ICS 35.080
L 77

中华人民共和国国家标准

GB/T 38634.4—2020

系统与软件工程　软件测试
第4部分：测试技术

Systems and software engineering—Software testing—Part 4：Test techniques

（ISO/IEC/IEEE 29119-4：2015，Software and systems engineering—Software testing—Part 4：Test techniques，MOD）

2020-04-28 发布　　2020-11-01 实施

国家市场监督管理总局
国家标准化管理委员会　发布

目　　次

前　言

GB/T 38634《系统与软件工程　软件测试》分为以下4个部分：

——第1部分：概念和定义；

——第2部分：测试过程；

——第3部分：测试文档；

——第4部分：测试技术。

本部分为GB/T 38634的第4部分。

本部分按照GB/T 1.1—2009给出的规则起草。

本部分使用重新起草法修改采用ISO/IEC/IEEE 29119-4：2015《软件与系统工程　软件测试　第4部分：测试技术》。

本部分与ISO/IEC/IEEE 29119-4：2015的技术性差异及其原因如下：

——关于规范性引用文件，本部分做了具有技术性差异的调整，以适应我国的技术条件，调整的情况集中反映在第3章“规范性引用文件”中，具体调整如下：

- 用修改采用国际标准的GB/T 38634.1代替了ISO/IEC/IEEE 29119-1；
- 用修改采用国际标准的GB/T 38634.2代替了ISO/IEC/IEEE 29119-2；
- 用修改采用国际标准的GB/T 38634.3代替了ISO/IEC/IEEE 29119-3；
- 删除了ISO/IEC/IEEE 29119-4：2015。

本部分还做了下列编辑性修改：

——将标准名称改为《系统与软件工程　软件测试　第4部分：测试技术》；

——删除了A.1.1、A.3.1、A.4.1、B.1.1、C.1.1、D.1.1、F.1.1的标题；

——删除了E.1.1标题，将E.1.2和E.1.2.1～E.1.2.8标题顺改；

——将附录B中国外的地名和货币修改为国内的地名和人民币；

——将附录G中的内容由“本部分与BS 7925-2映射过程”修改为与GB/T 15532—2008的映射过程。

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

本部分由全国信息技术标准化技术委员会(SAC/TC 28)提出并归口。

本部分起草单位：上海计算机软件技术开发中心、中国电子技术标准化研究院、中国航天系统科学与工程研究院、深圳赛西信息技术有限公司、国家应用软件产品质量监督检验中心、苏州洞察云信息技术有限公司、中国航发控制系统研究所、中国电子科技集团公司第五十四研究所、中国电子科技集团公司第十研究所、重庆市软件评测中心有限公司、北京航空航天大学、广东省科技基础条件平台中心、厦门理工学院、北京跟踪与通信技术研究所、北京轩宇信息技术有限公司、西宁市大数据服务管理局、中国司法大数据研究院有限公司、北方民族大学、中电莱斯信息系统有限公司、中国航天科工集团第三研究院第三〇四所、浙江省电子信息产品检验所、山东道普测评技术有限公司、南京大学、福建省电子产品监督检验所、内蒙古安盾信息安全评测有限公司、上海同思廷软件技术有限公司、上海第二工业大学。

本部分主要起草人：龚家瑜、李文鹏、张旸旸、左振雷、蔡立志、康京山、胡芸、孙云、沈颖、张元元、白万芳、王瑞、潘宇聪、赵明、卢俊文、吕雪、于志杰、杨隽、丁晓明、尹平、路云峰、张峻、郑丽娜、刘伟、徐宝文、王建强、赵毅、韩强、王凤玲、柳毓龙、徐跃伟、郭新伟、周震漪、孟宪伟、李丽萍。

引　言

本部分使用了 GB/T 38634.2 的测试设计和实现过程来描述测试设计技术(也称为测试用例设计技术或测试方法)。本部分没有给出测试设计和实现的过程,而是描述了可以在 GB/T 38634.2 中使用的技术。其目的是描述一系列软件测试行业广泛接受的技术。

本部分中说明的测试设计技术可用于导出测试用例,执行测试用例后会产生结果,符合测试项的要求或测试项存在缺陷(即没有达到要求)。基于风险的测试用于确定特定情况下使用的技术集(基于风险的测试在 GB/T 38634.1 和 GB/T 38634.2 中进行说明)。

注:"测试项"是需要测试的工作产品(见 GB/T 38634.1)。

示例:"测试项"包括系统、软件项、对象、类、需求文档、设计规格说明和用户指南。

每项技术都符合 GB/T 38634.2 中定义的测试设计和实现过程,如图 1 所示。

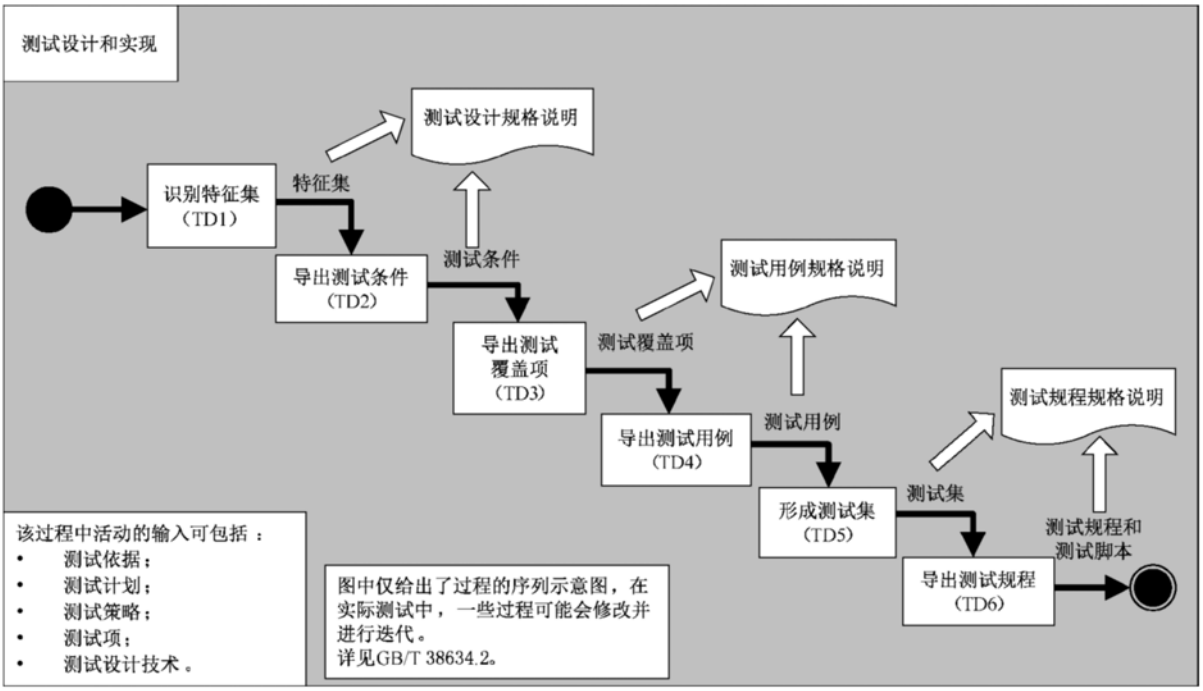


图 1　GB/T 38634.2 中所述的测试设计和实现过程

ISO/IEC TR 19759 定义了两种类型的需求:功能性需求和质量需求。GB/T 25000.10—2016 定义了八个质量特性(包括功能性),用于识别适合测试特定测试项的测试类型。附录 A 提供了用于测试 GB/T 25000.10—2016 中定义的测试质量特性对应的测试设计技术。

基于经验的测试实践比如探索性测试,或者其他测试实践比如基于模型的测试不在本部分中定义,因为本部分只描述设计测试用例的技术。测试实践如探索性的测试在 GB/T 38634.1 中描述。

测试过程中生成的测试文档的模板和例子在 GB/T 38634.3 中定义。本部分中的测试技术对测试用例的文档记录不进行说明(例如不包括分配唯一标识、测试用例描述、优先级、可追溯性或前置条件的信息或指导)。关于如何记录测试用例的信息见 GB/T 38634.3。

本部分的目的是提供给任何组织的利益相关方设计测试用例的能力。

系统与软件工程　软件测试
第4部分:测试技术

1　范围

GB/T 38634的本部分定义了在GB/T 38634.2测试设计和实现过程中使用的测试设计技术。

本部分适用于但不限于测试人员、测试经理、开发人员和项目经理,特别是那些负责管理和实施软件测试的人员。

2　符合性

2.1　预期用途

本部分的规范性要求包含在第5章和第6章中。由于特定项目或组织可能不需要使用本部分定义的所有技术,因此,本部分的实施通常涉及选择适合于项目或组织的一组技术。组织或个人可以通过两种方式声明符合本部分的规定:完全符合或剪裁符合。组织或个人应声明是否要求完全或剪裁符合本部分。

2.2　完全符合性

通过证明满足选定的第5章中技术集(不为空)和/或相应的第6章中测试覆盖率测量方法的所有要求(即描述为"应"的语句),则可声称完全符合。

示例:组织可以选择仅符合一种技术,例如边界值分析。在这种情况下,组织只需要提供证据证明他们已满足该技术的要求,可以声称符合本部分。

2.3　剪裁符合性

通过证明已经满足来自所选技术集(不为空)和/或相应的测试覆盖率测量方法的所选择要求的子集来实现剪裁符合。在剪裁情况下,如不完全遵循第5章中定义的技术或第6章中定义的测量规范性要求,应当提供理由(无论是直接剪裁或者是通过参考剪裁)。所有剪裁决策都应记录其理由,包括考虑任何适用的风险。剪裁应由利益相关方商定。

3　规范性引用文件

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

GB/T 38634.1　系统与软件工程　软件测试　第1部分:概念和定义(GB/T 38634.1—2020,ISO/IEC/IEEE 29119-1:2013,MOD)

GB/T 38634.2　系统与软件工程　软件测试　第2部分:测试过程(GB/T 38634.2—2020,ISO/IEC/IEEE 29119-2:2013,MOD)

GB/T 38634.3　系统与软件工程　软件测试　第3部分:测试文档(GB/T 38634.3—2020,ISO/IEC/IEEE 29119-3:2013,MOD)

4 术语和定义

下列术语和定义适用于本文件。

4.1

巴科斯范式 Backus-Naur Form

一种形式化的元语言。用于以文本格式定义一种语言的语法。

4.2

基本选择 base choice

见**基本值**(4.3)。

4.3

基本值 base value

在“基本选择测试”中输入参数的取值,通常依据参数的代表值或典型值进行选择。也称为基本选择。

4.4

计算使用 c-use

见**计算数据使用**(4.5)。

4.5

计算数据使用 computation data use

在任何类型语句中,变量值的使用。

4.6

条件 condition

不包含布尔运算符的布尔表达式。

示例:“A<B”是条件,“A and B”不是条件。

4.7

控制流 control flow

在测试项运行期间所执行操作的序列。

4.8

控制流子路径 control flow sub-path

测试项中的可执行语句的序列。

4.9

数据定义 data definition

变量赋值语句。也被称为变量定义。

4.10

数据定义计算使用对 data definition c-use pair

数据定义和后续计算的数据使用,其中数据使用是使用数据定义中定义的值。

4.11

数据定义谓词使用对 data definition p-use pair

数据定义和后续谓词的数据使用,其中数据使用是使用数据定义中定义的值。

4.12

数据定义使用对 data definition-use pair

数据定义和后续的数据使用,其中数据使用是使用数据定义中定义的值。

4.13

数据使用　data use

访问变量值的可执行语句。

4.14

判定结果　decision outcome

判定式的结果，可用于决定控制流选择方向。

4.15

判定规则　decision rule

在判定表测试和因果图中产生特定结果的条件（也称为原因）和动作（也称为结果）的组合。

4.16

定义使用对　definition-use pair

数据定义和后续谓词的数据使用或计算的数据使用，其中数据使用是使用数据定义中定义的值。

4.17

定义使用路径　definition-use path

从变量定义到谓词使用（p-use）或计算使用（c-use）的控制流子路径。

4.18

入口点　entry point

测试项中测试项开始执行的点。

注：入口点是测试项的可执行语句，可由外部进程选择作为通过测试项的一个或多个路径的起点。它通常是测试项中的第一条可执行语句。

4.19

可执行语句　executable statement

在编译后将转换为目标代码的语句，该目标代码将在测试项运行时以程序化方式执行，并可能对程序数据执行操作。

4.20

出口点　exit point

测试项的最后一条执行的语句。

注：出口点是通过测试项路径的终点，是测试项中的可执行语句。它或终止测试项，或将控制权返回到外部进程。这通常是测试项中的最后一条可执行语句。

4.21

谓词使用　p-use

见**谓词数据使用**（4.25）。

4.22

键值对　P-V pair

测试项参数与赋给该参数的值的组合。在组合测试设计技术中，用作测试条件和测试覆盖项。

4.23

路径　path

测试项中可执行语句的序列。

4.24

谓词　predicate

计算结果为“真”或“假”的逻辑表达式，通常用于引导代码中的执行路径。

4.25

谓词数据使用　predicate data use

与判定语句中谓词部分的判定结果相关联的“数据使用”。

4.26

子路径　sub-path

一个较大路径的一部分。

4.27

测试模型　test model

在测试用例设计过程中使用的测试项的表示。

4.28

变量定义　variable definition

见**数据定义**(4.9)。

5　测试设计技术

5.1　概述

本部分定义了基于规格说明的测试设计技术(5.2)、基于结构的测试设计技术(5.3)和基于经验的测试设计技术(5.4)。在基于规格说明的测试中,测试依据(如需求,规格说明,模型或用户需求)是设计测试用例的首要信息来源。在基于结构的测试中,测试项的结构(如源代码或模型结构)是设计测试用例的首要信息来源。在基于经验的测试中,测试人员的知识和经验是设计测试用例的首要信息来源。对于基于规格说明的测试、基于结构的测试和基于经验的测试,测试依据用于生成预期结果。上述测试设计技术是互补的,组合使用这些技术会使测试更加有效。

虽然在本部分中介绍的技术被划分为基于规格说明、基于结构和基于经验三类,但在实际使用中这些技术可以互换使用(如分支测试可以设计测试用例来测试一个互联网系统图形用户界面的逻辑路径)。附录 E 中对此进行了举例。此外,尽管每种技术都是独立于所有其他技术定义的,但实际上它们可以与其他技术结合使用。

示例 1:通过等价类划分导出的测试覆盖项可用于场景测试导出的测试用例的输入参数。

本部分使用基于规格说明的测试和基于结构的测试的术语,这种技术分类也称为“黑盒测试”和“白盒测试”。术语“黑盒测试”和“白盒测试”指的是测试项内部结构的可见性。对于黑盒测试,测试项的内部结构是不可见的;对于白盒测试,测试项的内部结构是可见的。当一个测试设计技术同时基于测试项的规格说明和结构时,该测试设计技术被称为“灰盒测试”。

本部分定义了 GB/T 38634.2(参见引言)中所述的通用测试设计和实现过程中步骤 TD2(导出测试条件)、TD3(导出测试覆盖项)和 TD4(导出测试用例)是如何应用于各测试设计技术的。本部分不提供使用这些技术的特定周境定义,即不说明每个技术在所有情况下使用的方法。本部分的用户可以参考附录 B、附录 C、附录 D 和附录 E,以获得如何应用这些技术的详细示例。附录 F 对基于结构的测试设计技术之间的覆盖关系给出了说明。

本部分定义的技术如图 2 所示,这组技术并不是全面的,部分测试人员和研究人员使用的技术未包含在本部分中。附录 G 给出了本部分定义的技术与 GB/T 15532—2008 中测试设计技术的映射关系。

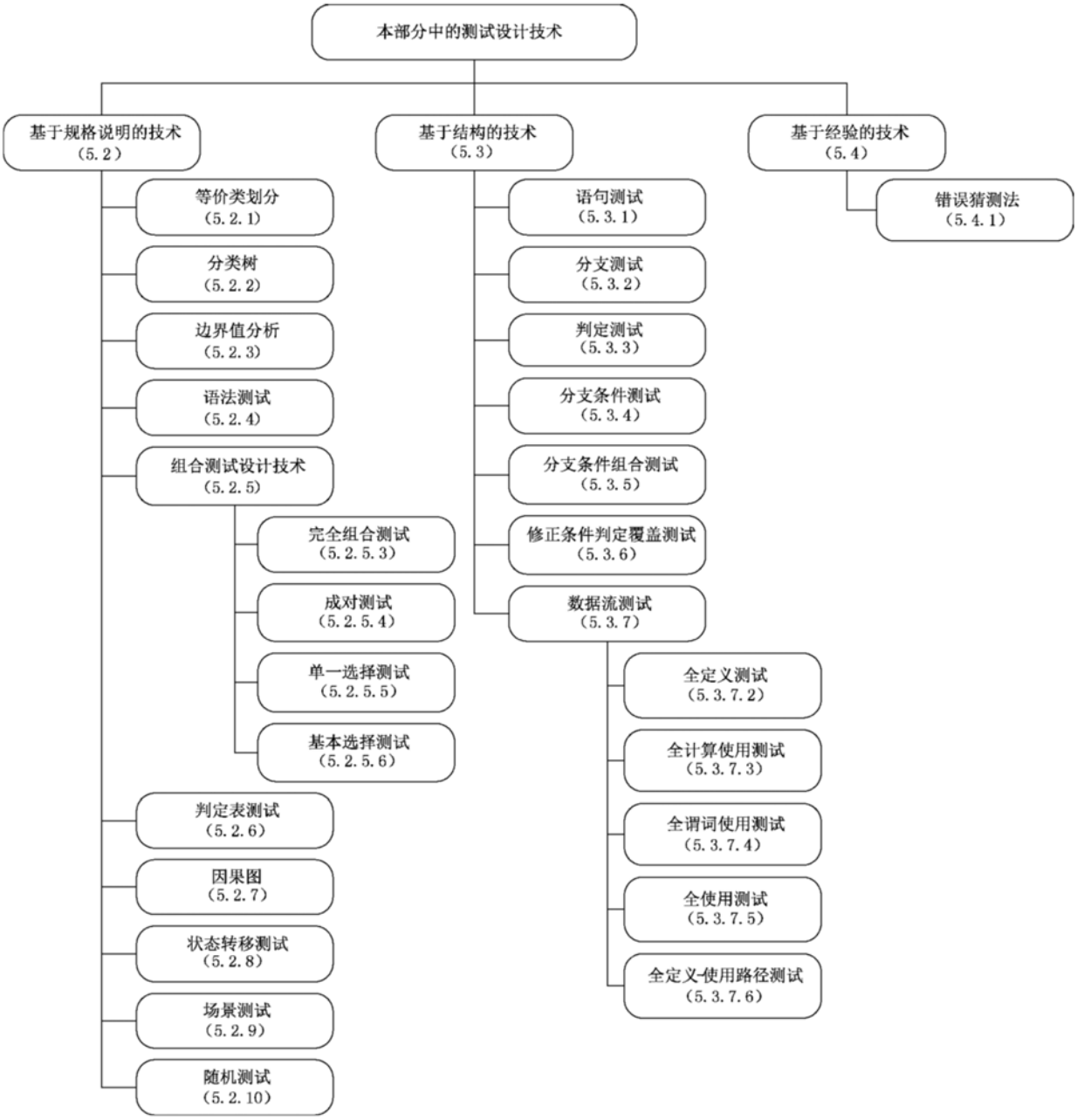


图 2　本部分包含的测试设计技术

在测试设计和实现过程的 6 个活动中，测试设计技术为导出测试条件（TD2）、导出测试覆盖项（TD3）和导出测试用例（TD4）提供了独特而具体的指导。因此，每项技术都是采用这三个活动来定义的。

步骤 TD2（导出测试条件）、TD3（导出测试覆盖项）和 TD4（导出测试用例）中存在不同级别的粒度。在每种技术中，术语“模型”用于描述准备测试项的逻辑，以便在步骤 TD2 中导出测试条件（例如，需要控制流模型来导出所有基于结构技术的测试条件）。某些情况可能要求整个模型作为测试条件，而在其他情况下，模型的一部分可作为测试条件。

示例 2：在状态转移测试中，如果需要覆盖所有状态，则整个状态模型是测试条件。如果需要覆盖状态之间的特定转换，则每次转换都是一个测试条件。

此外，由于某些技术有着共同的基础概念，所以它们的定义包含了相似的内容。

示例 3：等价类划分和边界值分析都基于等价类。

在每种技术的导出测试用例步骤(TD4)中，创建的测试用例可以是“有效的”(即测试项的输入被当作正确而接受)或“无效”(即测试项至少有一个输入被当作不正确而拒绝，理想情况下给出适当的错误提示)。在某些技术中，例如等价类划分和边界值分析，通常使用“一对一”方法导出无效测试用例，因为它通过确保每个测试用例仅包含一个无效输入值来避免故障屏蔽，同时有效测试通常使用“最小化”方法导出用例，因此减少了覆盖有效测试覆盖项所需的测试用例数量(见 5.2.1.3 和 5.2.3.3)。

注：无效用例也被称为“负面测试用例”。

尽管本部分中定义的技术各自在单独的条目中进行了描述(看上去是互相排斥的)，实际上它们可以结合使用。

示例 4：可以使用边界值分析来选择测试输入值，之后使用成对测试来根据测试输入值设计测试用例。可以使用等价类划分来选择分类树的分类和所分的类，然后根据所分的类使用单一选择测试(技术)来构造测试用例。

本部分提出的技术也可以与附录 A 中提供的测试类型结合使用。例如，等价类划分可用于易用性测试中识别用户组(测试条件)和代表性用户(测试覆盖项)。

本章给出了技术的规范性定义。第 6 章给出了每种技术的相应规范性覆盖测量。附录 B、附录 C、附录 D 和附录 E 给出了每种技术的资料性案例。尽管本部分给出每种技术的示例都是人工操作的，但在实践中，可采用自动化操作来支持某些类型的设计和执行(例如，语句覆盖分析器可用于支持基于结构的测试)。附录 A 给出了如何将本部分中定义的测试设计技术应用于测试 GB/T 25000.10—2016 中定义的质量特性的示例。

5.2 基于规格说明的测试设计技术

5.2.1 等价类划分

5.2.1.1 导出测试条件（TD2）

等价类划分使用测试项模型将测试项输入和输出划分为等价类(也称为“分区”)，其中每个等价类都应当作为一个测试条件。这些等价类应该从测试依据导出，对于每个分区中的所有值，都可以被测试项类似地处理(即等价类中的值是“等同的”)。有效的输入输出以及无效的输入输出均可导出等价类划分。

示例：对于期望小写字母字符作为(有效)输入的测试项，可以派生的无效输入等价类包括包含整数、实数、大写字母字符、符号和控制字符的等价类，具体取决于测试期间所需的严谨程度。

注 1：对于输出等价类，基于测试项规格说明中描述的过程导出相应的输入分区。然后从输入分区中选择测试输入。

注 2：无效的输出等价类通常对应于未明确指定的任何输出。由于未指定，通常会根据各测试人员的主观判断得到等价类。当应用基于经验的技术(如错误猜测)时，也可能出现这种主观形式的测试设计。

注 3：域分析法通常归类为等价类划分和边界值分析的结合。

5.2.1.2 导出测试覆盖项（TD3）

每个等价类应该是一个测试覆盖项(即在等价类划分中，测试条件和测试覆盖项是同样的等价类)。

5.2.1.3 导出测试用例（TD4）

导出的测试用例应该实现每个测试覆盖项(即等价类)。下面是导出测试用例的步骤：

a) 确定选择测试用例所实现测试覆盖项的组合方法，下面是两种常见的方法：
   1) 一对一，导出的每个测试用例用于覆盖一个特定的等价类；
   2) 最小化，其中等价类由测试用例覆盖，使得导出的最小测试用例数至少覆盖所有等价类一次。

注：在5.2.5(组合测试设计技术)中给出了选择测试用例实现测试覆盖项的其他方法。

b) 采用步骤a)中的方法选择包含在当前测试用例中的测试覆盖项。

c) 确定为执行测试用例所涵盖的测试覆盖项的输入值，以及测试用例所需的任何其他输入变量的任意有效值。

d) 将输入应用于测试依据来确定测试用例的预期结果。

e) 重复步骤b)～d)直到达到要求的测试覆盖率。

5.2.2 分类树

5.2.2.1 导出测试条件（TD2）

分类树方法使用测试项模型将测试项的输入进行划分，并且用分类树的方式进行图形化表示。测试项的输入被分为若干个“分类”，每个划分由若干个独立(不重叠)的“类”和子类组成，同时分类集是完整的(被建模测试项的所有输入域都被识别并包括在所有分类内)。每个分类应是一个测试条件。根据测试的严格程度，通过分解分类得到的“类”可能会进一步分为“子类”。根据要求的测试覆盖程度，导出的划分和类可能同时包括有效和无效的输入数据。将分类、类和子类之间的层次关系塑造成一棵树，测试项的输入域作为树的根节点，分类作为分支节点，类或者子类作为叶节点。

注：分类树方法中的分区过程类似于等价类划分。其关键的区别在于，在分类树方法中，划分(分类和类别)必须完全不相交，而在等价类划分中，它们可能会重叠，具体取决于技术的应用方式。此外，分类树方法还包括分类树的设计，其提供测试条件的视觉展示。

5.2.2.2 导出测试覆盖项（TD3）

测试覆盖项应采用所选的组合方法由组合分类来导出。

示例：将组合分类导出测试覆盖项的两个示例：

——最小化，导出最小数量的测试覆盖项，至少覆盖所有分类一次。

——最大化，每个可能的分类组合至少由一个测试覆盖项覆盖。

注1：5.2.5(组合测试设计技术)描述了通过测试用例选择测试覆盖项组合的其他方法。

注2：测试覆盖项通常由组合表来表示(见B.2.2.5的图B.5)。

注3：分类树方法的最早的出版物中，采用了术语“最小值”和“最大值”来代替“最小化”和“最大化”。

5.2.2.3 导出测试用例（TD4）

导出的测试用例应该实现每个测试覆盖项。下面是导出测试用例的步骤：

a) 在步骤TD3产生的分类组合的基础上，为当前测试用例选择一个组合，要求该组合没有被测试用例覆盖；

b) 确定每个类别中尚未赋值的输入值；

c) 通过将输入应用到测试依据中来确定测试用例的预期结果；

d) 重复步骤a)～c)，直到达到要求的测试覆盖率水平。

5.2.3 边界值分析

5.2.3.1 导出测试条件（TD2）

边界值分析是通过对测试项模型边界值的分析，将测试项的输入和输出划分为具有可识别边界的多个有序集和子集(分区和子分区)，其中每个边界是测试条件。边界应来自测试依据。

示例：一个分区的定义是从1～10的整数，该分区有两个边界，下边界是1，上边界是10，这些是测试条件。

注：对于输出边界，可基于测试项规格说明中描述的过程导出相应的输入分区。然后从输入分区中选择测试输入。

5.2.3.2 导出测试覆盖项（TD3）

可选择以下两种方法的一种来导出测试覆盖项：

——二值边界测试；

——三值边界测试。

对于二值边界测试，应为每个边界（测试条件）导出两个测试覆盖项，这些边界对应于边界上的值和该边界外一定增量距离的等价类。该增量距离应定义为所考虑的数据类型的最小有效值。

对于三值边界测试，应为每个边界（测试条件）导出三个测试覆盖项，这些边界对应于边界上的值和该边界每一侧有一定增量距离的等价类。该增量距离应定义为所考虑的数据类型的最小有效值。

注 1：某些分区在测试依据上只能识别出一个边界。例如，数据分区“年龄＞＝70 岁”有一个下边界但没有明显的上边界。在某些情况中，实际实现中强加的边界可以作为边界值，比如输入域可接受的最大值（此类决策需进行记录；例如，记录在测试规格说明文档中）。

注 2：二值边界测试在大多数情况下是充分的；但是，在某些情况下可能需要进行三值边界值测试（例如，测试人员和开发人员在严格测试确认测试项中，变量的边界没有错误发生时）。

注 3：在进行二值边界或三值边界测试时，连续分区（分区共享边界）将导致重复的测试覆盖项。在这种情况下，典型的做法是仅执行一次这些重复值。有关重复边界的示例，见 B.2.3.4.3。

5.2.3.3 导出测试用例（TD4）

导出的测试用例应实现每个测试覆盖项。下面是导出测试用例的步骤：

a) 确定选择测试用例所实现测试覆盖项的组合方法，有两种常见的方法：

1) 一对一，每个测试用例实现一个指定的边界值；

2) 最小化，导出最小数量的测试用例以覆盖所有边界值至少一次。

注 1：在最小化边界值分析中，每个测试用例可以覆盖多个测试覆盖项。

注 2：5.2.5（组合测试设计技术）描述了通过测试用例选择测试覆盖项组合的其他方法。

b) 采用步骤 a）中的方法选择当前测试用例包含的测试覆盖项。

c) 步骤 b）中测试用例没有选择的其他输入变量取任意有效值。

d) 通过将输入应用到测试依据中来确定测试用例的预期结果。

e) 重复步骤 b）～d），直到达到要求的测试覆盖率水平。

5.2.4 语法测试

5.2.4.1 导出测试条件（TD2）

语法测试使用测试项输入形式化的语法作为测试设计的依据。语法模型表示为多个规则，其中每个规则根据语法中的元素“序列”、元素“迭代”或元素“之间的选择”来定义输入参数的形式。语法可以以文本或图形的形式表示。语法测试中的测试条件应为测试项输入的全部或部分模型。

示例 1：巴科斯范式是一种形式化元语言，以文本形式定义一个测试项的语法。

示例 2：抽象语法树可以图形化地表示形式化语法。

5.2.4.2 导出测试覆盖项（TD3）

在语法测试中，基于两个目标导出测试覆盖项：正面测试，导出的测试覆盖项以各种方式覆盖有效语法；负面测试，导出的测试覆盖项以各种方式故意违反规则语法。正面测试的测试覆盖项应是已定义语法的“选项”，而负面测试的测试覆盖项应是已定义语法的“变异”。

以下指导原则可以用于导出“选项”（在合适的地方可以使用替代的指导原则）：

——每当语法强制选择时，就为该选择的每个备选方案导出“选项”。

示例 1：对于输入变量“颜色＝蓝色|红色|绿色”(|表示“或”布尔运算符)，则导出三个选项“蓝色”“红色”和“绿色”作为测试覆盖项。

——每当语法强制执行迭代时，为此迭代导出至少两个“选项”；一个包含了最小重复次数，另一个则大于最小重复次数。

示例 2：输入变量“字母＝[A - Z | a - z]$^{+}$”(“+”表示“一个或多个”)，则导出两个选项“一个字母”和“多个字母”作为测试覆盖项。

——每当迭代被要求具有最大重复次数时，为此迭代导出至少两个“选项”；一个具有最大重复次数，另一个则超过最大重复次数。

示例 3：输入变量“字母＝[A - Z | a - z]^{100}”(“100”表示一个字母最多选择 100 次)，至少生成两个选项“100 个字母”和“少于 100 个字母”作为测试覆盖项。

下面的指导原则可以用于生成“变异”(在合适的地方可以使用替代的指导原则)：

——对于任何输入，可以改变已定义的语法用以导出无效输入(“变异”)。

示例 4：输入变量“颜色＝蓝色|红色|绿色”，一个变异可能引入一个无效的变量值作为测试覆盖项，例如选择颜色“黄色”，“黄色”没有在输入变量列表中出现。其他变异的例子将在 B.2.4.5 中说明。

5.2.4.3 导出测试用例（TD4）

语法测试导出测试用例应覆盖所选择的选项和变异。下面是导出测试用例的步骤：

a) 确定选择测试用例所实现测试覆盖项组合的方法，有两种常用方法：
   1) 一对一，每个测试用例用于实现一个特定的选项、迭代和/或变异；
   2) 最小化，选项、迭代和/或变异包含在测试用例中，以便导出最小数量的测试用例，使之覆盖所有选项，迭代和/或变异至少一次。

   注：5.2.5(组合测试设计技术)描述了通过测试用例选择测试覆盖项组合的其他方法。
b) 选择当前测试用例包含的测试覆盖项。
c) 确定测试用例所实现的测试覆盖项的输入值，测试用例所需的其他输入变量取任意有效值。
d) 通过将输入应用到测试依据中，来确定测试用例的预期结果。
e) 重复步骤 b)～d)，直到实现了所有选项、迭代和/或变异。

5.2.5 组合测试设计技术

5.2.5.1 概述

组合测试设计技术用于系统地导出有效且可控的测试用例子集，其在测试期间覆盖了导出的测试条件和测试覆盖项。组合是根据测试项参数和参数可取的值来定义的。当多参数(每个参数都有大量离散值)必须相互作用的情况下，这种技术可以显著减少所需的测试用例数量，而不会影响功能覆盖率。

5.2.5.2 导出测试条件（TD2）

测试项参数表示与测试相关的测试项的特定方面，通常对应于测试项的输入参数，但是也可以是其他方面。

示例 1：在配置测试中，参数可能是各种环境因素，例如操作系统和浏览器。

每个测试项参数可以取不同的值。使用本技术的值域应是有限并且可控的。一些测试项参数可能受到自然约束只能取一小部分的值，而另一些测试项参数可能约束较少。

示例 2：测试项的参数受到自然约束只能取少量的值——“星期几”，只能取 7 个特定的值“[星期一|星期二|星期三|星期四|星期五|星期六|星期日]”。

示例 3：测试项参数有很少的约束——参数由任何实数组成，可以无限取值。

对于没有约束的测试项，有必要先使用其他测试设计技术，比如等价类划分或边界值分析，可以将一个很大的取值范围减少到一个可控的子集。

组合测试的测试条件对于所有的组合测试技术相同；每个测试条件应该是选择的测试项参数(P)，其具有特定值(V)，形成键值对。

5.2.5.3 完全组合测试

5.2.5.3.1 导出测试覆盖项（TD3）

在完全组合测试中，测试覆盖项是所有唯一的键值对的集合，使得每个参数在此集合中至少包含一次。

5.2.5.3.2 导出测试用例（TD4）

应导出测试用例，其中每个测试用例实现一个唯一的键值对组合。下面是导出测试用例的步骤：

a) 选择没有被测试用例覆盖的测试覆盖项，使其包含在当前测试用例中；

b) 通过将输入应用到测试依据中来确定测试用例的预期结果；

c) 重复步骤 a)和 b)直到达到要求的测试覆盖率水平。

注：要达到 100%完全组合测试覆盖，最少需要的测试用例数量和测试覆盖项参数的键值对数量一致。

5.2.5.4 成对测试

5.2.5.4.1 导出测试覆盖项（TD3）

在成对测试中，测试覆盖项应该是唯一的不同的测试项参数的键值对组合。本技术不是覆盖所有参数的可能组合(如完全组合测试中的要求)，而是覆盖总测试集中所有选定值的可能组合，从而用更少的测试覆盖项有效地实现测试覆盖。成对测试也被称为“完全对”测试。

5.2.5.4.2 导出测试用例（TD4）

首先识别键值对，导出测试以执行用例用来测试两个键值对组合，其中每个测试用例可测试一个或者多个唯一对组合。下面是导出测试用例的步骤：

a) 选择当前测试用例包含的测试覆盖项，其中键值对组合中，每一对都覆盖了一个没有被先前测试用例包含的参数取值对；

b) 测试用例中其他的输入变量取任意有效值；

c) 通过将输入应用到测试依据中来确定测试用例的预期结果；

d) 重复步骤 a)～c)直到键值对组合中所有唯一对都被实现。

很难计算达到 100%成对测试覆盖率所需的最小测试用例数。可以用以下三种方法中的其中一种计算得到一个接近最优集合：

——使用算法手动确定一个接近最优集合；

——使用自动化工具(实现算法)来确定接近最优集合；

——使用正交数组来确定一个接近最优集合。

5.2.5.5 单一选择测试

5.2.5.5.1 导出测试覆盖项（TD3）

在单一选择(或 1-对)测试中，测试覆盖项应该是键值对集合的成员，每个参数值至少包含在集合中一次。

5.2.5.5.2 导出测试用例（TD4）

导出的测试用例应实现一个或者多个未被先前测试用例包含的键值对。下面是导出测试用例的

步骤：

a) 选择当前测试用例包含的测试覆盖项，其键值对覆盖了至少一个先前测试用例未包含的测试覆盖项；

b) 测试用例中其他的输入变量取任意有效值；

c) 通过将输入应用到测试依据中，来确定测试用例的预期结果；

d) 重复步骤 a)～c)直到达到要求的测试覆盖率水平。

单一选择测试达到 100％覆盖率所需的最少测试用例数取决于测试项中拥有最多值的参数。

5.2.5.6 基本选择测试

5.2.5.6.1 导出测试覆盖项（TD3）

在基本选择测试中，测试覆盖项应为每个输入参数的键值对的集合。其中，除了一个参数之外的所有其他参数都被设为“基本”值，而最后的参数被设为其他有效值之一。

注：有许多选取参数基本值的方法。例如，从操作手册、场景测试的基本路径、等价类划分导出的测试覆盖项或者参数的默认值(最常用)中选择。

5.2.5.6.2 导出测试用例（TD4）

首先确定键值对之后，应选择每个参数的基本选择。通过将除了一个参数之外的所有参数设置为其基本选择，随后将最后的参数设置为有效值，然后导出测试用例，直到达到键值对的所需测试覆盖水平。下面是导出测试用例的步骤：

a) 通过设置参数为“基本”值来导出基本选择测试用例。

b) 建立一个新的测试用例，设置某个一个参数为有效值(非基本值)，其他参数仍为基本值。

c) 通过将输入应用到测试依据中来确定测试用例的预期结果。

d) 重复步骤 b)、c)直到达到预期的测试覆盖率水平。

5.2.6 判定表测试

5.2.6.1 导出测试条件（TD2）

判定表测试以判定表的形式使用了测试项条件(原因)和动作(结果)之间的逻辑关系(判定规则)模型，包括：

——测试项的每个布尔条件定义了一对输入等价类，一个对应“真”，一个对应“假”；

——每个动作是测试项的预期结果或结果的组合，其表示为布尔值；

——一组判定规则表示了条件和动作之间的关系。

测试条件应为条件和动作。

注：如果条件由多个值而不是简单的布尔值组成，则会产生“扩展条目”判定表，其测试可以通过等价类划分来处理。

5.2.6.2 导出测试覆盖项（TD3）

在判定表测试中，定义测试项条件和动作唯一组合之间关系的每个判定规则作为测试覆盖项。

5.2.6.3 导出测试用例（TD4）

应导出测试用例以执行判定规则(测试覆盖项)。其中，每个测试用例定义了输入和输出之间的关系，且每个判定规则对应于布尔条件的唯一组合。下面是导出测试用例的步骤：

a) 从判定表中选择测试覆盖项作为测试用例实现；

b) 确定输入值以满足测试用例所覆盖判定规则的输入条件，以及执行测试用例所需的任何其他输入变量的任意有效值；

c) 通过将输入应用于判定表，来确定测试用例的预期结果；

d) 重复步骤 a)～c)，直到达到所需测试覆盖率水平。

注：如果判定表包含了关联的输入条件，则这会导致不可行的组合（例如："年龄小于 18 岁"和"年龄大于 65 岁"均为真）。该情况宜识别并记录无意义的判定规则，不用于导出测试用例。

