测试用例设计的问题

测试用例（Test Case）是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。

测试用例（Test Case）目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

不同类别的软件，测试用例是不同的。不同于诸如系统、工具、控制、游戏软件，管理软件的用户需求更加不统一，变化更大、更快。笔者主要从事企业管理软件的测试。因此我们的做法是把测试数据和测试脚本从测试用例中划分出来。测试用例更趋于是针对软件产品的功能、业务规则和业务处理所设计的测试方案。对软件的每个特定功能或运行操作路径的测试构成了一个个测试用例。

确定测试用例之所以很重要，原因有以下几方面。

测试用例构成了设计和制定测试过程的基础。
测试的“深度”与**测试用例的**数量成比例。由于每个测试用例反映不同的场景、条件或经由产品的事件流，因而，随着测试用例数量的增加，您对产品质量和测试流程也就越有信心。
判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和/或执行的**测试用例的**数量为依据的。类似下面这样的说明：“95 % 的关键测试用例已得以执行和验证”，远比“我们已完成 95 % 的测试”更有意义。
测试工作量与**测试用例的**数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。
测试设计和开发的类型以及所需的资源主要都受控于测试用例。
测试用例通常根据它们所关联关系的测试类型或测试需求来分类，而且将随类型和需求进行相应地改变。最佳方案是为每个测试需求至少编制两个测试用例：

·一个测试用例用于证明该需求已经满足，通常称作正面测试用例；
·另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

一、测试用例是软件测试的核心

软件测试的重要性是毋庸置疑的。但如何以最少的人力、资源投入，在最短的时间内完成测试，发现软件系统的缺陷，保证软件的优良品质，则是软件公司探索和追求的目标。每个软件产品或软件开发项目都需要有一套优秀的测试方案和测试方法。

影响软件测试的因素很多，例如软件本身的复杂程度、开发人员（包括分析、设计、编程和测试的人员）的素质、测试方法和技术的运用等等。因为有些因素是客观存在的，无法避免。有些因素则是波动的、不稳定的，例如开发队伍是流动的，有经验的走了，新人不断补充进

来；一个具体的人工作也受情绪等影响，等等。如何保障软件测试质量的稳定？有了测试用例，无论是谁来测试，参照测试用例实施，都能保障测试的质量。可以把人为因素的影响减少到最小。即便最初的测试用例考虑不周全，随着测试的进行和软件版本更新，也将日趋完善。

因此**测试用例的**设计和编制是软件测试活动中最重要的。测试用例是测试工作的指导，是软件测试的必须遵守的准则。更是软件测试质量稳定的根本保障。

二、编制测试用例

着重介绍一些编制**测试用例的**具体做法。

1、测试用例文档

编写测试用例文档应有文档模板，须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。
软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位，形成一个测试用例文档，但并不是绝对的。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息：用例编号、用例名称、测试等级、入口准则、验证步骤、期望结果（含判断标准）、出口准则、注释等。以上内容涵盖了**测试用例的**基本元素：测试索引，测试环境，测试输入，测试操作，预期结果，评价标准。

2、测试用例的设置

我们早期的测试用例是按功能设置用例。后来引进了路径分析法，按路径设置用例。目前演变为按功能、路径混合模式设置用例。

按功能测试是最简捷的，按用例规约遍历测试每一功能。

对于复杂操作的程序模块，其各功能的实施是相互影响、紧密相关、环环相扣的，可以演变出数量繁多的变化。没有严密的逻辑分析，产生遗漏是在所难免。路径分析是一个很好的方法，其最大的优点是在于可以避免漏测试。

但路径分析法也有局限性。在一个非常简单字典维护模块就存在十余条路径。一个复杂的模块会有几十到上百条路径是不足为奇的。笔者以为这是路径分析比较合适的使用规模。若一个子系统有十余个或更多的模块，这些模块相互有关联。再采用路径分析法，其路径数量成几何级增长，达5位数或更多，就无法使用了。那么子系统模块间的测试路径或测试用例还是要靠传统方法来解决。这是按功能、路径混合模式设置用例的由来。

3、设计测试用例

测试用例可以分为基本事件、备选事件和异常事件。设计基本事件的用例，应该参照用例规约（或设计规格说明书），根据关联的功能、操作按路径分析法设计测试用例。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达 100%。

设计备选事件和异常事件的用例，则要复杂和困难得多。例如，字典的代码是唯一的，不允许重复。测试需要验证：字典新增程序中已存在有关字典代码的约束，若出现代码重复必须报错，并且报错文字正确。往往在设计编码阶段形成的文档对备选事件和异常事件分析描述不够详尽。而测试本身则要求验证全部非基本事件，并同时尽量发现其中的软件缺陷。

可以采用软件测试常用的基本方法：等价类划分法、边界值分析法、错误推测法、因果图法、逻辑覆盖法等设计测试用例。视软件的不同性质采用不同的方法。如何灵活运用各种基本方法来设计完整的测试用例，并最终实现暴露隐藏的缺陷，全凭测试设计人员的丰富经验和精心设计。

三、测试用例在软件测试中的作用

1、指导测试的实施

测试用例主要适用于集成测试、系统测试和回归测试。在实施测试时测试用例作为测试的标准，测试人员一定要按照测试用例严格按用例项目和测试步骤逐一实施测试。并对测试情况记录在测试用例管理软件中，以便自动生成测试结果文档。

