序设计故障的概率和增大检测任何残留故障的概率。

描述：语言应被检验，例如使用静态分析方法，以判别那些既易出错又难分析的程序设计构造。然后，排除这些构造，从而定义一个可减少引入程序设计故障的语言子集。

D.7.3 经认证的工具和经认证的翻译程序

目的：在开发软件的各个阶段，这些工具对帮助开发者是有必要的。只要可能，就应认证这些工具，以便能够认定有关输出正确性的某级置信度。

描述：一般是由一个独立的，通常是国家的团体，对照单独设立的标准(国家、国际标准或国军标)来执行工具的认证。理想情况下，所有开发阶段(规格说明、设计、编码、测试和确认)使用的工具和使用于配置管理的工具都应该经受认证。

注意，经认证的工具和经认证的翻译程序通常只根据它们各自的语言或过程标准进行过认证，一般并未对安全性进行过任何认证。

D.7.4 工具和翻译程序：通过使用提高置信度

目的：为了避免在开发、验证和维护一个软件包的过程中出现的翻译程序失效引起的任何问题。

描述：在先前的许多项目中并未发现异常性能的证据的情况下，可以使用该翻译程序。除非存在正确性能的一些其它保证(例如：见 D.7.4.1)，应避免使用没有运行经验或者带有任何已知的严重故障的翻译程序。

在一个与安全相关的项目开发过程中，当翻译程序显示出低的置信度时，记录下有关的语言构造并应小心避免使用这些部分。另一种方案是把语言的使用限定为仅仅使用它常用的特征。

这些建议是根据许多项目的经验提出来的。不成熟的翻译程序已显示出它对于任何软件开发而言都是一个严重的障碍。它们使得开发一个安全软件成为不可能。

D.7.4.1 源程序和执行代码的比较

目的：为了检验用来产生 PROM(可编程只读存储器)不会引入任何错误映像的工具。

描述：PROM 映像可被逆向工程来得到子“目标模块”。这些“目标”模块又可被逆向工程得到汇编语言文件。使用适当的技术，可以把这些逆向生成的汇编语言文件同最初用来产生 PROM 的实际源

文件进行比较。

本技术的主要优点是对所有程序而言，用于产生 PROM 映像的工具不必被确认。本技术可验证用于特殊安全相关系统的源文件的翻译的正确性。

D.7.5 可信的/经验证的软件模块和部件库

目的：为了避免对于每个新应用都需要对软件模块和部件设计进行彻底的重新确认或重新设计。也为了提升还未正式或精确确认但已经有相当长的运行历史可用的设计。

描述：良设计和良构的 PES 由许多硬件和软件部件和模块构成，这些部件和模块彼此之间有明显的差别并以一些明确规定的方式相互作用。

为不同应用设计的各种 PES 包含许多同样的或很相似的软件模块或部件。建立这种通用软件模块库使得必须用于确认设计的大部份资源被多个应用所共享。

此外，这些软件模块在多个应用中使用为成功运行使用提供了实践证明。这种经验证据无疑增强了用户对软件模块的置信度。

D.7.6 合适的编程语言

目的：为了尽可能多地支持本指导性技术文件的要求，特别是防卫性编程、强类型程序设计、结构化程序设计，或许还有断言。所选编程语言应该轻松产生易于检验的代码并有助于程序的开发、验证和维护。

描述：应充分地、清楚地定义编程语言。该语言应是面向用户或问题的而不是面向处理机/平台的机器语言。广泛使用的语言或它们的子集比专用语言更好。

除了已经提到的特征之外，编程语言应提供：程序块结构；翻译时间检验；运行时间类型和数组边界检验。编程语言还应鼓励：使用小型的和可管理的软件模块；对专用软件模块数据的访问限制；定义变量的子范围；任何其它类型的限制错误的语言构造。

如果系统的安全操作与实时约束有关，那么语言也应该提供异常/中断处理。

由一个适当的翻译程序(编译器)、合适的预编译软件模块库、一个调试器和一些工具(用于版本控制和开发两方面)支持的语言是满足要求的。

在开发本指导性技术文件时，还不清楚面向对象语言是否在开发安全相关软件时比其它传统的语言更好。

使验证困难并因此应避免的特征是：

a) 除子程序调用外的无条件转移；
b) 递归；
c) 指针、堆或任何类型的动态变量或对象；
d) 在源码级的中断处理；
e) 循环、程序块和子程序的多入口和出口；
f) 隐含变量初始化或声明；
g) 过程参数。