5.2.7 因果图

5.2.7.1 导出测试条件（TD2）

因果图法使用以因果图形式表示测试项原因（例如输入）和结果（例如输出）之间的逻辑关系（判定规则）模型，包括：

——测试项的每个布尔原因定义了一对输入等价类，一个对应"真"的情况，一个对应"假"的情况；

——每个结果定义为测试项的预期结果条件或输出条件组合，表示为布尔值。

测试条件应为原因和结果。

因果图将原因和结果之间的逻辑关系建模为布尔运算符加权的布尔逻辑网络，并根据原因之间的关系和结果之间的关系进行语义和约束建模（见 B.2.7.4 中的图 B.11 和图 B.12）。

5.2.7.2 导出测试覆盖项（TD3）

在因果图中，每个判定规则定义了测试项的原因和结果之间唯一的组合关系，其可作为一个测试覆盖项。

5.2.7.3 导出测试用例（TD4）

导出的测试用例应当实现对应的测试覆盖项。可以根据因果图产生相应的判定表，并用于导出测试用例。下面是导出测试用例的步骤：

a) 选择当前测试用例需实现的测试覆盖项。

b) 确定输入值以执行测试用例所实现的测试覆盖项，以及执行测试用例所需的任意其他输入变量的任意有效值。

c) 通过将输入应用于因果图和/或判定表来确定测试用例的预期结果。

d) 重复步骤 a)～c)直到达到要求的测试覆盖率水平。

5.2.8 状态转移测试

5.2.8.1 导出测试条件（TD2）

状态转移测试使用测试项的状态模型，模型包含测试项可能的状态、状态间的转移，导致转移的实际和转移可能导致的动作。模型的状态应该是离散的、可识别的和数量有限的。单个转移可由事件守护进行约束，事件守护定义了一组条件，当事件发生时，该组条件应为真，从而发生转移。在状态转移测试中，测试条件可以是状态模型的所有状态、状态模型的所有转移或整个状态模型，这取决于测试的覆盖要求。模型可以表示为状态转移图或状态表（也可以使用其他表示方式）。

5.2.8.2 导出测试覆盖项（TD3）

在状态转移测试中，测试覆盖项可依据所选的测试完成准则和测试设计方法而变化。测试完成准则可包含但不限于以下内容：

——状态，导出的测试覆盖项能使状态模型中所有的状态都被"访问"到。

——单步转移(0-switch),导出的测试覆盖项应能覆盖状态模型中的有效单步转移。

——全转移,导出的测试覆盖项能够覆盖状态模型中的有效转移和无效转移(从状态模型中未指定有效转移的事件启动的状态转移)。

——多步转移(N-switch),导出的测试覆盖项应覆盖状态模型中 N+1 步转移的有效序列。

注:"2 步转移"覆盖是"多步转移"覆盖的一种特殊情况,要求实现成对的转移。

5.2.8.3 导出测试用例(TD4)

状态转移测试导出的测试用例应实现测试覆盖项。下面是导出测试用例的步骤:

a) 选择当前测试用例要包含的测试覆盖项;

b) 确定覆盖测试覆盖项的测试用例输入值;

c) 通过将输入应用到测试依据中,来确定测试用例的预期结果(预期结果可以使用输出和状态模型中描述的访问状态来定义);

d) 重复步骤 a)~c)直到达到要求的测试覆盖率水平。

5.2.9 场景测试

5.2.9.1 导出测试条件(TD2)

场景测试使用测试项与其他系统之间的交互序列模型(在此周境中,用户通常被认为是其他系统),以测试所涉及的测试项使用流程。测试条件应是一个交互序列(即一个场景)或所有交互序列(即所有场景)。

场景测试应该包括以下场景:

——"主"场景是测试项的预期典型动作序列,或无典型动作序列时所采取的一个任意选择;和

——"备选"场景表示测试项可选择的(非主)场景。

注 1:备选的场景包括非正常的使用、极端或者压力条件和异常。

注 2:场景测试通常在功能测试中用于产生"端到端测试",比如系统测试或者验收测试。

场景测试的一种常见形式称为用例测试,其采用了测试项的用例模型来描述测试项如何与一个或多个参与者交互,以测试相关测试项的交互序列(即场景)。

注 3:在用例测试中,用例模型用于描述参与者如何触发测试项的各种动作。参与者可以是用户或其他系统。

注 4:事务流测试也是一种场景测试。

5.2.9.2 导出测试覆盖项(TD3)

测试覆盖项应是主场景和备选场景(即测试覆盖项与测试条件相同)。因此,在该步骤中不需要做进一步操作。

5.2.9.3 导出测试用例(TD4)

场景测试导出的测试用例中,一个测试用例至少覆盖一个场景(测试覆盖项)。下面是导出测试用例的步骤:

a) 选择当前测试用例实现的测试覆盖项;

b) 确定测试用例覆盖的测试覆盖项的输入值;

c) 通过将输入应用到测试依据中,来确定测试用例的预期结果;

d) 重复步骤 a)~c),直到达到要求的测试覆盖率水平。

5.2.10 随机测试

5.2.10.1 导出测试条件（TD2）

随机测试使用测试项的输入域模型来定义所有可能输入值的集合。应选择用于生成随机输入值的输入分布。整个输入域应是随机测试的测试条件。

示例：输入分布包括正态分布、均匀分布和运行剖面。

5.2.10.2 导出测试覆盖项（TD3）

随机测试没有已知的测试覆盖项。

5.2.10.3 导出测试用例（TD4）

随机测试的测试用例应根据所选输入分布从测试项的输入域（或如果使用工具生成伪随机）中随机选择输入值而来。下面是导出测试用例的步骤：

a) 为测试输入选择一种输入分布；

b) 根据步骤 a)中的输入分布，生成测试输入的随机值；

c) 通过将输入应用到测试依据中，来确定测试用例的预期结果；

d) 重复步骤 b)、c)直到完成要求的测试。

注 1：要求的测试是指一些需要执行的测试、测试花费的时间或其他完成测试的测度。

注 2：步骤 b)通常是自动的。

5.3 基于结构的测试设计技术

5.3.1 语句测试

5.3.1.1 导出测试条件（TD2）

应导出测试项的源代码模型，并将语句标识为可执行或不可执行。每个执行语句应为一个测试条件。

注：可在步骤 TD4 期间采用自动化工具进行不可执行语句的识别。

5.3.1.2 导出测试覆盖项（TD3）

每个可执行语句应为测试覆盖项（即测试覆盖项和测试条件一致）。因此，在该步骤中不需要做进一步操作。

5.3.1.3 导出测试用例（TD4）

以下是导出测试用例的步骤：

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径；

b) 确定实现所识别的控制流子路径的测试输入；

c) 通过将相应的测试输入应用到测试依据，来确定执行控制流子路径的预期结果；

d) 重复步骤 a)～c)直到达到要求的测试覆盖率水平。

5.3.2 分支测试

5.3.2.1 导出测试条件（TD2）

应导出识别控制流分支的测试项控制流模型。控制流模型的每个分支是一个测试条件，分支包括：

——从控制流的任一节点到另一节点的条件转移；

——从控制流任一节点到另一节点控制的明确无条件转移；

——当一个测试项有不止一个入口点时，到测试项一个入口点的控制转移。

注1：分支测试要达到100%的分支覆盖率，要求测试控制流图中的所有的弧（链接或边），包括任何入口和出口之间不包含判定的顺序语句。

注2：分支测试可以要求同时测试条件分支和非条件分支，包括测试项的入口和出口，这取决于要求的测试覆盖水平。

注3：在分支测试中，功能和方法调用不能被识别为独立的测试条件。

5.3.2.2 导出测试覆盖项（TD3）

控制流模型中的每个分支都是一个测试覆盖项（即测试覆盖项和测试条件一样）。因此，在该步骤中不需要做进一步操作。

5.3.2.3 导出测试用例（TD4）

下面是导出测试用例的步骤：

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径；

b) 确定实现所识别的控制流子路径的测试输入；

c) 通过将相应的测试输入应用到测试依据中，来确定执行控制流子路径的预期结果；

d) 重复步骤a)～c)直到达到要求的测试覆盖率水平。

注1：如果测试项中没有分支，则仍需要一个单独的测试条件、测试覆盖项和测试用例。

注2：如果测试项有多个入口，则需要足够的测试用例覆盖每个入口。

5.3.3 判定测试

5.3.3.1 导出测试条件（TD2）

应导出识别判定的测试项控制流模型。判定是测试项中可以产生控制流模型两个或多个结果（因此出现子路径）的点。典型的判定用于简单的选择（例如，源代码中的if-then-else），决定何时退出循环（例如，源代码中的while循环），以及case(switch)语句（例如，源代码中的case-1-2-3 -...- N）。在判定测试中，控制流模型中的每个判定都应是测试条件。

5.3.3.2 导出测试覆盖项（TD3）

每个判定得到的判定结果应当作为测试覆盖项。

5.3.3.3 导出测试用例（TD4）

下面是导出测试用例的步骤：

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径；

b) 确定实现所识别的控制流子路径的测试输入；

c) 通过将相应的测试输入应用到测试依据中，来确定执行控制流子路径的预期结果；

d) 重复步骤a)～c)直到达到要求的测试覆盖率水平。

注：如果测试项没有判定，则仍需要一个单独的测试条件、测试覆盖项和测试用例。

5.3.4 分支条件测试

5.3.4.1 导出测试条件（TD2）

应导出识别判定和判定中条件的测试项控制流模型。判定是测试项中可以产生控制流模型两个或

多个结果(因此出现子路径)的点。典型的判定用于简单的选择(例如,源代码中的 if-then-else),决定何时退出循环(例如,源代码中的 while 循环),以及 case(switch)语句(例如,源代码中的 case-1-2-3-...-N,也称 switch 语句)。在分支条件中,每个判定都应是测试条件。

示例:在程序源代码中,判定语句"if A OR B AND C then"是一个测试条件,包含了由逻辑运算符关联的 3 个条件。

5.3.4.2 导出测试覆盖项(TD3)

在分支条件测试中,判定中所有条件的赋值(取值)都是测试覆盖项。每个判定的判定结果也作为测试覆盖项。

5.3.4.3 导出测试用例(TD4)

下面是导出测试用例的步骤:

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径;

b) 确定实现所识别的控制流子路径的测试输入;

c) 识别步骤 b)中的测试输入的子集,以覆盖判定中条件的布尔值和判定结果;

d) 通过将相应的测试输入应用到测试依据中,来确定执行控制流子路径的预期结果;

e) 重复步骤 a)~d)直到达到要求的测试覆盖率水平。

注:如果测试项中没有判定,则仍需要一个单独的测试条件、测试覆盖项和测试用例。

5.3.5 分支条件组合测试

5.3.5.1 导出测试条件(TD2)

应导出识别判定和条件的测试项控制流模型。在分支条件组合测试中,每个判定都应该是一个测试条件。

示例:在程序源代码中,判定语句"if A OR B AND C then"是一个测试条件,包含了由逻辑运算符关联的 3 个条件。

5.3.5.2 导出测试覆盖项(TD3)

每个判定中条件的布尔值的每个唯一可行组合应被识别为测试覆盖项。包括简单判定,即由两个单一布尔量组合形成一个判定结果。

5.3.5.3 导出测试用例(TD4)

下面是导出测试用例的步骤:

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径;

b) 确定实现所识别的控制流子路径的测试输入;

c) 识别步骤 b)中的测试输入子集,以覆盖判定中包含条件的布尔值选定组合;

d) 通过将相应的测试输入应用到测试依据中,来确定执行控制流子路径的预期结果;

e) 重复步骤 a)~d)直到达到要求的测试覆盖率水平。

注:如果测试项中没有判定,则仍需要一个单独的测试条件、测试覆盖项和测试用例。

5.3.6 修正条件判定覆盖测试

5.3.6.1 导出测试条件(TD2)

应导出标识判定和条件的测试项控制流模型。修正条件判定覆盖(MCDC)测试中,每个判定应为测试条件。

示例:在程序源代码中,判定语句"if A OR B AND C then"则包含了逻辑运算符关联的 3 个条件的测试条件。

5.3.6.2 导出测试覆盖项（TD3）

由单个布尔条件可以独立影响判定结果的判定条件，其布尔值的每个唯一可行组合作为测试覆盖项。通过仅改变一个条件而保持其他可能的条件不变，来表明条件是否独立影响一个判定结果。

注：这包括简单判定，由两个单一布尔量组合形成一个判定结果。

5.3.6.3 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 识别能到达一个或多个尚未被执行到测试覆盖项的控制流子路径；

b) 确定实现所识别的控制流子路径被执行的测试输入；

c) 确定步骤 b)中的测试输入子集，以覆盖判定中各个布尔值的选定组合，每个组合条件能够独立影响判定结果，允许单个布尔条件独立地影响判定结果；

d) 通过将相应的测试输入应用于测试依据来确定预期结果；

e) 重复步骤 a)～d)，直到达到所需的测试覆盖水平。

注：如果测试项中没有判定，那么仍然需要单个测试条件、测试覆盖项和测试用例。

5.3.7 数据流测试

5.3.7.1 导出测试条件（TD2）

在数据流测试中，应该导出测试项的模型，该模型识别测试项的控制流子路径，在该子路径中，给定变量的每个定义与该变量的后续使用相关，并且后续使用没有重新定义变量的值。

“定义”可能给变量赋了新的值(有时定义将变量保持与之前相同的值)。“使用”是变量出现，不是赋新的值；“使用”可以进一步划分为“p-use”(谓词使用)和“c-use”(计算使用)。谓词使用是指使用变量来确定判定条件(谓词)的结果，例如 while-loop，if-then-else 等。计算使用是指一个变量作为任何变量定义或输出的计算输入。

在数据流测试中，测试项中变量的每个定义-使用对都是测试条件。

有许多形式的数据流测试，它们都是基于相同的测试条件。本部分中定义的 5 种形式是：全定义测试、全谓词使用测试、全计算使用测试、全使用测试、全定义-使用路径测试。

5.3.7.2 全定义测试

5.3.7.2.1 导出测试覆盖项（TD3）

测试覆盖项是从每个变量定义到该定义的某种使用(谓词使用或计算使用)的控制流子路径。每个子路径称为“定义-使用”路径。“全定义”测试要求所有变量定义都覆盖从定义到其谓词使用或者计算使用的至少一个定义到任意类型使用的子路径(与特定变量有关)。

5.3.7.2.2 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 确定尚未被测试覆盖的定义；

b) 确定控制流子路径的测试输入，该控制流子路径是从将要执行的定义中获得；

c) 通过将相应的测试输入应用于测试依据，来确定执行控制流子路径的预期结果；

d) 重复步骤 a)～c)，直到达到所需的测试覆盖水平。

注：实际上，这些步骤可能需要自动化。

5.3.7.3 全计算使用测试

5.3.7.3.1 导出测试覆盖项（TD3）

测试覆盖项是从每个变量定义到该定义的每个计算使用的控制流子路径。“全计算使用”测试要求所有相关变量定义都覆盖从定义到其每个计算使用的至少一个自由定义子路径(与特定变量有关)。

5.3.7.3.2 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 确定从变量定义到该定义计算使用的控制流子路径(不包括中间定义)，该路径尚未被测试覆盖；
b) 确定将要执行的控制流子路径的测试输入；
c) 通过将相应的输入应用于测试依据中，来确定执行控制流子路径的预期结果；
d) 重复步骤 a)～c)，直到达到所需的测试覆盖水平。

5.3.7.4 全谓词使用测试

5.3.7.4.1 导出测试覆盖项（TD3）

测试覆盖项是从每个变量定义到该定义的每个谓词使用的控制流子路径。“全谓词使用”测试要求所有相关变量定义都覆盖从定义到其每个谓词使用的至少一个自由定义子路径(与特定变量有关)。

5.3.7.4.2 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 确定从变量定义到其谓词使用的控制流子路径(不包括中间定义)，该路径尚未被测试覆盖；
b) 确定将要执行的控制流子路径的测试输入；
c) 通过将相应的测试输入应用于测试依据中，来确定执行控制流子路径的预期结果；
d) 重复步骤 a)～c)，直到达到所需的测试覆盖水平。

5.3.7.5 全使用测试

5.3.7.5.1 导出测试覆盖项（TD3）

测试覆盖项是从变量定义到该定义每次使用(包括计算使用和谓词使用)的控制流子路径。“全使用”测试要求包括从每个变量定义到它的每个使用的所有子路径(不包括变量的中间定义)。

5.3.7.5.2 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 确定从变量定义到谓词使用或计算使用的控制流子路径，该路径尚未被测试覆盖；
b) 确定将要执行的控制流子路径的测试输入；
c) 通过将相应的测试输入应用于测试依据中，来确定执行控制流子路径的预期结果；
d) 重复步骤 a)～c)，直到达到所需的测试覆盖层次。

5.3.7.6 全定义-使用路径测试

5.3.7.6.1 导出测试覆盖项（TD3）

测试覆盖项是从每个变量定义到该定义的每次使用(包括就计算使用和谓词使用)的控制流子路

径。“全定义-使用路径”测试要求包括从每个变量定义到它的每个使用的所有子路径(不包括变量的中间定义)。

注：全定义-使用路径测试要求对从变量定义到使用的所有无循环子路径进行测试，以尝试实现100%的测试项覆盖率。全定义-使用路径测试不同于全使用测试，后者只需要从每个变量定义到其使用的一条路径进行测试，以尝试实现100%的测试项覆盖率。

5.3.7.6.2 导出测试用例（TD4）

导出测试用例时应遵循以下步骤：

a) 确定从变量定义到谓词使用或计算使用的控制流子路径，该路径尚未被测试覆盖；
b) 确定将要执行的控制流子路径的测试输入；
c) 通过将测试输入应用于测试依据中，来确定执行控制流子路径的预期结果；
d) 重复步骤a)～c)，直到达到所需的测试覆盖水平。

5.4 基于经验的测试设计技术

5.4.1 错误猜测法

5.4.1.1 导出测试条件（TD2）

错误猜测法涉及设计测试项中可能存在缺陷类型的检查清单，如果这些缺陷存在于测试项中，允许测试人员识别可能导致失败的测试项输入。每种缺陷类型都应作为测试条件。

注：缺陷类型检查清单可以通过多种方式得到，例如已知错误的分类，事件管理系统中包含的信息，测试人员的知识、经验和/或对测试项和/或类似测试项的理解或来自其他利益相关者的知识(例如系统用户或者程序员)的知识。

5.4.1.2 导出测试覆盖项（TD3）

错误猜测法没有公认的测试覆盖项。

5.4.1.3 导出测试用例(TD4)

用于错误猜测法的测试用例通常通过从要覆盖的缺陷类型检查清单中选择缺陷类型，并导出可以在测试项中检测缺陷类型(如果存在的话)的测试用例，导出测试用例时，应使用以下步骤：

a) 选择当前测试用例覆盖的缺陷类型；
b) 确定可能导致与所选缺陷类型相对应故障的输入值；
c) 通过将输入应用于测试依据来确定测试用例的预期结果；
d) 重复步骤a)～c)，直到完成所需的测试。

6 测试覆盖率测量

6.1 概述

本部分定义的覆盖率测量基于不同的测试设计技术以达到不同覆盖度。覆盖率的范围从0%到100%不等。在计算覆盖率的过程中，一些测试覆盖项可能是无效的。如果证明了一个测试覆盖项是不能执行的或者不可能被测试用例覆盖的，则应该将其定义为不可行。覆盖率计算可以计入不可执行项，也可以不计入不可执行项。该选择通常会被记录在测试计划中(定义在GB/T 38634.3中)。如果有一个测试覆盖项不进行计算，则通常会在测试报告中记录其不可执行的原因。在计算覆盖率的过程中，如果测试项中存在没有给定类型的测试覆盖项，则定义该测试项的覆盖率为100%，因为对于测试项，该

测试覆盖项是不适用的。

在计算任何技术的覆盖率时，应使用式(1)：

$$覆盖率=(\frac{N}{T}\times 100)\% \quad \cdots\cdots(1)$$

式中：

覆盖率——特定测试设计技术所达到的覆盖率；

N ——执行的测试用例所涵盖的测试覆盖项的数量；

T ——由测试设计技术确定的测试覆盖项总数量。

每种技术的覆盖率、N 和 T 的具体值将在下面的条款中定义。以下条款中提供的覆盖率测量旨在与本标准中的测试设计技术一起使用。它们并非旨在成为详尽的清单，有些组织可能会使用本条款中未提及的其他测试测量。

示例：评估测试完整性可能需要的其他测量包括测量测试过程中所覆盖需求的总体比例。

6.2 基于规格说明的测试设计技术的测试测量

6.2.1 等价类划分覆盖率

等价类划分的覆盖率应使用以下定义计算：

——覆盖率是等价类的覆盖率；

——N 是执行的测试用例所覆盖的分区数量；

——T 是识别的分区总数量。

6.2.2 分类树方法覆盖率

分类树方法的覆盖率根据组合分类导出测试覆盖项的方式不同，分为以下定义类别：

对于最小化组合覆盖率，应使用以下定义：

——覆盖率是最小组合分类树方法的覆盖率；

——N 是执行的测试用例覆盖的类数量；

——T 是类的总数量。

对于最大化组合覆盖率，应使用以下定义：

——覆盖率是最大组合分类树方法的覆盖率；

——N 是执行的测试用例覆盖的类组合数量；

——T 是类组合的总数量。

6.2.3 边界值分析覆盖率

边界值分析的覆盖率应使用以下定义计算：

——覆盖率是边界值的覆盖率；

——N 是执行的测试用例覆盖的不同边界值的数量；

——T 是边界值的总数量。

应记录使用二值还是三值边界测试的决策。

6.2.4 语法测试覆盖率

目前还没有业界认可的方法用于计算语法测试的覆盖率。

注：对于语法测试来说，以百分比计算覆盖率是不可能的，因为语法测试可能存在大量的选项和无限的变异。

6.2.5 组合测试设计技术覆盖率

6.2.5.1 完全组合测试覆盖率

完全组合测试的覆盖率应使用以下定义计算：

——覆盖率是完全组合的覆盖率；

——N 是执行测试用例覆盖的键值对的唯一组合的数量；

——T 是唯一的键值对的组合的总数(即：每个测试项参数的键值对数的乘积)。

注：有关键值对的定义，见 4.22。

6.2.5.2 成对测试覆盖率

成对测试的覆盖率应使用以下定义计算：

——覆盖率是成对的覆盖率；

——N 是执行的测试用例所覆盖的键值对的唯一对的数量；

——T 是键值对的唯一对的总数量。

6.2.5.3 单一选择测试覆盖率

单一选择测试的覆盖率应使用以下定义计算：

——覆盖率是单一选择的覆盖率；

——N 是执行的测试用例所覆盖的键值对的数量；

——T 是键值对的总数量。

6.2.5.4 基本选择测试覆盖率

基本选择测试的覆盖率应使用以下定义计算：

——覆盖率是基本选择的覆盖率；

——N 是执行的测试用例所覆盖的基本选择组合的数量(即除了一个测试项参数设置为基值外，其他的测试项参数设置为有效值)加上 1(即当所有测试项参数设置为基值时)；

——T 是基本选择组合的总数量(即除了一个测试项参数设置为基值外，其他的测试项参数设置为有效值)加上 1(即当所有测试项参数设置为基值时)。

6.2.6 判定表测试覆盖率

判定表测试的覆盖率应使用以下定义计算：

——覆盖率是判定表的覆盖率；

——N 是执行的测试用例所覆盖的可行判定规则的数量；

——T 是可行判定规则的总数量。

6.2.7 因果图覆盖率

因果图的覆盖率应使用以下定义计算：

——覆盖率是因果图的覆盖率；

——N 是执行的测试用例所覆盖的判定规则的数量；

——T 是可行判定规则的总数量。

6.2.8 状态转移测试覆盖率

全状态转移测试的覆盖率应使用以下定义计算：

——覆盖率是所有状态的覆盖率；
——N 是执行的测试用例所覆盖的状态数量；
——T 是状态的总数量。

单步转移(0-switch 覆盖)的覆盖率应使用以下定义计算：
——覆盖率是单步转移的覆盖率；
——N 是执行的测试用例所覆盖的单步有效转移的数量；
——T 是单步有效转移的总数量。

全转移的覆盖率应使用以下定义计算：
——覆盖率是全转移的覆盖率；
——N 是执行的测试用例试图覆盖的有效和无效转移数量；
——T 是有效事件引起的已知状态之间的有效和无效转移总数量。

多步转移(N-switch 覆盖)的覆盖率应使用以下定义计算：
——覆盖率是多步转移的覆盖率；
——N 是执行的测试用例所覆盖的 $N+1$ 步有效转移的数量；
——T 是 $N+1$ 有效转移序列的总数量。

6.2.9 场景测试覆盖率

场景测试(包括用例测试)的覆盖率应使用以下定义计算：
——覆盖率是场景的覆盖率；
——N 是执行的测试用例所覆盖的场景数量；
——T 是场景的总数量。

6.2.10 随机测试覆盖率

目前还没有业界认可的方法用于计算随机测试的覆盖率。

6.3 基于结构的测试设计技术的测试测量

6.3.1 语句测试覆盖率

语句测试的覆盖率应使用以下定义计算：
——覆盖率是语句的覆盖率；
——N 是执行的测试用例所覆盖的可执行语句的数量；
——T 是可执行语句的总数量。

6.3.2 分支测试覆盖率

分支测试的覆盖率应使用以下定义计算：
——覆盖率是分支的覆盖率；
——N 是执行的测试用例覆盖的分支数量；
——T 是分支的总数量。

注：如果测试项中没有分支，则只需要一次测试就可达到 100%的分支覆盖率。

6.3.3 判定测试覆盖率

判定测试的覆盖率应使用以下定义计算：
——覆盖率是判定的覆盖率；

——N 是执行的测试用例所覆盖的判定结果的数量；

——T 是判定结果的总数量。

注：对于测试项中没有判定的情况，则只需要一次测试就可达到100%的判定覆盖率。

6.3.4 分支条件测试覆盖率

分支条件测试的覆盖率应使用以下定义计算：

——覆盖率是分支条件的覆盖率；

——N 是执行的测试用例覆盖的判定中条件的取值数量加上判定结果的数量；

——T 是判定中条件的布尔值数量加上判定结果的总数量。

注：如果测试项中没有判定，则只需要一次测试就可达到100%的分支条件覆盖率。

6.3.5 分支条件组合测试覆盖率

分支条件组合测试的覆盖率应使用以下定义计算：

——覆盖率是分支条件组合的覆盖率；

——N 是执行的测试用例覆盖的判定条件的布尔值组合数量；

——T 是判定中条件的布尔值唯一组合的总数量。

注：如果测试项中没有判定，则只需要一次测试就可达到100%的分支条件组合覆盖率。

6.3.6 修正条件判定(MCDC)测试覆盖率

修正条件判定测试的覆盖率应使用以下定义计算：

——覆盖率是修正条件判定的覆盖率；

——N 是执行的测试用例覆盖的判定中条件布尔值的唯一可行组合的数量，其允许单个布尔条件独立地影响判定结果；

——T 是判定条件单个布尔值的唯一可行组合的总数量，其允许单个布尔条件独立地影响判定结果。

注：如果测试项中没有判定，则只需要一次测试就可达到100%的修正条件判定覆盖率。

6.3.7 数据流测试覆盖率

6.3.7.1 全定义测试覆盖率

全定义测试的覆盖率应使用以下定义计算：

——覆盖率是全定义的覆盖率；

——N 是执行的测试用例所覆盖的与数据定义-使用对相关的定义数量；

——T 是不同变量定义的数据定义-使用对的总数量。

6.3.7.2 全计算使用测试覆盖率

全计算使用测试的覆盖率应使用以下定义计算：

——覆盖率是全计算使用的覆盖率；

——N 是执行的测试用例覆盖的定义-计算-使用对数量；

——T 是定义-计算-使用对的总数量。

6.3.7.3 全谓词使用测试覆盖率

全谓词使用测试的覆盖率应使用以下定义计算：

——覆盖率是全谓词使用的覆盖率；

——N 是执行的测试用例覆盖的唯一定义-谓词-使用对数量；
——T 是定义-谓词-使用对的总数量。

6.3.7.4 全使用测试覆盖率

全使用测试的覆盖率应使用以下定义计算：
——覆盖率是全使用的覆盖率；
——N 是执行的测试用例覆盖的定义-使用对数量；
——T 是定义-使用对的总数量，包括计算使用和谓词使用的定义。

6.3.7.5 全定义使用路径测试覆盖率

全定义使用路径测试的覆盖率应使用以下定义计算：
——覆盖率是全定义使用路径的覆盖率；
——N 是执行的测试用例覆盖的子路径的数量，该子路径是每个变量定义到该变量可达的使用路径（不包括变量的中间定义）；
——T 是从每个变量定义到该变量谓词-使用和计算-使用的子路径总数量。

6.4 基于经验的测试设计技术的测试测量

6.4.1 错误猜测覆盖率

目前还没有业界认可的方法用于计算错误猜测的覆盖率。

附 录 A
（资料性附录）
测试质量特性

A.1 质量特性

软件测试需要验证测试项是否符合质量标准的要求。本附录列举了一些示例，用于说明GB/T 25000.10—2016中定义的质量特性映射到本部分定义的测试设计技术。本部分的测试设计技术可以用于测试GB/T 25000.10—2016中的特性，也可以测试其他质量特性。

GB/T 25000.10—2016定义了一个产品质量模型，如图A.1所示，将系统/软件产品质量特性分为八类：功能性、性能效率、兼容性、易用性、可靠性、信息安全性、维护性和可移植性。每个特性由相关的子特性集组成。有些情况下可能存在监管要求（例如政策或法律），要求系统符合一些特定的质量特性。各项测试设计技术和测试类型可以用于测试各种特性（见A.3和A.4）。

A.2说明了图A.1中可以用于测试质量特性的测试类型。每个质量特性到测试的适用类型的映射关系在A.3中说明。A.4说明了质量特性和本部分定义的基于规格说明和基于结构的测试设计技术之间的关系。

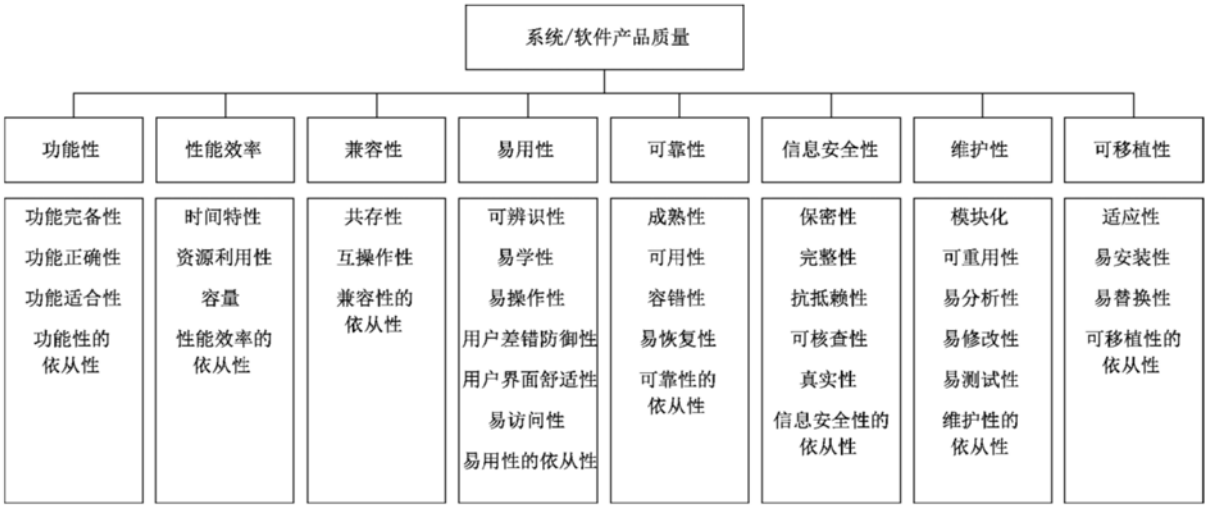


图 A.1 GB/T 25000.10 产品质量模型

注：ISO/IEC 25030可以识别和记录适用于测试项的软件测试需求，可以识别GB/T 25000.10—2016中的质量特性以及适用于测试每个质量需求的对应测试类型。

A.2 测试的质量相关类型

A.2.1 易访问性测试

易访问性测试的目的是确定用户能否根据测试项广泛的特性和功能（GB/T 25000.10—2016）进行操作，包括一些特定的易访问性需求（例如存在年龄、视力障碍或听力障碍）。易访问性测试使用的测试项模型指定了易访问性需求以及测试项应符合的易访问性设计标准。易访问性需求通过关注用户的能

力以实现易访问性目标，特别是一些特定的易访问性需求。

A.2.2 备份及恢复测试

备份及恢复测试的目的是确定系统在出现失效时，是否能够从备份中恢复测试项失效前的状态。备份及恢复测试使用的测试项模型指定了备份和恢复的要求，指定了备份测试项某个时间点应对应的操作状态的需求，包括数据、配置和环境要求，以及从备份恢复测试项的状态。备份及恢复测试关注于测试项备份内容的正确性以及针对失效前状态的测试项状态恢复的正确性。备份及恢复测试还可以用于验证测试项的备份和恢复步骤是否达到指定的要求。该类型的测试可以作为灾难恢复测试的一部分（见 A.2.5）。

A.2.3 兼容性测试

兼容性测试的目的是确定在共享的环境中，测试项能否和其他相关（但不一定交互）和无关的软件共同运行（即共存）。兼容性测试可以测试同一个测试项的多个副本或者共享同一个环境的多个测试项。兼容性测试对测试项的要求通常包括一个或者多个子要求，如下所示：

——安装顺序。指显式的安装顺序（否则假设所有可能的安装顺序是有效的），每个测试项能够按顺序正确执行要求的功能。

——实例化顺序。指显式的实例化顺序（否则假设所有可能的实例化顺序是有效的），在运行时让每个测试项按顺序正确执行要求的功能。

——并发测试。指两个或者多个测试项在同一个环境中运行的能力，分别执行要求的功能（但不一定交互）。

——环境约束。指环境的特点（例如存储、处理器、架构、平台或配置）可能会影响测试项正确执行所需的功能。

A.2.4 转换测试

转换测试的目的是确定数据或软件在修改形式后（例如将一个程序的编程语言转换成另一种编程语言，或将纯数据文件或数据库转换为另一种格式）是否能继续提供要求的功能。一种常用的转换测试子类型是数据迁移测试。转换测试使用的测试项模型指定了转换要求，包括经过转换后仍保持不变的要求和转换后新的、修改的或者废弃的要求以及测试项应符合的任意转换设计标准。

A.2.5 灾难恢复测试

灾难恢复测试的目的是确定系统出现失效，在测试项的操作转换成其他操作的情况下，进行失效恢复后，操作是否能转换回来。灾难恢复测试使用的测试项模型（典型为灾难恢复计划）中指定了灾难恢复要求，包括测试项应符合的任意灾难恢复设计标准。灾难恢复测试的测试项可能是整个操作系统，包括相关的设备、人员和程序。灾难恢复测试覆盖的因素可以是重新定位操作人员执行的程序、数据、软件、人员、办公室和其他设备，或者是远程恢复曾经备份过的数据。