根据**测试用例的**测试等级，集成测试应测试那些用例，系统测试和回归测试又该测试那些用例，在设计测试用例时都已作明确规定，实施测试时测试人员不能随意作变动。

2、规划测试数据的准备

在我们的实践中测试数据是与测试用例分离的。按照测试用例配套准备一组或若干组测试原始数据，以及标准测试结果。尤其象测试报表之类数据集的正确性，按照测试用例规划准备测试数据是十分必须的。
除正常数据之外，还必须根据测试用例设计大量边缘数据和错误数据。

3、编写测试脚本的"设计规格说明书"

为提高测试效率，软件测试已大力发展自动测试。自动测试的中心任务是编写测试脚本。如果说软件工程中软件编程必须有设计规格说明书，那么测试脚本的设计规格说明书就是测试用例。

4、评估测试结果的度量基准

完成测试实施后需要对测试结果进行评估，并且编制测试报告。判断软件测试是否完成、衡量测试质量需要一些量化的结果。例：测试覆盖率是多少、测试合格率是多少、重要测试合格率是多少，等等。以前统计基准是软件模块或功能点，显得过于粗糙。采用测试用例作度量基准更加准确、有效。

5、分析缺陷的标准

通过收集缺陷，对比测试用例和缺陷数据库，分析确证是漏测还是缺陷复现。漏测反映了**测试用例的**不完善，应立即补充相应测试用例，最终达到逐步完善软件质量。而已有相应测试用例，则反映实施测试或变更处理存在问题。

四、相关问题

1、测试用例的评审

测试用例是软件测试的准则，但它并不是一经编制完成就成为准则。测试用例在设计编制过程中要组织同级互查。完成编制后应组织专家评审，需获得通过才可以使用。评审委员会可由项目负责人、测试、编程、分析设计等有关人员组成，也可邀请客户代表参加。

2、测试用例的修改更新

测试用例在形成文档后也还需要不断完善。主要来自三方面的缘故：第一、在测试过程中发现设计测试用例时考虑不周，需要完善；第二、在软件交付使用后反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成；第三、软件自身的新增功能以及软件版本的更新，测试用例也必须配套修改更新。

一般小的修改完善可在原测试用例文档上修改，但文档要有更改记录。软件的版本升级更新，测试用例一般也应随之编制升级更新版本。

3、测试用例的管理软件

运用测试用例还需配备测试用例管理软件。它的主要功能有三个：第一、能将测试用例文档的关键内容，如编号、名称等等自动导入管理数据库，形成与测试用例文档完全对应的记录；第二、可供测试实施时及时输入测试情况；第三、最终实现自动生成测试结果文档，包含各测试度量值，测试覆盖表和测试通过或不通过的测试用例清单列表。

有了管理软件，测试人员无论是编写每日的测试工作日志、还是出软件测试报告，都会变得轻而易举。

五、测试用例的设计

（一）白盒技术

白盒测试是结构测试，所以被测对象基本上是源程序，以程序的内部逻辑为基础设计测试用例。

1、逻辑覆盖

程序内部的逻辑覆盖程度，当程序中有循环时，覆盖每条路径是不可能的，要设计使覆盖程度较高的或覆盖最有代表性的路径的测试用例。下面根据图 7-1 所示的程序，分别讨论几种常用的覆盖技术。

(1)语句覆盖

为了个提高发现错误的可能性，在测试时应该执行到程序中的每一个语句。语句覆盖是指设计足够的测试用例，使被测试程序中每个语句至少执行一次。

如图 7-1 是一个被测试程序流程图：

(2)判定覆盖

判定覆盖指设计足够的测试用例，使得被测程序中每个判定表达式至少获得一次“真”值和“假”值，从而使程序的每一个分支至少都通过一次，因此判定覆盖也称分支覆盖。

(3)条件覆盖

条件覆盖是指设计足够的测试用例，使得判定表达式中每个条件的各种可能的值至少出现一次。

(4)判定/条件测试

该覆盖标准指设计足够的测试用例，使得判定表达式的每个条件的所有可能取值至少出现一次，并使每个判定表达式所有可能的结果也至少出现一次。

(5)条件组合覆盖

条件组合覆盖是比较强的覆盖标准，它是指设计足够的测试用例，使得每个判定表达式中条件的各种可能的值的组合都至少出现一次。

(6)路径覆盖

路径覆盖是指设计足够的测试用例，覆盖被测程序中所有可能的路径。

在实际的逻辑覆盖测试中，一般以条件组合覆盖为主设计测试用例，然后再补充部分用例，以达到路径覆盖测试标准。

2.循环覆盖

3.基本路径测试

（二）黑盒技术

1.等价类划分

(1)划分等价类。

①如果某个输入条件规定了取值范围或值的个数。则可确定一个合理的等价类(输入值或数在此范围内)和两个不合理等价类(输入值或个数小于这个范围的最小值或大于这个范围的最大值)。