低级语言，特别是汇编语言，由于它们具有面向特定处理机/平台的特性，而存在问题。

符合要求的语言属性应使程序的设计和执行可预知。对于一种适当定义的编程语言，应该存在一个能保证可预测程序执行结果的子集。表 D.1 给出了针对特定编程语言的建议。

表 D.1 针对特定编程语言的建议

编程语言	SIL1	SIL2	SIL3	SIL4
1 ADA	HR	HR	R	R
2 ADA 子集	HR	HR	HR	HR
3 MODULA－2	HR	HR	R	R
4 MODULA－2 子集	HR	HR	HR	HR

表 D.1(续)

编程语言	SIL1	SIL2	SIL3	SIL4
5 PASCAL	HR	HR	R	R
6 PASCAL 子集	HR	HR	HR	HR
7 FORTRAN77	R	R	R	R
8 FORTRAN77 子集	HR	HR	HR	HR
9 C	R	—	NR	NR
10 具有编码标准并使用静态分析工具的 C 子集	HR	HR	HR	HR
11 PL/M	R	—	NR	NR
12 具有编程标准的 PL/M 子集	HR	R	R	R
13 汇编程序	R	R	—	—
14 具有编码标准的汇编程序子集	R	R	R	R
15 梯形图	R	R	R	R
16 具有定义的语言子集的梯形图	HR	HR	HR	HR
17 功能块图	R	R	R	R
18 具有定义的语言子集的功能块图	HR	HR	HR	HR
19 结构化文本	R	R	R	R
20 具有定义的语言子集的结构化文本	HR	HR	HR	HR
21 顺序功能图	R	R	R	R
22 具有定义的语言子集的顺序功能图	HR	HR	HR	HR
23 指令表	R	—	NR	NR
24 具有定义的语言子集的指令表	HR	R	R	R

注 1：系统软件包括作为系统组成部分而配置的操作系统，驱动程序，嵌入功能和软件模块。典型地，软件由安全相关系统供方提供。应仔细选择语言子集以避免可能会引起实现故障的复杂结构。应执行检查，以确定语言子集使用合理。

注 2：应用软件是为专门的安全应用而开发的软件。在许多情况下，由最终用户或者一个面向应用的供方开发软件。当多种编程语言时受到同样的推荐时，供方应选择在需方产业或设施中人们常用的那一种。应仔细选择语言子集以避免可能会引起实现故障的复杂结构。应执行检查，以确定语言子集使用合理。

注 3：如果在表中未列出某种专用语言，并不能认为它被排除在外了。

D.8 验证和修改

D.8.1 概率测试

目的：为了得到被研究软件的可靠性属性的定量数值。

描述：该定量数值应该考虑有关的置信度和重要性级别并能给出：每一操作的失效概率；在某一时期的失效概率；错误遏制的概率。

从这些数值可导出其它一些参数，比如：

a) 无失效执行的概率；

b) 残留概率；

c) 可用性；

d) MTBF(平均无故障工作时间)或者失效率；

e) 安全执行的概率。

概率考虑要么以概率测试为基础，要么以运行经验为基础。通常，测试用例或者观察的操作用例数目都很大。典型地，按操作指令方式运行的测试占用的时间比连续运行方式经过的时间少得多。

一般使用自动测试工具来提供测试数据和监控测试输出。大量的测试要在具有合适过程模拟外围设备的大型主计算机上运行。选择测试数据要具有系统性和随机性。整个测试控制，要保证测试数据分布，而随机选择能够管理各个测试用例的细节。

各个测试设施、测试执行和测试监控由如上所述的详细测试目的确定。其它重要条件由数学前提给出，此数学前提是在测试评估满足预定测试目的时必须满足的。

有关任何测试目标行为的概率数据也可从运行经历导出。倘若满足同样的条件，评价测试结果就可使用同样的数学技术。实际上，使用这些技术很难证明极高级别的可靠性。

D.8.2 数据记录和分析

目的：为使验证、确认、评价和维护较容易，软件项目中的所有数据、判定和基本原理都应该编制成文档。

描述：在一个项目期间，应该维护详细的记录文档。这些文档包括：

a) 对每个软件模块执行的测试；

b) 判定和它们的基本原理；

c) 问题和解决办法。

在项目进行过程中和结束时，可分析文档以便建立各种各样的信息。特别是当开发项目期间作出某些判定的基本原理还不为维护工程师所知时，数据记录对计算机系统的维护是很重要的。