A.2.6 功能性测试

功能性测试的目的是确定测试项的功能是符合要求。例如，识别一个功能是否按照其指定的要求实现。可使用第 5 章中基于规格说明和基于结构的测试设计技术来实现功能性测试。

A.2.7 安装测试

安装测试的目的是确定在所有指定的环境中，测试项是否能按照要求安装、卸载/移除和升级。安装测试使用的测试项模型中指定了安装、卸载/移除和升级过程要求（如安装手册或指南中所说明的），包括进行安装、卸载/移除或升级的人员，以及测试项安装、卸载/移除或升级的目标平台。

A.2.8 互操作性测试

互操作性测试的目的是确定测试项是否能在相同或者不同的环境中和其他的测试项或系统进行正确交互，包括测试项是否能有效使用从其他系统接收的信息。互操作性测试使用的测试项模型指定了互操作性要求，包括测试项应符合的互操作性设计标准以及评估一个在环境中运行的测试项是否能和另一个环境中的测试项或系统进行准确交互。

A.2.9 本地化测试

本地化测试的目的是确定使用测试项的地理区域是否能理解测试项。本地化测试包括（但不限于）各个使用测试项的国家和区域是否能分析理解测试项的用户接口和支持文档。

A.2.10 维护性测试

维护性测试的目的是确定测试项是否能进行可接受范围内的维护工作。维护性测试使用的测试项模型，指定了为实现变化所需进行的工作，可以分为下面几类：

——纠正性维护（即纠正问题）；

——完善性维护（即增强功能）；

——适应性维护（即适应环境的变化）；

——预防性维护（即减少未来维护成本的措施）。

维护性可以直接通过静态分析来测量。

A.2.11 性能测试

性能测试的目的是确定测试项在各种不同情况的“负载”下是否能按要求执行。包括性能、负载、压力、持久性、容量、可扩展性和存储管理测试，每种测试使用的测试项模型指定了性能要求，包括测试项应符合的任何所需的性能设计标准。例如，根据每秒的交互量、响应时间、往返时间和资源利用率来评估测试项的性能。测试项在“正常”条件下的“典型”负载可以在测试项的操作手册中定义。

多项评估测试项性能的技术如下：

——性能测试的目的是评估测试项在“典型”负载下的性能。

——负载测试的目的是评估测试项在不同负载下（通常是在谷值、平值和峰值之间的预期条件下）的行为（比如性能和可靠性）。

——压力测试的目的是评估测试项在极限状态下（超出预期峰值或者可用资源少于最低要求时）的性能。

——持久性测试（浸泡测试）的目的是评估测试项在一段连续的时间内维持要求的负载量的性能。

——容积测试的目的是评估测试项在处理指定数量的数据时的性能。例如，评估测试项在其数据库接近最大容量时的性能。

——可扩展测试的目的是评估测试项在未来支持的条件下的执行情况。例如，评估支持未来预期负载所需的额外资源（例如存储、硬盘容量、网络带宽）的水平。

——存储管理测试的目的是根据测试中的内存使用（例如硬盘存储、RAM 和 ROM）数量（一般是

最大值)、存储类型(例如动态或静态)或存储泄露的程度来评估测试项的执行情况。在特定的操作条件下,通常需要指定存储要求(例如,交互负载下特定操作期间的峰值存储要求)。

A.2.12 可移植性测试

可移植性测试的目的是确定测试项可以有效地从一个硬件、软件、其他操作系统或实用环境转换到其他环境的难易程度。可移植性测试使用的测试项模型指定了可移植性需求,包括测试项应符合的任意可移植性设计标准。可移植性要求关注的是测试项从一个环境移植到另一个环境和改变现有环境的配置到其他需要的配置。例如,评估测试项是否能在不同的浏览器上运行。

A.2.13 规程测试

规程测试的目的是确定规程指令是否符合用户需求和使用目的。规程测试使用测试项的规程需求模型作为完整的交付单元。规程需求定义了任意规程文件的要求,并写成规程指令的形式。这些规程指令通常使用下面文档中的其中一种形式来表示:

——用户指南;

——使用说明书;

——用户参考手册。

上述信息通常会定义用户的意图:

——设置正常使用的测试项;

——在正常条件下执行测试项;

——成为系统(教程文件)的管理用户;

——当出现故障时排除测试项的故障;

——重新配置测试项。

A.2.14 可靠性测试

可靠性测试的目的是评估测试项在特定时期的状态条件下执行所需功能的能力,包括评估出现失效的频率。可靠性测试使用的测试项模型指定了其要求的可靠性程度(例如平均失效时间、平均失效间隔时间)。该模型应该包含测试项的操作手册或生成操作手册方法以及失效的定义。

A.2.15 信息安全性测试

信息安全性测试的目的是评估测试项和其相关数据的受保护程度,是否能够在授权相关人员或系统权限的同时,保证非授权人员或系统不能使用、读取和修改数据。信息安全性测试使用的测试项模型指定了信息安全性要求,包括测试项应符合的任意信息安全性设计标准。信息安全性要求关注的是保护数据的能力以及测试项的功能不被非授权用户恶意使用。例如,评估测试项是否阻止了非授权用户访问数据,是否阻止了其他用户类型使用只能被特定的用户类型使用的测试项的某些功能。多项评估测试项信息安全的技术通常包括以下几种:

——渗透测试是指测试人员模仿非授权用户试图访问测试项(包括其功能和/或私有数据);

——隐私测试包括尝试访问隐私数据和验证用户访问隐私数据时留下的审计跟踪;

——信息安全审计是一种静态的测试,测试人员检查、审查、走查测试项的需求和代码以确定是否存在任何的信息安全漏洞;

——漏洞扫描包括使用自动化工具扫描测试项是否有特定已知的漏洞。

A.2.16 易用性测试

易用性测试的目的是评估用户能否在指定周境下测试项能达到有效性、效率和满意度的指定目标。

易用性测试使用的测试项模型指定了其易用性要求，包括测试项应符合的任意易用性设计标准。易用性需求指定了测试项的易用性目标。易用性目标应是基于测试项的目标(原因是测试项可以给组织或者个人带来的不同之处，用以帮助实现易用性相关的目标并完成相关的任务)，和测试项使用的周境(谁使用测试项、将使用什么样的环境以及用户特性和用户任务)。易用性目标定义为特定用户在一个或多个指定的周境中达到指定的有效性、效率和满意度目标。

注：ISO/IEC 9241 系列标准定义了人机交互要求的标准。

A.3 质量特性到测试类型的映射

表 A.1 说明了图 A.1 的每一个质量特性与 A.2 中测试类型的映射关系。

表 A.1 GB/T 25000.10 产品质量特性到测试类型的映射

基于质量特性的测试类型	质量特性	子特性
易访问性测试	易用性	易访问性
备份及恢复测试	可靠性	成熟性
		容错性
		易恢复性
兼容性测试	兼容性	共存性
转换测试	功能性	功能完备性
		功能正确性
		功能适合性
灾难恢复测试	可靠性	成熟性
		容错性
		易恢复性
功能性测试	功能性	功能完备性
		功能正确性
		功能适合性
安装测试	可移植性	易安装性
互操作性测试	兼容性	互操作性
本地化测试	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	可辨识性
		易学性
		易操作性
		用户差错防御性
		用户界面舒适性
		易访问性
	可移植性	适应性

表 A.1（续）

基于质量特性的测试类型	质量特性	子特性
维护性测试	维护性	模块化
		可重用性
		易分析性
		易修改性
		易测试性
性能测试	性能效率	时间特性
		资源利用性
		容量
可移植性测试	可移植性	适应性
		易安装性
		易替换性
规程测试	无	无
可靠性测试	可靠性	成熟性
		可用性
		容错性
		易恢复性
信息安全性测试	信息安全性	保密性
		完整性
		抗抵赖性
		可核查性
		真实性
易用性测试	易用性	可辨识性
		易学性
		易操作性
		用户差错防御性
		用户界面舒适性
		易访问性

A.4 质量特性到测试设计技术的映射

本部分中说明的测试设计技术可以用于测试图 A.1 的质量特性。表 A.2 提供了一个示例，说明两者之间的映射关系。

表 A.2　测试设计技术到 GB/T 25000.10 产品质量特性的映射

测试设计技术	GB/T 25000.10 质量特性	GB/T 25000.10 子特性
基于规格说明的测试设计技术		
边界值分析	功能性	功能完备性
		功能正确性
		功能适合性
	性能效率	时间特性
		容量
	易用性	用户差错防御性
	可靠性	容错性
	信息安全性	保密性
		完整性
因果图	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	用户差错防御性
	兼容性	共存性
分类树方法	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	用户差错防御性
组合测试设计技术	功能性	功能完备性
		功能正确性
		功能适合性
	兼容性	共存性
	性能效率	时间特性
	易用性	用户差错防御性
决策表测试	功能性	功能完备性
		功能正确性
		功能适合性
	兼容性	共存性
	易用性	用户差错防御性
等价类划分	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	用户差错防御性
	可靠性	可用性

表 A.2（续）

测试设计技术	GB/T 25000.10 质量特性	GB/T 25000.10 子特性
等价类划分	信息安全性	保密性
		完整性
		抗抵赖性
		可核查性
		真实性
随机测试	功能性	功能完备性
		功能正确性
		功能适合性
	性能效率	时间特性
		资源利用性
		容量
	可靠性	成熟性
		可用性
		容错性
		易恢复性
	信息安全性	保密性
		完整性
		抗抵赖性
		可核查性
		真实性
场景测试	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	易学性
		易操作性
		用户差错防御性
		用户界面舒适性
		易访问性
		可辨识性
状态转移测试	功能性	功能完备性
		功能正确性
		功能适合性
	可靠性	成熟性
		可用性
		容错性
		易恢复性

表 A.2（续）

测试设计技术	GB/T 25000.10 质量特性	GB/T 25000.10 子特性
语法测试	功能性	功能完备性
		功能正确性
		功能适合性
用例测试	功能性	功能完备性
		功能正确性
		功能适合性
	易用性	易学性
		易操作性
		用户差错防御性
		用户界面舒适性
		易访问性
基于结构的测试设计技术		
分支条件组合测试	功能性	功能完备性
		功能正确性
		功能适合性
分支条件测试	功能性	功能完备性
		功能正确性
		功能适合性
分支测试	功能性	功能完备性
		功能正确性
		功能适合性
数据流测试	功能性	功能完备性
		功能正确性
		功能适合性
判定测试	功能性	功能完备性
		功能正确性
		功能适合性
修正条件判定覆盖(MCDC)测试	功能性	功能完备性
		功能正确性
		功能适合性
语句测试	功能性	功能完备性
		功能正确性
		功能适合性

表 A.2（续）

测试设计技术	GB/T 25000.10 质量特性	GB/T 25000.10 子特性
基于经验的测试设计技术		
错误猜测	功能性	功能完备性
		功能正确性
		功能适合性
	性能效率	时间特性
		资源利用性
		容量
	易用性	易学性
		易操作性
		用户差错防御性
	可靠性	容错性

附 录 B
（资料性附录）
基于规格说明的测试设计技术的应用指南和示例

B.1 基于规格说明的测试设计技术的指南和示例

本附录通过展示各个基于规格说明的测试设计技术对特定问题的应用示例，以对 5.2 和 6.2 的要求提供指南。所有示例遵循 GB/T 38634.2 定义的测试设计和实现的过程。尽管每个示例都是在一个特定的基于规格说明的测试周境下实施，如 5.1 所述，但在实践中，大部分在本部分中已定义的技术可以交互使用（例如边界值分析可以通过用户接口或者程序源代码中变量的边界来测试一个程序的输入）。

B.2 基于规格说明的测试设计技术示例

B.2.1 等价类划分

B.2.1.1 介绍

等价类划分的目标是根据已选定等价类划分覆盖率，导出一组测试用例覆盖测试项的输入和输出分区。等价类划分的前提是可以根据测试项的测试依据，将测试项的输入和输出划分成测试项等价处理的分区。因此，测试等价类任何一个值的结果相当于测试该等价类其他值的结果。

B.2.1.2 规格说明

考虑一个测试项——generate_grading，测试依据如下：

组件接受考试成绩（不高于 75 分）和课程作业（c/w）成绩（不高于 25 分）作为输入，并输出一个“A”到“D”的课程等级。等级是通过计算考试和课程作业成绩的总和得到的，具体如下：

大于或等于 70 ——‘A’
大于或等于 50，小于 70 ——‘B’
大于或等于 30，小于 50 ——‘C’
小于 30 ——‘D’

在检测到无效输入（例如分数超出了预期的范围）时给出一条故障信息（‘FM’）表示。所有的输入都用整数。

B.2.1.3 步骤 1：识别特征集（TD1）

因为测试基准中只有一个测试项，只需要定义一个特征集：

FS1：generate_grading 功能

B.2.1.4 步骤 2：导出测试条件（TD2）

在等价类划分中，等价类就是测试条件（TCOND）。根据测试项的输入和输出实现等价类划分。同时考虑有效和无效的输入和输出。

两个输入的划分初步确定。有效的划分如下：

TCOND1：0＜＝考试成绩＜＝75 （FS1）

TCOND2：0<＝课程作业成绩<＝25　　（FS1）

基于输入的最显著的无效等价类如下：

TCOND3：考试成绩<0　　（FS1）

TCOND4：考试成绩>75　　（FS1）

TCOND5：课程作业成绩<0　　（FS1）

TCOND6：课程作业成绩>25　　（FS1）

分类值的范围可以用图表示，根据考试成绩得到图 B.1。

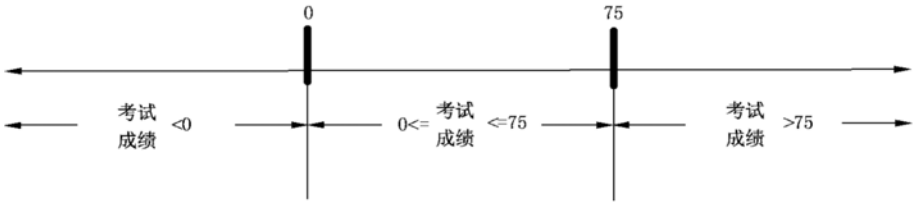


图 B.1　输入“考试成绩”

根据课程作业成绩得到图 B.2。

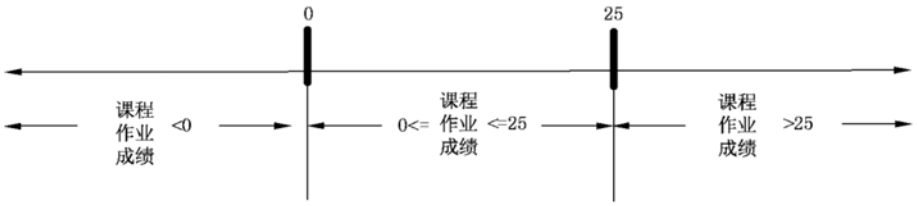


图 B.2　输入“课程作业成绩”

不明显的无效输入类可能包括任意其他的输入类型，例如，非整数输入或者非数字输入。因此可以导出下面的无效输入等价类：

TCOND7：考试成绩＝有小数部分的实数　　（FS1）

TCOND8：考试成绩＝字母　　（FS1）

TCOND9：考试成绩＝特殊字符　　（FS1）

TCOND10：课程作业成绩＝有小数部分的实数　　（FS1）

TCOND11：课程作业成绩＝字母　　（FS1）

TCOND12：课程作业成绩＝特殊字符　　（FS1）

下面是有效的输出分类：

TCOND13：“A”的条件　　70<＝总成绩<＝100　　（FS1）

TCOND14：“B”的条件　　50<＝总成绩<70　　（FS1）

TCOND15：“C”的条件　　30<＝总成绩<50　　（FS1）

TCOND16：“D”的条件　　0<＝总成绩<30　　（FS1）

TCOND17：“出错信息”(FM)的条件　　总成绩>100　　（FS1）

TCOND18：“出错信息”(FM)的条件　　总成绩<0　　（FS1）

TCOND19：“出错信息”(FM)的条件　　非整数输入　　（FS1）

其中总成绩＝考试成绩＋课程作业成绩。注意“出错信息”是一个有效的输出，因为是一个具体的输出。

总成绩的等价类和边界如图 B.3。

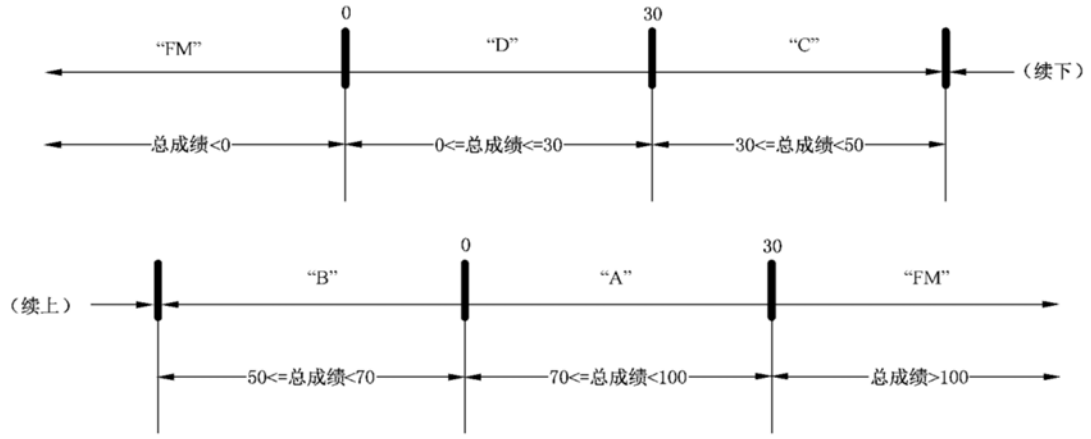


图 B.3　总成绩等价类划分

无效输出是除了五个指定的类以外的任意输出。识别未指定的输出是有难度的。但是，应考虑无效输出，如果可以找出一个无效输出，就代表找到了测试项或者测试基准的一个缺陷，或者两者共同的缺陷。在本例中定义了三个没有指定的输出，将在下面列出。等价类划分在这方面是主观的，因此，每个测试者可能会产生他们认为有可能发生的不同等价类(在 5.2.1.1 的注 2 中说明)。

TCOND20：输出＝“E”，0＜总成绩＜15 被错分成无效的不存在的“E”等级　　(FS1)

TCOND21：输出＝“A＋”，90＜总成绩＜100 被错分成无效的不存在的“A＋”等级　　(FS1)

TCOND22：输出＝“null”，生成“null”输出　　(FS1)

B.2.1.5　步骤 3：导出测试覆盖项(TD3)

在等价类划分中，测试覆盖项是在上面步骤中导出的等价类(即本技术中测试条件和测试覆盖项一致)。测试覆盖项如下所示：

TCOVER1：0＜＝考试成绩＜＝75　　(TCOND1)

TCOVER2：0＜＝课程作业成绩＜＝25　　(TCOND2)

TCOVER3：考试成绩＜0　　(TCOND3)

TCOVER4：考试成绩＞75　　(TCOND4)

TCOVER5：课程作业成绩＜0　　(TCOND5)

TCOVER6：课程作业成绩＞25　　(TCOND6)

TCOVER7：考试成绩＝有小数部分的实数　　(TCOND7)

TCOVER8：考试成绩＝字母　　(TCOND8)

TCOVER9：考试成绩＝特殊字符　　(TCOND9)

TCOVER10：课程作业成绩＝有小数部分的实数　　(TCOND10)

TCOVER11：课程作业成绩＝字母　　(TCOND11)

TCOVER12：课程作业成绩＝特殊字符　　(TCOND12)

TCOVER13：“A”的条件　　70＜＝总成绩＜＝100　　(TCOND13)

TCOVER14：“B”的条件　　50＜＝总成绩＜70　　(TCOND14)

TCOVER15：“C”的条件　　30＜＝总成绩＜50　　(TCOND15)

TCOVER16：“D”的条件　　0＜＝总成绩＜30　　(TCOND16)

TCOVER17：“出错信息”(FM)的条件　　总成绩＞100　　(TCOND17)

TCOVER18：“出错信息”(FM)的条件　　总成绩＜0　　(TCOND18)

TCOVER19：“默认值”(FM)的条件　　非整数输入　　(TCOND19)

TCOVER20：输出＝“E” （TCOND20）
TCOVER21：输出＝“A＋” （TCOND21）
TCOVER22：输出＝“null” （TCOND22）

B.2.1.6 步骤 4：导出测试用例（TD4）

B.2.1.6.1 选项

已经识别了等价类和需要测试的测试覆盖项，可以尝试导出测试用例来覆盖每个测试覆盖项。设计测试用例的常用两种方法是一对一和最小等价类划分（其他选择测试用例满足的测试覆盖项组合的方法在 4.2.5 中说明）。基于第一种方法一对一，为每一个识别的类导出一个测试用例（见 B.2.1.6.2 选项 4a），同时，第二种方法导出覆盖所有识别的类的最小测试集（见 B.2.1.6.3 选项 4b）。所有 generate_grading 的测试用例的前置条件是相同的：该应用已经为输入考试成绩和课程作业成绩做好准备。

B.2.1.6.2 选项 4a：导出一对一等价类划分的测试用例（TD4）

首先说明一对一的方法是，可以清晰地看到等价类和测试用例之间的关系。对于每个测试用例，只对作为目标的单一选择测试覆盖项进行明确说明。22 个测试覆盖项对应 22 个测试用例。

根据输入的考试成绩划分的等价类对应的测试用例如表 B.1 所示。注意下面输入的课程作业成绩已经设置为一个任意的有效值 15。本节中所有的测试用例的输入都使用任意的有效值（除了测试的变量）。

表 B.1 输入变量考试成绩的测试用例

测试用例	1	2	3
输入变量（考试成绩）	60	－10	93
输入变量（课程作业成绩）	15	15	15
总成绩（计算）	75	5	108
测试覆盖项	TCOVER1	TCOVER3	TCOVER4
测试等价类（考试成绩的）	0＜＝e＜＝75	e＜0	e＞75
预期结果	“A”	“FM”	“FM”

根据输入的课程作业成绩划分的等价类对应的测试用例如表 B.2 所示。

表 B.2 输入变量课程作业成绩的测试用例

测试用例	4	5	6
输入变量（考试成绩）	40	40	40
输入变量（课程作业成绩）	20	－15	47
总成绩（计算）	60	25	87
测试覆盖项	TCOVER2	TCOVER5	TCOVER6
测试等价类（课程作业成绩的）	0＜＝c＜＝25	c＜0	c＞25
预期结果	“B”	“FM”	“FM”

根据可能的无效输入划分的等价类对应的测试用例如表 B.3 和 B.4 所示。

表 B.3　无效的考试成绩输入对应的测试用例

测试用例	7	8	9
输入变量(考试成绩)	60.5	Q	$
输入变量(课程作业成绩)	15	15	15
总成绩(计算)	75.5	无法计算	无法计算
测试覆盖项	TCOVER7	TCOVER8	TCOVER9
测试等价类	考试成绩＝有小数部分的实数	考试成绩＝字母	考试成绩＝特殊字符
预期结果	“FM”	“FM”	“FM”

表 B.4　无效的课程作业成绩输入对应的测试用例

测试用例	10	11	12
输入变量(考试成绩)	40	40	40
输入变量(课程作业成绩)	20.23	G	@
总成绩(计算)	60.23	无法计算	无法计算
测试覆盖项	TCOVER10	TCOVER11	TCOVER12
测试等价类	课程作业成绩＝有小数部分的实数	课程作业成绩＝字母	课程作业成绩＝特殊字符
预期结果	“FM”	“FM”	“FM”

根据可能的有效输出划分的等价类对应的测试用例如表 B.5 和表 B.6 所示。

表 B.5　有效输出对应的测试用例(一)

测试用例	13	14	15
输入变量(考试成绩)	60	44	32
输入变量(课程作业成绩)	20	22	13
总成绩(计算)	80	66	45
测试覆盖项	TCOVER13	TCOVER14	TCOVER15
测试等价类(总成绩的)	70＜＝t＜＝100	50＜＝t＜70	30＜＝t＜50
预期结果	“A”	“B”	“C”

表 B.6　有效输出对应的测试用例(二)

测试用例	16	17	18
输入变量(考试成绩)	12	80	－10
输入变量(课程作业成绩)	5	60	－10
总成绩(计算)	17	140	－20

表 B.6（续）

测试覆盖项	TCOVER16	TCOVER17	TCOVER18
测试等价类(总成绩的)	0<=t<30	t>100	t<0
预期结果	“D”	“FM”	“FM”

考试成绩和课程作业成绩的输入值来源于两者的总和——总成绩。

根据无效的输出划分的等价类对应的测试用例如表 B.7 所示。

表 B.7　无效输出对应的测试用例

测试用例	19	20	21	22
输入变量(考试成绩)	47.3	5	72	Null
输入变量(课程作业成绩)	@@@	5	23	Null
总成绩(计算)	—	10	95	—
测试覆盖项	TCOVER19	TCOVER20	TCOVER21	TCOVER22
测试等价类(输出)	“FM”	“E”	“A+”	“Null”
等价类(总成绩的)	—	0<=t<=15	90<=t<=100	—
预期结果	“FM”	“D”	“A”	“FM”

根据实现的情况，可能无法执行包含了无效输出值的测试用例(比如上面例子中的测试用例 2、3、5～12 和 17～22)。例如，在 Ada 编程语言中，如果输入变量定义为一个正整数，则可能无法赋给变量一个负数。尽管如此，为了完整性还是值得考虑所有的测试用例。

B.2.1.6.3　选项 4b：导出等价类划分的最小化测试用例(TD4)

从上面可以看到一些测试用例很相似，例如测试用例 1 和 13，主要的不同之处在于从目标分区选择的具体测试覆盖项不同。因为测试覆盖项有两个输入和一个输出，每个测试用例实际上“命中”三个分类：两个输入分类和一个输出分类。因此，通过产生测试用例来测试多个等价类，可能会产生一个更小的最小化测试集仍可以“命中”所有识别的等价类。

表 B.8～表 B.10 是通过最小化等价类划分方法导出的 12 个测试用例组成的测试集，其中每个设计的测试用例覆盖多个新的等价类，而不仅仅只覆盖一个等价类。

表 B.8　最小化测试用例集(一)

测试用例	1	2	3	4
输入变量(考试成绩)	60	50	35	19
输入变量(课程作业成绩)	20	16	10	8
总成绩(计算)	80	66	45	27
测试覆盖项	TCOVER1，TCOVER2，TCOVER13	TCOVER1，TCOVER2，TCOVER14	TCOVER1，TCOVER2，TCOVER15	TCOVER1，TCOVER2，TCOVER16
等价类(考试成绩的)	0<=e<=75	0<=e<=75	0<=e<=75	0<=e<=75

表 B.8（续）

等价类(课程作业成绩的)	0<=c<=25	0<=c<=25	0<=c<=25	0<=c<=25
等价类(总成绩的)	70<=t<=100	50<=t<70	30<=t<50	0<=t<30
预期结果	“A”	“B”	“C”	“D”

表 B.9 最小化测试用例集(二)

测试用例	5	6	7	8
输入变量(考试成绩)	−10	93	60.5	Q
输入变量(课程作业成绩)	−15	47	20.23	G
总成绩(计算)	−25	140	80.73	—
测试覆盖项	TCOVER3，TCOVER5，TCOVER18	TCOVER4，TCOVER6，TCOVER17	TCOVER7，TCOVER10，TCOVER13，TCOVER19	TCOVER8，TCOVER11，TCOVER19
等价类(考试成绩的)	e<0	e>75	e=有小数部分的实数	e =字母
等价类(课程作业成绩的)	c<0	c>25	c=有小数部分的实数	c =字母
等价类(总成绩的)	t<0	t>100	70<=t<=100	—
预期结果	“FM”	“FM”	“FM”	“FM”

表 B.10 最小化测试用例集(三)

测试用例	9	10	11	12
输入变量(考试成绩)	$	5	72	“Null”
输入变量(课程作业成绩)	@	5	23	“Null”
总成绩(计算)	—	10	95	—
测试覆盖项	TCOVER9，TCOVER12，TCOVER19	TCOVER1，TCOVER2，TCOVER16 TCOVER20	TCOVER1，TCOVER2，TCOVER13，TCOVER21	TCOVER19，TCOVER22
等价类(考试成绩的)	e=特殊字符	0<=e<=75	0<=e<=75	—
等价类(课程作业成绩的)	e=特殊字符	0<=e<=25	0<=e<=25	—
等价类(总成绩的)	—	0<=e<=15	90<=t<=100	—
等价类(输出的)	—	“E”	“A+”	“Null”
预期结果	“FM”	“D”	“A”	“FM”

一对一和最小化等价类划分代表用于导出等价类划分的测试用例的两种不同的方法。一对一的测试用例对测试错误条件特别有用(例如，在尝试强制输出特定消息时)，例如，它减少一个错误条件中断

处理和/或掩饰或阻断了其他错误条件的可能性。另一方面，一对一方法的缺点是它要求更多的测试用例，如果这是一个问题，则可以用一个更简约的办法。最小化等价类划分的缺点是出现一个测试失效时，很难找到原因，因为同时测试了几个新的分类。因此，有一种结合这两种方法的常用方法，将最小化等价类划分用于设计有效的测试，用一对一等价类划分设计无效测试用例。

B.2.1.7 步骤 5：汇集测试集(TD5)

B.2.1.7.1 选项

假设可以自动检查每个测试用例接收/拒绝的结果，但是不能自动处理出错信息(FM)，则可以产生两个测试集(TS)：一个用于手动测试，另一个用于自动测试。

B.2.1.7.2 选项 5a：使用一对一等价类划分组合测试项(TD5)

TS1：手动测试——测试用例 2，3，5，6，7，8，9，10，11，12，17，18，19，22。

TS2：自动测试——测试用例 1，4，13，14，15，16，20，21。

B.2.1.7.3 选项 5b：使用最小等价类划分组合测试项(TD5)

TS3：手动测试——测试用例 5，6，7，8，9，12。

TS4：自动测试——测试用例 1，2，3，4，10，11。

B.2.1.8 步骤 6：导出测试规程(TD6)

B.2.1.8.1 选项

现在可以导出一对一等价类划分和最小等价类划分的测试过程。

B.2.1.8.2 选项 6a：导出一对一等价类划分的测试过程(TD6)

对于一对一等价类划分测试集 TS1 中的手动测试用例，可以定义一个测试规程(TP)，如下：

TP1：手动测试，按照测试集中指定的顺序覆盖 TS1 中的所有测试用例。

对于一对一测试集 TS2 的自动测试用例，可以写一个测试脚本执行测试集中的所有测试用例，如下：

TP2：自动测试，按照测试集中指定的顺序覆盖 TS2 中的所有测试用例。

对于自动测试规程 TP2，实现了测试过程的自动化代码需要写在测试自动脚本中。

B.2.1.8.3 选项 6b：导出最小化等价类划分的测试规程(TD6)

对于最小化测试集 TS3 中的手动测试用例，可以定义一个测试规程(TP)，如下：

TP3：手动测试，按照测试集中指定的顺序覆盖 TS3 中的所有测试用例。

对于最小化测试集 TS4 的自动测试用例，可以写一个测试脚本执行测试集中的所有测试用例，如下：

TP4：自动测试，按照测试集中指定的顺序覆盖 TS4 中的所有测试用例。

B.2.1.9 等价类划分覆盖率

根据 6.2.1 提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(一对一等价类划分)}=\frac{22}{22}\times 100\%=100\%$$

$$覆盖率_{(最小化等价类划分)}=\frac{12}{12}\times 100\%=100\%$$

对于一对一和最小化等价类划分，都达到了100%等价类划分覆盖率，22个确定的等价类至少在一个测试用例中测试。如果不是测试了所有确定的等价类，则达到低一点的覆盖率。如果不是确定了所有等价类，则基于这个不完整的等价类集合的测试，覆盖率可能是错误的。但是，因为对测试项不同的分析可能会产生不同的等价类，特别是“无效”值，对于等价类划分的覆盖测量应是针对确定的等价类的。

B.2.2 分类树方法

B.2.2.1 介绍

分类树方法的目的是根据选择的等价类覆盖程度，导出测试用例覆盖测试项的输入类。分类树方法构造了等价类的分类树，帮助测试者进行测试设计。

B.2.2.2 规格说明

考虑测试项 tavel_preference 的测试依据，记录了某组织员工的旅行选项，他们因为工作的需要到国内主要城市出差。旅行偏好的每个集合通过一些单选按钮来选择，这些单选按钮由下面的输入选项组成：

目的地= 北京，上海，广州，深圳，武汉，西安，成都，重庆

舱位= 头等舱，公务舱，经济舱

座位= 靠走廊，靠窗

食物偏好= 糖尿病餐，无麸质，蛋奶素食，低脂，低糖，严格素食，标准

从每个分类中选择一个类的任意组合都会出现消息“成功预订”，但其他输入都会出现一个错误的信息“无效输入”。员工不能选择不进餐，因此在本例中没有这个选项。

B.2.2.3 步骤1：识别特征集(TD1)

测试依据中只有一个测试项，只需要定义一个特征集：

FS1：tavel_preference 功能

B.2.2.4 步骤2：导出测试条件(TD2)

对于分类树测试，测试条件是通过生成每个输入参数的划分和类来确定的：

TCOND1：目的地	(FS1)
TCOND2：舱位	(FS1)
TCOND3：座位	(FS1)
TCOND4：食物偏好	(FS1)

注1：可能会导出无效的测试条件，尽管在本例中没有举例。

本例中的每个测试条件就是一个“划分”(即一个类别)。可以生成食物偏好的“类”(即子类别)和子类，如下：

偏好(分类)= 素食，非素食

素食(类)= 蛋奶素食，严格素食

非素食(类)= 糖尿病餐，无麸质，低脂，低糖，标准

注2：设计划分和类通常是一个主观的过程，使用这项技术的其他测试者可以设计与本例中不同的划分和类。

现在可以为这些测试条件生成一棵分类树，如图B.4。

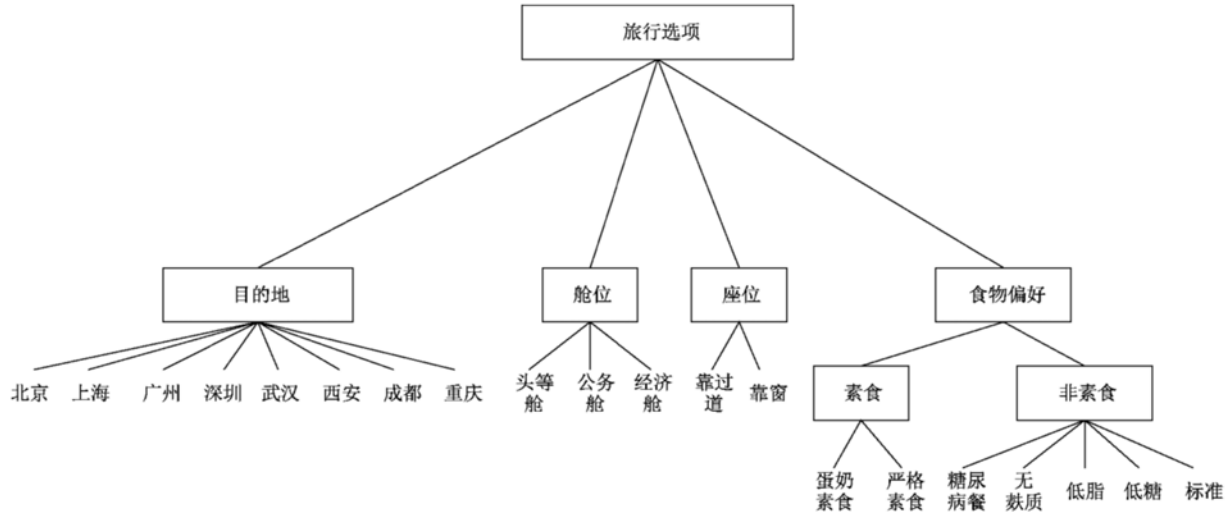


图 B.4　分类树示例

B.2.2.5　步骤 3:导出测试覆盖项(TD3)

选择一种组合方式导出测试覆盖项,并用该方法生成类的组合。根据分类树构建一个“组合表”,证明类的组合是为了形成每个测试覆盖项(例如见图 B.5)。每个测试覆盖项覆盖的类被一系列的记号标记(黑点),水平分布在分类树下面。