②如果规定了输入数据的一组值，而且程序对不同的输入值做不同的处理，则每个允许输入值是一个合理等价类，此处还有一个不合理等价类(任何一个不允许的输入值)。

③如果规定了输入数据必须遵循的规则，可确定一个合理等价类(符合规则)和若干个不合理

等价类(从各种不同角度违反规则)。
④如果已划分的等价类中各元素在程序中的处理方式不同，则应将此等价类进一步划分为更小的等价类。
(2)确定测试用例。
①为每一个等价类编号。
②设计一个测试用例，使其尽可能多地覆盖尚未被覆盖过的合理等价类。重复这步，直到所有合理等价类被测试用例覆盖。
③设计一个测试用例，使其只覆盖一个不合理等价类。
2.边界值分析
使用边界值分析方法设计测试用例时一般与等价类划分结合起来。但它不是从一个等价类中任选一个例子作为代表，而是将测试边界情况作为重点目标，选取正好等于、刚刚大于或刚刚小于边界值的测试数据。
(1)如果输入条件规定了值的范围，可以选择正好等于边界值的数据作为合理的测试用例，同时还要选择刚好越过边界值的数据作为不合理的测试用例。如输入值的范围是[1，100]，可取 0，1，100，101 等值作为测试数据。
(2)如果输入条件指出了输入数据的个数，则按最大个数、最小个数、比最小个数少 1、比最大个数多 1 等情况分别设计测试用例。如，一个输入文件可包括 1--255 个记录，则分别设计有 1 个记录、255 个记录，以及 0 个记录的输入文件的测试用例。
(3)对每个输出条件分别按照以上原则(1)或(2)确定输出值的边界情况。如，一个学生成绩管理系统规定，只能查询 95--98 级大学生的各科成绩，可以设计测试用例，使得查询范围内的某一届或四届学生的学生成绩，还需设计查询 94 级、99 级学生成绩的测试用例(不合理输出等价类)。
由于输出值的边界不与输入值的边界相对应，所以要检查输出值的边界不一定可能，要产生超出输出值之外的结果也不一定能做到，但必要时还需试一试。
(4)如果程序的规格说明给出的输入或输出域是个有序集合(如顺序文件、线形表、链表等)，则应选取集合的第一个元素和最后一个元素作为测试用例。
3.错误推测
在测试程序时，人们可能根据经验或直觉推测程序中可能存在的各种错误，从而有针对性地编写检查这些错误的测试用例，这就是错误推测法。
4.因果图
等价类划分和边界值方法分析方法都只是孤立地考虑各个输入数据的测试功能，而没有考虑多个输入数据的组合引起的错误。
5.综合策略
每种方法都能设计出一组有用例子，用这组例子容易发现某种类型的错误，但可能不易发现另一类型的错误。因此在实际测试中，联合使用各种测试方法，形成综合策略，通常先用黑盒法设计基本的测试用例，再用白盒法补充一些必要的测试用例。

六、测试用例设计的误区

·能发现到目前为止没有发现的缺陷的用例是好的用例；

首先要申明，其实这句话是十分有道理的，但我发现很多人都曲解了这句话的原意，一心要设计出发现“难于发现的缺陷”而陷入盲目的片面中去，忘记了测试的目的所在，这是十分可

怕的。我倾向于将测试用例当作一个集合来认识，对它的评价也只能对**测试用例的**集合来进行，测试本身是一种“V&V”的活动，测试 需要保证以下两点：

程序做了它应该做的事情
程序没有做它不该做的事情
因此，作为测试实施依据的测试用例，必须要能完整覆盖测试需求，而不应该针对单个的测试用例去评判好坏。

·测试用例应该详细记录所有的操作信息，使一个没有接触过系统的人员也能进行测试；

不知道国内有没有公司真正做到这点，或者说，不知道有国内没有公司能够将每个测试用例都写得如此详细。在我的测试经历中，对测试用例描述的详细和复杂程度 也曾有过很多的彷徨。写得太简单吧，除了自己没人能够执行，写得太详细吧，消耗在测试用例维护（别忘了，测试用例是动态的，一旦测试环境、需求、设计、实 现发生了变化，测试用例都需要相应发生变化）上的时间实在是太惊人，在目前国内大部分软件公司的测试资源都不足的情况下，恐怕很难实现。但我偏偏就能遇到 一些这样的老总或者是项目负责人，甚至是测试工程师本身，全然不顾实际的资源情况，一定要写出“没有接触过系统的人员也能进行测试”的用例。

在讨论这个问题之前，我们可以先考虑一下测试的目的。测试的目的是尽可能发现程序中存在的缺陷，测试活动本身也可以被看作是一个 Project，也需要在 给定的资源条件下尽可能达成目标，根据我个人的经验，大部分的国内软件公司在测试方面配备的资源都是不足够的，因此我们必须在测试计划阶段明确测试的目 标，一切围绕测试的目标进行。

除了资源上的约束外，**测试用例的**详细程度也需要根据需要确定。如果**测试用例的**执行者、测试用例设计者、测试活动相关人对系统了解都很深刻，那测试用例就没有必要太详细了，文档**的作用**本来就在于沟通，只要能达到沟通的目的就 OK。在我担任测试经理的项目中，在测试计划阶段，一般给予测试设计 30% - 40%左右的时间，测试设计工程师能够根据项目的需要自行确定用例的详细程度，在**测试用例的**评审阶段由参与评审的相关人对其把关。

·测试用例设计是一劳永逸的事情；

这句话摆在这里，我想没有一个人会认可，但在实际情况中，却经常能发现这种想法的影子。我曾经参与过一个项目，软件需求和设计已经变更了多次，但测试用例 却没有任何修改。导致的直接结果是新加入的测试工程师在执行测试用例时不知所措，间接的后果是测试用例成了废纸一堆，开发人员在多次被无效的缺陷报告打扰 后，对测试人员不屑一顾。

这个例子可能有些极端，但测试用例与需求和设计不同步的情况在实际开发过程中确是屡见不鲜的，测试用例文档是“活的”文档，这一点应该被测试工程师牢记。

·测试用例不应该包含实际的数据；

测试用例是“一组输入、执行条件、预期结果”、毫无疑问地应该包括清晰的输入数据和预期输出，没有测试数据的用例最多只具有指导性的意义，不具有可执行 性。当然，测试用例中包含输入数据会带来维护、与测试环境同步之类的问题，关于这一点，《Effective Software Test》一书中提供了详细的测试用例、测试数据的维护方法，可以参考。