D.8.3 接口测试

目的：为了检测子程序接口中的错误。

描述：对测试的完整性或者详细程度分级是可行的。对最重要的一些级进行如下测试：

a) 处于它们的极限值的所有接口变量；

b) 分别处于它们的极限值的所有接口变量以及处于正常值的另一些接口变量；

c) 每个接口变量范围中的所有值和处于正常值的接口变量；

d) 组合(针对小的接口，这种组合才是可行的)中的所有变量的所有值；

e) 与每个子程序的每次调用有关的规定的测试条件。

当接口不包括检测到的错误参数值的断言时，这些测试是特别重要的，在预编译子程序生成新配置后，这些测试也是重要的。

D.8.4 边界值分析

目的：为了检测发生在参数极限值或边界值处的软件错误。

描述：根据等价关系(见 D.8.7)把程序的输入范围分成若干输入类别。测试应包括这些类别的边界值和极限值。测试将检查和程序中的规格说明一致的规格说明的输入域的边界。在直接和间接转换中使用数值 0 通常易出错误，从而要求特别注意：

a) 0 除数；

b) 空白 ASCII 字符；

c) 空栈或者链表元素；

d) 0 表项。

通常，输入的边界直接相应于输出范围的边界。测试用例通常被编制为强制输出到它的限定范围。还应考虑是否有可能设计一个能使输出超过规格说明边界值的测试用例。

如果输出是一个数据序列，例如一个打印的表，应特别注意第一个和最后一个元素并且还应特别注意不包含元素，只包含一个和两个元素的表。

D.8.5 错误推测

目的：为了消除普遍的编程失误。

描述：测试经验和直觉同受试系统的知识和奇特性结合可把一些未分类的测试用例附加到计划的测试用例集里。

特殊的值或者值的组合易出错。可以从审查检查单得出某些感兴趣的测试用例。也要考虑系统是否足够健壮。例如，在前面板上按按钮太快或者太频繁?同时按两个按钮会发生什么情况?

D.8.6 错误撒播

目的：为了弄清一组测试用例是否适合。

描述：在程序中插入(撒播)一些已知类型的错误并在测试条件下对测试用例执行程序。如只发现撒播错误中的一部分，那么测试用例集是不合适的。发现的撒播错误与撒播错误的总数之比可用来估算发现的真实错误与错误总数之比。如式 D.1，它给出估算剩余错误数的可能性，并由此也估算出剩余的测试工作量。

$$\frac{\text{发现的撒播错误}}{\text{撒播的错误总数}}=\frac{\text{发现的真实错误}}{\text{真实的错误总数}} \quad \cdots\cdots (D.1)$$

检测到所有撒播的错误，要么指示测试用例集是合适的，要么指示撒播的错误太容易发现了。本方法的限制是，为了得到任何可用的结果，错误的类型和撒播的位置必须反映真实错误的统计分布情况。

D.8.7 等价类别和输入分区测试

目的：为了使用最少的测试数据恰当地测试软件。通过选择测试软件必需的输入值域(范围)的分区来得到测试数据。

描述：本测试策略是以输入的等价关系为基础的，该关系确定了输入值域的一个分区。

为了包括早先规定的所有分区，应对测试用例进行选择。从每个等价类型至少要选一个测试用例。

对于输入分区存在两种基本的可能性，它们是：

a) 从规格说明得出的等价类型：规格说明的解释要么是面向输入的，例如选择的值按相同的方法进行处理，要么是面向输出的，例如值的集合产生相同的功能结果；

b) 从程序的内部结构得出的等价类型：从程序的静态分析确定等价类型结果，例如值的集合产生同样的执行路径。

D.8.8 基于结构的测试

目的：通过测试来检验程序结构的某些子集。

描述：在程序分析的基础上，选择一组输入数据，从而可检验大部分(并且常常是预定的目标)程序代码。根据要求的严格程度，代码覆盖测试的范围变化如下：

a) 语句：它是严格性最差的测试，因为不用检验一个条件语句的两个分支就能可执行所有代码；

b) 分支：应检验每个分支的两边。对某些类型的防卫性代码，这可能是不实际的；

c) 组合条件：检验一个组合条件分支(即用 AND/OR 连接的)的每个条件；

d) 线性代码序列和跳转：线性代码序列和跳转是包括条件语句并用一个跳转结束的任何代码语句序列。由于前面代码的执行强加给输入数据的限制，许多潜在的子路径是不可行的；