如果假设选择的组合方法是“最小化”,其中每个测试覆盖项覆盖尽可能多的类,直到所有类都至少被一个测试用例包含,则可以确定图 B.5 中的测试覆盖项。

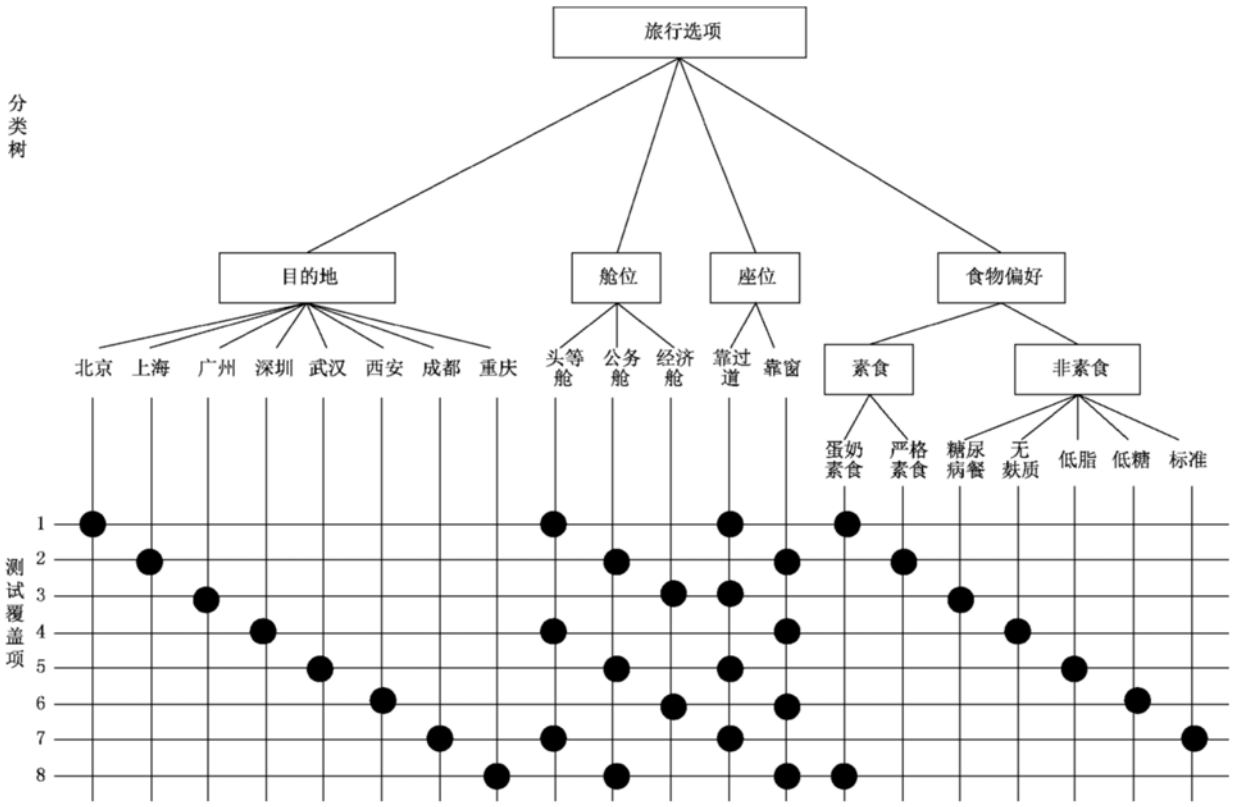


图 B.5　分类树和对应组合表示例

在本例中，测试覆盖项覆盖了所有的测试条件。

B.2.2.6 步骤 4：导出测试用例（TD4）

现在可以导出测试用例集，其中每个测试用例准确地覆盖类的一个测试覆盖项。通过依次选择一个没有被测试用例覆盖的测试覆盖项导出测试用例，并用覆盖类组合的测试输入计算它。一直重复进行直到达到要求的覆盖率。通过将输入放到测试依据来生成预期结果。在这种特殊的情况下，任何有效输入的组合都会生成状态“预订成功”，见表 B.11。

表 B.11 分类树测试的测试用例

测试用例	输入值				预期结果	测试覆盖项
	目的地	舱位	座位	食物偏好		
1	北京	头等舱	靠过道	蛋奶素食	预订成功	TCOVER1
2	上海	公务舱	靠窗	严格素食	预订成功	TCOVER2
3	广州	经济舱	靠过道	糖尿病餐	预订成功	TCOVER3
4	深圳	头等舱	靠窗	无麸质	预订成功	TCOVER4
5	武汉	公务舱	靠过道	低脂	预订成功	TCOVER5
6	西安	经济舱	靠窗	低糖	预订成功	TCOVER6
7	成都	头等舱	靠过道	标准	预订成功	TCOVER7
8	重庆	公务舱	靠窗	蛋奶素食	预订成功	TCOVER8

B.2.2.7 步骤 5：汇集测试集（TD5）

既然本例中生成的测试用例较少，则可以考虑将其合并成一个测试集。

TS1：测试用例 1，2，3，4，5，6，7，8。

B.2.2.8 步骤 6：导出测试规程（TD6）

所有的测试用例都在一个测试集中，可以导出一个测试规程。

TP1：按照测试集中指定的顺序覆盖 TS1 中的所有测试集。

B.2.2.9 分类树方法的覆盖率

根据 6.2.2 提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(分类树方法)}=\frac{8}{8}\times 100\%=100\%$$

因此，对于分类树方法，测试覆盖项的覆盖率达到了 100%。

B.2.3 边界值分析

B.2.3.1 介绍

边界值分析的目的是根据选择的边界值覆盖率导出测试用例集，这些测试用例应覆盖测试项的每个输入和输出的所有边界值。该方法有如下前提要求：第一，根据测试项的测试依据，可以将测试项的输入和输出划分为等价类，测试项对等价类中的所有值的处理方式相似；第二，有些等价类的元素可以

从小到大无间隔排列；第三，根据经验，有序的连续区间的边界是软件开发中容易出错的地方。导出测试用例是为了测试边界。

下面是一个有三值的边界值测试的示例，它采用了一对一的方法导出测试用例（分别见 5.2.3.2 和 5.2.3.3）。为了导出测试项的边界，应先确定测试项的等价类，然后从每个等价类导出边界值。

B.2.3.2 规格说明

考虑测试项——generate_grading，测试依据如下：

组件收到一个考试成绩（不高于 75 分）输入和一个课程作业（c/w）成绩（不高于 25 分）输入，根据成绩输出一个“A”到“D”的课程等级。等级是通过计算考试和课程作业成绩的总和得到的，具体如下：

大于或等于 70 ——‘A’
大于或等于 50，小于 70 ——‘B’
大于或等于 30，小于 50 ——‘C’
小于 30 ——‘D’

在检测到无效输入（如分数超出了预期的范围）时给出错误信息（‘FM’）。所有的输入都使用整数。

B.2.3.3 步骤 1：识别特征集（TD1）

测试基准中只有一个测试项，只需要定义一个特征集（FS）：

FS1：generate_grading 功能

B.2.3.4 步骤 2：导出测试条件（TD2）

B.2.3.4.1 子步骤

边界值分析的测试条件是测试期间需要覆盖的已经选择的边界（等价类之间）。为了确定边界，首先需要确定等价类（见 B.2.3.4.2 步骤 2a），然后根据等价类导出测试条件（边界）（见 B.2.3.4.3 步骤 2b）。

B.2.3.4.2 步骤 2a：确定等价类

等价类可从特征集 FS1 有效和无效的输入和输出中被导出。下面是基于输入识别的有效等价类（EP）：

EP1：0＜＝考试成绩＜＝75 （FS1）
EP2：0＜＝课程作业成绩＜＝25 （FS1）

基于输入的最明显的无效等价类如下：

EP3：考试成绩＞75 （FS1）
EP4：考试成绩＜0 （FS1）
EP5：课程作业成绩＞25 （FS1）
EP6：课程作业成绩＜0 （FS1）

尽管等价类 EP3 到 EP6 看起来只有一个边界，但实际上这些分类受限于与实现相关的最小值和最大值。对于 16 比特的整数来说，这些整数的最大值和最小值分别是 32767 和－32768。因此，EP3 到 EP6 可以更加完整的定义，如下：

EP3：75＜考试成绩＜＝32767 （FS1）
EP4：－32768＜＝课程作业成绩＜0 （FS1）
EP5：25＜课程作业成绩＜＝32767 （FS1）
EP6：－32768＜＝课程作业成绩＜0 （FS1）

分类值的范围可以用图 B.6 表示。

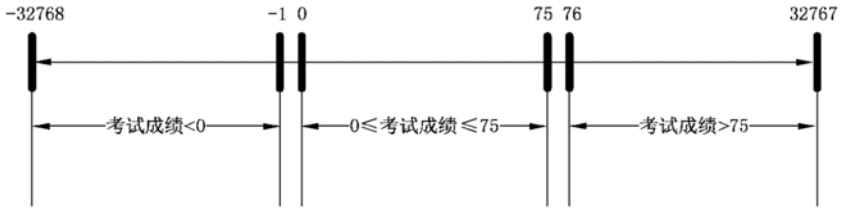


图 B.6　考试成绩的等价类和边界

根据输入——课程作业成绩，得到图 B.7。

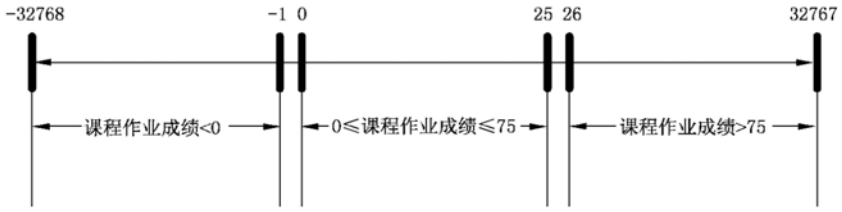


图 B.7　课程作业成绩的等价类和边界

不明显的无效输入类可能包括有效分类以外的其他输入类型，例如，非整数输入或者非数字输入。等价类划分在这方面是主观的，因此，每个测试者可能会产生他们认为相关的不同的分类。等价类划分中测试项应同等对待一个等价类的所有的值。可以根据两个输入域导出下面的无效等价类：

EP7：考试成绩＝有小数部分的实数　　(FS1)

EP8：考试成绩＝字母　　(FS1)

EP9：考试成绩＝特殊字符　　(FS1)

EP10：课程作业成绩＝有小数部分的实数　　(FS1)

EP11：课程作业成绩＝字母　　(FS1)

EP12：课程作业成绩＝特殊字符　　(FS1)

尽管等价类 EP7 到 EP12 是可能存在的情况，但是没有明确的边界，因此在边界值分析中不需要为它们导出测试覆盖项和测试用例。

下面确定输出的等价类。考虑每个测试项的有效输出，导出的有效等价类如下：

EP13：“A”的条件　　70＜＝总成绩＜＝100　　(FS1)

EP14：“B”的条件　　50＜＝总成绩＜70　　(FS1)

EP15：“C”的条件　　30＜＝总成绩＜50　　(FS1)

EP16：“D”的条件　　0＜＝总成绩＜30　　(FS1)

EP17：“错误信息(FM)的条件 总成绩＞100　　(FS1)

EP18：“错误信息”(FM)的条件　　总成绩＜0　　(FS1)

其中总成绩＝考试成绩＋课程作业成绩。

和输入相似，输出的边界受限于与实现相关的最小值和最大值。假设输出使用了 16 bits 整数存储，这些整数的最大值和最小值分别是 32768 和－32767，则 EP17 和 EP18 应重新定义为：

EP17：“错误信息”(FM)的条件　　100＜总成绩＜＝32767　　(FS1)

EP18：“错误信息”(FM)的条件　　－32768＜总成绩＜0　　(FS1)

使用出错信息，出错信息是一个指定的输出。总成绩的等价类和边界如图 B.8 所示。

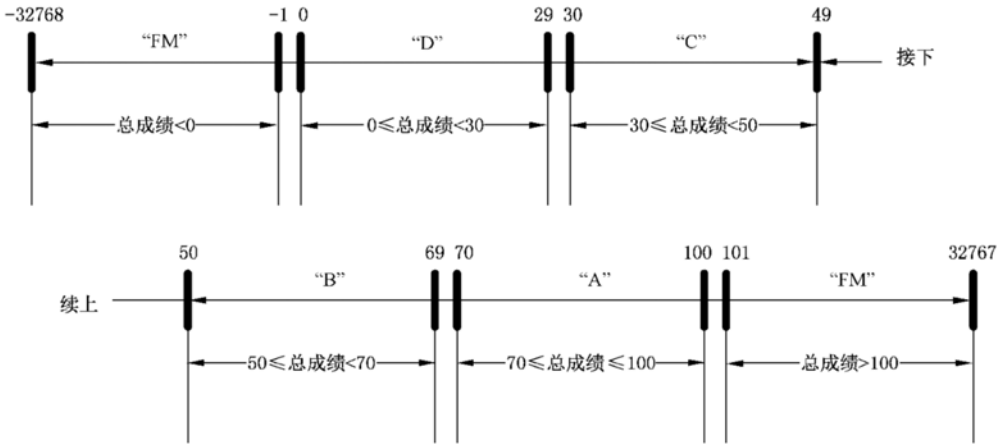


图 B.8　总成绩的等价类和边界值

无效输出是除了五个指定测试项输出以外的任意输出。这很难发现非指定的输出，但是很明显应考虑这些无效输出，因为如果发现了一个无效输出，一定可以识别一个测试项或测试依据的缺陷，或二者都有的缺陷。在本例中，3 个没有指定的输出被确定为("E""A+"和"null")，但是不可能将这些可能的输出分成有序有边界的等价类，因此不能导出测试用例。

B.2.3.4.3　步骤 2b：导出测试条件

一旦确定每个输入和输出域的等价类后，即可确定测试条件(每个等价类的边界)。

对应考试成绩和课程作业成绩输入域的有效类，可以导出以下的测试条件。注意重复边界值(比如边界"0"同时是 EP1 和 EP4 的边界)只能被一个测试条件覆盖。

TCOND1：考试成绩＝0　　(EP1 和 EP4)

TCOND2：考试成绩＝75　　(EP1 和 EP3)

TCOND3：课程作业成绩＝0　　(EP2 和 EP6)

TCOND4：课程作业成绩＝25　　(EP3 和 EP5)

对应总成绩的有效类，可以确定以下的边界。

TCOND5：总成绩＝0　　(EP16 和 EP18)

TCOND6：总成绩＝29　　(EP15 和 EP16)

TCOND7：总成绩＝30　　(EP15 和 EP16)

TCOND8：总成绩＝49　　(EP14 和 EP15)

TCOND9：总成绩＝50　　(EP14 和 EP15)

TCOND10：总成绩＝69　　(EP13 和 EP14)

TCOND11：总成绩＝70　　(EP13 和 EP14)

TCOND12：总成绩＝100　　(EP13 和 EP17)

对应输入域的无效类，可以确定以下的边界。

TCOND13：考试成绩＝32767　　(EP3)

TCOND14：考试成绩＝－32768　　(EP4)

TCOND15：课程作业成绩＝32767　　(EP5)

TCOND16：课程作业成绩＝－32768　　(EP6)

最后，对应总成绩的无效类，可以确定以下的边界

TCOND17：总成绩＝101　　(EP17)

TCOND18：总成绩＝32767 （EP17）

TCOND19：总成绩＝－1 （EP18）

TCOND20：总成绩＝－32768 （EP19）

B.2.3.5 步骤 3：导出测试覆盖项（TD3）

如果使用三值边界值分析，测试覆盖项是等价类划分边界点上的值，以及边界点两侧的值，两侧的取值应是距离边界点最近的有效值，如图 B.9 所示。

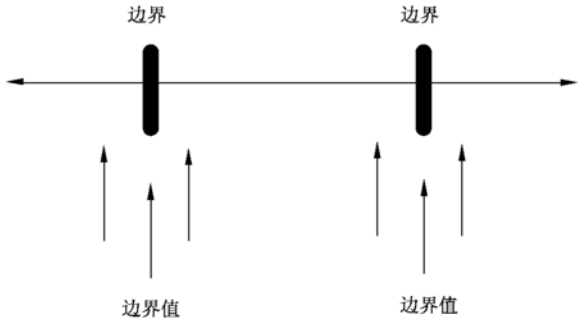


图 B.9 三值边界值分析的测试覆盖项

注 1：另外，也可以使用二值边界值分析，会导出更少的测试覆盖项，对应导出更少的测试用例。

在前面步骤中，使用识别的边界作为测试条件，下面使用测试条件识别测试覆盖项（TCOVER）。由于本例中使用整数作为输入，所以测试覆盖项是每个边界两侧相邻的整数。

注 2：如果例子中数据类型涉及小数类型的（比如实数），则边界值的测试覆盖项要考虑数据类型的最小值。

TCOVER1：考试成绩＝－1 （TCOND1）

TCOVER2：考试成绩＝0 （TCOND1）

TCOVER3：考试成绩＝1 （TCOND1）

TCOVER4：考试成绩＝74 （TCOND2）

TCOVER5：考试成绩＝75 （TCOND2）

TCOVER6：考试成绩＝76 （TCOND2）

TCOVER7：课程作业成绩＝－1 （TCOND3）

TCOVER8：课程作业成绩＝0 （TCOND3）

TCOVER9：课程作业成绩＝1 （TCOND3）

TCOVER10：课程作业成绩＝24 （TCOND4）

TCOVER11：课程作业成绩＝25 （TCOND4）

TCOVER12：课程作业成绩＝26 （TCOND4）

对应输出域，可以确定一下的测试覆盖项：

TCOVER13：总成绩＝－1 （TCOND5 和 TCOND19）

TCOVER14：总成绩＝0 （TCOND5 和 TCOND19）

TCOVER15：总成绩＝1 （TCOND5）

TCOVER16：总成绩＝28 （TCOND6）

TCOVER17：总成绩＝29 （TCOND6 和 TCOND7）

TCOVER18：总成绩＝30 （TCOND6 和 TCOND7）

TCOVER19：总成绩＝31 （TCOND7）

TCOVER20：总成绩＝48 （TCOND8）

TCOVER21:总成绩=49 (TCOND8 和 TCOND9)
TCOVER22:总成绩=50 (TCOND8 和 TCOND9)
TCOVER23:总成绩=51 (TCOND9)
TCOVER24:总成绩=68 (TCOND10)
TCOVER25:总成绩=69 (TCOND10 和 TCOND11)
TCOVER26:总成绩=70 (TCOND10 和 TCOND11)
TCOVER27:总成绩=71 (TCOND11)
TCOVER28:总成绩=99 (TCOND12)
TCOVER29:总成绩=100 (TCOND12 和 TCOND17)
TCOVER30:总成绩=101 (TCOND12 和 TCOND17)

注意,由于没有明确的边界(如前面的步骤所示),因此没有任何测试覆盖项覆盖等价类 EP7 到 EP12。

根据剩下确定的无效等价类(如还没有被覆盖的测试条件 TCOND13～TCOND20 的边界),确定无效的测试覆盖项:

TCOVER31:考试成绩=32766 (TCOND13)
TCOVER32:考试成绩=32767 (TCOND13)
TCOVER33:考试成绩=32768 (TCOND13)
TCOVER34:考试成绩=－32769 (TCOND14)
TCOVER35:考试成绩=－32768 (TCOND14)
TCOVER36:考试成绩=－32767 (TCOND14)
TCOVER37:课程作业成绩=32766 (TCOND15)
TCOVER38:课程作业成绩=32767 (TCOND15)
TCOVER39:课程作业成绩=32768 (TCOND15)
TCOVER40:课程作业成绩=－32769 (TCOND16)
TCOVER41:课程作业成绩=－32768 (TCOND16)
TCOVER42:课程作业成绩=－32767 (TCOND16)
TCOVER43:总成绩=102 (TCOND17)
TCOVER44:总成绩=32766 (TCOND18)
TCOVER45:总成绩=32767 (TCOND18)
TCOVER46:总成绩=32768 (TCOND18)
TCOVER47:总成绩=－2 (TCOND19)
TCOVER48:总成绩=－32769 (TCOND20)
TCOVER49:总成绩=－32768 (TCOND20)
TCOVER50:总成绩=－32767 (TCOND20)

B.2.3.6 步骤 4:导出测试用例(TD4)

现在可以导出测试用例来达到前面步骤中导出的测试覆盖项要求的覆盖率。例如,如果要求达到100%边界覆盖,那么导出的测试用例应覆盖所有的测试覆盖项。可以使用一对一的边界值分析法对每个测试覆盖项导出一个测试用例,也可以使用最少边界值分析法导出最少的测试用例覆盖所有的测试覆盖项。

generate_grading 功能所有测试用例的前置条件是相同的:该应用已经准备好接收考试成绩和课程作业成绩的输入。

假设要求达到 100%的边界值覆盖并使用一对一的边界值分析法来导出测试用例,对应输入考试

成绩可以导出 6 个测试用例，如表 B.12 所示。导出的测试用例过程如下：第一步，每个测试用例中选择一个边界值（测试覆盖项）；第二步，分配一个任意有效的值给测试用例中所有其他的输入；第三，确定测试的预期结果。

表 B.12　考试成绩的边界值

测试用例	1	2	3	4	5	6
输入（考试成绩）	−1	0	1	74	75	76
输入（c/w 成绩）	15	15	15	15	15	15
总成绩（计算）	14	15	16	89	90	91
测试覆盖项	1	2	3	4	5	6
被测试的边界（考试成绩的）	0			75		
预期结果	“FM”	“D”	“D”	“A”	“A”	“FM”

注 1：课程作业成绩输入已经设置为一个任意的有效值 15，表 B.12 中测试用例主要用于测试输入考试成绩的边界值。

根据输入的课程作业成绩导出的测试用例如表 B.13 所示。

表 B.13　课程作业成绩的边界值

测试用例	7	8	9	10	11	12
输入（考试成绩）	40	40	40	40	40	40
输入（c/w 成绩）	−1	0	1	24	25	26
总成绩（计算）	39	40	41	64	65	66
测试覆盖项	7	8	9	10	11	12
被测试的边界（c/w 成绩的）	0			25		
预期结果	“FM”	“C”	“C”	“B”	“B”	“FM”

注 2：考试成绩已经设置为一个任意的有效值 40。

根据输出导出的测试用例如表 B.14、表 B.15 和表 B.16 所示。

表 B.14　总成绩的测试用例（一）

测试用例	13	14	15	16	17	18	19
输入（考试成绩）	−1	0	0	28	29	15	6
输入（c/w 成绩）	0	0	1	0	0	15	25
总成绩（计算）	−1	0	1	28	29	30	31
测试覆盖项	13	14	15	16	17	18	19
被测试的边界（总成绩的）	0			29	29，30		30
预期结果	“FM”	“D”	“D”	“D”	“D”	“C”	“C”

表 B.15 总成绩的测试用例(二)

测试用例	20	21	22	23	24	25	26	27
输入(考试成绩)	23	24	50	26	48	49	45	71
输入(c/w 成绩)	25	25	0	25	20	20	25	0
总成绩(计算)	48	49	50	51	69	69	70	71
测试覆盖项	20	21	22	23	24	25	26	27
被测试的边界(总成绩的)	49	49,50		50	69	69,70		70
预期结果	"C"	"C"	"B"	"B"	"B"	"A"	"A"	"A"

表 B.16 总成绩的测试用例(三)

测试用例	28	29	30
输入(考试成绩)	74	75	75
输入(c/w 成绩)	25	25	26
总成绩(计算)	99	100	101
测试覆盖项	28	29	30
被测试的边界(总成绩的)	100	100 101	
预期结果	"A"	"A"	"FM"

考试成绩和课程作业成绩的输入值来源于它们的总和——总成绩。

表 B.17、表 B.18 和表 B.19 所示的测试用例用于覆盖剩余的测试覆盖项 TCOVER31 到 TCOVER50,这些覆盖项是根据有效边界的外部和等价类的极端边缘确定的。

表 B.17 考试成绩的测试用例

测试用例	31	32	33	34	35	36
输入(考试成绩)	32766	32767	32768	−32769	−32768	−32767
输入(c/w 成绩)	15	15	15	15	15	15
总成绩(计算)	32781	32782	32783	−32754	−32753	−32752
测试覆盖项	31	32	33	34	35	36
被测试的边界(考试成绩的)	32767			−32768		
预期结果	"FM"	"FM"	"FM"	"FM"	"FM"	"FM"

表 B.18 课程作业成绩的测试用例

测试用例	37	38	39	40	41	42
输入(考试成绩)	40	40	40	40	40	40
输入(c/w 成绩)	32766	32767	32768	−32769	−32768	−32767
总成绩(计算)	32806	32807	32808	−32729	−32728	−32727

表 B.18（续）

测试覆盖项	37	38	39	40	41	42
被测试的边界(c/w 成绩的)	32767			−32768		
预期结果	“FM”	“FM”	“FM”	“FM”	“FM”	“FM”

表 B.19 总成绩的测试用例(四)

测试用例	43	44	45	46	47	48	49	50
输入(考试成绩)	75	16 383	32767	1	−1	0	−16 384	−32766
输入(c/w 成绩)	27	16 383	0	32767	−1	−32769	−16 384	−1
总成绩(计算)	102	32766	32767	32768	−2	−32769	−32768	−32767
测试覆盖项	43	44	45	46	47	48	49	50
被测试的边界(总成绩的)	101	32767			1	32768		
预期结果	“FM”	“FM”	“FM”	“FM”	“FM”	“FM”	“FM”	“FM”

需要注意当使用无效输入时(如上,测试用例 1、6、7、12、13 和 30～50),由于软件实现的原因,可能会出现实际不能执行的测试用例。例如,在 Ada 编程语言中,如果输入定义为一个正整数,则无法赋给变量负数。但是,为了完整性还是需要考虑所有的测试用例。

上面的测试用例集合对于 3 个值的边界值分析,达到了 100%的边界值覆盖,所有确定的覆盖项至少被一个测试用例测试。如果有部分确定的边界没有测试,则达到低一点的覆盖率。如果有些边界没有被确定,那么基于这个不完整的边界值集合的任何覆盖测量都具有误导性。

B.2.3.7 步骤 5:汇集测试集(TD5)

假设可以自动检查每个测试用例接收/拒绝的结果,但是不能自动处理无效测试用例产生的错误信息(FM),则可以产生两个测试集(TS):一个用于手动测试,另一个用于自动测试。

TS1:人工测试——测试用例 1,6,7,12,13,30～50。

TS2:自动化测试——测试用例 2,3,4,5,8,9,10,11,14～29。

B.2.3.8 步骤 6:导出测试规程(TD6)

对于测试集 TS1 中的手动测试用例,可以定义一个测试规程(TP),如下:

TP1:手动测试,按照测试集中指定的顺序覆盖 TS1 中的所有测试用例。

对于测试集 TS2 的自动测试用例,可以写一个测试脚本执行测试集中的所有测试用例,如下:

TP2:自动测试,按照测试集中指定的顺序覆盖 TS2 中的所有测试用例。

对于自动测试规程 TP2,执行了测试规程的自动化代码需要写在测试自动脚本中。

B.2.3.9 边界值分析覆盖率

根据 6.2.3 中提供的公式和上面导出的测试覆盖项:

$$覆盖率_{(边界值分析)}=\frac{50}{50}\times 100\%=100\%$$

因此,对于边界值分析,测试覆盖项的覆盖率达到了 100%。

B.2.4 语法测试

B.2.4.1 介绍

语法测试的目的是根据语法覆盖率的要求导出测试用例集，这些测试用例能够覆盖测试项所有输入的语法。语法测试是基于对测试项的测试基础的分析，通过对输入的语法描述来对其行为建模。通过一个样例来说明这个技术。只有当定义的语法和要求的语法一致时，语法测试才有效。

B.2.4.2 规格说明

考虑检查一个输入 float_in 是否和在下面定义的浮点数的语法 float 一致的测试项。测试项的输出 check_res 根据检查的结果取值“有效”或“无效”。

下面是以巴科斯范式(BNF)形式表示的浮点数语法的 float：

Float ＝int “e”int
int ＝[“＋”| “－”]nat
nat ＝{dig}
dig ＝“0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”

在引号中的字符代表着这些字符本身；这些是语法最基础的部分——实际输入字符组成了测试项的输入。| 将可选项分开。[ ]内部是可选项，不选择任何一个也是一个可选项。{ }内部是可以迭代一次或者多次的项。

B.2.4.3 步骤 1：识别特征集(TD1)

测试依据中只有一个定义的测试项，只需要定义一个功能：

FS1： float_in

B.2.4.4 步骤 2：导出测试条件(TD2)

第一步是根据语法导出测试条件。测试条件可能定义为语法中的输入参数，如下：

TCOND1 float＝int“e”int
TCOND2 int＝[“＋”|“－”]nat
TCOND3 nat＝{dig}
TCOND4 dig＝“0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”

B.2.4.5 步骤 3：导出测试覆盖项(TD3)

语法测试的测试覆盖项是定义的语法的“选项”(有效的测试项覆盖项)和“变异”(无效的测试项覆盖项)(见 5.2.4.2 中对“选项”和“变异”的定义)。

根据 BNF 定义右侧元素导出有效的测试项覆盖项。对于 TCOND2 中的“＋”和“－”可以导出三个测试覆盖项：

TCOVER1： 没有“＋”和“－”符号 (TCOND2 和 TCOND1)
TCOVER2： 有一个“＋”符号 (TCOND2 和 TCOND1)
TCOVER3： 有一个“－”符号 (TCOND2 和 TCOND1)

注 1：如果需要，“＋”和“－”的第一个和第二个实例可以分别导出测试覆盖项。

nat 有两个测试覆盖项：

TCOVER4： nat 是一位数 (TCOND3 和 TCOND2)
TCOVER5： nat 是多位数 (TCOND3 和 TCOND2)

注 2：如果需要，nat 的第一个和第二个实例可以分别导出测试覆盖项。

dig 有 10 选项：

TCOVER6： 整数“0” （TCOND4 和 TCOND3）
TCOVER7： 整数“1” （TCOND4 和 TCOND3）
TCOVER8： 整数“2” （TCOND4 和 TCOND3）
TCOVER9： 整数“3” （TCOND4 和 TCOND3）
TCOVER10： 整数“4” （TCOND4 和 TCOND3）
TCOVER11： 整数“5” （TCOND4 和 TCOND3）
TCOVER12： 整数“6” （TCOND4 和 TCOND3）
TCOVER13： 整数“7” （TCOND4 和 TCOND3）
TCOVER14： 整数“8” （TCOND4 和 TCOND3）
TCOVER15： 整数“9” （TCOND4 和 TCOND3）

以上一共定义了 15 个有效的测试覆盖项。

导出无效测试覆盖项的第一步是构建一个适用于测试条件的通用变异清单。可能的清单是：

m1：引入一个元素的无效值；

m2：用一个已定义的元素替代另一个元素；

m3：缺失一个定义的元素；

m4：增加一个额外的元素。

注 3：可以使用其他类型的语法错误，由期望通过测试发现的缺陷类型而定。

在语法的独立元素上使用上述变异，导出特定的变异。

TCOVER16：在第一个“int”处使用 m1 （TCOND1）
TCOVER17：在“e”处使用 m1 （TCOND1）
TCOVER18：在第二个“int”处使用 m1 （TCOND1）
TCOVER19：在[“+”|“−”]处使用 m1 （TCOND2）
TCOVER20：在“nat”处使用 m1 （TCOND2）
TCOVER21：使用 m2 用“e”替代第一个“int” （TCOND1）
TCOVER22：使用 m2 用[“+”|“−”]替代第一个“int” （TCOND1 和 TCOND2）
TCOVER23：使用 m1 用第一个“int” 替代“e” （TCOND1）
TCOVER24：使用 m2 用[“+”|“−”]替代“e” （TCOND1 和 TCOND2）
TCOVER25：使用 m2 用“e”替代第二个“int” （TCOND1）
TCOVER26：使用 m2 用[“+”|“−”]替代第二个“int” （TCOND1）
TCOVER27：使用 m2 用“e”替代[“+”|“−”] （TCOND1 和 TCOND2）
TCOVER28：使用 m2 用“e”替代“nat” （TCOND1 和 TCOND2）
TCOVER29：使用 m2 用[“+”|“−”]替代“nat” （TCOND1 和 TCOND2）
TCOVER30：在第一个“int”处使用 m3 （TCOND1）
TCOVER31：在“e”处使用 m3 （TCOND1）
TCOVER32：在第二个“int”处使用 m3 （TCOND1）
TCOVER33：使用 m4 在第一个“int”前面增加一个元素 （TCOND1）
TCOVER34：使用 m4 在“e”前面增加一个元素 （TCOND1）
TCOVER35：使用 m4 在第二个“int”前面增加一个元素 （TCOND1）
TCOVER36：使用 m4 在第二个“int”后面增加一个元素 （TCOND1）
TCOVER37：使用 m4 在第一个“int”和[“+”|“−”]前面增加一个元素 （TCOND 和 TCOND2）
TCOVER38：使用 m4 在[“+”|“−”]和第一个“int”之间增加一个元素 （TCOND1 和 TCOND2）

TCOVER39:使用 m4 在第一个“int”和“e”前面增加一个元素　　(TCOND1)

[“+”|“−”]视作是一个元素,因为该单独可选项的变异不能产生无效语法的独立测试用例(使用通用的变异方法)。

B.2.4.6　步骤 4:导出测试用例(TD4)

通过选择当前测试用例中包含的一个或多个选项导出有效测试用例,确定输入来满足选项和决定预期结果(在本例中,“check_res”)。由此产生的有效测试用例见表 B.20。

表 B.20　语法测试的有效测试用例

测试用例	输入“float_in”	测试覆盖项	预期结果“check_res”
TC1	3e2	TCOVER1	“有效”
TC2	+2e+5	TCOVER2	“有效”
TC3	−6e−7	TCOVER3	“有效”
TC4	62−2	TCOVER4	“有效”
TC5	1234567890e3	TCOVER5	“有效”
TC6	0e0	TCOVER6	“有效”
TC7	1e1	TCOVER7	“有效”
TC8	2e2	TCOVER8	“有效”
TC9	3e3	TCOVER9	“有效”
TC10	4e4	TCOVER10	“有效”
TC11	5e5	TCOVER11	“有效”
TC12	6e6	TCOVER12	“有效”
TC13	7e7	TCOVER13	“有效”
TC14	8e8	TCOVER14	“有效”
TC15	9e9	TCOVER15	“有效”

表 B.20 绝不是测试 15 个选项的最小测试集(可以减少到只有 3 个测试用例,比如,上面的 TC2、TC3 和 TC5),有些测试用例会测试“测试覆盖项”列中的多个选项而不是一个单独的选项。这里对每个选项分别进行了处理,以帮助我们理解它们的来源。这种方法有助于定位失效原因。

通过选择包含在当前测试用例中的一个或多个变异导出无效测试用例,确定输入来测试变异和决定预期结果(在本例中,“check_res”)。由此产生的无效测试用例见表 B.21。

表 B.21　语法测试的无效测试用例

测试用例	输入“float_in”	变异	测试覆盖项	预期结果“check_res”
TC16	xe0	m1	TCOVER16	“无效”
TC17	0x0	m1	TCOVER17	“无效”
TC18	0ex	m1	TCOVER18	“无效”
TC19	x0e0	m1	TCOVER19	“无效”

表 B.21（续）

测试用例	输入“float_in”	变异	测试覆盖项	预期结果“check_res”
TC20	＋xe0	m1	TCOVER520	“无效”
TC21	ee0	m2	TCOVER21	“无效”
TC22	＋e0	m2	TCOVER22	“无效”
TC23	000	m2	TCOVER23	“无效”
TC24	0＋0	m2	TCOVER24	“无效”
TC25	0ee	m2	TCOVER25	“无效”
TC26	0e＋	m2	TCOVER26	“无效”
TC27	e0e0	m2	TCOVER27	“无效”
TC28	＋ee0	m2	TCOVER28	“无效”
TC29	＋＋e0	m2	TCOVER29	“无效”
TC30	e0	m3	TCOVER30	“无效”
TC31	00	m3	TCOVER31	“无效”
TC32	0e	m3	TCOVER32	“无效”
TC33	y0e0	m4	TCOVER33	“无效”
TC34	0ye0	m4	TCOVER34	“无效”
TC35	0ey0	m4	TCOVER35	“无效”
TC36	0e0y	m4	TCOVER36	“无效”
TC37	y＋0e0	m4	TCOVER37	“无效”
TC38	＋y0e0	m4	TCOVER38	“无效”
TC39	＋0ye0	m4	TCOVER39	“无效”

一些变异是正确形式的扩展，因此已经被抛弃了。例如，通用变异 m2（用 TCOND2 代替 TCOND4）产生了正确的语法，m2 是“用另一个定义的元素替代一个元素”，和 TCOND2 和 TCOND4 是一致的（int）。

剩下的一些变异彼此不能明显区分，可以用同一个测试用例进行覆盖。例如，使用通用变异 m1（“引入一个元素的无效值”）代替本来应该是整数的 TCOND4，用“＋”产生了形式“0e＋”。和上面的测试用例 26 有相同的输入。

通过使用单独的变异或者组合变异来导出更多的测试用例。

B.2.4.7 步骤 5：汇集测试集（TD5）

可以将有效的测试用例组合成一个测试集，无效的测试用例组合成另一个测试集：

TS1：测试用例 1，2，3，4，5，6，7，8，9，10，11，12，13，14，15。

TS2：测试用例 16，17，18，19，20，21，22，23，24，25，26，27，28，29，30，31，32，33，34，35，36，37，38，39。

B.2.4.8 步骤 6:导出测试规程(TD6)