·测试用例中不需要明显的验证手段；

我见过很多测试工程师编写的测试用例中，“预期输出”仅描述为程序的可见行为，其实，“预期结果”的含义并不只是程序的可见行为。例如，对一个订货系统， 输入订货数据，点击“确

定”按钮后，系统提示“订货成功”，这样是不是一个完整的用例呢？是不是系统输出的“订货成功”就应该作为我们唯一的验证手段呢？ 显然不是。订货是否成功还需要查看相应的数据记录是否更新，因此，在这样的一个用例中，还应该包含对测试结果的显式的验证手段：在数据库中执行查询语句进行查询，看查询结果是否与预期的一致。

七、从用例中生成测试用例

用于功能性测试的测试用例来源于测试目标的用例。应该为每个用例场景编制测试用例。用例场景要通过描述流经用例的路径来确定，这个流经过程要从用例开始到结束遍历其中所有基本流和备选流。

例如，下图中经过用例的每条不同路径都反映了基本流和备选流，都用箭头来表示。基本流用直黑线来表示，是经过用例的最简单的路径。每个备选流自基本流开始，之后，备选流会在某个特定条件下执行。备选流可能会重新加入基本流中（备选流 1 和 3），还可能起源于另一个备选流（备选流 2），或者终止用例而不再重新加入某个流（备选流 2 和 4）。

用例的事件流示例

遵循上图中每个经过用例的可能路径，可以确定不同的用例场景。从基本流开始，再将基本流和备选流结合起来，可以确定以下用例场景：

场景 1	基本流			
场景 2	基本流	备选流 1		
场景 3	基本流	备选流 1	备选流 2	
场景 4	基本流	备选流 3		
场景 5	基本流	备选流 3	备选流 1	
场景 6	基本流	备选流 3	备选流 1	备选流 2
场景 7	基本流	备选流 4		
场景 8	基本流	备选流 3	备选流 4	

注：为方便起见，场景 5、6 和 8 只描述了备选流 3 指示的循环执行一次的情况。

生成每个场景的测试用例是通过确定某个特定条件来完成的，这个特定条件将导致特定用例场景的执行。

例如，假定上图描述的用例对备选流 3 规定如下：

“如果在上述步骤 2‘输入提款金额’中输入的美元量超出当前帐户余额，则出现此事件流。系统将显示一则警告消息，之后重新加入基本流，再次执行上述步骤 2‘输入提款金额’，此时银行客户可以输入新的提款金额。”

据此，可以开始确定需要用来执行备选流 3 的测试用例：

测试用例 ID	场景	条件	预期结果
TC x	场景 4	步骤 2 - 提款金额 > 帐户余额	在步骤 2 处重新加入基本流
TC y	场景 4	步骤 2 - 提款金额 < 帐户余额	不执行备选流 3，执行基本流
TC z	场景 4	步骤 2 - 提款金额 = 帐户余额	不执行备选流 3，执行基本流

注：由于没有提供其他信息，以上显示的测试用例都非常简单。测试用例很少如此简单。

下面是一个由用例生成**测试用例的**更符合实际情况的示例。

示例：

一台 ATM 机器的主角和用例。

下表包含了上图中提款用例的基本流和某些备用流：

	本用例的开端是 ATM 处于准备就绪状态。 准备提款 - 客户将银行卡插入 ATM 机的读卡机。 验证银行卡 - ATM 机从银行卡的磁条中读取帐户代码，并检查它是否属于可以接收的银行卡。 输入 PIN - ATM 要求客户输入 PIN 码（4 位） 验证帐户代码和 PIN - 验证帐户代码和 PIN 以确定该帐户是否有效以及所输入的 PIN 对该帐户来说是否正确。对于此事件流，帐户是有效的而且 PIN 对此帐户来说正确无误。 ATM 选项 - ATM 显示在本机上可用的各种选项。在此事件流中，银行客户通常选择“提款”。

	输入金额 – 要从 ATM 中提取的金额。对于此事件流，客户需选择预设的金额（10 美元、20 美元、50 美元或 100 美元）。 授权 – ATM 通过将卡 ID、PIN、金额以及帐户信息作为一笔交易发送给银行系统来启动验证过程。对于此事件流，银行系统处于联机状态，而且对授权请求给予答复，批准完成提款过程，并且据此更新帐户余额。 出钞 – 提供现金。 返回银行卡 – 银行卡被返还。 .收据 – 打印收据并提供给客户。ATM 还相应地更新内部记录。 用例结束时 ATM 又回到准备就绪状态。
备选流 1 – 银行卡无效	在基本流步骤 2 中 – 验证银行卡，如果卡是无效的，则卡被退回，同时会通知相关消息。
备选流 2 – ATM 内没有现金	在基本流步骤 5 中 – ATM 选项，如果 ATM 内没有现金，则“提款”选项将无法使用。
备选流 3 – ATM 内现金不足	在基本流步骤 6 中– 输入金额，如果 ATM 机内金额少于请求提取的金额，则将显示一则适当的消息，并且在步骤 6 – 输入金额处重新加入基本流。
备选流 4 – PIN 有误	在基本流步骤 4 中– 验证帐户和 PIN，客户有三次机会输入 PIN。 如果 PIN 输入有误，ATM 将显示适当的消息；如果还存在输入机会，则此事件流在步骤 3 – 输入 PIN 处重新加入基本流。 如果最后一次尝试输入的 PIN 码仍然错误，则该卡将被 ATM 机保留，同时 ATM 返回到准备就绪状态，本用例终止。
备选流 5 – 帐户不存在	在基本流步骤 4 中 – 验证帐户和 PIN，如果银行系统返回的代码表明找不到该帐户或禁止从该帐户中提款，则 ATM 显示适当的消息并且在步骤 9 – 返回银行卡处重新加入基本流。
备选流 6 – 帐面金额不足	在基本流步骤 7 – 授权中，银行系统返回代码表明帐户余额少于在基本流步骤 6 – 输入金额内输入的金额，则 ATM 显示适当的消息并且在步骤 6 – 输入金额处重新加入基本流。
备选流 7 –	在基本流步骤 7 – 授权中，银行系统返回的代码表明包括本提款