e) 数据流：根据数据的使用情况选择执行路径。例如，在同一变量被写和读的路径；

f) 调用图：一个程序是由从其它子程序激活的子程序组成的。调用图是程序中子程序被调用的树形图。设计的测试应该包括树中的所有调用；

g) 基本路径：从开始到结束的一个有限路径的最小集合中的一条路径，所有基本路径的测试对查找错误都是有效的。

D.8.9 控制流分析

目的：为了检测差劲的和潜在有错误的程序结构。

描述：控制流分析是查找代码可疑区的一种静态测试技术，可疑区代码不遵循好的编程实践。程序可通过产生一个有向图来分析，该有向图能进一步分析：

a) 不可访问的代码，例如，无条件转移留下的执行不到的代码块；

b) 打结的代码：良构代码具有一个可简化的控制图，该代码能被不断简化成一个单节点。相反，非良构代码只能简化成几个节点构成的一个结。

D.8.10 数据流分析

目的：为了检测差劲的和潜在有错误的程序结构。

描述：数据流分析是一种静态测试技术，此技术把从控制流分析得到的信息同在各代码分区中读或写的变量有关的信息组合起来。分析要检查：

a) 在给变量赋值之前读出的那些变量：可以通过声明新变量时总是给它赋初值来避免该错误；

b) 被写多次而只被读一次的那些变量：可以指示已被省略的代码；

c) 只写而决不会读的那些变量：可以指示冗余代码。

一个数据流的异常结构不一定直接相应于一个程序错误，但如果避免了这种异常，代码可能更少包含错误。

D.8.11 潜通路分析

目的：为了检测一个系统中的一条意想不到的路径或逻辑流。在某些条件下，该系统启动一个不希望有的功能或者禁止一个需要的功能。

描述：一条潜通路路径可以由硬件、软件、操作员动作或者这些要素的组合构成。潜通路不是硬件失效的结果而是无意中设计入系统的或者编码到软件程序中的潜伏条件，在某些条件下，它们能引起系统或者软件程序出错。

潜通路的类型有：

a) 引起电流、能量或逻辑序列沿一条意想不到的通路或者在一个非预定方向流动的潜通路；

b) 潜通路计时，按这种计时，事件以一种不可预料的或者冲突的顺序发生；

c) 潜通路指示，它引起系统运行条件的显示发生歧义和虚假，并因而造成操作员采取一个不需要的行动；

d) 潜通路标记，它错误地或不准确地标记系统功能，例如系统输入、控制、显示、汇集信息等，并因此误导一个操作员对系统施加一个错误的激励。

潜通路分析有赖于对硬件和软件结构的基本拓扑模式(例如，对于软件提出了多种基本模式)识别。借助关于使用问题的检查单以及基本拓扑成分之间的关系进行分析。

D.8.12 符号执行

目的：为显示源码和规格说明之间的一致性。

描述：在所有赋值中，在用右侧替换左侧之后估算程序变量；条件分支和循环应翻译成布尔表达式；对于每个程序变量，其最后结果是一个符号表达式，应该能够对照预期的表达式来检验它。

D.8.13 形式化证明

目的：为了不用执行而证明一个程序或者规格说明的正确性。证明中使用了理论模型和数学模型及规则。

描述：在程序中的不同位置声明了许多断言，它们作为先决条件和后续条件被用于程序中的各条路径。验证包括显示出根据一组逻辑规则，程序把先决条件转换成后续条件，而后程序终止。