所有的测试用例可以汇集到一个测试规程,从有效的测试用例开始,以无效的测试用例结束。

TP1:以测试集指定的顺序覆盖 TS1 中的所有测试用例,然后是 TS2 中的所有测试用例。

B.2.4.9 语法测试覆盖率

如 6.2.4 中所述,尚无可以计算语法测试的测试覆盖项覆盖率的方法。

B.2.5 组合测试设计技术

B.2.5.1 介绍

组合测试的目的是通过导出较少(可能是最少的)的测试用例来减少测试的开销,测试用例覆盖选中的测试项参数和输入值。组合测试设计技术提供了一种通过现在已经被选中的输入值导出测试用例的方法,比如通过其他基于规格说明的测试设计技术,例如等价类划分或者边界值分析。每项技术将通过一个应用示例来说明。每项技术有通用的步骤:识别特征集和导出测试条件,在下面对所有技术说明通用的步骤,下面的步骤是导出测试覆盖项和测试用例,每个组合技术都不同。

B.2.5.2 规格说明

考虑测试项 tavel_preference 的测试依据,记录了某组织员工的出行偏好,他们因为工作的需要到国内主要城市出差。出行偏好的每个集合通三组单选按钮来选择,由下面的输入选项组成:

目的地=北京、上海、广州

舱位=头等舱,公务舱,经济舱

座位=靠过道,靠窗

如果在程序中输入一个有效的输入组合,将输出“同意”,否则输出“拒绝”。

B.2.5.3 步骤 1:识别特征集(TD1)

测试依据中只有一个测试项,只需要定义一个特征集:

FS1:tavel_preference 功能

B.2.5.4 步骤 2:导出测试条件(TD2)

每项组合技术都有导出测试条件的通用方法。即测试条件和测试项的每个参数(P)相关,采用一个设定的值(V),形成键值对。重复执行直到所有的参数与其设定的值都成对。对于上面的例子,会生成以下的键值对:

TCOND 1: 目的地=北京 (FS1)

TCOND 2: 目的地=上海 (FS1)

TCOND 3: 目的地=广州 (FS1)

TCOND 4: 舱位=头等舱 (FS1)

TCOND 5: 舱位=公务舱 (FS1)

TCOND 6: 舱位=经济舱 (FS1)

TCOND 7: 座位=靠过道 (FS1)

TCOND 8: 座位=靠窗 (FS1)

B.2.5.5 完全组合测试

B.2.5.5.1 步骤 3：导出测试覆盖项(TD3)

在完全组合测试中，测试覆盖项是键值对的不重复组合，每个测试项参数取一个键值对。键值对在前面确定为测试条件。

TCOVER1：目的地＝北京　舱位＝头等舱　座位＝靠过道　(TCOND1,4,7)
TCOVER2：目的地＝北京　舱位＝头等舱　座位＝靠窗　(TCOND1,4,8)
TCOVER3：目的地＝北京　舱位＝公务舱　座位＝靠过道　(TCOND1,5,7)
TCOVER4：目的地＝北京　舱位＝公务舱　座位＝靠窗　(TCOND1,5,8)
TCOVER5：目的地＝北京　舱位＝经济舱　座位＝靠过道　(TCOND1,6,7)
TCOVER6：目的地＝北京　舱位＝经济舱　座位＝靠窗　(TCOND1,6,8)
TCOVER7：目的地＝上海　舱位＝头等舱　座位＝靠过道　(TCOND2,4,7)
TCOVER8：目的地＝上海　舱位＝头等舱　座位＝靠窗　(TCOND2,4,8)
TCOVER9：目的地＝上海　舱位＝公务舱　座位＝靠过道　(TCOND2,5,7)
TCOVER10：目的地＝上海　舱位＝公务舱　座位＝靠窗　(TCOND2,5,8)
TCOVER11：目的地＝上海　舱位＝经济舱　座位＝靠过道　(TCOND2,6,7)
TCOVER12：目的地＝上海　舱位＝经济舱　座位＝靠窗　(TCOND2,6,8)
TCOVER13：目的地＝广州　舱位＝头等舱　座位＝靠过道　(TCOND3,4,7)
TCOVER14：目的地＝广州　舱位＝头等舱　座位＝靠窗　(TCOND3,4,8)
TCOVER15：目的地＝广州　舱位＝公务舱　座位＝靠过道　(TCOND3,5,7)
TCOVER16：目的地＝广州　舱位＝公务舱　座位＝靠窗　(TCOND3,5,8)
TCOVER17：目的地＝广州　舱位＝经济舱　座位＝靠过道　(TCOND3,6,7)
TCOVER18：目的地＝广州　舱位＝经济舱　座位＝靠窗　(TCOND3,6,8)

B.2.5.5.2 步骤 4：导出测试用例(TD4)

通过选择一个键值对，并组合其他参数的键值对(其中每个组合代表一个测试用例)来导出测试用例，确认测试用例其他的输入变量取任意有效值，确定预期结果，重复上述步骤直到达到覆盖要求。在本例中，测试用例的结果如表 B.22 所示。

表 B.22 完全组合测试的测试用例

测试用例	输入值			预期结果	测试覆盖项
	目的地	舱位	座位		
1	北京	头等舱	靠过道	预订成功	TCOVER1
2	北京	头等舱	靠窗	预订成功	TCOVER2
3	北京	公务舱	靠过道	预订成功	TCOVER3
4	北京	公务舱	靠窗	预订成功	TCOVER4
5	北京	经济舱	靠过道	预订成功	TCOVER5
6	北京	经济舱	靠窗	预订成功	TCOVER6
7	上海	头等舱	靠过道	预订成功	TCOVER7
8	上海	头等舱	靠窗	预订成功	TCOVER8

表 B.22（续）

测试用例	输入值			预期结果	测试覆盖项
	目的地	舱位	座位		
9	上海	公务舱	靠过道	预订成功	TCOVER9
10	上海	公务舱	靠窗	预订成功	TCOVER10
11	上海	经济舱	靠过道	预订成功	TCOVER11
12	上海	经济舱	靠窗	预订成功	TCOVER12
13	广州	头等舱	靠过道	预订成功	TCOVER13
14	广州	头等舱	靠窗	预订成功	TCOVER14
15	广州	公务舱	靠过道	预订成功	TCOVER15
16	广州	公务舱	靠窗	预订成功	TCOVER16
17	广州	经济舱	靠过道	预订成功	TCOVER17
18	广州	经济舱	靠窗	预订成功	TCOVER18

B.2.5.5.3 步骤 5：汇集测试集（TD5）

可以将所有测试用例分为两类，座位靠过道和座位靠窗。组合生成下面的测试集：

TS1：测试用例 1，3，5，7，9，11，13，15，17。

TS2：测试用例 2，4，6，8，10，12，14，16，18。

B.2.5.5.4 步骤 6：导出测试规程（TD6）

每个测试集都由不同的测试人员来执行，可以分为两个测试过程。

TP1：按照测试集中指定的顺序，覆盖 TS1 中的所有测试用例。

TP2：按照测试集中指定的顺序，覆盖 TS2 中的所有测试用例。

B.2.5.5.5 完全组合测试覆盖率

根据 6.2.5.1 中提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(完全组合)}=\frac{18}{18}\times 100\%=100\%$$

因此，对于完全组合测试，测试覆盖项的覆盖率达到了 100%。

B.2.5.6 成对测试

B.2.5.6.1 步骤 3：导出测试覆盖项（TD3）

在成对测试中，测试覆盖项是两个不同参数的不重复键值对。对于 travel_preference，可以确定下面的测试覆盖项：

TCOVER1： 北京，头等舱 （TCOND1，TCOND4）

TCOVER2： 北京，公务舱 （TCOND1，TCOND5）

TCOVER3： 北京，经济舱 （TCOND1，TCOND6）

TCOVER4： 上海，头等舱 （TCOND2，TCOND4）

TCOVER5： 上海，公务舱 （TCOND2，TCOND5）

TCOVER6： 上海，经济舱 （TCOND2，TCOND6）
TCOVER7： 广州，头等舱 （TCOND3，TCOND4）
TCOVER8： 广州，公务舱 （TCOND3，TCOND5）
TCOVER9： 广州，经济舱 （TCOND3，TCOND6）
TCOVER10： 北京，靠过道 （TCOND1，TCOND7）
TCOVER11： 北京，靠窗 （TCOND1，TCOND8）
TCOVER12： 上海，靠过道 （TCOND2，TCOND7）
TCOVER13： 上海，靠窗 （TCOND2，TCOND8）
TCOVER14： 广州，靠过道 （TCOND3，TCOND7）
TCOVER15： 广州，靠窗 （TCOND3，TCOND8）
TCOVER16： 头等舱，靠过道 （TCOND4，TCOND7）
TCOVER17： 头等舱，靠窗 （TCOND4，TCOND8）
TCOVER18： 公务舱，靠过道 （TCOND5，TCOND7）
TCOVER19： 公务舱，靠窗 （TCOND5，TCOND8）
TCOVER20： 经济舱，靠过道 （TCOND6，TCOND7）
TCOVER21： 经济舱，靠窗 （TCOND6，TCOND8）

B.2.5.6.2 步骤 4：导出测试用例（TD4）

通过选择当前测试用例中的一个或者多个不重复配对参数的键值对（测试覆盖项）来导出测试用例，测试用例其他的输入取任意有效值，确定预期结果，重复上述步骤直到所有两个不同参数的键值对都至少被一个测试用例包含。在本例中，所有的测试用例包含了 3 个（两个不同参数的）键值对，见表 B.23。

表 B.23 成对测试的测试用例

测试用例	输入值			预期结果	测试覆盖项
	目的地	舱位	座位		
1	北京	头等舱	靠过道	预订成功	TCOVER1，TCOVER10，TCOVER16
2	北京	公务舱	靠窗	预订成功	TCOVER2，TCOVER11，TCOVER19
3	北京	经济舱	靠过道	预订成功	TCOVER3，TCOVER10，TCOVER20
4	上海	头等舱	靠过道	预订成功	TCOVER4，TCOVER12，TCOVER16
5	上海	公务舱	靠窗	预订成功	TCOVER5，TCOVER13，TCOVER19
6	上海	经济舱	靠过道	预订成功	TCOVER6，TCOVER12，TCOVER20
7	广州	头等舱	靠窗	预订成功	TCOVER7，TCOVER15，TCOVER17
8	广州	公务舱	靠过道	预订成功	TCOVER8，TCOVER14，TCOVER18
9	广州	经济舱	靠窗	预订成功	TCOVER9，TCOVER15，TCOVER21

B.2.5.6.3 步骤 5：汇集测试集（TD5）

只有少量的测试用例，可以组合生成一个测试集，如下所示：
TS1：测试用例 1，2，3，4，5，6，7，8，9。

B.2.5.6.4 步骤 6：导出测试规程(TD6)

只有一个测试集，可以合成一个测试规程。

TP1：按照测试集中指定的顺序，覆盖 TS1 中的所有测试用例。

B.2.5.6.5 成对测试覆盖率

根据 6.2.5.2 中提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(成对测试)}=\frac{9}{9}\times 100\%=100\%$$

因此，对于成对测试，测试覆盖项的覆盖率达到了 100%。

B.2.5.7 单一选择测试

B.2.5.7.1 步骤 3：导出测试覆盖项(TD3)

在单一选择(或者 1-wise)测试中，测试覆盖项是键值对的集合。对于 travel_preference，可以确定下面的测试覆盖项：

TCOVER1： 目的地＝北京 (TCOND1)

TCOVER2： 目的地＝上海 (TCOND2)

TCOVER3： 目的地＝广州 (TCOND3)

TCOVER4： 舱位＝头等舱 (TCOND4)

TCOVER5： 舱位＝公务舱 (TCOND5)

TCOVER6： 舱位＝经济舱 (TCOND6)

TCOVER7： 座位＝靠过道 (TCOND7)

TCOVER8： 座位＝靠窗 (TCOND8)

B.2.5.7.2 步骤 4：导出测试用例(TD4)

通过选择当前测试用例中的一个或者多个不重复的键值对(测试覆盖项)来导出测试用例，测试用例其他的输入变量取确定任意有效值，确定预期结果，重复上述步骤直到所有键值对都至少被一个测试用例包含。在本例中，只需要表 B.24 所示的 3 个测试用例。

表 B.24 单一选择测试的测试用例

测试用例	输入值			预期结果	测试覆盖项
	目的地	舱位	座位		
1	北京	头等舱	靠过道	预订成功	TCOVER1，TCOVER4，TCOVER7
2	上海	公务舱	靠窗	预订成功	TCOVER2，TCOVER5，TCOVER8
3	广州	经济舱	靠过道	预订成功	TCOVER3，TCOVER6，TCOVER7

注：导出其他测试用例也可以达到要求的测试覆盖率。

B.2.5.7.3 步骤 5：汇集测试集(TD5)

本例中只有少量的测试用例，可以组合生成一个测试集，如下所示：

TS1：测试用例 1，2，3。

B.2.5.7.4 步骤 6:导出测试规程(TD6)

只有一个测试集,可以合成一个测试过程。

TP1:按照测试集中指定的顺序,覆盖 TS1 中的所有测试用例。

B.2.5.7.5 单一选择测试覆盖率

根据 6.2.5.3 中提供的公式和上述导出的测试覆盖项:

$$覆盖率_{(单一选择)}=\frac{8}{8}\times 100\%=100\%$$

因此,对于单一选择测试,测试覆盖项的覆盖率达到了 100%。

B.2.5.8 基本选择测试

B.2.5.8.1 步骤 3:导出测试覆盖项(TD3)

对于基本选择测试,测试覆盖项是通过选择每个参数的"基本选择"来确定的。基本选择可以从操作手册、用例的基本路径或者等价类划分中得到的测试覆盖项中挑选。在本例中,操作手册中说明了基本选择应当选择下面的输入值:

TCOVER1:目的地=上海,舱位=经济舱,座位=靠窗 (TCOND2,6,8)

剩余的测试覆盖项是通过识别所有剩余的键值对来导出的:

TCOVER2:目的地=北京,舱位=经济舱,座位=靠窗 (TCOND1,6,8)

TCOVER3:目的地=广州,舱位=经济舱,座位=靠窗 (TCOND3,6,8)

TCOVER4:目的地=上海,舱位=头等舱,座位=靠窗 (TCOND2,4,8)

TCOVER5:目的地=上海,舱位=公务舱,座位=靠窗 (TCOND2,5,8)

TCOVER6:目的地=上海,舱位=经济舱,座位=靠过道 (TCOND2,6,7)

B.2.5.8.2 步骤 4:导出测试用例(TD4)

通过组合测试项导出一个基本选择测试用例:

基本选择: 上海,经济舱,靠窗

这是表 B.25 中的第一个测试用例。通过替代基本测试用例中的每个键值对导出剩下测试用例的,重复上述步骤直到覆盖所有的键值对,执行的测试用例见表 B.25。

表 B.25 基本选择测试的测试用例

测试用例	输入值			预期结果	测试覆盖项
	目的地	舱位	座位		
1	上海	经济舱	靠窗	预订成功	TCOVER1
2	北京	经济舱	靠窗	预订成功	TCOVER2
3	广州	经济舱	靠窗	预订成功	TCOVER3
4	上海	头等舱	靠窗	预订成功	TCOVER4
5	上海	公务舱	靠窗	预订成功	TCOVER5
6	上海	经济舱	靠过道	预订成功	TCOVER6

B.2.5.8.3 步骤 5:汇集测试集(TD5)

本例中只有少量的测试用例,可以组合生成一个测试集,如下所示:

TS1:测试用例 1,2,3,4,5,6。

B.2.5.8.4 步骤 6:导出测试规程(TD6)

所有的测试用例都在一个测试集中,可以导出一个测试规程。

TP1:按照测试集中指定的顺序,覆盖 TS1 中的所有测试用例。

B.2.5.8.5 基本选择测试覆盖率

根据 6.2.5.4 中提供的公式和上述导出的测试覆盖项:

$$覆盖率_{(基本选择)}=\frac{6}{6}\times 100\%=100\%$$

因此,对于基本选择测试,测试覆盖项的覆盖率达到了 100%。

B.2.6 判定表测试

B.2.6.1 介绍

判定表测试的目的是根据所选择的条件和动作覆盖级别导出测试用例集,以覆盖输入和输出(即一系列条件和动作)的逻辑关系,输入和输出与判定规则相关。

B.2.6.2 规格说明

以支票借记功能为例,输入是借记金额、账户类型和当前余额,输出是新的余额和操作代码。账户类型是邮政("p")类型或柜台("c")类型。操作代码是"D&L""D""S&L"或"L",分别对应"处理借记并发送信件""只处理借记""冻结账户和发送信件"和"只发送信件"。该功能的测试依据如下:

如果账户中有足够的金额或者新的余额在授权透支的范围内,则处理借记。如果新的余额超过了授权透支的范围,则不处理借记,如果是一个邮政账户则进行冻结。邮政账户的所有交易都会发送信件,非邮政账户如果有足够的资金也会发送信件(即账户将不再是信贷)。

B.2.6.3 步骤 1:识别特征集(TD1)

在测试依据信息中只有一个测试项,只需要定义一个特征集:

FS1:支票借记功能

B.2.6.4 步骤 2:导出测试条件(TD2)

测试条件是测试依据中得到的条件和动作。

条件(C)是:

TCOND1(C1):账户中的新余额 (FS1)

TCOND2(C2):授权范围内的新的透支余额 (FS1)

TCOND3(C3):账户是邮局类型 (FS1)

动作(A)是:

TCOND4(A1):处理借记 (FS1)

TCOND5(A2):冻结账户 (FS1)

TCOND6(A3):发送信件 (FS1)

B.2.6.5 步骤 3：导出测试覆盖项(TD3)

判定表将测试覆盖项作为判定表的判定规则。判定表的每一列是一个判定规则。判定表也可以用行的形式显示判定规则而不是列。该表包含了两个部分：在第一部分中，每个判定规则对应多个条件。“T”表示对于使用的判定规则来说条件必须是真的；“F”表示对于使用的判定规则来说条件必须是假的。第二部分中，每个判定规则对应多个动作。“T”表示动作将被执行；“F”表示动作不会被执行；星号(*)表示条件的组合是无效的，因此该判定规则没有对应的动作。如果两个或多个列都包含了一个不会影响结果的布尔条件，则可以进行合并。本例对应表 B.26，有 8 条判定规则，其中 6 条是有效的，因此得到 6 个测试覆盖项：

表 B.26 支票借记功能的判定表(一)

判定规则	1	2	3	4	5	6	7	8
C1：账户中的新余额	F	F	F	F	T	T	T	T
C2：授权范围内的新的透支余额	F	F	T	T	F	F	T	T
C3：账户是邮局类型	F	T	F	T	F	T	F	T
A1：处理借记	F	F	T	T	T	T	*	*
A2：冻结账户	F	T	F	F	F	F	*	*
A3：发送信件	T	T	T	T	F	T	*	*

注 1：在表 B.26 中使用了“T”和“F”来表示“True”和“False”，也可以用其他符号表示(例如用“true”和“false”代替)。

注 2：在表 B.26 中，条件和动作都是二元(T 或 F)条件，形成了一个“有限条目”的判定表。在“扩展条目”的判定表中，条件和/或动作可以设多个值。

B.2.6.6 步骤 4：导出测试用例(TD4)

从判定表中一次选择一个或者多个有效的判定规则来导出测试用例，这些规则未被测试用例覆盖，确定测试判定规则的条件和动作的输入值，测试用例其他的输入变量取任意有效值，确定预期结果，重复上述步骤直到达到覆盖要求。表 B.27 的测试用例要求达到 100%的判定表覆盖率，对应表 B.26 中的判定规则(判定规则 7 和 8 是无效的，因此不导出对应的测试用例)。

表 B.27 支票借记功能的测试用例表(一)

测试用例	原因/输入				影响/结果		测试覆盖项
	账户类型	透支范围	当前余额	借记金额	新余额	操作代码	
1	“c”	¥100	−¥70	¥50	−¥70	“L”	1
2	“p”	¥1 500	¥420	¥2 000	¥420	“S&L”	2
3	“c”	¥250	¥650	¥800	−¥150	“D&L”	3
4	“p”	¥750	−¥500	¥200	−¥700	“D&L”	4
5	“c”	¥1 000	¥2 100	¥1 200	¥900	“D”	5
6	“p”	¥500	¥250	¥150	¥100	“D&L”	6

B.2.6.7 步骤 5：汇集测试集(TD5)

只需要 6 个测试用例就覆盖了所有的判定规则，可以手动测试所有测试用例，组合生成一个测试集。

TS1：测试用例 1，2，3，4，5，6。

B.2.6.8 步骤 6：导出测试规程（TD6）

所有的测试用例都在一个测试集中，可以导出一个测试规程。

TP1：按照测试集中指定的顺序，覆盖 TS1 中的所有测试用例。

B.2.6.9 判定表测试覆盖率

根据 6.2.6 中提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(判定表测试)}=\frac{6}{6}\times 100\%=100\%$$

因此，对于判定表测试，测试覆盖项的覆盖率达到了 100%。

B.2.7 因果图

B.2.7.1 介绍

因果图的目的是根据要求的覆盖程度导出测试用例集，以覆盖原因（输入）和结果（输出）的逻辑关系。该技术利用了一种约定来表示测试的因果图，该图显示了原因和结果之间的关系以及它们之间的显式约束。和判定表测试不同，判定表的约束没有显式地声明。因此，该技术只是对模型覆盖测试项的测试依据的一种扩展。

B.2.7.2 规格说明

以支票借记功能为例，输入是借记金额、账户类型和当前余额，输出是新的余额和操作代码。账户类型是邮政（“p”）类型或柜台（“c”）类型。操作代码是“D&L”“D”“S&L”或“L”，分别对应“处理借记并发送信件”“只处理借记”“冻结账户和发送信件”和“只发送信件”。该功能的基础信息如下：

如果账户中有足够的金额或者新的余额在授权透支的范围内，则处理借记。如果新的余额超过了授权透支的范围，则不处理借记，如果是一个邮政账户则进行冻结。邮政账户的所有交易都会发送信件，非邮政账户如果有足够的资金也会发送信件（即账户将不再是信贷）。

B.2.7.3 步骤 1：识别特征集（TD1）

在测试依据中只有一个测试项，只需要定义一个特征集：

FS1：支票借记功能

B.2.7.4 步骤 2：导出测试条件（TD2）

测试条件是测试依据中得到的原因和结果。

原因是：

TCOND1（C1）：账户中的新余额 （FS1）

TCOND2（C2）：授权范围内的新的透支余额 （FS1）

TCOND3（C3）：账户是邮政类型 （FS1）

结果是：

TCOND4（A1）：处理借记 （FS1）

TCOND5（A2）：冻结账户 （FS1）

TCOND6（A3）：发送信件 （FS1）

因果图用类似于设计者使用的硬件逻辑电路符号来表示原因和结果之间的关系。对测试依据建模如图 B.10 所示。

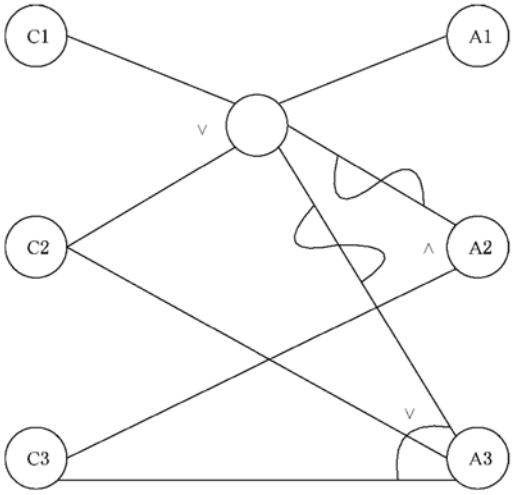


图 B.10　支票借记功能因果图

注 1：连接 C1/C2 到 A1/A2/A3 的“空”节点是一个连接节点，在图中用于连接两个或者多个原因。

因果图中原因和结果之间的关系符号表示见图 B.11。

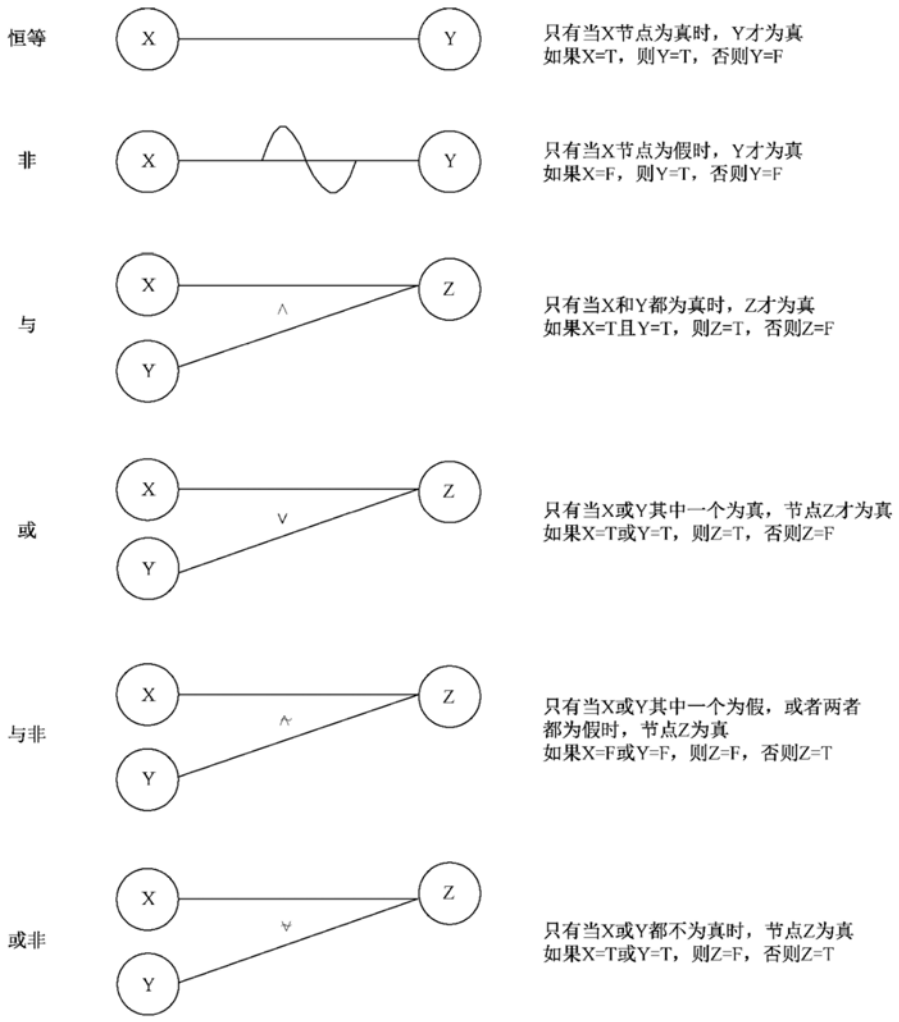


图 B.11　因果图中原因和结果之间的关系符号表示

注 2：在本例中不需要“约束”符号，但仍在本条中进行说明，“约束”符号在表示原因和结果之间的要求、允许和禁止关系上有优势。约束关系在判定表中没有进行显式声明。这些符号是一种检验因果图、判定表和测试用例（从因果图中产生）完整性的方法。

因果图中原因和结果的约束符号表示见图 B.12。

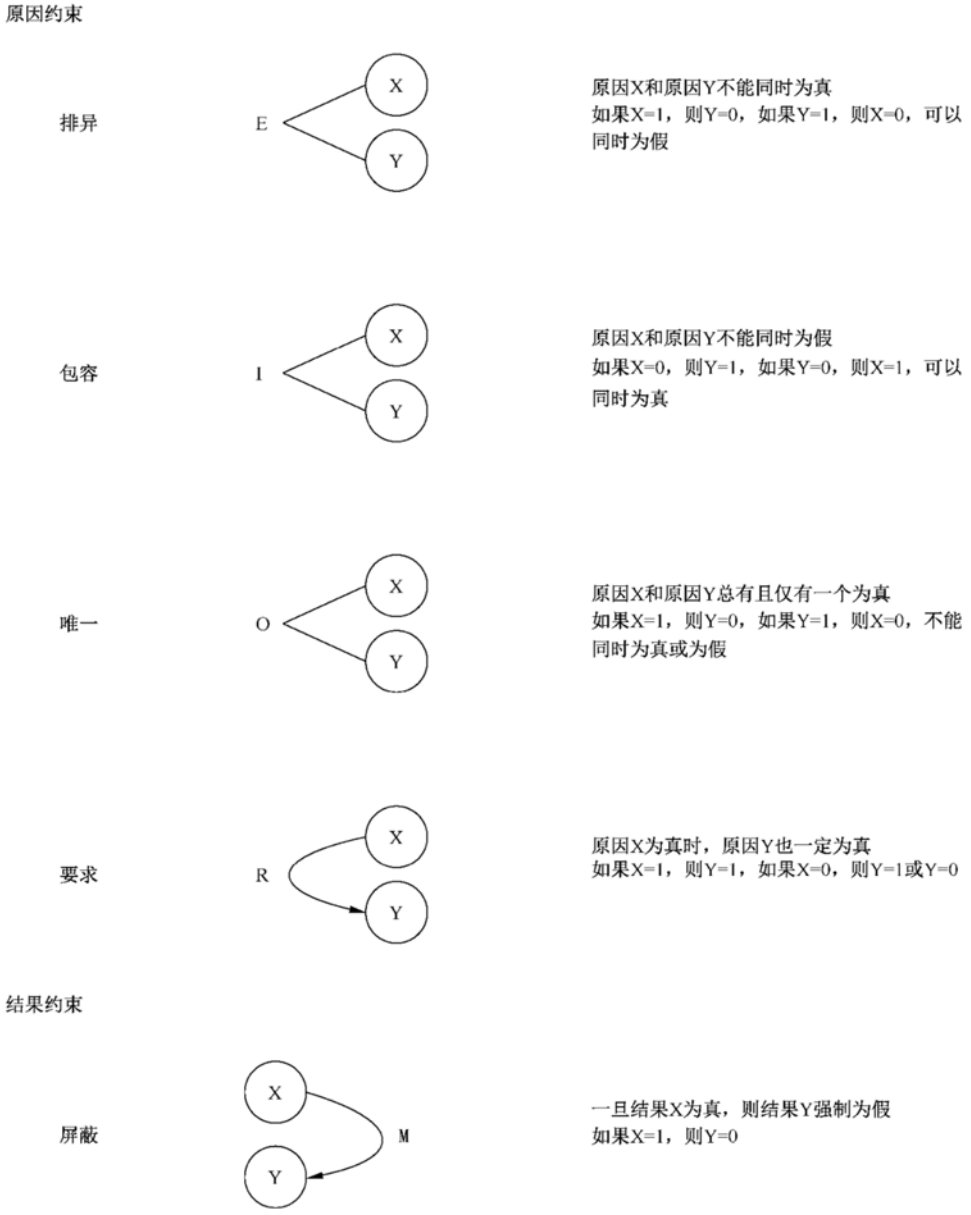


图 B.12　因果图中原因和结果的约束符号表示

B.2.7.5　步骤 3：导出测试覆盖项（TD3）

因果图也可用判定表的方式表示，用以帮助识别测试覆盖项（即判定表中判定规则）。判定表的每一列是一个判定规则。该表包含了两个部分。在第一个部分中，每个判定规则对应多个条件。“T”表示对于使用的判定规则来说条件必须是真的；“F”表示对于使用的判定规则来说条件必须是假的。第二部分中，每个判定规则对应多个动作。“T”表示动作将被执行；“F”表示动作不会被执行；星号（*）表示条件的组合是无效的，因此该判定规则没有对应的动作。本例对应判定表表 B.28，包含了 6 个测试

覆盖项类(判定规则 7 和 8 无效,因此不是测试覆盖项)。

表 B.28　支票借记功能的判定表(二)

判定规则	1	2	3	4	5	6	7	8
C1:信贷中的新余额	F	F	F	F	T	T	T	T
C2:授权范围内的新的透支余额	F	F	T	T	F	F	T	T
C3:账户是邮局类型	F	T	F	T	F	T	F	T
A1:处理借记	F	F	T	T	T	T	*	*
A2:冻结账户	F	T	F	F	F	F	*	*
A3:发送信件	T	T	T	T	F	T	*	*

B.2.7.6　步骤 4:导出测试用例(TD4)

从判定表中一次选择一个或者多个有效的判定规则来导出测试用例,这些规则未被测试用例覆盖,确定测试判定规则的条件和动作的输入值,测试用例其他的输入变量取任意有效值,确定预期结果,重复上述步骤直到覆盖所有有效的判定规则。表 B.29 的测试用例要求达到 100%的原因-结果覆盖率,对应表 B.28 中的判定规则(判定规则 7 和 8 是无效的,因此不导出对应的测试用例)。

表 B.29　支票借记功能的测试用例表(二)

测试用例	原因/输入				影响/结果		测试覆盖项
	账户类型	透支范围	当前余额	借记金额	新余额	操作编码	
1	“c”	¥100	−¥70	¥50	−¥70	“L”	1
2	“p”	¥1 500	¥420	¥2 000	¥420	“S&L”	2
3	“c”	¥250	¥650	¥800	−¥150	“D&L”	3
4	“p”	¥750	−¥500	¥200	−¥700	“D&L”	4
5	“c”	¥1 000	¥2 100	¥1 200	¥900	“D”	5
6	“p”	¥500	¥250	¥150	¥100	“D&L”	6

B.2.7.7　步骤五:汇集测试集(TD5)

只需要 6 个测试用例就覆盖了所有的判定规则,可以手动测试所有测试用例,组合生成一个测试集。

TS1:测试用例 1,2,3,4,5,6。

B.2.7.8　步骤 6:导出测试规程(TD6)

所有的测试用例都在一个测试集中,可以导出一个测试规程。

TP1:覆盖 TS1 中的所有测试用例,以测试集指定的顺序。

B.2.7.9　因果图覆盖率

根据 6.2.7 中提供的公式和上述导出的测试覆盖项:

$$覆盖率_{(因果图)}=\frac{6}{6}\times 100\%=100\%$$

因此，对于因果图，测试覆盖项的覆盖率达到了 100%。

B.2.8 状态转移测试

B.2.8.1 介绍

状态转移测试的目的是根据要求的覆盖程度导出测试用例，以覆盖测试项的转移和/或状态。这项技术基于分析测试项的测试依据，通过状态转移建模测试项的行为。

B.2.8.2 规格说明

考虑测试项——manage_display_changes，下面是其测试依据：

测试项响应输入请求以切换一个时间显示装置的外部显示模式。外部显示模式可以设置为 4 个值中的其中一个：两个对应时间或者日期的显示，其他两个对应改变时间或者日期时使用的模式。

4 种可能的输入请求是："改变模式""重置""设置时间"和"设置日期"。输入请求是"改变模式"会使"显示时间"和"显示日期"两种显示模式相互转变。如果显示模式设置为"显示时间"或者"显示日期"，则输入请求是"重置"会让显示模式对应地变为"改变时间"或者"改变日期"模式。输入请求是"设置时间"会使显示模式从"改变时间"变为"显示时间"，类似的，输入请求是"设置日期"会使显示模式从"改变日期"变为"显示日期"

B.2.8.3 步骤 1：识别特征集

在测试依据信息中只有一个测试项，只需要定义一个特征集：

FS1：manage_display_changes

B.2.8.4 步骤 2：导出测试条件(TD2)

导出一个状态模型对应测试条件。状态转移图通常用作状态模型，具体的符号表示在下面进行说明。状态转移图由状态、转移、时间和动作(见图 B.13)组成。事件是由输入引起的。类似的，动作会导致输出。动作的输出对识别测试项当前的状态是必要的。转移是由当前的状态和事件决定的，通常由事件和动作标记。如 4.2.8.1 所述，在状态转移测试中，测试条件可能是状态模型的所有状态、状态模型的所有转移或者整个状态模型，具体取决于测试的覆盖要求。

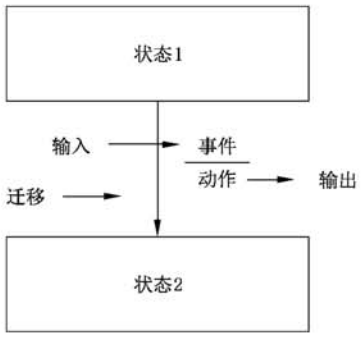


图 B.13 一般状态模型

测试项 manage_display_changes 的状态转移图如图 B.14 所示(本例中该图是测试条件 TCOND1)。

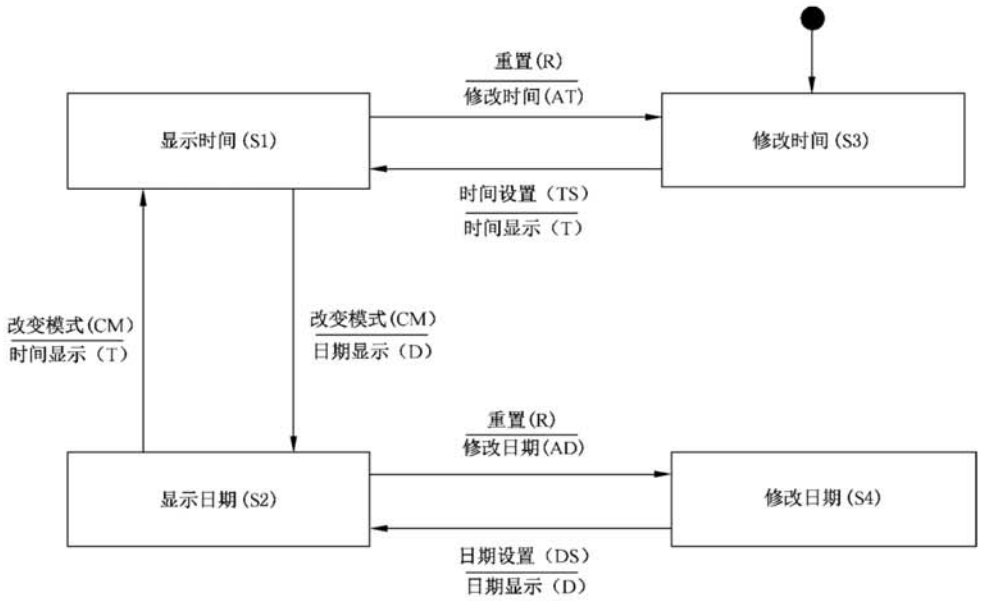


图 B.14　manage_display_change 状态转移图

B.2.8.5　步骤 3：导出测试覆盖项(TD3)