达到每日最大的提款金额	请求在内，客户已经或将超过在 24 小时内允许提取的最多金额，则 ATM 显示适当的消息并在步骤 6 – 输入金额上重新加入基本流。
备选流 x – 记录错误	如果在基本流步骤 10 – 收据中，记录无法更新，则 ATM 进入“安全模式”，在此模式下所有功能都将暂停使用。同时向银行系统发送一条适当的警报信息表明 ATM 已经暂停工作。
备选流 y – 退出	客户可随时决定终止交易（退出）。交易终止，银行卡随之退出。
备选流 z – “翘起”	ATM 包含大量的传感器，用以监控各种功能，如电源检测器、不同的门和出入口处的测压器以及动作检测器等。在任一时刻，如果某个传感器被激活，则警报信号将发送给警方而且 ATM 进入“安全模式”，在此模式下所有功能都暂停使用，直到采取适当的重启/重新初始化的措施。

在第一次迭代中，根据迭代计划，我们需要核实提款用例已经正确地实施。此时尚未实施整个用例，只实施了下面的事件流：

基本流 – 提取预设金额（10 美元、20 美元、50 美元、100 美元）
备选流 2 – ATM 内没有现金
备选流 3 – ATM 内现金不足
备选流 4 – PIN 有误
备选流 5 – 帐户不存在/帐户类型有误
备选流 6 – 帐面金额不足

可以从这个用例生成下列场景

场景 1 – 成功的提款	基本流	
场景 2 – ATM 内没有现金	基本流	备选流 2
场景 3 – ATM 内现金不足	基本流	备选流 3
场景 4 – PIN 有误（还有输入机会）	基本流	备选流 4
场景 5 – PIN 有误（不再有输入机会）	基本流	备选流 4
场景 6 – 帐户不存在/帐户类型有误	基本流	备选流 5
场景 7 – 帐户余额不足	基本流	备选流 6

注：为方便起见，备选流 3 和 6（场景 3 和 7）内的循环以及循环组合未纳入上表。

对于这 7 个场景中的每一个场景都需要确定测试用例。可以采用矩阵或决策表来确定和管理测试用例。下面显示了一种通用格式，其中各行代表各个测试用例，而各列则代表**测试用例的**信息。本示例中，对于每个测试用例，存在一个测试用例 ID、条件（或说明）、测试用例中涉及的所有数据元素（作为输入或已经存在于数据库中）以及预期结果。

通过从确定执行用例场景所需的数据元素入手构建矩阵。然后，对于每个场景，至少要确定包含执行场景所需的适当条件的测试用例。例如，在下面的矩阵中，V（有效）用于表明这个条件必须是 VALID（有效的）才可执行基本流，而 I（无效）用于表明这种条件下将激活所需备选流。下表中使用的“n/a”（不适用）表明这个条件不适用于测试用例。

TC（测试用例）ID 号	场景/条件	PIN	帐号	输入的金额（或选择的金额）	帐面金额	ATM 内的金额	预期结果
CW1.	场景 1 - 成功的提款	V	V	V	V	V	成功的提款。
CW2.	场景 2 - ATM 内没有现金	V	V	V	V	I	提款选项不可用，用例结束
CW3.	场景 3 - ATM 内现金不足	V	V	V	V	I	警告消息，返回基本流步骤 6 - 输入金额
CW4.	场景 4 - PIN 有误（还有不止一次输入机会）	I	V	n/a	V	V	警告消息，返回基本流步骤 4，输入 PIN
CW5.	场景 4 - PIN 有误（还有一次输入机会）	I	V	n/a	V	V	警告消息，返回基本流步骤 4，输入 PIN
CW6.	场景 4 - PIN 有误（不再有输入机会）	I	V	n/a	V	V	警告消息，卡予保留，用例结束

在上面的矩阵中，六个测试用例执行了四个场景。对于基本流，上述测试用例 CW1 称为正面测试用例。它一直沿着用例的基本流路径执行，未发生任何偏差。基本流的全面测试必须包括负面测试用例，以确保只有在符合条件的情况下才执行基本流。这些负面测试用例由 CW2 至 6 表示（阴影单元格表明这种条件下需要执行备选流）。虽然 CW2 至 6 对于基本流而言都是负面测试用例，但它们相对于备选流 2 至 4 而言是正面测试用例。而且对于这些备选流中的每一个而言，至少存在一个负面测试用例（CW1 - 基本流）。

每个场景只具有一个正面测试用例和负面测试用例是不充分的，场景 4 正是这样的一个示例。要全面地测试场景 4 - PIN 有误，至少需要三个正面测试用例（以激活场景 4）：