在D.5.4描述了几种形式化方法，例如CCS，CSP，LOTOS，OBJ，时态逻辑，VDM和Z。

D.8.14 复杂性度量

目的：从软件本身的属性或者从它的开发史或者测试史预测程序属性。

描述：这些模型评估软件的某些结构属性，并把它同一个描述的属性比如可靠性或者复杂性联系起来。为了评估大部分测量方法，采用软件工具是必须的。下面给出了可以使用的一些度量：

a) 图形理论复杂性：本方法可用于生命周期的初期进行比较评价，它以程序控制图的复杂性为基础，复杂性用它的秩数来表示；

b) 启动某个软件模块的方式数(可存取性)：较好的说法是可访问一个软件模块的次数，更确切的

说法是它被调用的次数；

c) 赫尔斯梯德 (Halstead)度量：这种方法通过操作码和操作的数目来计算程序长度。它提供了复杂性和大小的一种度量，它为估算今后所需开发资源时提供了一个参照基准；

d) 每个软件模块的进出口数：最小化进口／出口数是结构化设计和编程技术的一个关键特征。

D.8.15 菲根检查法

目的：菲根 (Fagan)检查法，是为了揭示程序开发的各个阶段中的错误和故障。

描述：以查找错误和故障为目的对质量保证文件的一种“形式”审核。检查过程包括5个阶段：计划、准备、检验、修改、跟踪。每个阶段有它自己单独的目标。必须检查整个系统开发(规格说明、设计、编码和测试)。

D.8.16 走查/设计评审

目的：为了尽早和尽可能经济地检测某个开发的产品中的故障。

描述：建议对所有新产品/过程、新应用、现存产品及开发过程的修改进行正式设计评审；因为它们可以影响功能、性能、安全性、可靠性、检查可维护性的能力、可用性、支付能力，以及影响最终产品/过程、用户等的其它特性。

正式设计评审，包括正式设计评审的概述，它们的目标，各种设计评审类型的细节，设计评审小组的组成和相关的任务和职责。

代码走查包括走查小组选择针对程序的一个小型纸件测试用例文件集、代表性的输入集合和相应预计的输出集合，然后通过程序逻辑手动给出测试数据。

D.8.17 原型设计/动画

目的：为了根据给定的一些约束检查实现系统的可行性；还为了与需方就系统规格说明进行解释和交流，并以便找出存在误解的地方。

描述：选择系统功能、约束和性能要求的一个子集。使用高级工具建立一个原型。在该阶段，不需要考虑比如目标计算机、实现语言，程序规模、可维护性、可靠性和可用性这些约束。对照需方标准来评估原型并且通过这种评估来修改系统要求。

D.8.18 过程模拟

目的：为了测试一个软件系统的功能连同它与外界的接口而不用对真实世界作任何修改。

描述：建立一个系统，该系统只用于测试目的，它模拟受控设备的行为。

模拟可以是软件和硬件的组合或者只是软件。它必须：

a) 提供等同于实际安装受控设备时存在的输入；

b) 提供与受试软件在某种程度上相符合的输出，符合程度就是指忠实的代表受控设备的程度；

c) 要保证能为操作员输入提供任何干扰，这种干扰是受试系统需要克服的。

当测试软件时，模拟可以是具有目标硬件输入和输出的模拟。

D.8.19 性能要求

目的：为了确定一个软件系统的可证明的性能要求。

描述：对系统和软件需求规格说明两者执行分析以便确定所有通用和专用、显式和隐式的性能要求。

逐个检验每个性能要求，并且反过来确定：达到性能要求的标准；是否可对照标准进行测量；测量的可能精度；测量能够被评价的项目阶段；测量能够被实施的项目阶段。

为了得到性能要求、达到的标准和可能的测量的清单，则要分析每个性能要求的可行性。主要的目标是：

a) 与至少一次测量相关的每个性能要求；

b) 在可能的情况下，选择精确有效的测量方法以尽可能早地被用于开发；

c) 规定根本的和任选的性能要求和达到的标准；

d) 在可能的情况下，对于多个性能要求可能利用使用单次测量的好处。

D.8.20 性能建模

目的：为了保证系统工作能力足以满足特定的要求。

描述：需求规格说明包括特定功能(也许还要和使用系统总资源的约束相结合)的吞吐量和响应要求。提出的系统设计将借助以下办法同需求规格说明的要求进行对比：

a) 产生一个系统过程和各过程间的相互作用的模型；

b) 通过每个过程确定资源的使用，例如，处理机时间、通信带宽、存储器件等；

c) 确定平均的和最差情况条件下，性能分配要求到系统中的位置；

d) 计算在平均的和最差情况下各个系统功能的吞吐量和响应时间。

对简单的系统而言，一个分析可能就足够了，而对较复杂的系统而言，某种形式的模拟可能更适于获得精确的结果。

在详细的模型化之前，可以使用一种较简单的"资源预算"检验，它可把所有过程的资源总需求计算出来。当需求超过设计的系统能力时，设计是不可行的。即使设计通过了这种检验，由于资源不足，性能模型将显示响应时间过长和产生的延迟。为了避免这种情况，工程师常常使用总资源的一部分(例如50%)来设计系统以便避免资源不足的可能性。