假设要求覆盖“全转移”，可以画一个状态表表示所有有效和无效的转移(所要求的测试覆盖项)。为了达到完全的单步转移覆盖，只需要测试有效的转移。

单步转移覆盖的限制在于测试只用于测试测试项中的有效转移。测试项更完整的测试会导致发生无效转移(“全转移”)。STD 只显示有效转移(没有显示被认为是无效的)。既显示有效转移又显示无效转移的状态模型的一种形式是状态表，另一种形式是状态转移图，包括所有以无效转移结束的“异常”状态。状态表使用的符号如表 B.30 所示。

表 B.30　状态表符号

状态	输入 1	输入 2	…
开始状态 1	进入 A	进入 B	…
开始状态 2	进入 C	进入 D	…
…	…	…	…

其中进入 X＝最终状态/开始状态和输入的输出或动作。

manage_display_change 的状态表如表 B.31 所示。

表 B.31　manage_display_change 的状态表

状态	CM	R	TS	DS
S1	S2/D	S3/AT	S1/—	S1/—
S2	S1/T	S4/AD	S2/—	S2/—
S3	S3/—	S3/—	S1/T	S3/—
S4	S4/—	S4/—	S4/—	S2/D

状态保持不变,动作为空(—)的任意输入表示是一个空的转移,如果实际产生了任何转移都是失败的。只是想达到对有效测试覆盖项的覆盖的测试集,会忽略测试这些空的转移。因此,需要一个更加完整的测试集("全转移")既会测试可能的转移,也会测试空的转移,即测试测试项在所有状态下对测试依据中给出的所有输入的响应。状态表提供了一种直接导出测试覆盖项的理想方式,可以覆盖空的转移(覆盖"全转移")。

在表 B.31 中的 16 个条目代表在四种状态下发生四种可能的输入的情况。为了覆盖"全转移",导出了 16 个测试覆盖项,这些覆盖项可以列在表 B.32 中。

表 B.32　状态表至测试用例表的映射

状态	CM	R	TS	DS
S1	S2/D (TCOVER1)	S3/AT (TCOVER2)	S1/— (TCOVER3)	S1/—(TCOVER4)
S2	S1/T (TCOVER5)	S4/AD(TCOVER6)	S2/—(TCOVER7)	S2/—(TCOVER8)
S3	S3/—(TCOVER9)	S3/—(TCOVER10)	S1/T(TCOVER11)	S3/—(TCOVER12)
S4	S4/—(TCOVER13)	S4/—(TCOVER14)	S4/—(TCOVER15)	S2/D(TCOVER16)

因此,从上面的覆盖"全转移"的状态表中得到(有效和无效)测试覆盖项如下所示:

TCOVER1:　输入 CM,S1 转移到 S2　　(FS1,有效转移)
TCOVER2:　输入 R,S1 转移到 S3　　(FS1,有效转移)
TCOVER3:　输入 TS,S1 转移到 S1　　(FS1,无效转移)
TCOVER4:　输入 DS,S1 转移到 S1　　(FS1,无效转移)
TCOVER5:　输入 CM,S2 转移到 S1　　(FS1,有效转移)
TCOVER6:　输入 R,S2 转移到 S4　　(FS1,有效转移)
TCOVER7:　输入 TS,S2 转移到 S2　　(FS1,无效转移)
TCOVER8:　输入 DS,S2 转移到 S2　　(FS1,无效转移)
TCOVER9:　输入 CM,S3 转移到 S3　　(FS1,无效转移)
TCOVER10:　输入 R,S3 转移到 S3　　(FS1,无效转移)
TCOVER11:　输入 TS,S3 转移到 S1　　(FS1,有效转移)
TCOVER12:　输入 DS,S3 转移到 S3　　(FS1,无效转移)
TCOVER13:　输入 CM,S4 转移到 S4　　(FS1,无效转移)
TCOVER14:　输入 R,S4 转移到 S4　　(FS1,无效转移)
TCOVER15:　输入 TS,S4 转移到 S4　　(FS1,无效转移)
TCOVER16:　输入 DS,S4 转移到 S2　　(FS1,有效转移)

B.2.8.6　步骤 4:导出有效测试用例(TD4)

B.2.8.6.1　选项

现在可以导出检验每个可能的转移的测试用例。可以从状态转移图中获得输入,用于检验每个测试用例中覆盖的转移,同时可以从状态转移图的预期结果和转移的结束状态获得预期结果。导出的测试用例中,每个测试用例可以覆盖 1 到 n 转移,其中 n 是可能转移的最大数量。例如,根据单步转移或者 2 步转移覆盖可以导出测试用例(实际上同时导出单步转移和 2 步转移测试用例是没有必要的,这将在下面进行简单的证明)。测试用例的导出也可以是覆盖无效转移。这三种情况将在下面进行证明。

B.2.8.6.2 步骤 4a：导出单步转移测试用例（有效转移）

表 B.33 的 6 个测试用例提供了单步转移测试覆盖项。为选定的每个转移导出一个测试用例，从 STD 识别输入、预期结果和结束状态，直到所有的转移都被测试用例覆盖。

表 B.33 manage_display_change 的单步转移测试用例

测试用例	1	2	3	4	5	6
开始状态	S1	S1	S2	S2	S3	S4
输入	CM	R	CM	R	TS	DS
预期结果	D	AT	T	AD	T	D
结束状态	S2	S3	S1	S4	S1	S2
测试覆盖项	1	2	5	6	11	16

注：可以针对表 B.33 中 6 个用例编制一个测试过程，使得这些用例按照如下的方式执行：每一个用例的最终状态是另一个用例的开始状态，（比如执行顺序为 5，1，4，6，3，2）。这将在步骤 5 中进行详细说明。

图中显示测试用例 1 的开始状态是“显示时间（S1）”，输出是“改变模式（CM）”，预期结果是“显示日期”（D），结束状态是“显示日期（S2）”。

这 6 个测试用例检验了“有效”的转移，达到了单步转移覆盖。达到该覆盖程度的测试具有有限的能力去发现一些错误的类型，因为尽管可以发现最明显的错误转移和输出，但不能发现那些只能通过执行转移序列才能发现的更深层错误。

B.2.8.6.3 步骤 4b：导出无效转移的测试用例

覆盖无效转移的测试用例如表 B.34 所示，其中“—”表示空的转移。

表 B.34 manage_display_change 的无效测试用例

测试用例	7	8	9	10	11	12	13	14	15	16
开始状态	S1	S1	S2	S2	S3	S3	S3	S4	S4	S4
输入	TS	DS	TS	DS	CM	R	DS	CM	R	TS
预期结果	—	—	—	—	—	—	—	—	—	—
结束状态	S1	S1	S2	S2	S3	S3	S3	S4	S4	S4
测试覆盖项	3	4	7	8	9	10	12	13	14	15

如上表所示，覆盖无效测试项的测试用例不会触发从开始状态的转移。组合上面两个表中的测试用例可以达到“全转移”覆盖。

B.2.8.6.4 步骤 4c：导出测试覆盖项——2 步转移测试（TD3）

为了达到 2 步转移覆盖，从 STD 中导出下面的测试覆盖项：

TCOVER17： 输入 CM 和 CM，状态从 S1 转移到 S2，S2 转移到 S1 （FS1）

TCOVER18： 输入 CM 和 R，状态从 S1 转移到 S2，S2 转移到 S4 （FS1）

TCOVER19： 输入 R 和 TS，状态从 S1 转移到 S3，S3 转移到 S1 （FS1）

TCOVER20： 输入 TS 和 CM，状态从 S3 转移到 S1，S1 转移到 S2 （FS1）

TCOVER21： 输入 TS 和 R，状态从 S3 转移到 S1，S1 转移到 S3 （FS1）

TCOVER22： 输入 CM 和 CM，状态从 S2 转移到 S1，S1 转移到 S2 （FS1）

TCOVER23： 输入 CM 和 R，状态从 S2 转移到 S1，S1 转移到 S3 （FS1）
TCOVER24： 输入 R 和 DS，状态从 S2 转移到 S4，S4 转移到 S2 （FS1）
TCOVER25： 输入 DS 和 CM，状态从 S4 转移到 S2，S2 转移到 S1 （FS1）
TCOVER26： 输入 DS 和 R，状态从 S4 转移到 S2，S2 转移到 S4 （FS1）

B.2.8.6.5 步骤 4d：导出 2 步转移测试用例（TD4）

如果步骤 TD3 中选择的测试覆盖项是覆盖所有 2 步转移，则设计测试用例来检验所有转移可能的顺序对。本例中，有 10 个测试用例，如表 B.35 所示。

表 B.35 manage_display_change 的 2 步转移测试用例

测试用例	17	18	19	20	21	22	23	24	25	26
开始状态	S1	S1	S1	S3	S3	S2	S2	S2	S4	S4
输入	CM	CM	R	TS	TS	CM	CM	R	DS	DS
预期结果	D	D	AT	T	T	T	T	AD	D	D
下一个状态	S2	S2	S3	S1	S1	S1	S1	S4	S2	S2
输入	CM	R	TS	CM	R	CM	R	DS	CM	R
预期结果	T	AD	T	D	AT	D	AT	D	T	AD
结束状态	S1	S4	S1	S2	S3	S2	S3	S2	S1	S4
测试覆盖项	17	18	19	20	21	22	23	24	25	26

上表说明测试用例 17 包含了两个转移。对于第一个转移，开始状态是显示时间（S1），第一次的输入是“改变模式（CM）”，中间的预期结果是显示日期（D），下一个状态是显示日期（S2）。对于第二个转移，第二次的输入是“改变模式（CM）”，最终的预期结果是显示时间（T），结束状态是显示时间（S1）。注意中间状态和每个转移的输入输出是明确定义的。

较长序列的测试转移可以达到更高程度的转移覆盖，应视要求的测试完整性程度而定。

B.2.8.7 步骤 5：汇集测试集（TD5）

可以将所有单步转移的测试用例（覆盖有效转移）组合成一个测试集，将覆盖全部无效转移的“全转移”测试用例组合成第二个测试集，所有 2 步转移的测试用例组成第三个测试集，如下所示：

TS1：单步转移的测试用例 ——测试用例 1，2，3，4，5，6。
TS2：全转移的测试用例 ——测试用例 7，8，9，10，11，12，13，14，15，16。
TS3：2 步转移的测试用例 ——测试用例 17，18，19，20，21，22，23，24，25，26。

注 1：在某些情况下，可以对单个测试用例进行排序，使得一个测试用例的“结束状态”是下一测试用例的“开始状态”。这可以提升测试执行的效率。该做法的可能性依赖于待测试的特定状态模型。在上面的例子中，下面的测试用例执行顺序可以达到此目标：
TS1：单步转移的测试用例—测试用例 5，1，4，6，3，2。

注 2：因为上面定义的 2 步转移的测试用例覆盖了单步转移的测试用例中的所有路径，不需要为单步转移的测试用例定义测试集和测试过程。但为了完整性，这个示例包含了单步转移的内容。

B.2.8.8 步骤 6：导出测试规程（TD6）

导出一个测试规程，根据步骤 5 中定义的顺序执行所有测试用例：
TP1：以测试集指定的顺序覆盖 TS1、TS2 和 TS3 中所有的测试用例。

B.2.8.9 状态转移测试覆盖率

根据 6.2.8 提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(单步转移覆盖)}=\frac{6}{6}\times 100\%=100\%$$

$$覆盖率_{(全转移覆盖)}=\frac{16}{16}\times 100\%=100\%$$

$$覆盖率_{(2步转移覆盖)}=\frac{10}{10}\times 100\%=100\%$$

因此，对于单步转移测试、全转移测试和 2 步转移，测试覆盖项的覆盖率都达到了 100%。

B.2.9 场景测试

B.2.9.1 介绍

场景测试的目的是根据选择的覆盖程度导出覆盖测试项场景的测试用例。场景测试是以测试依据的分析为基础，以动作序列的形式产生测试项行为的模型，该动作序列组成了贯穿测试项的工作流。

下面说明了两种类型的场景测试。第一个例子是基于场景测试的一种通用模式，第二个是基于用例的具体例子。

B.2.9.2 例 1

B.2.9.2.1 规格说明

考虑测试项 withdraw_cash，是自动取款机(ATM)系统的一部分，测试依据如下所示：

withdraw_cash 功能允许有银行账户的客户通过 ATM 从他们的账户取款。取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台可工作的 ATM。当取款完成后，账户余额扣除取款金额，打印取款回执条，ATM 可用于为下一位用户服务。

根据用户的需求定义下面的场景：

基本场景

——成功从账户取款

可选场景

不支持取款，因为：

——银行卡不能被 ATM 识别，被拒绝；

——用户输入密码错误不多于 2 次；

——用户输入密码错误 3 次，ATM 吞卡；

——用户选择存款或者转账，不选择取款；

——用户选择了错误账户，此账户在插入的卡中不存在；

——用户输入的取款金额是无效的；

——ATM 中现金不足；

——用户输入不符合面额的取款金额；

——用户输入的取款金额是超过了每日最大取款金额；

——用户银行账户中的金额不足。

注：实际上，也可能存在代表其他情况的场景，例如用户在取款过程中的某个点上选择取消。

B.2.9.2.2 步骤 1：识别特征集(TD1)

在测试依据中只有一个测试项，只需要定义一个特征集：

FS1：withdraw_cash

B.2.9.2.3 步骤 2:导出测试条件(TD2)

为了识别测试条件,需要生成测试项的模型以识别每个场景中的场景(和场景中的活动)。图 B.15 例子中的模型是一个事件流模型。在这些符号中,“主”路径用粗黑线表示,工作流的开始和结束点进行了标记,每个动作用唯一的标识符来表示,代表用户(U)或系统(S)(即测试项)的动作。

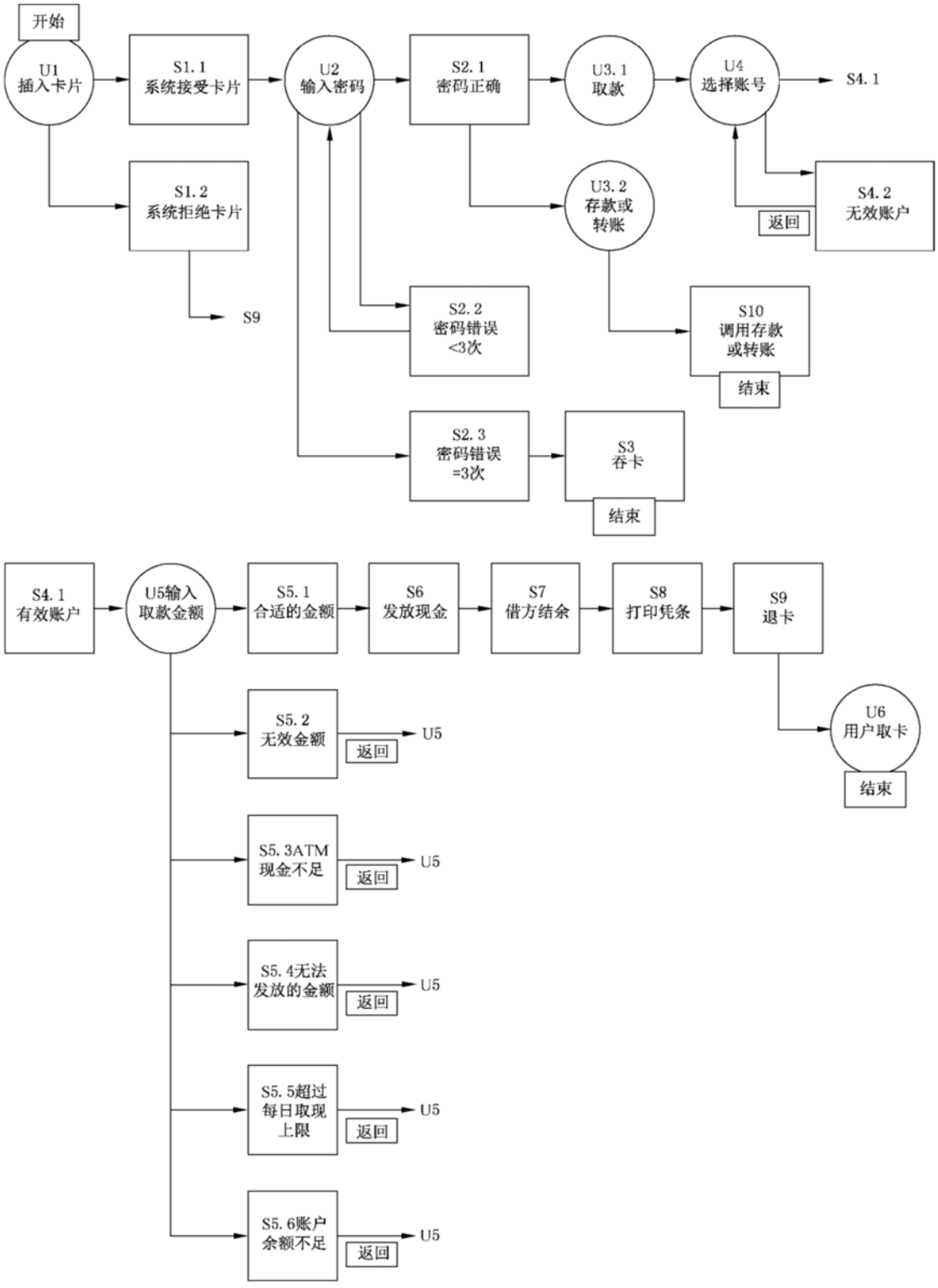


图 B.15 取款功能场景流程图

在场景测试中，测试条件是需要在测试中覆盖的基本场景和可选场景（即用户和系统交互的事件流用序列组成一个场景）。在规格说明中描述了 11 个场景，包括 1 个基本场景和 10 个可选场景。这些场景即测试条件（覆盖 FS1），如下所示：

TCOND1：取款成功	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.1，S6，S7，S8，S9，U6）
TCOND2：银行卡不能被 ATM 识别	（覆盖 U1，S1.2，S9，U6）
TCOND3：用户输入密码错误次数小于 3 次	（覆盖 U1，S1.1，U2，S2.2）
TCOND4：用户输入密码错误次数等于 3 次	（覆盖U1，S1.1，U2，S2.2，U2，S2.2，U2，S2.3，S3）
TCOND5：用户选择存款或者转账	（覆盖 U1，S1.1，U2，S2.1，U3.2，S10）
TCOND6：用户选择错误账户	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.2）
TCOND7：用户输入无效的取款金额	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.2）
TCOND8：ATM 中现金不足	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.3）
TCOND9：用户输入不符合面额的取款金额	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.4）
TCOND10：用户输入的取款金额超过了每日最大的金额	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.5）
TCOND11：用户银行账户中的金额不足	（覆盖 U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.6）

B.2.9.2.4 步骤 3：导出测试覆盖项（TD3）

在场景测试中，测试覆盖项和测试条件一样，都是基本场景和可选场景。

TCOVER1＝TCOND1	TCOVER7＝TCOND7
TCOVER2＝TCOND2	TCOVER8＝TCOND8
TCOVER3＝TCOND3	TCOVER9＝TCOND9
TCOVER4＝TCOND4	TCOVER10＝TCOND10
TCOVER5＝TCOND5	TCOVER11＝TCOND11
TCOVER6＝TCOND6	

B.2.9.2.5 步骤 4：导出测试用例（TD4）

选定要覆盖的一个场景来导出测试用例，确定测试用例的输入场景，确定测试的预期结果，重复上述步骤直到覆盖所有要求的场景。测试用例的步骤通常用自然语言的格式说明。如果假设所确定每一个测试覆盖项都需要一个测试用例，可以导出表 B.36～表 B.46 的测试用例。

注 1：对于每个测试用例，用于填充每个输入字段可以有很多种变化。可以用等价类划分生成填充每个输入字段的一组值。

表 B.36　场景测试的测试用例(一)

测试用例	1
测试用例名称	取款成功
检验的场景路径	U1,S1.1,U2,S2.1,U3.1,U4,S4.1,U5,S5.1,S6,S7,S8,S9,U6
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——¥50 000
	用户账户余额——¥100
	取款金额——¥50
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	成功从客户账户取款
	ATM 余额为¥49 950
	客户账户余额为¥50
	ATM 是激活的、可操作的,等待插入一张客户银行卡
测试覆盖项	TCOVER1

表 B.37　场景测试的测试用例(二)

测试用例	2
测试用例名称	银行卡不能被 ATM 识别
检验的场景路径	U1,S1.2,S9,U6
输入	卡无效
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	银行卡被 ATM 拒绝,错误提示卡无效。ATM 等待客户取走无效卡
测试覆盖项	TCOVER2

表 B.38　场景测试的测试用例(三)

测试用例	3
测试用例名称	用户输入错误密码次数小于 3 次
检验的场景路径	U1,S1.1,U2,S2.2
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	输入两次错误密码——假设 0000 是错误的且和银行卡不匹配
	ATM 余额——¥100
	用户账户余额——¥500
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	密码被 ATM 拒绝,系统提示用户重新输入密码
测试覆盖项	TCOVER3

表 B.39　场景测试的测试用例(四)

测试用例	4
测试用例名称	用户输入错误密码达 3 次
检验的场景路径	U1,S1.1,U2,S2.2,U2,S2.2,U2,S2.3,S3
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	输入 3 次错误密码——假设 0000 是错误的且和银行卡不匹配
	ATM 余额——¥100
	用户账户余额——¥500
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	每次输入错误密码,系统显示密码错误并提示用户重新输入密码。在第 3 次尝试后,ATM 吞卡,消息提示卡已经被吞,用户需要联系银行重新激活。ATM 保存用户的银行卡。ATM 是激活的、可操作的,等待插入一张客户银行卡
测试覆盖项	TCOVER4

表 B.40　场景测试的测试用例(五)

测试用例	5
测试用例名称	用户选择存款或者转账
检验的场景路径	U1,S1.1,U2,S2.1,U3.2,S10
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	交易类型——存款
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示用户输入存款金额
测试覆盖项	TCOVER5

表 B.41　场景测试的测试用例(六)

测试用例	6
测试用例名称	用户选择错误账户
检验的场景路径	U1,S1.1,U2,S2.1,U3.1,U4,S4.2
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	输入三次无效密码——假设 0000 是无效的且和银行卡不匹配
	ATM 余额——¥100
	客户选择不存在卡里的错误账户
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示输入账号是无效的,提示用户选择新的账户
测试覆盖项	TCOVER6

表 B.42　场景测试的测试用例(七)

测试用例	7
测试用例名称	用户输入无效的取款金额
检验的场景路径	U1,S1.1,U2,S2.1,U3.1,U4,S4.1,U5,S5.2
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——¥100
	用户账户余额——¥20
	取款金额——¥17
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示输入金额是无效的,提示用户重新输入金额
测试覆盖项	TCOVER7

表 B.43　场景测试的测试用例(八)

测试用例	8
测试用例名称	ATM 中现金不足
检验的场景路径	U1,S1.1,U2,S2.1,U3.1,U4,S4.1,U5,S5.3
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——¥100
	用户账户余额——¥500
	取款金额——¥200
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示 ATM 内没有充足的现金,提示用户重新输入金额
测试覆盖项	TCOVER8

表 B.44　场景测试的测试用例(九)

测试用例	9
测试用例名称	用户输入不符合面额的取款金额
检验的场景路径	U1,S1.1,U2,S2.1,U3.1,U4,S4.1,U5,S5.4
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——¥100,ATM 只有¥50 纸币
	用户账户余额——¥1 000
	取款金额——¥20

表 B.44（续）

前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示输入的金额不符合 ATM 余额的面额，提示用户重新输入金额
测试覆盖项	TCOVER9

表 B.45　场景测试的测试用例（十）

测试用例	10
测试用例名称	用户输入的取款金额超过了每日最大的金额
检验的场景路径	U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.5
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——￥100，ATM 只有￥50 纸币
	用户账户余额——￥3 000
	用户每日最大取款金额——￥1 000
	取款金额——￥2 000
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示输入的金额超出了每日最大的取款金额，提示用户重新输入金额
测试覆盖项	TCOVER10

表 B.46　场景测试的测试用例（十一）

测试用例	11
测试用例名称	用户银行账户中的金额不足
检验的场景路径	U1，S1.1，U2，S2.1，U3.1，U4，S4.1，U5，S5.6
输入	有效客户账户的有效银行卡——假设 293910982246 是有效的
	正确密码——假设 5672 是正确的且和银行卡匹配
	ATM 余额——￥50 000
	用户账户余额——￥20
	取款金额——￥50
前置条件	取款要求用户有一个激活的账户、一张有效的卡和匹配的密码以及一台工作的 ATM
预期结果	系统显示信息提示用户银行账户中的金额不足，提示用户重新输入金额
测试覆盖项	TCOVER11

注 2：为了可读性，上面的测试用例都在独立的表中显示。实际上，也可以在一个表中显示。

B.2.9.2.6　步骤 5：汇集测试集（TD5）

根据是否覆盖基本场景和可选场景将测试分组：

TS1：测试用例 1。

TS2：测试用例 2，3，4，5，6，7，8，9，10，11。

B.2.9.2.7 步骤 6：导出测试规程（TD6）

要求的两个测试过程如下所示：

TP1：覆盖 TS1 中的所有测试用例，以测试集指定的顺序。

TP2：覆盖 TS2 中的所有测试用例，以测试集指定的顺序。

B.2.9.2.8 场景测试覆盖率

根据 6.2.9 提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(场景测试)}=\frac{11}{11}\times 100\%=100\%$$

因此，对于场景测试，测试覆盖项的覆盖率达到了 100%。

B.2.9.3 例 2

B.2.9.3.1 介绍

用例测试是场景测试的一种形式，测试用例的导出是基于测试项的用例模型。下面将用一个单独的例子，给本标准的用户提供一个该测试技术的完整可用示例。

B.2.9.3.2 规格说明

考虑如表 B.47 示例中“change_password”测试项的用例。

表 B.47 密码修改用例的例子

用例编号	UC001	
用例名称	change_password	
测试目的	允许用户修改现有的密码，以输入新的密码	
执行者	用户	
用例说明	本用例允许用户修改现有的密码，以输入新的密码	
触发动作	用户在主菜单界面上点击修改密码按钮	
前置条件	用户已经登录系统	
场景名称	步骤	动作
基本流	1	用户点击修改密码按钮
	2	系统显示修改密码界面
	3	用户正确输入当前密码
	4	用户输入正确的新密码
	5	用户重新输入正确的新密码
	6	用户点击 OK
	7	系统消息提示“成功修改密码”

表 B.47（续）

可选流——当前密码输入错误	3.1	用户输入错误的当前密码
	3.2	用户输入正确的新密码
	3.3	用户重新输入正确的新密码
	3.4	用户点击 OK
	3.5	系统错误信息提示“当前密码输入错误，请重试”，并且定位到当前密码输入框
可选流——新密码长度少于 8 位	4.1	用户输入新密码且长度少于 8 位
	4.2	用户点击 OK
	4.3	系统错误信息提示“新密码长度至少需要 8 位，请重试”
可选流——新密码和当前密码一致	5.1	用户输入新密码和当前密码一致
	5.2	用户点击 OK
	5.3	系统错误信息提示“新密码必须不同于当前密码，请重试”
可选流——两次输入的新密码不一致	6.1	用户重新输入新密码，和步骤 4 中输入的新密码不一致
	6.2	用户点击 OK
	6.3	系统错误信息提示“新密码不一致，请重试”
异常	无	
规则	新密码必须和当前密码不同 新密码长度至少有 8 位 系统用星号（*）显示所有当前密码和新密码	
常规	新用户登录系统时首次使用 一般每个用户每年平均使用两次 用户可以随时点击“修改密码”按钮	

注：实际上，也可能存在代表其他情况的场景，例如用户输入无效形式的新密码。

B.2.9.3.3 步骤 1：识别特征集（TD1）

在测试依据中只有一个测试项，只需要定义一个特征集：

FS1：change_password 功能

B.2.9.3.4 步骤 2：导出测试条件（TD2）

测试条件是当前用例中的基本场景和可选场景。在本例中，测试条件是：

TCOND1： 基本流 （FS1）

TCOND2： 可选流——当前密码输入错误 （FS1）

TCOND3： 可选流——新密码长度少于 8 位 （FS1）

TCOND4： 可选流——新密码和当前密码一致 （FS1）

TCOND5： 可选流——两次输入的新密码不一致 （FS1）

B.2.9.3.5 步骤 3：导出测试覆盖项（TD3）

用例测试中的测试覆盖项是基本场景和可选场景，如下所示：

TCOVER1： 基本流 （TCOND1）
TCOVER2： 可选流——当前密码输入错误 （TCOND2）
TCOVER3： 可选流——新密码长度少于 8 位 （TCOND3）
TCOVER4： 可选流——新密码和当前密码一致 （TCOND4）
TCOVER5： 可选流——两次输入的新密码不一致 （TCOND5）

B.2.9.3.6 步骤 4：导出测试用例（TD4）

通过覆盖一个场景来导出测试用例，确定输入来检验测试用例覆盖的路径，确定测试的预期结果，重复上述步骤直到覆盖所有要求的场景，相关的测试用例见表 B.48～表 B.52。

表 B.48 用例测试的测试用例（一）

用例名称	change_password	
测试用例名称	基本流	
描述	用户修改密码成功	
执行者	用户	
测试覆盖项	TCOND1	
覆盖的用例步骤	1,2,3,4,5,6,7	
前置条件	用户已经登录系统	
#	步骤	预期结果
1	用户点击修改密码按钮	系统显示修改密码界面
2	用户正确输入当前密码	当前密码用星号（*）显示
3	用户输入符合规则的新密码	新密码用星号（*）显示
4	用户重新输入新密码	重新输入的新密码用星号（*）显示
5	用户点击 OK	系统消息提示“成功修改密码”

表 B.49 用例测试的测试用例（二）

用例名称	change_password	
测试用例名称	可选流—当前密码错误	
描述	用户尝试修改密码，但是输入的当前密码错误	
执行者	用户	
测试覆盖项	TCOND2	
覆盖的用例步骤	1,2,3.1,3.2,3.3,3.4,3.5	
前置条件	用户已经登录系统	
#	步骤	预期结果
1	用户点击修改密码按钮	系统显示修改密码界面
2	用户输入错误的当前密码	当前密码用星号（*）显示
3	用户输入符合规则的新密码	新密码用星号（*）显示
4	用户重新输入新密码	重新输入的新密码用星号（*）显示
5	用户点击 OK	系统错误信息提示“当前密码输入错误，请重试”

表 B.50　用例测试的测试用例(三)

用例名称	change_password	
测试用例名称	可选流—新密码长度少于 8 位	
描述	用户尝试修改密码,但是新密码长度少于 8 位	
执行者	用户	
测试覆盖项	TCOND3	
覆盖的用例步骤	1,2,3,4.1,4.2,4.3	
前置条件	用户已经登录系统	
#	步骤	预期结果
1	用户点击修改密码按钮	系统显示修改密码界面
2	用户输入正确的当前密码	当前密码用星号(*)显示
3	用户输入新密码且长度少于 8 位	新密码用星号(*)显示
4	用户点击 OK	系统错误信息提示“新密码长度至少需要 8 位,请重试”

表 B.51　用例测试的测试用例(四)

用例名称	change_password	
测试用例名称	可选流—新密码和当前密码一致	
描述	用户尝试修改密码,但是新密码和当前密码一致	
执行者	用户	
测试覆盖项	TCOND4	
覆盖的用例步骤	1,2,3,5.1,5.2,5.3	
前置条件	用户已经登录系统	
#	步骤	预期结果
1	用户点击修改密码按钮	系统显示修改密码界面
2	用户输入正确的当前密码	当前密码用星号(*)显示
3	用户输入新密码和当前密码一致	新密码用星号(*)显示
4	用户点击 OK	系统错误信息提示“新密码不能和当前密码一致,请重试”

表 B.52　用例测试的测试用例(五)

用例名称	change_password
测试用例名称	可选流—两次输入的新密码不一致
描述	用户尝试修改密码,但是新密码长度少于 8 位
执行者	用户
测试覆盖项	TCOND5
覆盖的用例步骤	1,2,3,4,6.1,6.2,6.3

表 B.52（续）

前置条件	用户已经登录系统	
#	步骤	预期结果
1	用户点击修改密码按钮	系统显示修改密码界面
2	用户输入正确的当前密码	当前密码用星号(*)显示
3	用户输入符合规则的新密码	新密码用星号(*)显示
4	用户重新输入新密码，但和步骤 3 中输入的新密码不一致	重新输入的新密码用星号(*)显示
5	用户点击 OK	系统错误信息提示“两次输入的新密码不一致，请重试”

B.2.9.3.7 步骤 5：汇集测试集(TD5)

根据是否覆盖基本路径和可选路径将测试分组：

TS1：测试用例 1

TS2：测试用例 2，3，4，5

B.2.9.3.8 步骤 6：导出测试规程(TD6)

要求的两个测试规程如下所示：

TP1：覆盖 TS1 中的所有测试用例，以测试集指定的顺序。

TP2：覆盖 TS2 中的所有测试用例，以测试集指定的顺序。

B.2.9.3.9 场景测试覆盖率

根据 6.2.9 中提供的计算场景测试覆盖率的公式和上述导出的测试覆盖项：

$$覆盖率_{(场景测试)}=\frac{5}{5}\times 100\%=100\%$$

B.2.10 随机测试

B.2.10.1 介绍

随机测试的目的是在选定的输入分布内生成测试项的输入参数，形成一个测试集。本技术不需要对测试项的输入域进行划分，仅要求输入值是从输入域中随机选择的。

B.2.10.2 规格说明

考虑测试项变换坐标，下面是测试依据：

组件需要将屏幕中用直角坐标系(x，y)表示的位置变为用极坐标系(r，H)表示，替换公式为：$r=\mathrm{sqrt}(x_2+y_2)$，$\cos H = x/r$。直角坐标系的原点和极坐标系的极点都应该在屏幕的中心，x 轴是极坐标系逆时针方向的起始线。所有输入和输出都被表示为一个有范围和精度的定点数，如下所示：

输入

x——范围为 −320～ +320，精度为 $1/2^6$

y——范围为 −240～ +240，精度为 $1/2^7$

输出

r——范围为 0～400，精度为 $1/2^6$

H——范围为 0～((2 * pi)－$1/2^6$),精度为 $1/2^6$

B.2.10.3 步骤 1:识别特征集(TD1)

在测试依据中只有一个测试项,只需要定义一个特征集:

FS1:变换坐标功能

B.2.10.4 步骤 2:导出测试条件(TD2)

随机测试中的测试条件是为每个输入参数选择所有可能的输入值,具体如下:

TCOND1: x——范围为－320～ ＋320,精度为 $1/2^6$ (FS1)

TCOND2: y——范围为－240～ ＋240,精度为 $1/2^7$ (FS1)

B.2.10.5 步骤 3:导出测试覆盖项(TD3)

在随机测试中没有明显的测试覆盖项。

B.2.10.6 步骤 4:导出测试用例(TD4)

构造测试用例首先选择输入分布区间,然后在每个测试条件上使用输入区间,最后确定每个测试用例的预期结果(两个输出参数"r"和"H"的"输出"如表 B.53 所示)。因为例子中测试项的输入参数的操作分布没有任何可用的信息,所以选择一个均匀分布。从定义可知,x 可以随机选择 41024 个值(641×2^6)中的任意一个,同时 y 可以随机选择 61568 个值(481×2^7)中的任意一个。如果使用预期的操作分布而不是均匀分布,则需要注意。预期的分布会忽略部分输入域,可能导致一些意想不到的错误没有进行测试。

因为每个测试用例应同时包含测试条件中 x 和 y 的一个随机测试输入值,所以每个测试用例覆盖了 x 和 y 两个测试条件。在均匀分布中,x 和 y 定义范围内的所有输入值有相同的概率被选为测试用例的输入值。表 B.53 是四个测试用例。