- 输入了错误的 PIN，但仍存在输入机会，此备选流重新加入基本流中的步骤 3 - 输入 PIN。
- 输入了错误的 PIN，而且不再有输入机会，则此备选流将保留银行卡并终止用例。
- 最后一次输入时输入了“正确”的 PIN。备选流在步骤 5 - 输入金额处重新加入基本流。

注：在上面的矩阵中，无需为条件（数据）输入任何实际的值。以这种方式创建测试用例矩阵的一个优点在于容易看到测试的是什么条件。由于只需要查看 V 和 I（或此处采用的阴影单元格），这种方式还易于判断是否已经确定了充足的测试用例。从上表中可发现存在几个条件不具备阴影单元格，这表明测试用例还不完全，如场景 6 - 不存在的帐户/帐户类型有误和场景 7 - 帐户余额不足就缺少测试用例。

一旦确定了所有的测试用例，则应对这些用例进行复审和验证以确保其准确且适度，并取消多余或等效的测试用例。

测试用例一经认可，就可以确定实际数据值（在测试用例实施矩阵中）并且设定测试数据

TC（测试用例）ID 号	场景/条件	PIN	帐号	输入的金额（或选择的金额）	帐面金额	ATM 内的金额	预期结果
CW1.	场景 1 - 成功的提款	4987	809 - 498	50.00	500.00	2,000	成功的提款。帐户余额被更新为 450.00
CW2.	场景 2 - ATM 内没有现金	4987	809 - 498	100.00	500.00	0.00	提款选项不可用，用例结束
CW3.	场景 3 - ATM 内现金不足	4987	809 - 498	100.00	500.00	70.00	警告消息，返回基本流步骤 6 - 输入金

							额
CW4.	场景 4 - PIN 有误（还有不止一次输入机会）	4978	809 - 498	n/a	500.00	2,000	警告消息，返回基本流步骤 4，输入 PIN
CW5.	场景 4 - PIN 有误（还有一次输入机会）	4978	809 - 498	n/a	500.00	2,000	警告消息，返回基本流步骤 4，输入 PIN
CW6.	场景 4 - PIN 有误（不再有输入机会）	4978	809 - 498	n/a	500.00	2,000	警告消息，卡予保留，用例结束

以上测试用例只是在本次迭代中需要用来验证提款用例的一部分测试用例。需要的其他测试用例包括：

- 场景 6 - 帐户不存在/帐户类型有误：未找到帐户或帐户不可用
- 场景 6 - 帐户不存在/帐户类型有误：禁止从该帐户中提款
- 场景 7 - 帐户余额不足：请求的金额超出帐面金额

在将来的迭代中，当实施其他事件流时，在下列情况下将需要测试用例：

- 无效卡（所持卡为挂失卡、被盗卡、非承兑银行发卡、磁条损坏等）
- 无法读卡（读卡机堵塞、脱机或出现故障）
- 帐户已消户、冻结或由于其他方面原因而无法使用
- ATM 内的现金不足或不能提供所请求的金额（与 CW3 不同，在 CW3 中只是一种币值不足，而不是所有币值都不足）
- 无法联系银行系统以获得认可
- 银行网络离线或交易过程中断电

在确定功能性测试用例时，确保满足下列条件：

- 已经为每个用例场景确定了充足的正面和负面测试用例。
- 测试用例可以处理用例所实施的所有业务规则，确保对于业务规则，无论是在内部、外部还是在边界条件/值上都存在测试用例。
- 测试用例可以处理所有事件或动作排序（如在设计模型的序列图中确定的内容），还应能处理用户界面对象状态或条件。

- 测试用例可以处理为用例所指定的任何特殊需求，如最佳/最差性能，有时这些特殊需求会与用例执行过程中的最小/最大负载或数据容量组合在一起。

八、从补充规约中生成测试用例

并不是所有的测试目标需求都将在用例中有所反映。非功能性需求（如性能、安全性和访问控制）以及配置要求等将会说明测试目标的其他行为或特征。补充规约是为其他行为生成**测试用例的**主要来源。

关于如何生成这些其他**测试用例的**指南说明如下：

- 为性能测试生成测试用例
- 为安全性/访问控制测试生成测试用例
- 为配置测试生成测试用例
- 为安装测试生成测试用例
- 为其他非功能性测试生成测试用例

为性能测试生成测试用例

性能**测试用例的**主要输入是补充规约，补充规约中包含了非功能性需求（请参见工件：补充规约）。为性能测试生成测试用例时，请使用下列指南：

- 对于补充规约内阐明性能标准的各条说明都应确保至少要确定一个测试用例。性能标准通常表示为时间/事务、事务量/用户或百分数的形式。
- 对每个关键用例，都应确保至少要确定一个测试用例。关键用例是在上述说明中和/或在工作量分析文档中确定的、必须采用性能评测方法来评估的用例（请参见工件：工作量分析文档）。

与功能性测试的测试用例类似，通常对于每个用例/需求都会存在不止一个测试用例。常见的情况是：存在一个低于性能阈值的测试用例、一个处于阈值上的测试用例，还有一个测试用例高于阈值。

除了以上性能标准以外，确保已确定影响响应时间的特定条件，包括：

- 数据库的大小 - 存在多少个记录？
- 工作量 - 同时执行操作的最终用户的数量和类型，以及要同时执行的事务的数量和类型
- 环境特征（硬件、网件以及软件配置）