D.8.21 强度测试

目的：为了给测试目标加上一个例外的高工作负荷以便显示测试目标可以毫不费力地承受正常工作负荷。

描述：存在各种可用于强度测试的测试条件。这些测试条件中的一些是：

a) 如果工作在轮询方式下，则每个时间单位里测试目标接受的输入的变化要比正常条件下增大很多；

b) 如果工作在按需方式下，则每个时间单位里向测试目标的请求数目增加大大超过正常条件；

c) 如果一个数据库的规模起某种重要作用，则让它增大到超过正常条件；

d) 受影响的设备分别调到它们的最高或者最低速度；

e) 在极端情况时，所有影响因素同时以边界条件提供。

在这些测试条件下，可估算测试目标的时间行为，还可观察负荷变化的影响，并能检查内部缓冲器或者动态变量、堆栈等的正常大小。

D.8.22 响应时间和存储限制

目的：为了保证系统满足它的时序和存储要求。

描述：系统和软件的需求规格说明包含特定功能的存储和响应要求，也许还要结合对使用系统总资源的限制。

为了确定在平均的和最差条件下的分配要求，要实施分析。该分析需要估算每个系统功能使用的资源和经过时间，有几种办法可得到这些估算，例如，把一个现存系统或者原型设计同时向关键系统的基准程序作比较。

D.8.23 影响分析

目的：为了确定改变或者增强软件系统时对该软件系统中的其它软件模块以及其它系统的影响。

描述：在对软件进行一次修改或增强之前，为了确定这种修改或增强对该软件的影响，还为了确定哪些软件系统和软件模块将受影响，应进行影响分析。

在完成分析之后，需要对重新验证软件系统的问题作出决定。该决定依赖于受影响的软件模块数、受影响的软件模块的临界状态和改变的本质有关。可能的决定有：

a) 只重新验证被改变的软件模块；

b) 重新验证所有受影响的软件模块；

c) 重新验证整个系统。

D.8.24 软件配置管理

目的：软件配置管理的目的是为了保证当那些可交付的项目有变更时，开发可交付项目的一致性。一般地，配置管理可用于硬件和软件开发两方面。

描述：软件配置管理是在整个开发过程中使用的一种技术。实质上，它要求编写每个重要的可交付项的每个版本及各个可交付项的不同版本之间的每种关系的生产文件。产生的文件允许开发者确定一个可交付项(特别是它的一个部件)的一个改变对其它可交付项的影响。特别是可以从一致的几组部件版本可靠地重建各系统或子系统。

D.9 功能安全评估

D.9.1 判定表 (真值表)

目的：为了提供复杂逻辑组合和关系的一个清楚的和一致的规范和分析。

描述：本方法使用两维表来简洁地描述布尔程序变量之间的逻辑关系。本方法的简明性和表格特性使它适于作为一种分析用代码表示的复杂逻辑组合的方法。

D.9.1 危险和可操作性研究(HAZOP)

目的：为了确定在一个规划的或者现存的系统中的安全危险，它们可能的原因和后果以及为减少它们发生的概率而建议的行动。

描述：一个由覆盖整个目标系统各方面的专家组成的工程师小组，通过一系列计划好的会议，探讨设计的结构化检验。他们考虑设计的功能特性和系统在实际中是怎样操作的(包括人的活动和维护)的两个问题。组长鼓励小组成员创造性地暴露潜在危险和通过展现系统的每一构成部分与几个引导词："none(没有)"、"more of(更多的)"、"less of(更少的)"、"part of(.....的一部份)"、"more than(大于.....)"或者"as well as"(既.....又)和"other than(而不是.....)"的连接来驱动该过程。每个应用条件或者失效模式都要考虑：它的可行性、它的产生机理、可能的后果(有无危险?)、怎样避免以及避免它的技术花费是否值得。