表 B.53 随机测试的测试用例

测试用例	1	2	3	4
输入(x)	－126.125	11.015625	283.046875	－99.109375
输入(y)	238.046875	78.03125	－156.054688	－9.0625
测试条件	TCOND1 TCOND2	TCOND1 TCOND2	TCOND1 TCOND2	TCOND1 TCOND2
输出 r,计算公式($r=\mathrm{sqrt}(x^2+y^2)$)	269.395305	78.804949	323.216025	99.522847
输出 H,计算公式($\cos H=x/r$)	2.058024	1.430554	0.503870	3.050407

B.2.10.7 步骤 5:汇集测试集(TD5)

假设所有的测试必须按照测试用例表中的顺序手动执行,可以定义一个测试集,如下所示:

TS1:手动测试——测试用例 1,2,3,4。

B.2.10.8 步骤 6:导出测试规程(TD6)

只需要执行一个测试集,则可以定义一个测试规程,如下所示:

TP1：按照测试集中指定的顺序手动测试，覆盖所有 TS1 中的测试用例。

B.2.10.9 随机测试测试率

如 6.2.10 中所述，目前尚无达成业界共识的方法用于计算随机测试测试覆盖项的覆盖率。

B.2.10.10 自动化随机测试

随机测试可以手动测试或自动化测试。完全自动化随机测试不需要人工干预，是最有效率的。但是，为了达到完全自动化，应满足：

——自动生成随机测试的输入值；

——从测试依据中自动生成预期结果；或

——自动按测试依据核对测试结果。

只要测试项的输入是已经定义的，使用伪随机数发生器自动生成随机测试的输入值并不难。如果用伪随机数发生器自动生成随机测试的输入值，这些值就不需要进行明确记录，因为可以重新生成相同的值。如果使用一个已经记录的“种子”填充伪随机数发生器，这就是可能的。

自动生成预期结果和自动核对输出是比较难的。通常情况下，根据测试依据自动生成预期结果和自动核对输出是不容易操作的，但是对于一些测试项是有可能的，例如：

——可以使用执行与测试项相同功能的、可信的、独立制作的软件（很可能不符合相同的约束，例如处理速度、实现语言等）；

——仅检查测试项是否崩溃（所以其预期结果是“不崩溃”）；

——测试项输出结果的性质使得核对结果相对容易。排序函数就是这样一个例子，检查输出是否进行了正确的排序是非常简单的任务；

——从每个输出生成输入是比较简单的（使用测试项的逆向功能）。例如，平方根函数可以通过对输出进行平方逆向计算出函数的输入。

在 B.2.10.2 中的例子中，坐标转换测试项可以用逆向功能函数进行自动检验。在这个例子中，从测试项的测试依据中可以直接获得 $r\cos H = x$。经过分析，可以推导出 $r\sin H = y$。如果这两个公式在合理的数值误差内，则测试项正确转换了坐标系。

尽管完全自动化随机测试是不切实际的，但是仍需要考虑，因为它在设计测试用例的过程中不需要大量的开销，而在非随机测试技术中则需要这样的开销。

对于比本例的输入集更大的测试项，“符号输入属性分解”（SIAD）树（Cho 1987）是一种有效的方法，可以在测试用例设计前组织随机抽样的输入域。

附 录 C
（资料性附录）
基于结构的测试设计技术的应用指南和示例

C.1 基于结构的测试的指南和示例

本附录提供了5.3和6.3中描述的基于结构的测试设计技术的指南和示例。每个示例遵照GB/T 38634.2中定义的测试设计和实施过程。示例中使用了不同的应用程序和编程语言。如5.1所述，尽管这里的每个示例都是适合于基于结构的测试环境，但实际上本部分中定义的大多数技术都可以交互使用。

C.2 基于结构的测试设计技术示例

C.2.1 语句测试

C.2.1.1 介绍

语句测试的目的是根据选择的语句覆盖率导出覆盖测试项语句的测试用例集。该结构化的测试设计技术分解了组成测试项的语句。

需要考虑下面两个主要的问题：

——什么是语句？

——哪些语句是可执行的？

通常情况下，语句应该是一个原子操作；一个语句应该完全执行或不执行。例如：

IF a THEN b ENDIF

是多个语句，因为根据条件a，b既可能执行也可能不执行。用于语句测试的语句定义不一定是语言定义中所使用的语句定义。

我们预期的与机器代码相关联语句是可执行的。比如，我们预期以下所有的语句都是可执行的：

——赋值；

——循环和选择；

——过程和函数调用；

——带有初始化的变量声明；

——堆变量存储的动态分配。

但是，其他大多数的变量声明是不可执行的。例如下面的代码：

```
a;
if (b) {
    c;
}
d;
```

任何在b为真（TRUE）情况下的测试用例都覆盖了全部的语句。注意如果不使用b为假（FALSE）的情况下可以实现完全语句覆盖。

C.2.1.2 规格说明

考虑如下Ada语言的测试项，将正整数分为质数和合数，并且求合数的因子：

```
1 READ (Num);
2 WHILE NOT End of File DO
3   Prime := TRUE;
4   FOR Factor := 2 TO Num DIV 2 DO
5     IF Num - (Num DIV Factor) * Factor = 0 THEN
6       WRITE (Factor, ' is a factor of', Num);
7       Prime := FALSE;
8     ENDIF;
9   ENDFOR;
10  IF Prime = TRUE THEN
11    WRITE (Num, ' is prime');
12  ENDIF;
13  READ (Num);
14 ENDWHILE;
15 WRITE ('End of prime number program');
```

C.2.1.3 第 1 步：识别特征集(TD1)

测试依据中只定义了一个测试项，只需要定义一个特征集：

FS1：找出质数和合数的函数

C.2.1.4 第 2 步：导出测试条件(TD2)

语句测试的测试条件是代码中可执行的语句。给每一行代码编号，对应测试条件的数量。例如，语句 1 允许定义一个测试条件：

TCOND1： READ (Num)；语句 1 (FS1)

不再重复说明剩余的测试条件对应的源代码：

TCOND2： 语句 2 (FS1)

TCOND3： 语句 3 (FS1)

TCOND4： 语句 4 (FS1)

TCOND5： 语句 5 (FS1)

TCOND6： 语句 6 (FS1)

TCOND7： 语句 7 (FS1)

TCOND8： 语句 10 (FS1)

TCOND9： 语句 11 (FS1)

TCOND10： 语句 13 (FS1)

TCOND11： 语句 15 (FS1)

C.2.1.5 第 3 步：导出测试覆盖项(TD3)

语句测试中的测试覆盖项和测试条件一致：

TCOVER1：语句 1 (TCOND1)

TCOVER2：语句 2 (TCOND2)

TCOVER3：语句 3 (TCOND3)

TCOVER4：语句 4 (TCOND4)

TCOVER5：语句 5 (TCOND5)

TCOVER6：语句 6　　（TCOND6）
TCOVER7：语句 7　　（TCOND7）
TCOVER8：语句 10　　（TCOND8）
TCOVER9：语句 11　　（TCOND9）
TCOVER10：语句 13　　（TCOND13）
TCOVER11：语句 15　　（TCOND11）

C.2.1.6　第 4 步：导出测试用例（TD4）

在语句测试中，每条语句应至少由一个测试用例覆盖。首先通过识别控制流图中的子路径导出测试用例，该子路径执行了没有被测试用例覆盖的一个或多个可执行的语句。确定执行子路径的输入和预期结果。重复此过程，直到达到测试要求的测试覆盖率。在本例中，只需要一个测试用例就可以覆盖代码中的所有语句，因为代码中的迭代允许两个输入值覆盖所有语句（即达到 100％的语句覆盖），见表 C.1。

表 C.1　语句测试的测试用例

测试用例	输入	预期结果	测试覆盖项
1	2	2 是质数	1，2，3，4，9，10，11，12，13，14
	4	2 是 4 的倍数	2，3，4，5，6，7，8，9，10，12，13，14
	EOF	质数程序结束	2，15

注：在该举例中，有一个测试用例以及三个独立的输入值和预期结果的集合，因为代码中包含迭代。

C.2.1.7　第 5 步：汇集测试集（TD5）

只有一个测试用例，可以组合生成一个测试集，如下所示：
TS1：测试用例 1。

C.2.1.8　第 6 步：导出测试规程（TD6）

只需要一个规程，如下所示：
TP1：覆盖 TS1 的测试用例。

C.2.1.9　语句测试覆盖率

根据 6.3.1 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(语句)}=\frac{11}{11}\times 100\%=100\%$$

因此，对于语句测试，测试覆盖项的覆盖率达到了 100％。

C.2.2　分支/判定测试

C.2.2.1　介绍

分支覆盖和判定覆盖密切相关。对于只有一个入口点的测试项，100％的分支覆盖相当于 100％的判定覆盖，如果低于 100％的覆盖率，则可能不一样。用一个例子说明两个不同的覆盖水平。

C.2.2.2　规格说明

组件确定一个单词在单词表中的位置，单词表是按字母顺序排序的。除了输入单词和表，还应当向

组件输入查找的表中的单词数量。如果在表中查找到单词，组件应当返回该单词的位置(从零开始)，否则返回“-1”。

代码来自(Kernighan and Richie 1998)。3 个判定进行了加粗：

```
int binsearch (char * word, struct key tab[], int n) {
    int cond;
    int low, high, mid;
    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (low+high) / 2;
        if ((cond = strcmp(word, tab[mid].word)) < 0)
            high = mid - 1;
        else if (cond > 0)
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}
```

C.2.2.3 第 1 步：识别特征集(TD1)

测试依据只定义了一个测试项，只需要定义一个特征集：

FS1：二分查找功能

C.2.2.4 第 2 步：导出测试条件(TD2)

C.2.2.4.1 导出测试条件的选项

构造程序的控制流图，识别分支/判定测试的测试条件。构造程序的控制流图的第一步是把程序划分成基本块。基本块是指令的序列，没有分支进入基本块(除了开始)，没有分支从基本块出来(除了结束)。每个基本块的语句将一起执行或者都不执行。上面的程序有下面几个基本块：

```
int binsearch (char * word, struct key tab[], int n) {
        int cond;
        int low, high, mid;
B1      low = 0;
        high = n - 1;
B2      while (low <= high) {
B3        mid = (low+high) / 2
        if ((cond = strcmp(word, tab[mid].word)) < 0)
B4        high = mid - 1;
B5      else if (cond > 0)
B6        low = mid + 1;
B7      else
```

```
            return mid;
B8      }
B9      return -1;
}
```

构造控制流图，每个基本块代表一个节点，画一条从一个基本块到另一个基本块的控制转移边。下面是可能控制转移：

B1→B2　B3→B4　B5→B6　B6→B8

B2→B3　B3→B5　B5→B7　B8→B2

B2→B9　B4→B8

得到的控制流图如图 C.1 所示。图中有一个入口点 B1 和两个出口点，B7 和 B9。

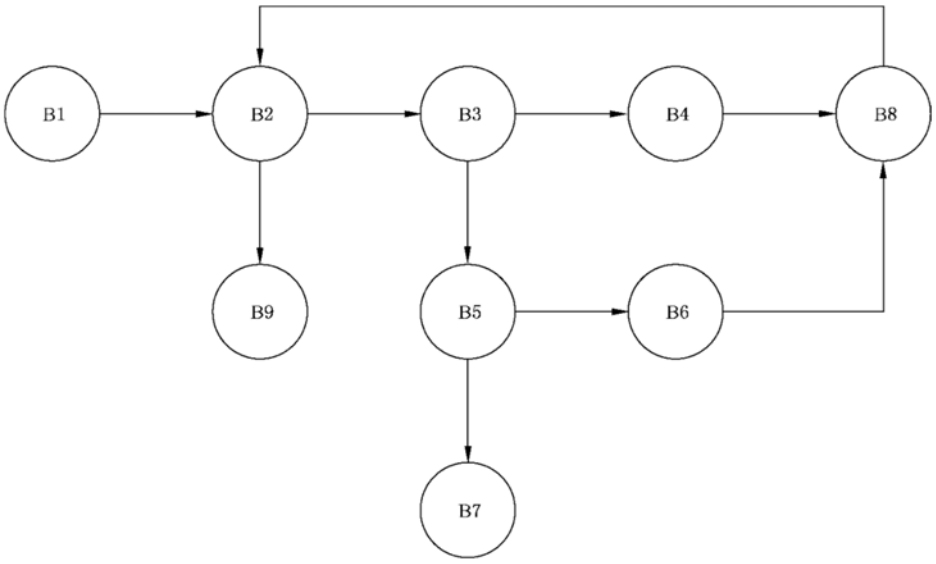


图 C.1　二分查找控制流图

上面的控制流图不一定是人工绘制的，通常使用工具显示已经执行的判定/分支。

分支覆盖的测试条件与判定覆盖的测试条件不同，操作步骤见 C.2.2.4.2 和 C.2.2.4.3。

C.2.2.4.2　选项 2a：导出分支覆盖的测试条件（TD2）

对于分支覆盖，测试条件（BRANCH-TCOND）是分支，在控制流图中用边表示。本例中有 10 测试条件，具体如下：

BRANCH-TCOND1：　B1→B2　（FS1）

BRANCH-TCOND2：　B2→B3　（FS1）

BRANCH-TCOND3：　B2→B9　（FS1）

BRANCH-TCOND4：　B3→B4　（FS1）

BRANCH-TCOND5：　B3→B5　（FS1）

BRANCH-TCOND6：　B4→B8　（FS1）

BRANCH-TCOND7：　B5→B6　（FS1）

BRANCH-TCOND8：　B5→B7　（FS1）

BRANCH-TCOND9：　B6→B8　（FS1）

BRANCH-TCOND10：　B8→B2　（FS1）

C.2.2.4.3 选项 2b：导出判定覆盖的测试条件（TD2）

对于判定覆盖，测试条件（DECISION-TCOND）是控制流图中的判定节点，判定节点有多个汇出的箭头。在本例中，有三个测试条件：

DECISION-TCOND1：B2 （FS1）

DECISION-TCOND2：B3 （FS1）

DECISION-TCOND3：B5 （FS1）

C.2.2.5 第 3 步：导出测试覆盖项（TD3）

C.2.2.5.1 导出测试覆盖项的选项

分支覆盖的测试覆盖项和判定覆盖不同，操作步骤见 C.2.2.5.2 和 C.2.2.5.3。

C.2.2.5.2 选项 3a：导出分支覆盖的测试覆盖项（TD3）

对于分支覆盖，测试覆盖项目是控制流图的分支，和测试条件一致。在本例中，分支覆盖有 10 个测试覆盖项，具体如下：

BRANCH-TCOVER1： B1→B2 （BRANCH-TCOND1）

BRANCH-TCOVER2： B2→B3 （BRANCH-TCOND2）

BRANCH-TCOVER3： B2→B9 （BRANCH-TCOND3）

BRANCH-TCOVER4： B3→B4 （BRANCH-TCOND4）

BRANCH-TCOVER5： B3→B5 （BRANCH-TCOND5）

BRANCH-TCOVER6： B4→B8 （BRANCH-TCOND6）

BRANCH-TCOVER7： B5→B6 （BRANCH-TCOND7）

BRANCH-TCOVER8： B5→B7 （BRANCH-TCOND8）

BRANCH-TCOVER9： B6→B8 （BRANCH-TCOND9）

BRANCH-TCOVER10： B8→B2 （BRANCH-TCOND10）

C.2.2.5.3 选项 3b：导出判定覆盖的测试覆盖项（TD3）

对于判定覆盖，测试覆盖项是每个判定的结果（即 true，false）。在本例中，每个判定都有 true 和 false 的两个结果；因此有六个测试覆盖项，具体如下：

DECISION-TCOVER1： B2 = true （DECISION-TCOND1）

DECISION-TCOVER2： B2 = false （DECISION-TCOND1）

DECISION-TCOVER3： B3 = true （DECISION-TCOND2）

DECISION-TCOVER4： B3 = false （DECISION-TCOND2）

DECISION-TCOVER5： B5 = true （DECISION-TCOND3）

DECISION-TCOVER6： B5 = false （DECISION-TCOND3）

一个判定通常是有两个以上的结果，增加了需要导出的测试覆盖项的数量。

C.2.2.6 第 4 步：导出测试用例（TD4）

分支测试通过识别控制流子路径导出测试用例，子路径是测试过程中没有执行的一个或多个分支（测试覆盖项），确定满足子路径的输入，确定每个测试的预期结果，重复上述步骤直到达到要求的测试覆盖率。类似的，判定测试通过识别控制流子路径导出测试用例，子路径到达尚未在测试过程中执行的一个或多个判定，确定每个测试输入和预期结果。对于分支覆盖和判定覆盖，测试项中的任何一个单独

测试，都会执行一个子路径，也会包含许多判定和分支。

考虑测试用例执行子路径 B1→B2 →B9。该用例的 $n = 0$，表示查找的表中没有条目。子路径执行判定(B2 →B9)，达到了 1/6=16.7%的覆盖率。路径执行 10 个分支中的 2 个分支，达到了 20%的覆盖率(和判定覆盖不同)。

现在考虑测试用例，执行了子路径：

B1→B2→B3→B4→B8→B2→B3→B5→B6→B8→B2→B3→B5→B7

该子路径出现的条件是首先查找表中的前半部分，然后查找该部分的前半部分(第二个二分之一，即四分之一)，找到条目。注意两个测试用例提供了 100%判定和分支覆盖。

这些测试用例见表 C.2。

表 C.2　二分搜索测试用例

测试用例	输入			执行判定(下划线)	测试覆盖项目	预期结果
	单词	标签	n			
1	chas	Alf Bert Chas Dick Eddy Fred Geoff	7	B1→B2→B3→B4 →B8→B2→B3 →B5→B6→B8→B2 →B3→B5→B7	BRANCH-TCOVER 1，2，4，5，6，7，8，9，10 以及 DECISION-TCOVER 1，3，4， 5，6	2
2	chas	‘empty table’	0	B1→B2→B9	BRANCH-TCOVER 1，3 以及 DECISION-TCOVER 2	−1

分支和判定覆盖通常使用软件工具进行测量。

C.2.2.7　第 5 步：汇集测试集(TD5)

只需要两个测试用例，可以选择组合成一个测试集。

TS1：测试用例 1 和 2。

C.2.2.8　第 6 步：导出测试规程(TD6)

只有两个测试用例和一个测试集，可以选择只定义一个测试规程。

TP1：按照测试集指定的顺序覆盖 TS1 中的所有测试用例。

C.2.2.9　分支测试覆盖率

根据 6.3.2 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(分支)}=\frac{10}{10}\times 100\%=100\%$$

因此，对于分支测试，测试覆盖项的覆盖率达到了 100%。

C.2.2.10　判定测试覆盖率

根据 6.3.3 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(判定)}=\frac{6}{6}\times 100\%=100\%$$

因此，对于判定测试，测试覆盖项的覆盖率达到了 100%。

C.2.3 分支条件测试、分支条件组合测试及修正条件判定覆盖(MCDC)测试

C.2.3.1 介绍

分支条件测试、分支条件组合测试和修正条件判定测试是密切相关的,覆盖测量也是相关联的。三种测试设计技术的目的是根据选择的覆盖率,导出测试用例集覆盖测试项的判定条件。下面用一个例子说明测试用例设计和测试覆盖测量方法。

C.2.3.2 规格说明

考虑下面的代码片段:

```
if A or (B and C) then
    do_something;
else
    do_something_else;
end if;
```

在判定条件中的布尔条件是 A,B 和 C。A、B 和 C 自身可能由复杂的表达式组成,涉及到关系运算。例如,布尔条件 A 可以是一个表达式 X>= Y。为了更加清晰地说明,下面的例子都把 A、B 和 C 设置为简单的布尔条件。

C.2.3.3 第 1 步:识别特征集(TD1)

三种测试设计技术的例子都是基于相同的测试项(上面的代码片段),3 种测试技术可以定义一个特征集:

FS1:条件代码片段

C.2.3.4 第 2 步:导出测试条件(TD2)

三种测试设计技术中,控制流图的每个判定是一个测试条件。在本例中,有一个判定:

TCOND1:A or (B and C) (FS1)

本测试条件适用于分支条件测试、分支条件组合测试和修改条件判定覆盖测试。

C.2.3.5 分支条件测试

C.2.3.5.1 第 3 步:导出测试覆盖项目(TD3)

分支条件测试检测多个条件判定中的独立条件,目的是每个独立条件和每个判定都同时具有真值(TRUE)和假值(FALSE)。测试覆盖项是判定内的条件布尔值(真 TRUE/假 FALSE)。在本例中,分支条件测试要求布尔条件 A 分别取 TRUE 和 FALSE,布尔条件 B 分别取 TRUE 和 FALSE 和布尔条件 C 分别取 TRUE 和 FALSE。下面的测试覆盖项符合要求:

TCOVER1:A = TRUE (TCOND1)
TCOVER2:A = FALSE (TCOND1)
TCOVER3:B = TRUE (TCOND1)
TCOVER4:B = FALSE (TCOND1)
TCOVER5:C = TRUE (TCOND1)
TCOVER6:C = FALSE (TCOND1)
TCOVER7:A or (B and C) = TRUE (TCOND1)
TCOVER8:A or (B and C) = FALSE (TCOND1)

C.2.3.5.2 第 4 步：导出测试用例(TD4)

通过识别控制流子路径导出分支条件覆盖的测试用例，子路径是测试过程中没有执行的一个或多个分支(测试覆盖项)，确定满足子路径的输入，确定每个测试的预期结果，重复上述步骤直到达到要求的测试覆盖率。在本例中，表 C.3 的测试输入集符合要求(注意：其他的测试输入集也可以实现分支条件覆盖)。

表 C.3 分支条件测试的测试用例

测试用例	A	B	C	A 或(B 和 C)	测试覆盖项
1	FALSE	FALSE	FALSE	FALSE	TCOVER 2，4，6
2	TRUE	TRUE	TRUE	TRUE	TCOVER 1，3，5

注：表 C.3 中的测试用例不包括预期结果，是不“完整”的。

通常可以只用两个测试用例实现分支条件覆盖，和所有条件的实际布尔条件数量无关。

C.2.3.5.3 第 5 步：汇集测试集(TD5)

只需要两个测试用例，可以选择组合成一个测试集。

TS1：测试用例 1 和 2。

C.2.3.5.4 第 6 步：导出测试规程(TD6)

只有两个测试用例和一个测试集，可以选择只定义一个测试规程。

TP1：按照测试集指定的顺序覆盖 TS1 中的所有测试用例。

C.2.3.5.5 分支条件测试覆盖率

根据 6.3.4 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(分支条件)}=\frac{8}{8}\times 100\%=100\%$$

因此，对于分支条件测试，测试覆盖项的覆盖率达到了 100%。

C.2.3.6 分支条件组合测试

C.2.3.6.1 第 3 步：导出测试覆盖项(TD3)

在分支条件组合测试中，测试覆盖项是判定中的条件布尔值的不重复组合。在本例中，分支条件组合测试所有布尔条件 A、B 和 C 的组合。该技术的测试覆盖项有：

TCOVER1：A = FALSE，B = FALSE，C = FALSE (TCOND1)

TCOVER2：A = TRUE，B = FALSE，C = FALSE (TCOND1)

TCOVER3：A = FALSE，B = TRUE，C = FALSE (TCOND1)

TCOVER4：A = TRUE，B = TRUE，C = FALSE (TCOND1)

TCOVER5：A = FALSE，B = FALSE，C = TRUE (TCOND1)

TCOVER6：A = TRUE，B = FALSE，C = TRUE (TCOND1)

TCOVER7：A = FALSE，B = TRUE，C = TRUE (TCOND1)

TCOVER8：A = TRUE，B = TRUE，C = TRUE (TCOND1)

C.2.3.6.2 第 4 步：导出测试用例(TD4)

通过识别控制流子路径导出测试用例，子路径是测试过程中没有执行的一个或多个分支(测试覆盖项)，确定满足子路径的输入，确定每个测试的预期结果，重复上述步骤直到达到要求的测试覆盖率。在本例中，可以通过导出表 C.4 的测试用例来实现。

表 C.4 分支条件组合测试的测试用例

测试用例	A	B	C	测试覆盖项
1	FALSE	FALSE	FALSE	TCOVER1
2	TRUE	FALSE	FALSE	TCOVER2
3	FALSE	TRUE	FALSE	TCOVER3
4	TRUE	TRUE	FALSE	TCOVER4
5	FALSE	FALSE	TRUE	TCOVER5
6	TRUE	FALSE	TRUE	TCOVER6
7	ALSE	TRUE	TRUE	TCOVER7
8	TRUE	TRUE	TRUE	TCOVER8

注：表 C.4 中的测试用例不包括预期结果，是不“完整”的。

分支条件组合覆盖是很详尽的，需要 2^n 个测试用例实现 100%覆盖率，包含了 n 个布尔条件。复杂的条件不能实现这种增速。

C.2.3.6.3 第 5 步：汇集测试集(TD5)

所有测试用例可以组合成一个测试集，如下：

TS1：测试用例 1，2，3，4，5，6，7，8。

C.2.3.6.4 第 6 步：导出测试规程(TD6)

只有一个测试集，可以选择定义一个对应的测试规程，具体如下：

TP1：按照测试集指定的顺序覆盖 TS1 中的所有测试用例。

C.2.3.6.5 分支条件组合测试覆盖率

根据 6.3.5 提供的公式和上述生成的测试覆盖项计算覆盖率：

$$覆盖率_{(分支条件组合)}=\frac{8}{8}\times 100\%=100\%$$

因此，对于分支条件组合测试，测试覆盖项的覆盖率达到了 100%。

C.2.3.7 修正条件判定覆盖测试

修正条件判定覆盖(MCDC)测试比分支条件组合覆盖需要更少的测试用例。广泛应用于航空电子软件的发展，如 RTCA / DO-178C 所要求的。MCDC 测试需要测试用例说明每一个布尔条件(A、B 和 C)可以独立影响决策结果。测试覆盖项比所有条件的组合少(分支条件组合覆盖所要求的)。

C.2.3.7.1 第 3 步：导出测试覆盖项(TD3)

在修正条件判定覆盖(MCDC)测试中，测试覆盖项是判定中的条件独立布尔值的不重复组合，允许

一个布尔条件单独影响结果。

例如判定条件[A or(B and C)],有一个测试覆盖项对,B 和 C 保持不变,仅改变 A 的状态时会改变结果,即 A 可以独立影响条件的结果,如下:

TCOVER1:A=FALSE,B=FALSE,C=TRUE　OUTCOME=FALSE　　(TCOND1)

TCOVER2:A=TRUE,B=FALSE,C=TRUE　OUTCOME=TRUE　　(TCOND1)

同样对于 B,需要一个测试用例对说明 A 和 C 保持不变时 B 可以独立影响结果:

TCOVER3:A=FALSE,B=FALSE,C=TRUE　OUTCOME=FALSE　　(TCOND1)

TCOVER4:A=FALSE,B=TRUE,C=TRUE　OUTCOME=TRUE　　(TCOND1)

对于 C,需要一个测试用例对说明 A 和 B 保持不变时 C 可以独立影响结果:

TCOVER5:A=FALSE,B=TRUE,C=FALSE　OUTCOME=FALSE　　(TCOND1)

TCOVER6:A=FALSE,B=TRUE,C=TRUE　OUTCOME=TRUE　　(TCOND1)

单独为每个判定条件创建测试条件对,可以得出 TCOVER1 和 TCOVER3 是相同的,TCOVER4 和 TCOVER6 也是相同的。因此,在下一步中不会使用重复的测试覆盖项 TCOVER3 和 TCOVER6 导出测试用例。

C.2.3.7.2　第 4 步:导出测试用例(TD4)

通过识别控制流子路径导出测试用例,子路径是测试过程中没有执行的一个或多个分支(测试覆盖项),确定满足子路径的输入,确定每个测试的预期结果,重复上述步骤直到达到要求的测试覆盖率。在本例中,表 C.5 的测试用例集符合要求。

注:表 C.5 中的测试用例不包括预期结果,是不"完整"的。

表 C.5　整体测试用例集

测试用例	A	B	C	预期结果	测试覆盖项
1	FALSE	FALSE	TRUE	FALSE	TCOVER1, TCOVER3
2	TRUE	FALSE	TRUE	TRUE	TCOVER2
3	FALSE	TRUE	TRUE	TRUE	TCOVER4, TCOVER6
4	FALSE	TRUE	FALSE	FALSE	TCOVER5

总结:

——测试用例 1 和 2 说明 A 独立影响判定条件的结果;

——测试用例 1 和 3 说明 B 独立影响判定条件的结果;

——测试用例 3 和 4 说明 C 独立影响判定条件的结果。

注意可能会有实现 MCDC 的替代方案。例如,表 C.6 中的两个测试用例可以说明 A 独立影响判定条件的结果。

表 C.6　替代 MCDC 测试用例

用例	A	B	C	结果
X	FALSE	TRUE	FALSE	FALSE
Y	TRUE	TRUE	FALSE	TRUE

测试用例 X 与上面的测试用例 4 一样，但是测试用例 Y 是前面没有使用的。已经实现 MCDC，不需要测试用例 Y 实现覆盖。

达到 100％的修正条件判定覆盖率，至少需要 $n+1$ 个测试用例，最多 2^n 个测试用例，其中 n 是布尔条件的数量。分支条件组合覆盖需要 2^n 个测试用例。因此，MCDC 是一种实用的低风险的方法，是分支条件组合覆盖的折中，条件表达式涉及的不仅仅是一些布尔条件。

C.2.3.7.3 第 5 步：汇集测试集（TD5）

将表 C.6 中定义的所有测试用例组合成一个测试集，如下：

TS1：测试用例 1，2，3，4。

C.2.3.7.4 第 6 步：导出测试规程（TD6）

只有一个测试集，可以选择定义一个相应的测试规程，如下：

TP1：按照测试集指定的顺序覆盖 TS1 中的所有测试用例。

C.2.3.7.5 修正条件判定覆盖测试覆盖率

根据 6.3.6 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(MCDC)}=\frac{4}{4}\times 100\%=100\%$$

因此，对于修正条件判定覆盖测试，测试覆盖项的覆盖率达到了 100％。

C.2.3.8 其他布尔表达式

三种测试设计技术和测试覆盖测量方法的一个缺点是，当控制判定放置在实际判定条件外部时，布尔表达式是有漏洞的。例如：

```
FLAG := A or (B and C);
if FLAG then
    do_something;
else
    do_something_else;
end if;
```

为了修补漏洞，这三种测试设计技术和覆盖测量的使用变体是要设计所有布尔表达式的测试，不只是直接用于控制流判定的布尔表达式。

C.2.3.9 优化表达式

一些编程语言和编译器通过忽略表达式中对结果没有直接影响的部分，“短路”布尔运算符的评估。

例如在 C 和 C++语言中短路布尔“and”(&&)和“or”(||)的运算符，Ada 编程语言提供专门的短路运算符。上述例子中，如果可以根据第一个条件确定结果，则不需要评估第二个条件。

结果是，不能显示第二个条件的值覆盖。对于短路“and”运算符，可行的组合是“真：真”“真：假”和“假：X”，其中 X 是未知的。对于短路“or”运算符，可行的组合是“假：假”“假：真”和“真：X”。

其他语言和编译器可能会短路布尔运算符的评价。在该情况下，可行的组合是未知的。布尔运算符的短路优化的程度可能取决于编译器转换器，或者可能不受用户控制。

短路控制形式对分支条件覆盖或修正条件判定覆盖没有阻碍，但会妨碍分支条件组合覆盖的测量。情况是，可以设计测试用例达到 100％的覆盖率（从理论的角度来看），但实际上测量不可能达到 100％的覆盖。

C.2.3.10 其他分支和判定

上述对分支条件测试、分支条件组合测试和 MCDC 测试及其相应的覆盖测量用分支或判定的形式说明，分支和判定由布尔条件控制。其他分支和判定，如多路分支（由“case”“switch”或“computed goto”语句实现），计算循环（由“for”或“do”实施循环不附带任何条件）不使用布尔条件，因此前面的描述不能解决其他分支和判定。

处理该情况的一个方法是使用这三个测试设计技术及其相关的覆盖测量补充分支测试和分支或判定覆盖。分支测试将解决所有简单的判定、多路判定和所有循环。条件测试将解决包括布尔条件的判定。

实际上，其中一个选项的测试用例实现 100％的覆盖率，其他选项也能达到 100％的覆盖率。两个选项之间不能比较小于 100％的覆盖率。

C.2.4 数据流测试

C.2.4.1 介绍

数据流测试的目的是通过选择的定义-使用覆盖率导出测试用例集，这些测试用例覆盖测试项中变量定义和使用之间的路径。数据流测试是一项基于结构的测试设计技术，目的是执行从测试项的变量定义点到变量引用点的子路径。子路径就是定义-使用对。不同的数据流覆盖准则要求执行不同定义-使用对和子路径。每个准则导出测试用例集以尽可能达到 100％的覆盖率。

注：数据流测试需要定义数据对象。工具通常把数组或记录当作单独的数据项，而不是作为有许多部分组成的复合对象。忽略复合对象的组成部分降低了数据流测试的有效性。

C.2.4.2 规格说明

考虑 Ada 编程语言中对以下测试项进行数据流测试：

```
procedure Solve_Quadratic(A, B, C: in Float; Is_Complex: out Boolean; R1, R2:out Float) is
-- Is_Complex is true if the roots are not real.
-- If the two roots are real, they are produced in R1, R2.
        Discrim : Float := B * B - 4.0 * A * C;                    -- 1
        R1, R2: Float;                                             -- 2
    Begin                                                          -- 3
        if Discrim < 0.0 then                                      -- 4
            Is_Complex := true;                                    -- 5
        else                                                       -- 6
            Is_Complex := false;                                   -- 7
        end if;                                                    -- 8
        if not Is_Complex then                                     -- 9
            R1 := (-B + Sqrt(Discrim))/ (2.0 * A);                 -- 10
            R2 := (-B - Sqrt(Discrim))/ (2.0 * A);                 -- 11
        end if;                                                    -- 12
    end Solve_Quadratic;                                           -- 13
```

注：第二行不是定义（R1 和 R2），而是声明。（如果声明语句直接初始化，则是定义。）

C.2.4.3 第 1 步：识别特征集（TD1）

只有一个功能的测试，只要定义一个特征集：

FS1：Solve_Quadratic 功能

C.2.4.4 第 2 步：导出测试条件(TD2)

第一步是列出测试项中使用的变量：A，B，C，Discrim，Is_Complex，R1 and R2。接下来，测试项中变量每次出现都是对程序列表的交叉引用，并为其分配一个类别(定义、谓词使用或计算使用)。变量的出现及其类别见表 C.7。

表 C.7 变量的出现及其类别

行	类别		
	定义	计算使用	谓词使用
0	A，B，C		
1	Discrim	A，B，C	
2			
3			
4			Discrim
5	Is_Complex		
6			
7	Is_Complex		
8			
9			Is_Complex
10	R1	A，B，Discrim	
11	R2	A，B，Discrim	
12			
13		R1，R2，Is_Complex	

测试条件是定义-使用对。

下一步，是确定定义-使用对及其类型(谓词使用或计算使用)，每个定义-使用对是一个测试覆盖项，识别定义列中的每个条目到该变量谓词使用或计算使用列的每个条目之间的链接。定义-使用对及其类型见表 C.8。

表 C.8 定义-使用对及其类型

定义-使用对 (起始行→结束行)	变量		测试条件
	计算使用	谓词使用	
0→1	A		TCOND1
	B		TCOND2
	C		TCOND3
0→10	A		TCOND4
	B		TCOND5

表 C.8（续）

定义-使用对（起始行→结束行）	变量		测试条件
	计算使用	谓词使用	
0→11	A		TCOND6
	B		TCOND7
1→4		Discrim	TCOND8
1→10	Discrim		TCOND9
1→11	Discrim		TCOND10
5→9		Is_Complex	TCOND11
7→9		Is_Complex	TCOND12
10→13	R1		TCOND13
11→13	R2		TCOND14
5→13	Is_Complex		TCOND15
7→13	Is_Complex		TCOND16

注：为了导出测试覆盖项，不一定要生成所有的定义使用对（如上表所示）（依赖所使用的技术）。

C.2.4.5 全定义测试

C.2.4.5.1 第 3a 步：导出测试覆盖项（TD3）

全定义测试中，测试覆盖项是从变量定义到使用（计算使用或谓词使用）的控制流子路径。表 C.9 所示的定义-使用对符合此准则的。

表 C.9 全定义测试

测试覆盖项	全定义		
	变量	定义-使用对	测试条件
TCOVER1	A	0→1	TCOND1
TCOVER2	B	0→1	TCOND2
TCOVER3	C	0→1	TCOND3
TCOVER4	Discrim	1→4	TCOND8
TCOVER5	Is_Complex	5→9	TCOND11
TCOVER6	Is_Complex	7→9	TCOND12
TCOVER7	R1	10→13	TCOND13
TCOVER8	R2	11→13	TCOND14

C.2.4.5.2 第 4a 步：导出测试用例（TD4）

通过识别控制流子路径，覆盖一个或多个未执行的测试覆盖项，确定控制子路径的输入和测试的预

期结果，从而导出测试用例。重复上述步骤直到达到要求的测试覆盖率。为达到100%全定义数据流覆盖率，至少执行每个变量定义到使用(谓词使用或计算使用)的一个子路径。表C.10的测试集合满足此要求。