将用于性能测试的测试用例记录在类似于功能测试所使用的矩阵中。

以下是各种性能测试的一些示例：

对于负载测试：

TC(测试用例) ID 号	工作量	条件	预期结果

PCW1.	1 （单个 ATM）	完成提款交易	全部交易(不依赖于主角的时间）在 20 秒之内完成
PCW2.	2 （1,000 个同时运行的 ATM）	完成提款交易	全部交易(不依赖于主角的时间）在 30 秒之内完成
PCW3.	3 （10.000 个同时运行的 ATM）	完成提款交易	全部交易(不依赖于主角的时间）在 50 秒之内完成

对于强度测试：

TC（测试用例）ID 号	工作量	条件	预期结果
SCW1.	2 （1,000 个同时运行的 ATM）	数据库锁定 – 2 个 ATM 请求同一帐户	ATM 请求排成队列
SCW2.	2 （1,000 个同时运行的 ATM）	无法实现银行系统的通信	交易排成队列或超时
SCW3.	2 （1,000 个同时运行的 ATM）	在交易过程中，银行系统通信被终止	显示警告消息

为安全性/访问控制测试生成测试用例

主角和用例一同说明系统外部用户与系统所执行的动作之间的交互,以便为特定主角生成测试结果。复杂系统包含许多主角，所以我们编制测试用例时必须确保只有指定执行用例的主角可以进行此操作,这一点非常关键。在基于主角类型的用例事件流存在差别时,尤其如此。

例如,在 ATM 用例中,如果主角“银行客户”的卡和帐户有的属于拥有这个 ATM 机的银行,有的是竞争银行的银行卡（和帐户），或是企图使用该 ATM 不支持的银行卡，则将对该主角“银行客户”执行不同的用例事件流。

对于功能性测试用例，请同样遵循上面列举的指南。

关于安全性和访问控制**测试用例的**示例：

TC（测试用例）ID 号	条件	卡 （V 表明卡有效）	读卡机 （V 表明读卡机工作正常）	银行的网络	预期结果
ACW1.	在银行网络之内	V	V	V	所有用例都可用
ACW2.	银行网络之外	V	V	I	只有提款用例可用
ACW3.	无法读卡	I	V	V	警告消息，卡被退出
ACW4.	因被盗，卡已挂失	I	V	V	警告消息，卡予保留
ACW5.	卡已过期	I	V	V	警告消息，卡予保留

为配置测试生成测试用例

在典型的分布式系统中，允许存在许多种受支持的硬件和软件组合。为了核实测试目标在不同的配置情况下（如不同的操作系统、浏览器或 CPU 的速度）能否正常工作或执行，应该对此进行测试。此外，测试还应涵盖构件的组合，以便检测在不同构件的交互中产生的缺陷。例如，确保由应用程序安装的 DDL 版本不会与另一个应用程序需要的相同 DDL 的版本发生冲突。

采用下列指南来生成用于配置测试的测试用例：

- 确保对每个关键配置，应至少存在一个测试用例可用于对其进行确定。这是通过确定测试目标的环境所要求的硬件和软件配置以及确定这些配置的优先级来完成的。应确保最先测试最常见的配置，包括：
- 打印机支持
- 网络连接 - 局域网和广域网
- 服务器配置 - 服务器驱动程序、服务器硬件
- 台式机和/或服务器上安装的其他软件
- 所有已安装软件的软件版本
- 确保对于每个可能有问题的配置至少存在一个测试用例。这些配置可能包括：
- 具有最低性能的硬件。
- 历史上存在兼容性问题的共驻内存的软件。
- 通过最慢的 LAN/WAN 连接访问服务器的客户机。

- 资源不足（缓慢的 CPU 速度、最小的内存或分辨率，磁盘空间不足等等）

为安装测试生成测试用例

安装测试需要核实测试目标可以在所有可能的安装情况下安装。安装情况可以指首次安装测试目标，或是在装有较早版本的机器上安装测试目标的某个较新的版本或工作版本。安装测试还应确保在遇到异常情况时（如磁盘空间不足），测试目标的执行情况仍可接受。

测试用例应包含以下各种软件的安装情况：

- 分发介质，例如磁盘、CD-ROM 或文件服务器。
- 首次安装。
- 完全安装。
- 自定义安装。
- 升级安装。

客户机服务器软件的安装程序具备一组特定的测试用例。不同于基于主机的系统，服务器和客户机上的安装程序是有所不同的。因而，安装测试应执行构成测试目标的所有构件的安装，包括客户机、中间层以及服务器，这一点至关重要。

为其他非功能性测试生成测试用例

理论上,应找到所有必需的输入来生成测试用例模型、设计模型以及补充规约工件的测试用例。不过，如果此时您需要补充已有的输入，那也不足为奇。

示例如下：

- 操作测试（用以检验在某次故障发生后以及在下一次故障发生前“较长时间”内软件的运行情况）的测试用例。
- 对性能瓶颈、系统容量或测试目标的强度承受能力进行调查的测试用例。

大多数情况下，您可以通过先前所确定的测试用例生成的某些测试用例来构建其变体或聚合关系体，借此来查找测试用例。

九、为单元测试生成测试用例

单元测试要求既测试单元的内部结构同时还要测试其行为特征。测试内部结构要求了解实施单元的方式，基于这种了解的测试被称为白盒测试。对单元行为特征的测试侧重于从外部可观察的单元行为，而不需要了解或考虑其实施方式。基于这种方法的测试称为黑盒测试。基于这两种方法所生成的**测试用例的**说明如下。