在以后的时间里，常常需要进一步进行危险分析(常称为概率风险评价或者定量风险评价)以便更详细地研究主要的危险。

在项目开发的许多阶段都要进行危险研究，但最有效的执行时间是早到足以影响主要设计和可操作性判定的阶段。在项目内为开会规定一个固定的时间日程表是有帮助的；每次会议安排至少半天，每周安排不多于 4 次，因此伴生文件流可得到维护更新。会议文件将构成系统危险/安全档案的重要组成部分。

D.9.3 共同原因失效分析

目的：为了确定多个系统或子系统中潜在的失效，因为在多个部分中可同时出现相同的失效，所以这种失效可能逐渐削弱冗余的好处。

描述：打算用来解决 PES 中常常使用冗余和多数表决带来的问题，以便避免组成部件和子系统中的随机部件失效，这些失效势必会妨碍数据的正确处理。

但有些失效对不止一个部件或子系统是共同的。例如，当在单间房内安装一个系统时，空调的缺点可能会削弱冗余的好处。系统的其它外部影响，比如火、注水、电磁干扰、平台撞击和地震也同样如此。系统也可能受与操作和维护有关的意外事故的影响。因此，为操作和维护，对操作和维护人员提供全面培训并且提供编写良好的操作规程是最根本的。

内部影响对共同原因失效也是一个主要的原因。它们能起源于共同的或同样的组成部件和它们的接口的设计故障。共同原因失效分析必须搜查系统的这些潜在的共同失效。共同原因失效分析的方法是：通常的质量控制；设计评查；由一个独立小组进行验证和测试；根据类似系统反馈的经验分析实际的意外事故。即使在一个冗余系统的各个通道中使用软件多样化，也可能在软件方法中存在一些共性，它们将引起共同原因失效，例如在共用的规格说明中的错误。

D.9.4 马尔可夫模型

目的：为了评估一个系统的可靠性、安全性或者可用性。

描述：构建系统的一幅图，此图描绘了关于失效情形(失效情形由图的节点代表)的系统状态。表示失效事件或修复事件的节点之间的边线由相应的失效率或修复率加权。假定状态 N 改变成状态 N+1 的一次变化和前面的状态 N-1 无关。注意，可以以获取系统的精确描述的方式，例如，发现的或未发现的失效，一个较大的失效的表现等，来详细说明故障事件、状态和失效率或修复率。

马尔可夫技术适合模型化多个系统，在这些系统中冗余级因组成部件的失效和修复而随时间变化。其它传统的方法，例如，失效模式与影响分析和故障树分析，因为不存在计算相应概率的组合公式，不能很好地用来模型化系统整个生命周期内的失效影响。

在最简单的情况中，描述系统概率的公式很容易被人工计算。在较复杂的情况中存在简化(即减少状态数)的一些方法。对非常复杂的情况，用计算机图形模拟才能计算结果。

D.9.5 可靠性方框图

目的：为了用图形形式对必定发生的一组事件和为成功运行一个系统或一个任务必须满足的一组条件建模。

描述：分析目标被表示成一条成功的通路，此通路由方框、线条和逻辑结点(junction)组成。一条成功的通路从图的一侧开始经过方框和结点连接到图的另一侧。一个方框代表一个条件或一个事件，如果该条件是真实的或者事件已发生，则通路就可通过。当通路到达一个结点时，如满足结点逻辑，它就继续下去。当通路到达一个顶点时，它可沿所有的引出线继续向前。如至少存在一条成功地通过图的通路，则分析目标就处在正确运行之中。