表 C.10　对全定义测试的测试用例

测试用例	全定义				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	Is_Complex	7→9	7-8-9	TCOVER6	1	2	1	FALSE	−1	−1
	R1	10→13	10-11-12-13	TCOVER7						
	R2	11→13	11-12-13	TCOVER8						
2	A,B,C, Discrim	0→1 1→4	0-1 1-2-3-4	TCOVER1, TCOVER2, TCOVER3 TCOVER4	1	1	1	TRUE	未定义	未定义
	Is_Complex	5→9	5-8-9	TCOVER5						

C.2.4.5.3　全定义测试覆盖率

根据6.3.7.1提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(全定义)}=\frac{8}{8}\times 100\%=100\%$$

因此，对于全定义测试，测试覆盖项的覆盖率达到了100%。

C.2.4.6　全计算使用测试

C.2.4.6.1　第3b步：导出测试覆盖项(TD3)

在全计算使用测试中，测试覆盖项是从变量定义到该定义所有计算使用的控制流子路径。

表C.11所示的定义-使用对符合此准则。

表 C.11　全计算使用测试

测试覆盖项	全计算使用			
	变量	定义使用对	子路径	测试条件
TCOVER1	A	0→1	0-1	TCOND1
TCOVER2	B	0→1	0-1	TCOND2
TCOVER3	C	0→1	0-1	TCOND3
TCOVER4	A	0→10	0-1-2-3-4-6-7-8-9-10	TCOND4
TCOVER5	B	0→10	0-1-2-3-4-6-7-8-9-100-1-2-3-4-6-7-8-9-10	TCOND5
TCOVER6	A	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND6
TCOVER7	B	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND7
TCOVER8	Discrim	1→10	1-2-3-4-6-7-8-9-10	TCOND9
TCOVER9	Discrim	1→11	1-2-3-4-6-7-8-9-10-11	TCOND10

表 C.11（续）

测试覆盖项	全计算使用			
	变量	定义使用对	子路径	测试条件
TCOVER10	R1	10→13	10-11-12-13	TCOND13
TCOVER11	R2	11→13	11-12-13	TCOND14
TCOVER12	Is_Complex	5→13	5-8-9-12-13	TCOND15
TCOVER13	Is_Complex	7→13	7-8-9-10-11-12-13	TCOND16

C.2.4.6.2 第 4b 步：导出测试用例（TD4）

通过识别控制流子路径，覆盖一个或多个未执行的测试覆盖项，确定控制子路径的输入和测试的预期结果，从而导出测试用例，重复上述步骤直到达到要求的测试覆盖率。为达到 100％全计算使用数据流覆盖率，至少执行每个变量定义到该定义的所有计算使用的一个子路径，由此产生的测试用例见表 C.12。

表 C.12 全计算使用测试的测试用例

测试用例	全计算使用				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	A，B，C	0→1	0-1	TCOVER1，TCOVER2，TCOVER3	1	2	1	FALSE	−1	−1
	A，B	0→10	0-1-2-3-4-6-7-8-9-10	TCOVER4，TCOVER5						
	A，B	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOVER6，TCOVER7						
	Discrim	1→10	1-2-3-4-6-7-8-9-10	TCOVER8						
		1→11	1-2-3-4-6-7-8-9-10-11	TCOVER9						
	R1	10→13	10-11-12-13	TCOVER10						
	R2	11→13	11-12-13	TCOVER11						
	Is_Complex	7→13	7-8-9-10-11-12-13	TCOVER13						
2	Is_Complex	5→13	5-8-9-12-13	TCOVER12	1	1	1	TRUE	未定义	未定义

C.2.4.6.3 全计算使用测试覆盖率

根据 6.3.7.2 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(全计算使用)}=\frac{13}{13}\times 100\%=100\%$$

因此，对于全计算使用测试，测试覆盖项的覆盖率达到了 100％。

C.2.4.7 全谓词使用测试

C.2.4.7.1 第 3c 步：导出测试覆盖项(TD3)

在全谓词使用测试中，测试覆盖项是从变量定义到该定义所有谓词-使用的控制流子路径。

表 C.13 所示的定义使用对符合此准则。

表 C.13 全谓词使用测试

测试覆盖项	全谓词使用		
	变量	定义使用对	测试条件
TCOVER1	Discrim	1→4	TCOND8
TCOVER2	Is_Complex	5→9	TCOND11
TCOVER3	Is_Complex	7→9	TCOND12

C.2.4.7.2 第 4c 步：导出测试用例(TD4)

通过识别控制流子路径，覆盖一个或多个未执行的测试覆盖项，确定控制子路径的输入和测试的预期结果，从而导出测试用例，重复上述步骤直到达到要求的测试覆盖率。为达到 100%全使用数据流覆盖率，至少执行每个变量定义到该定义的所有使用(包括谓词使用和计算使用)的一个子路径。表 C.14 的测试集满足此要求。

表 C.14 全谓词使用测试的测试用例

测试用例	全谓词使用				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	Is_Complex	7→9	7-8-9	TCOVER3	1	2	1	FALSE	−1	−1
2	Discrim	1→4	1-2-3-4	TCOVER1	1	1	1	TRUE	未定义	未定义
	Is_Complex	5→9	5-8-9	TCOVER2						

C.2.4.7.3 全谓词使用测试覆盖率

根据 6.3.7.3 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(全谓词使用)}=\frac{3}{3}\times 100\%=100\%$$

因此，对于全谓词使用测试，测试覆盖项的覆盖率达到了 100%。

C.2.4.8 全使用测试

C.2.4.8.1 第 3d 步：导出测试覆盖项(TD3)

在全使用测试中，测试覆盖项是从变量定义到该定义的所有使用(包括谓词使用和计算使用)的控制流子路径。

表 C.15 所示的定义-使用对符合该准则。

表 C.15　全使用测试

测试覆盖项	全使用/全定义使用路径			
	变量	定义使用对	子路径	测试条件
TCOVER1	A	0→1	0-1	TCOND1
TCOVER2	B	0→1	0-1	TCOND2
TCOVER3	C	0→1	0-1	TCOND3
TCOVER4	A	0→10	0-1-2-3-4-6-7-8-9-10	TCOND4
TCOVER5	B	0→10	0-1-2-3-4-6-7-8-9-10	TCOND5
TCOVER6	A	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND6
TCOVER7	B	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND7
TCOVER8	Discrim	1→4	1-2-3-4	TCOND8
TCOVER9	Discrim	1→10	1-2-3-4-6-7-8-9-10	TCOND9
TCOVER10	Discrim	1→11	1-2-3-4-6-7-8-9-10-11	TCOND10
TCOVER11	Is_Complex	5→9	5-8-9	TCOND11
TCOVER12	Is_Complex	7→9	7-8-9	TCOND12
TCOVER13	R1	10→13	10-11-12-13	TCOND13
TCOVER14	R2	11→13	11-12-13	TCOND14
TCOVER15	Is_Complex	5→13	5-8-9-12-13	TCOND15
TCOVER16	Is_Complex	7→13	7-8-9-10-11-12-13	TCOND16

C.2.4.8.2　第 4d 步：导出测试用例(TD4)

通过识别控制流子路径导出测试用例，子路径是测试过程中没有执行的一个或多个测试覆盖项，确定满足子路径的输入，确定每个测试的预期结果，重复上述步骤直到达到要求的测试覆盖率。达到100%的全使用数据流覆盖至少执行一个从每个变量定义到定义的每个使用(谓词使用和计算使用)的子路径。表 C.16 的测试集满足要求。

表 C.16　全使用测试的测试用例

测试用例	全使用				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	A，B，C	0→1	0-1	TCOVER1，TCOVER2，TCOVER3	1	2	1	FALSE	−1	−1
	A，B	0→10	0-1-2-3-4-6-7-8-9-10	TCOVER4，TCOVER5						
	A，B	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOVER6，TCOVER7						

表 C.16（续）

测试用例	全使用				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	Discrim	1→4	1-2-3-4	TCOVER8	1	2	1	FALSE	−1	−1
		1→10	1-2-3-4-6-7-8-9-10	TCOVER9						
		1→11	1-2-3-4-6-7-8-9-10-11	TCOVER10						
	Is_Complex	7→9	7-8-9	TCOVER12						
	R1	10→13	10-11-12-13	TCOVER13						
	R2	11→13	11-12-13	TCOVER14						
	Is_Complex	7→13	7-8-9-10-11-12-13	TCOVER16						
2	Is_Complex	5→9	5-8-9	TCOVER11	1	1	1	TRUE	未定义	未定义
	Is_Complex	5→13	5-8-9-12-13	TCOVER15						

C.2.4.8.3 全使用测试覆盖率

根据 6.3.7.4 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(全使用)}=\frac{16}{16}\times 100\%=100\%$$

因此，对于全使用测试，测试覆盖项的覆盖率达到了 100%。

C.2.4.9 全定义-使用路径测试

C.2.4.9.1 第 3e 步：导出测试覆盖项（TD3）

为达到 100%的全定义使用路径数据流覆盖，应执行所有变量定义到使用的“简单子路径”。不同于全使用测试，应执行定义使用对之间的所有简单子路径。在表 C.17 表示的全定义-使用路径中，似乎测试项中有 0-1-4-5-9-10 和 1-4-5-9-10 两个子路径未在所有测试用例中标识出来。然而，这两个子路径是不可行（因此无法导出测试用例来执行它们）。因此，对于全定义使用路径测试，它们不被认为是“简单子路径”。

表 C.17 全定义-使用路径测试

测试覆盖项	全定义使用路径			
	变量	定义使用对	子路径	测试条件
TCOVER1	A	0→1	0-1	TCOND1
TCOVER2	B	0→1	0-1	TCOND2
TCOVER3	C	0→1	0-1	TCOND3
TCOVER4	A	0→10	0-1-2-3-4-6-7-8-9-10	TCOND4
TCOVER5	B	0→10	0-1-2-3-4-6-7-8-9-10	TCOND5

表 C.17（续）

测试覆盖项	全定义使用路径			
	变量	定义使用对	子路径	测试条件
TCOVER6	A	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND6
TCOVER7	B	0→11	0-1-2-3-4-6-7-8-9-10-11	TCOND7
TCOVER8	Discrim	1→4	1-2-3-4	TCOND8
TCOVER9	Discrim	1→10	1-2-3-4-6-7-8-9-10	TCOND9
TCOVER10	Discrim	1→11	1-2-3-4-6-7-8-9-10-11	TCOND10
TCOVER11	Is_Complex	5→9	5-8-9	TCOND11
TCOVER12	Is_Complex	7→9	7-8-9	TCOND12
TCOVER13	R1	10→13	10-11-12-13	TCOND13
TCOVER14	R2	11→13	11-12-13	TCOND14
TCOVER15	Is_Complex	5→13	5-8-9-12-13	TCOND15
TCOVER16	Is_Complex	7→13	7-8-9-10-11-12-13	TCOND16

C.2.4.9.2 第 4e 步：导出测试用例（TD4）

导出全定义-使用路径的测试用例。全使用测试导出的同一组测试用例，对于本例中的全定义-使用路径测试，也可以达到最大的测试覆盖项覆盖率，如表 C.18 所示。

表 C.18 全定义-使用路径测试的测试用例

测试用例	全定义使用路径				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
1	A，B，C	0→1	0－1	TCOVER1，TCOVER2，TCOVER3	1	2	1	FALSE	－1	－1
	A，B	0→10	0-1-2-3-4-6-7-8-9-10	TCOVER4，TCOVER5						
		0→11	0-1-2-3-4-6-7-8-9-10-11	TCOVER6，TCOVER7						
	Discrim	1→4	1-2-3-4	TCOVER8						
		1→10	1-2-3-4-6-7-8-9-10	TCOVER9						
	Is_Complex	1→11	1-2-3-4-6-7-8-9-10-11	TCOVER10						
		7→9	7-8-9	TCOVER12						
	R1	10→13	10-11-12-13	TCOVER13						
	R2	11→13	11-12-13	TCOVER14						
	Is_Complex	7→13	7-8-9-10-11-12-13	TCOVER16						

表 C.18（续）

测试用例	全定义使用路径				输入			预期结果		
	变量	定义使用对	子路径	测试覆盖项	A	B	C	Is_Complex	R1	R2
2	Is_Complex	5→9	5-8-9	TCOVER11	1	1	1	TRUE	未定义	未定义
	Is_Complex	5→13	5-8-9-12-13	TCOVER15						

C.2.4.9.3 全定义-使用路径测试覆盖

根据 6.3.7.5 提供的公式和上述导出的测试覆盖项：

$$覆盖率_{(全定义使用)}=\frac{16}{16}\times 100\%=100\%$$

因此，对于全定义使用-路径测试，测试覆盖项的覆盖率达到了 100%。

C.2.4.9.4 第 5 步：汇集测试集（TD5）

全使用测试导出的测试用例覆盖了所有的测试覆盖项，将其作为识别测试集的例子。所有 Is_Complex 为 FALSE 测试用例组合成一个测试集，为 TRUE 的测试用例组合成另一个测试集：

TS1：测试用例 1。

TS2：测试用例 2。

C.2.4.9.5 第 6 步：导出测试规程（TD6）

所有测试集按照执行的顺序组合成一个测试规程，具体如下：

TP1：按照测试集指定的顺序覆盖 TS1 中的所有测试用例，再覆盖 TS2 中的所有测试用例。

附 录 D
（资料性附录）
基于经验的测试设计技术的应用指南和示例

D.1 基于经验的测试设计技术的使用规则和示例

本附录为5.4和6.4的要求提供了指南。本附录展示了在一个例子中运用基于经验的测试设计技术的情况。例子符合GB/T 38634.2定义的测试设计和实施过程。

D.2 基于经验的测试设计技术示例

D.2.1 错误猜测

D.2.1.1 介绍

错误猜测的目的是导出覆盖了可能错误的测试用例集，使用测试人员在此前测试项中的知识和经验。导出满足每种错误类型的测试用例，由测试人员确定当前的测试项可能存在的错误类型。该技术通常是在其他基于规格说明的测试设计技术（如等价类划分和边界值分析）后使用，用于补充这些技术的目标错误类型。

D.2.1.2 规格说明

考虑测试项——generate_grading，测试依据如下：

组件收到一个考试成绩（不高于75分）输入和一个课程作业（c/w）成绩（不高于25分）输入，根据输入输出一个“A”到“D”的课程等级。等级是通过计算考试和课程作业成绩的总和得到的，具体如下：

大于或等于70 ——‘A’
大于或等于50，小于70 ——‘B’
大于或等于30，小于50 ——‘C’
小于30 ——‘D’

在检测到无效输入（例如分数超出了预期的范围）时生成一个出错信息（‘FM’）表示。所有的输入都用整数。

D.2.1.3 第1步：识别特征集（TD1）

因为测试依据中只有一个测试项，只需要定义一个特征：

FS1：generate_grading功能

D.2.1.4 第2步：导出测试条件（TD2）

基于过去其他测试项中类似错误的知识和经验导出可能存在的错误类型列表，并以此导出测试条件。对于generate_grading功能，可以导出如下测试条件：

TCOND1：输入NULL （FS1）
TCOND2：输入0 （FS1）
TCOND3：输入负数 （FS1）
TCOND4：反向输入 （FS1）

TCOND5：输入非常大的数字(例如 10 位) (FS1)
TCOND6：输入非常长的字符串(例如 10 个字符) (FS1)

D.2.1.5 第 3 步：导出测试覆盖项(TD3)

缺陷的一般类型是测试覆盖项(即测试覆盖项和测试条件一致)。因此，可以定义下面的测试覆盖项：

TCOVER1：输入 NULL L (TCOND1)
TCOVER2：输入 0 (TCOND2)
TCOVER3：输入负数 (TCOND3)
TCOVER4：反向输入 (TCOND4)
TCOVER5：输入非常大的数字(例如 10 位) (TCOND5)
TCOVER6：输入非常长的字符串(例如 10 个字符) (TCOND6)

D.2.1.6 第 4 步：导出测试用例(TD4)

测试用例可依据不同的缺陷类型导出，根据选择的缺陷类型确定满足测试用例参数的输入，确定预期结果，重复上述步骤直到测试用例包括了所有测试覆盖项。对于本例，导出测试用例如表 D.1～表 D.4。

表 D.1 错误猜测的测试用例(一)

测试用例	1	2	3	4
输入(考试成绩)	NULL	25	NULL	0
输入(课程作业成绩)	20	NULL	NULL	20
总成绩(经计算)	20	25	NULL	20
测试覆盖项	TCOVER1 考试成绩	TCOVER1 课程作业成绩	TCOVER1 考试和课程作业成绩	TCOVER2 考试成绩
预期结果值	‘FM’	‘FM’	‘FM’	‘FM’

表 D.2 错误猜测的测试用例(二)

测试用例	5	6	7	8
输入(考试成绩)	25	0	−25	25
输入(课程作业成绩)	0	0	20	−25
总成绩(经计算)	25	0	−5	0
测试覆盖项	TCOVER2 课程作业成绩	TCOVER2 考试和课程作业成绩	TCOVER3 考试成绩	TCOVER3 课程作业成绩
预期结果值	‘FM’	‘FM’	‘FM’	‘FM’

表 D.3 错误猜测测试用例(三)

测试用例	9	10	11	12
输入(考试成绩)	−25	20	1234567890	25
输入(课程作业成绩)	−50	55	20	1234567890
总标志(经计算)	−75	75	1234567910	1234567915
测试覆盖项	TCOVER3 考试和课程作业成绩	TCOVER4 考试和课程作业成绩	TCOVER5 考试成绩	TCOVER5 课程作业成绩
预期结果值	'FM'	'FM'	'FM'	'FM'

表 D.4 错误猜测测试用例(四)

测试用例	13	14	15	16
输入(考试成绩)	1234567890	abcdefghij	25	abcdefghij
输入(课程作业成绩)	1234567890	20	abcdefghij	abcdefghij
总标志(经计算)	2469135780	NULL	NULL	NULL
测试覆盖项	TCOVER5 考试和课程作业成绩	TCOVER6 考试成绩	TCOVER6 课程作业成绩	TCOVER6 考试和课程作业成绩
预期结果值	'FM'	'FM'	'FM'	'FM'

D.2.1.7 第 5 步:汇集测试集(TD5)

所有测试用例都是无效的,均会出现出错消息,可归入一个测试集,如下所示:

TS1:测试用例集 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16。

D.2.1.8 第 6 步:导出测试规程(TD6)

只需要一个规程,如下所示:

TP1:按顺序覆盖 TS1 中测试用例。

D.2.1.9 错误猜测的测试覆盖率

如 6.4.1 所述,尚无错误猜测覆盖率的计算方法。

附 录 E
（资料性附录）
可交换的测试设计技术的应用指南和示例

E.1 可交换的测试设计技术的应用指南和示例

虽然本部分将技术分为基于结构、基于规格说明或基于经验的，实际上一些技术可以交换使用（见 5.1）。下面的例子说明了分支测试（通常是指基于结构的技术）如何应用于基于规格说明的测试，证明了技术可以交换使用。

E.2 分支测试作为以规格说明为基础的技术

E.2.1 规格说明

考虑下面的例子规格说明，定义了一个登录功能，需要输入用户名和密码来判断用户是否有效：

组件要求输入用户名和密码。用户必须输入正确的用户名和相应的密码才能登录系统。用户有 3 次输入用户名和密码的机会，每次不能超过 20 s。如果用户输入用户名和密码错误 3 次或者超过 20 s，则会锁定系统，不准许有任何进一步的登录尝试。

组件的控制流图如图 E.1 所示。

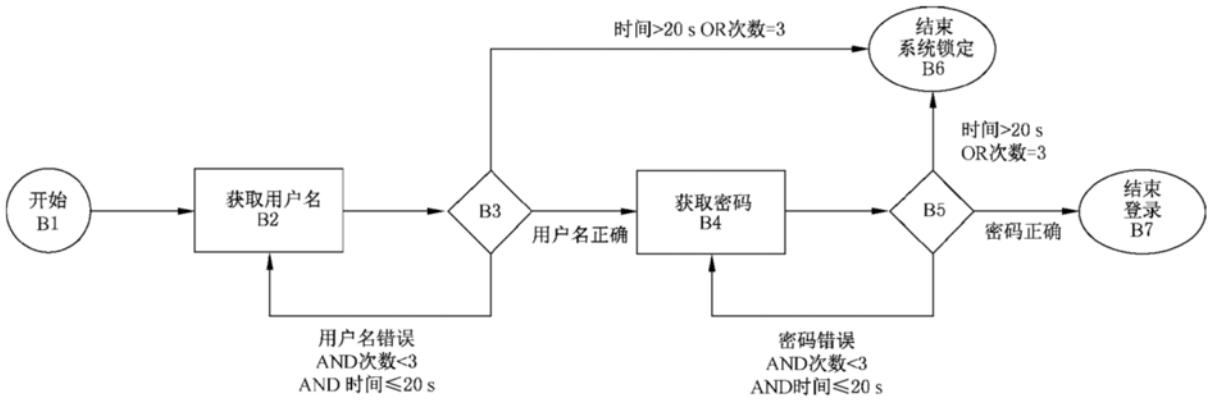


图 E.1 登录功能的控制流图

E.2.2 第 1 步：识别特征集（TD1）

测试基准中只定义了一个测试项，只需要定义一个特征集：

FS1：登录功能

E.2.3 第 2 步：导出测试条件（TD2）

控制流图提供了很好的功能概述，帮助识别分支测试的测试条件。控制流程图中的方块表示组件中的程序源代码段，菱形代表判定。菱形指向判定结果。可能的控制转移是：

B1→B2	B3→B4	B5→B4
B2→B3	B3→B6	B5→B6

B3→B2　　B4→B5　　B5→B7

对于分支覆盖，测试条件是分支(边)，在控制流图用箭头表示。共有 9 个，如下所示：

TCOND1：B1→B2　　(FS1)
TCOND2：B2→B3　　(FS1)
TCOND3：B3→B2　　(FS1)
TCOND4：B3→B4　　(FS1)
TCOND5：B3→B6　　(FS1)
TCOND6：B4→B5　　(FS1)
TCOND7：B5→B4　　(FS1)
TCOND8：B5→B6　　(FS1)
TCOND9：B5→B7　　(FS1)

E.2.4　第 3 步：导出测试覆盖项(TD3)

对于分支覆盖，测试覆盖项是控制流图中的分支，与测试条件相同。在本例中，分支覆盖有 9 个测试覆盖项，具体如下：

TCOVER1：B1→B2　　(TCOND1)
TCOVER2：B2→B3　　(TCOND2)
TCOVER3：B3→B2　　(TCOND3)
TCOVER4：B3→B4　　(TCOND4)
TCOVER5：B3→B6　　(TCOND5)
TCOVER6：B4→B5　　(TCOND6)
TCOVER7：B5→B4　　(TCOND7)
TCOVER8：B5→B6　　(TCOND8)
TCOVER9：B5→B7　　(TCOND9)

E.2.5　第 4 步：导出测试用例(TD4)

分支测试通过识别控制流子路径导出测试用例，子路径是测试过程中没有执行的一个或多个分支(测试覆盖项)，确定满足子路径的输入，确定每个测试的预期结果，重复上述步骤直到达到要求的测试覆盖率。对于分支覆盖，任何单独的测试用例满足一个子路径，因此可能会有许多潜在的判定和分支。

考虑执行子路径 B1→B2→B3→B4→B5→B7 的测试用例。导出测试用例：第一次输入有效的用户名和对应的密码。路径执行了 9 个分支中的 5 个分支，实现了 56%的覆盖率(和判定覆盖不同)。

现在考虑执行子路径 B1→B2→B3→B2→B3→B4→B5→B4→B5→B6 的测试用例。生成子路径：用户名无效、用户名有效但密码错误、用户名有效但正确的密码输入时间超过 20 秒。现在除了 B3→B6 以外的所有分支均被覆盖。该子路径可以被覆盖，例如，输入三次无效的用户名或者等待输入有效的用户过长时间。现在覆盖了所有的分支，包括所有判定。需要注意的是，一些判定条件没有覆盖。

覆盖所有控制流子路径的测试用例如表 E.1 所示。

表 E.1　登录功能的测试用例

测试用例	输入				子路径	测试覆盖项	预期结果
	用户名	用户名等待时间	密码	密码等待时间			
1	Andy	≤20	Warhol	≤20	B1→B2→B3→B4→B5→B7	TCOVER1，TCOVER2，TCOVER4，TCOVER6，TCOVER9	登录
2	InVaLiD Andy	≤20 ≤ 0	— InVAliD Warhol	— ≤20 ≥21	B1→B2→B3→B2→B3→B4→B5→B4→B5→B6	TCOVER1，TCOVER2，TCOVER3，TCOVER4，TCOVER6，TCOVER7，TCOVER8	系统锁定
3	Brandy	≥21		—	B1→B2→B3→B6	TCOVER1，TCOVER2，TCOVER5	系统锁定

E.2.6　第 5 步：汇集测试集（TD5）

只需要三个测试用例，可以选择这三个测试用例组合成一个测试集：

TS1：测试用例 1，2 和 3。

E.2.7　第 6 步：导出测试规程（TD6）

只有三个测试用例和一个测试集，可以选择只定义一个测试规程：

TP1：按照测试集的指定顺序覆盖 TS1 中的所有测试用例。

E.2.8　分支测试覆盖率

根据 6.3.2 提供的公式和上述导出的测试覆盖项计算覆盖率：

$$覆盖率_{(分支)}=\frac{9}{9}\times 100\%=100\%$$

因此，对于分支测试，测试覆盖项的覆盖率达到了 100%。

附 录 F
（资料性附录）
测试设计技术覆盖有效性

本部分未提供测试设计技术或测试完成准则（也称为测试充分准则）选择的指南，分别从第5章和第6章中选择测试设计技术或测试完成准则。没有指南的主要原因是：哪些技术和准则是最有效的，尚无公认的共识。唯一的共识是测试设计技术或测试完成准则选择要有所变化，它依赖于许多因素，如风险、临界性、应用领域和成本。对测试用例设计和测量技术的研究到目前为止没有明确的结果，下面提供了一些理论性的结果，这些结果没有考虑成本。

没有要求选择相应的测试用例设计和测试覆盖测量方法。基于规格说明的测试设计技术对检测遗漏错误是有效的，而基于结构的测试设计技术只能检测已知的错误。所以测试计划通常使用边界值分析导出初始的测试用例集，同时要求达到100％的分支覆盖率。不同的方法可能会导致分支测试导出的测试用例用以补充边界值分析测试用例集遗漏的分支。

理想的情况下，选择测试完成准则的测试覆盖率应尽可能达到100％。对测试覆盖级别的严格定义测试覆盖率有时会导致该测试覆盖级别不具有可操作性，然而，本部分的第6章定义了在计算测试覆盖率时，允许只考虑可行的覆盖项，这样就可以达到100％覆盖率的目标。

100％（且仅有100％）的测试完成准则可能按顺序涉及一些准则，其中准则显式包含了其他准则。一个准则包含了另一个准则，对于所有测试用例及其测试依据和测试套件，如果满足第一个准则，则也满足第二准则。例如，分支覆盖包含语句覆盖，因为如果达到了分支覆盖率（100％），则也会达到100％的语句覆盖率。

注意这里描述的“包含”关系和测试覆盖准则严格关联（而不是测试设计技术），所以只提供测试设计技术相对有效性的间接指标。

并非所有的测试覆盖准则都通过包含排序相关联，基于规格说明和基于结构的准则完全不相关的。准则的偏序关系适用于基于结构的测试设计技术，如图F.1所示，从一个准则指向另一准则表示第一准则包含第二准则。当一个测试覆盖准则没有出现在偏序关系中，则说明和其他准则没有包含关系。

尽管很直观，但在使用包含关系选择测试完成准则前需要考虑受到的限制：

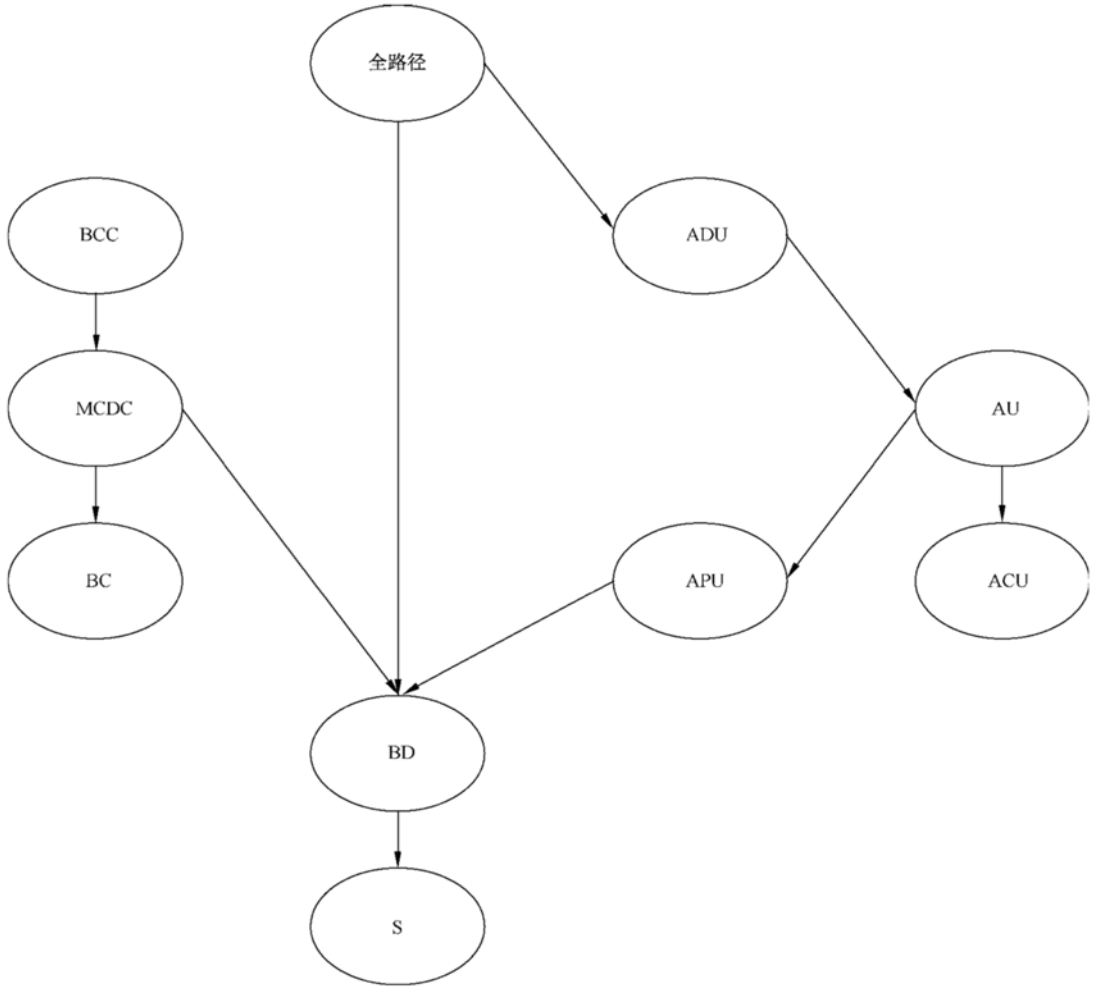
——首先，只和可用的测试完成准则的子集相关，在该子集中没有提供有效性的指示，因此仍需要考虑上图中没有显示的其他准则。

——其次，一个准则包含另一个准则的包含关系没有提供对数量的测量，因此没有提供相对成本效益的措施。

——第三，偏序只适用于单个准则，如果推荐使用多个准则，则至少有一个功能性准则和一个结构性准则。

——最后，也是最重要的，包含关系不一定根据测试完成准则发现故障（测试效率）对其排序。例如，100％的路径覆盖（当达到时）对于某些测试项可能比其包含的准则有效率，如关注数据流的准则。这是因为一些错误对数据敏感，简单执行有错误的路径不能发现错误，但需要变量取特殊值（例如，对整数变量“无保护的”除法可能会错误地包含在一个测试项中，只有当该变量取负值时才会失败）。数据流准则可以集中测试测试项行为的这些方面，从而增加了发现这类错误的概率。在一些情况下，通过测试特定准则所需要的路径子集提高测试的有效性，用更多的测试用例满足该子集。

包含关系高度依赖于全覆盖准则的定义，虽然上图符合第5章中的定义，但可能不适用于其他地方使用的替代定义。



说明：

AU ——全使用覆盖；

ACU ——全计算使用覆盖；

ADU ——全定义使用路径覆盖；

APU ——全谓词使用覆盖；

BD ——分支/判定覆盖；

BC ——分支条件覆盖；

BCC ——分支条件组合覆盖；

MCDC ——修正条件判定覆盖；

S ——语句覆盖。

图 F.1 结构测试覆盖准则的偏序关系

附 录 G
（资料性附录）
测试设计技术对照

本附录描述了本部分和 GB/T 15532—2008 的测试设计技术的映射关系。

表 G.1 本部分与 GB/T 15532—2008 测试设计技术映射

本部分		GB/T 15532—2008	
5.2.1	等价类划分	A.2.2.2	等价类划分
5.2.3	边界值分析	A.2.2.3	边界值分析
5.2.6	判定表测试	A.2.2.4	判定表
5.2.8	状态转移测试	—	—
5.2.7	因果图	A.2.2.5	因果图
5.2.4	语法测试	—	—
5.3.1	语句测试	A.2.3.1 a)	语句测试
5.3.2	分支测试	A.2.3.1 b)	分支/判定测试
5.3.3	判定测试		
5.3.7	数据流测试	A.2.3.2	数据流测试
5.3.4	分支条件测试	A.2.3.1 c)	分支条件测试
5.3.5	分支条件组合测试	A.2.3.1 d)	分支条件组合测试
5.3.6	修正条件判定覆盖测试	—	—
5.2.10	随机测试	A.2.2.6	随机测试
5.4	基于经验的测试	A.2.2.7	猜错法

参 考 文 献

[1] GB/T 25000.10—2016 系统与软件工程 系统与软件质量要求和评价(SQuaRE) 第10部分:系统与软件质量模型

[2] ISO/IEC TR 19759 Software Engineering—Guide to the Software Engineering Body of Knowledge (SWEBOK)

[3] ISO/IEC 25030 Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Quality requirements

[4] BS 7925-2:1998 Software testing—Software component testing

[5] BATH, G. and McKAY J., 2008. The Software Test Engineer's Handbook. O'Reilly Media, Inc.

[6] BEIZER, B., 1995. Black Box Testing. Techniques for Functional Testing of Software and Systems, John Wiley & Sons Inc.

[7] Non-Functional Testing. British Computer Society Special Interest Group in Software Testing, [September 2011].

[8] BURNSTEIN, I., 2003. Practical Software Testing: A Process—Oriented Approach. Springer—Verlag.

[9] COPELAND, L., 2004. A Practitioner's Guide to Software Test Design. Artech House, Inc.

[10] CHO, C. K., 1987. Quality Programming. Wiley.

[11] CHOW, T. S., 1978. Testing Software Design Modelled by Finite—State Machines. In: IEEE Transactions on Software Engineering, Vol. SE-4(3).

[12] CRAIG, R. and JASKIEL, S., 2002. Systematic Software Testing. Artech House Inc.

[13] DESIKAN, S. and RAMESH, G., 2007. Software Testing: Principles and Practices, Pearson Education.

[14] GRINDAL, M., OFFUTT, J. and ANDLER, S., 2005. Combination Testing Strategies: A Survey. In: Software Testing, Verification and Reliability, John Wiley & Sons Ltd., 15, pp. 167-199.

[15] GROCHTMANN, M. and GRIMM, Grimm, K., 1993. Classification Trees for Partition Testing. In: Software Testing, Verification & Reliability, Wiley, 3(2), pp. 63-82.

[16] HASS, A. M. Jonassen, 2008. Guide to Advanced Software Testing. Artech House.

[17] KANER, C., 1998. Testing Computer Software. TAB Books Inc.

[18] KERNIGHAN, B. W. and RICHIE, D. M., 1998. The C Programming Language. Prentice—Hall Software Series.

[19] KIT, E., 1995. Software Testing in the Real World: Improving the Process. ACM Press.

[20] MANDL, R., 1985. Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing. In Communications of the ACM, 28(10), pp. 1054-1058.

[21] MYERS, G., 1979. The Art of Software Testing. John Wiley & Sons Inc.

[22] NURSIMULU, K. and PROBERT, R. L., 1995. Cause—Effect Graphing Analysis and Validation of Requirements. In Proceedings of CASCON'1995.

[23] RTCA/DO-178C, Software Considerations in Airborne Systems and Equipment Certification. RTCA, Inc. 2011.

[24] REID, S., 1996. Popular Misconceptions in Module Testing. In Proceedings of the Software Testing Conference (STC), Washington DC.

中 华 人 民 共 和 国
国 家 标 准
系统与软件工程　软件测试
第 4 部分：测试技术
GB/T 38634.4—2020
*
中 国 标 准 出 版 社 出 版 发 行
北京市朝阳区和平里西街甲 2 号(100029)
北京市西城区三里河北街 16 号(100045)
网址：www.spc.org.cn
服务热线：400-168-0010
2020 年 4 月第一版
*
书号：155066・1-64592

GB/T 38634.4-2020