白盒测试

理论上，应通过代码测试每一条可能的路径。在所有这些非常简单的单元内实现这样的目标是不切实际或几乎是不可能的。作为最基本的测试，应将每个决定到决定路径（DD 路径）

测试至少一次，这样可确保将所有语句至少执行一次。决定通常是指 if 语句，而 DD 路径是两个决定之间的路径。

要达到这种程度的测试覆盖，建议您在选择测试数据时应使每个决定都可以用每种可能的方法来评估。为达到上述目标，测试用例应确保：

每个布尔表达式的求值结果为 true 和 false。例如，表达式 (a<3) OR (b>4) 的求值结果为 true/false 的四种组合
每一个无限循环至少要执行零次、一次和一次以上。
可使用代码覆盖工具来确定白盒测试未测试到的代码。在进行白盒测试的同时应进行可靠性测试。

示例：

假设您对类 Set of Integers 中的 member 函数执行结构测试。该测试在二进制搜索的帮助下，将检查该集合是否包含了某个指定的整数。

成员 (member) 函数以及相应的流程图。虚线箭头指示出如何通过采用两个测试用例将所有语句至少执行一次。

理论上，对于彻底测试的某个操作，测试用例应遍历代码内路径的所有组合情况。在 member 函数的 while-loop 中存在三个可选择的路径。测试用例可以多次遍历该循环，或是根本就不遍历。如果测试用例根本就没有遍历循环，则在代码中只能找到一条路径。如果遍历循环一次，您将发现有三条路径。如果遍历两次，则您将发现存在六条路径，如此类推。因而，路径的总数应该是：1+3+6+12+24+48+...，在实际情况中，这个路径组合总数根本无法无法处理。这就是为什么必须选择所有这些路径的子集的原因。本示例中，可以采用两个测试用例来执行所有的语句。其中一个测试用例中，您可以选择 Set of Integers = {1,5,7,8,11}，而且测试数据 t = 3。在另一个测试用例中，您可以选择 Set of Integers = {1,5,7,8,11}，且 t = 8。

黑盒测试

黑盒测试的目的是为了在不了解单元将如何实施指定行为的情况下，对指定行为进行验证。黑盒测试侧重并依赖于单元的输入和输出。

等价类划分是一种用来减少所需测试数量的技术。对于每一个操作都应确定参数和对象状态的等价类。等价类是一组值的集合，对这组值来说，对象的行为应类似。例如，一个集合可有三个等价类：空、若干元素以及满。

可使用代码覆盖工具来确定白盒测试未测试到的代码。在进行黑盒测试的同时应进行可靠性测试。

接下来的两个小节说明了如何通过选择特定参数的测试数据来确定测试用例。

基于输入参数的测试用例
输入参数是由某个操作使用的参数。对于以下每个输入条件，都应通过使用每个操作的输入参数来编制测试用例：

每个等价类的正常值。
每个等价类的边界值。

等价类之外的值。
非法值。
请记住要将对象状态视作输入参数。例如：如果在对集合这个对象测试添加操作，您必须使用集合内所有等价类的值来测试添加操作。所有等价类的值指的是：充满元素的集合、有若干元素的集合、以及空集合。

基于输出参数的测试用例
输出参数是某个操作所改变的参数。某个参数既可以是输入参数也可以是输出参数。根据以下每个条件选择输入，以便获得输出。

每个等价类的正常值。
每个等价类的边界值。
等价类之外的值。
非法值。
请记住将对象状态视为输出参数。例如，假设您对某个列表测试删除操作，您必须选择输入值以便执行操作之后，列表为充满状态、具有若干元素或为空（采用它的所有等价类的值进行测试）。

如果对象受状态控制（根据对象的状态产生不同的反应），您应利用状态矩阵，如下图所示：

用于测试的状态矩阵。您可以在此矩阵的基础上测试激励和状态的所有组合。

十、为产品验收测试生成测试用例

产品验收测试是部署软件前的最后测试操作。验收测试的目标在于核实软件是否已经准备就绪，而且可以由最终用户按软件设计来执行功能和任务。产品验收测试通常不仅涉及执行软件以确认其是否准备就绪，还涉及交付给客户的所有产品工件，如培训、文档和包装。

为软件工件生成测试用例是按上文中说明的方式实现的。测试用例可与上面确定的测试用例（正式）或某个子集（非正式）相同或类似，这取决于产品验收测试的正式程度。不管**测试用例的**深度如何，应该在实施和执行产品测试之前对测试用例和产品验收计划达成共识。

对非软件工件的评估将随着被评估工件的不同而相去甚远。请参见每个特定非软件工件的指南以及核对清单，查看这些工件的评估内容和评估方式。

十一、为回归测试编制测试用例

回归测试比较同一测试目标的两个工作版本或版本，并将差异确定为潜在缺陷。据此可假定：新版本应该象早先版本一样操作，并确保并未因为版本的变化而带来缺陷。

理想状态下，您可能希望一次迭代内的所有测试用例都能在后续迭代内使用。应遵照下列指导原则来确定、设计并实施测试用例，这些测试用例可以最大限度地发挥回归测试和复用的价值，同时将维护的成本减至最低：

确保测试用例只确定关键的数据元素（创建/支持被测试的条件所需的数据元素）
确保每个测试用例都说明或代表一个唯一的输入集或事件序列，其结果是独特的测试目标行为
消除多余或等效的测试用例
将具有相同的测试目标初始状态和测试数据状态的测试用例组合到一起