D.9.1 蒙特－卡络模拟

目的：用随机数模拟软件中的真实世界现象。

描述：蒙特－卡洛(Monte-Carlo)模拟用来解决两类问题。

a) 概率性的，在使用随机数来产生随机现象的情况中；

b) 确定性的，它被从数学上变换成一个等效的概率性的问题。

蒙特－卡洛模拟注入随机数流来模拟一个分析信号上的噪声，或者加上随机偏置值或者容差。执行蒙特－卡洛模拟可产生一个大的样本，根据这个样本可得到统计结果。

当使用蒙特－卡洛模拟时，必须注意保证偏置值、容差或者噪声要有一个合理的值。

蒙特－卡洛模拟的一个普遍原理是重述和再用形式表示最初所描述的问题使之得到的结果尽可能精确,而不是满足于解决最初所描述的问题。

附 录 E
（资料性附录）
人员资格要求导则

E.1 目的

本附录所考虑的是保证对于软件安全生存周期活动负责的人具有履行其责任的能力。

E.2 一般考虑

进行任何软件安全生存周期活动的人员，包括安排活动的人员，应受过相关的培训，具有技术知识、经验和从事相关工作的资格。

应对进行任何软件安全生存周期活动的人员，包括任何功能安全安排活动的人，进行在相关应用领域的培训、经验和资格的评估。

当进行人员资格评估时应考虑下列因素：

a) 相关应用领域的知识；
b) 相关技术的工程知识(如电子、电气、可编程电子、软件工程)；
c) 相关技术的软件工程知识；
d) 法律和安全法律框架方面的知识；
e) 从安全相关系统失效事件中的后果来看，后果越大越严重，对资格的评估和规定就越严格；
f) 从安全相关系统的安全完整性级别来看，安全完整性级别越高，对资格的评估和规定就越严格；
g) 设计、设计程序和应用领域越是新颖，越是未经过验证，对资格的评估和规定就越严格；
h) 对于使用的技术和履行规定职责的经验，要求的资格水平越高越好，承担规定职责的要求和从先前经验而来的资格越接近越好；
i) 对于履行特定任务的相关资格。

进行任何软件安全生存周期活动的人都应该被记录到有关文档中。

E.3 软件安全性分析管理者的资格要求

软件安全性分析管理者，根据其所在组织的不同，分为供方、需方、独立评价第三方软件安全性分析管理者；软件安全性分析管理者具有组织协调有关软件安全性分析活动和批准有关软件安全性文件的职权。

软件安全性分析管理者应符合下列条件：

a) 至少具有或相当理工科或管理专业硕士研究生毕业以上水平；
b) 经软件工程和软件安全性方面的专业培训并经考试合格；
c) 具有下列方面之一种或者多种工作经历(经历长短由有关组织自定)：
   1) 系统安全性管理；
   2) 系统安全性分析；
   3) 系统安全性设计；
   4) 系统安全性研究；
   5) 安全性系统的使用；
   6) 软件安全性管理；
   7) 软件安全性分析；
   8) 软件安全性设计；
   9) 软件安全性研究；

10） 事故调查；
11） 人素工程；
12） 产品质量保证工程；
13） 可靠性工程；
14） 维修工程。

注：软件安全性分析组织中人员的工作经历应该具有互补性(例如，有人要熟悉系统和硬件安全性)；对不同的组织的分析人员的要求不一定相同；不同组织之间的分析活动具有互补性，其组织人员从整体上也同样应该具有互补性。关于安全性培训的内容见 GJB 900－90《系统安全性通用大纲》的 5.4。

附 录 F
（资料性附录）
初步危险表

本附录的表 F.1 给出了一个初步危险表的示例。对于特定的系统环境，需要开发相应的初步危险表。

表 F.1 初步危险表

1.污染/腐蚀	化学分解 化学取代/化合 潮气 氧化 生物(霉菌/细菌,等等) 粒子 无机物(包括石棉)
2.释放电/电击	外部电击 内部电击 静电释放 电晕 短路
3.环境/气候	雾 闪电 沉降(雾/雨/雪/冻雨/冰雹) 沙/灰尘 真空 风 温度极值
4.火灾/爆炸	化学变化(放热的/吸热的) 燃料和氧化剂(高压、火源) 压力释放/挤压 高热源
5.撞击/碰撞	加速度(包括重力) 分离的设备 机械冲击/振动/音响 流星体/陨石 移动/旋转的设备
6.丧失可居住的环境	污染 高压 氧含量低 低压 毒性 低温 高温
7.病理/生理/心理	加速度/冲击/撞击/振动 大气压力(高、低、快速变化) 湿度 疾病 噪声 尖锐的棱边 睡眠(缺少) 可见性(强光，窗户/防护帽发雾) 温度 工作负荷(超额) 高处(可能跌落)

表 F.1(续)

8.辐射	电磁辐射 电离辐射(包括氡) 非电离辐射
9.温度极值	高 低 变化
注：健康问题需要与职业保健人员协商。	

参 考 文 献

1. GJB 438A－97 武器系统软件开发文档
2. GJB 5235 军用软件配置管理
3. GJB 5234 军用软件验证与确认
4. GJB/Z 141 军用软件测试指南