第3章 软件测试的分类、策略与方法

软件测试策略(software testing strategy)是指一种确定如何设计、执行和管理测试的计划或方法,以确保软件质量达到预期水平,并满足业务需求和用户期望。


软件测试策略及黑盒

软件测试策略是测试计划的一部分,包括评估测试需求、明确测试目标、选择测试技术、界定测试范围、分配测试资源和控制测试进度等内容。测试计划制订人员需要面向具体任务,综合考虑经济、技术、习惯等多方面因素进行确定。

测试目标:通过描述测试目标,明确测试重点,以确保测试的针对性和有效性。

测试方法:根据被测软件的特点和测试目标,选择适合的测试方法和技术,构建测试用例、测试场景和测试数据,以确保测试覆盖面广泛、深入、透彻,满足具体测试任务的需求。

测试环境:明确测试需要搭建的环境,包括硬件设备、软件系统、网络环境等,以确保测试可进行并得到准确的结果。

测试工具:选择适合的测试工具和自动化测试框架,以提高测试效率和精度,减少测试人员的工作强度。

测试资源分配:确定测试的时间、地点、人员和资源等要素,以建立全面的测试计划和进度,确保测试能够按时、高质量地完成。

软件测试策略的确定,需要面向具体测试任务和目标来确定测试方法,并围绕测试方法决定测试环境、测试工具及资源分配等。

本章首先对各种软件测试类型做简要说明;然后以测试方法为抓手,说明软件测试策略,重点说明静态测试与动态测试、黑盒测试与白盒测试。通过本章的学习,读者能够理解软件测试策略的含义,理解各种测试方法的特点及适用场景。

3.1 软件测试的分类

素质培养

从不同的角度,对软件测试有不同的分类。对于任务事物的认知,都需要辩证、相对、多角度地全面理解问题实质。

软件测试可以根据不同的维度进行分类,以下是几种比较常见的分类方式。

1. 按照是否运行程序分类

(1) 静态测试也称为静态分析,不执行程序,只是检查和审阅程序,目的是收集有关程序代码的结构信息,纠正软件系统中描述、表示和规格方面的错误,为进一步测试做准备。

(2) 动态测试直接执行被测程序检验运行结果是否正确。通常情况下,动态测试是在完成静态测试之后进行的,包括功能测试、接口测试、覆盖率分析、性能测试等。

2. 按照测试阶段分类

(1) 单元测试是测试代码中最小的模块,通常由开发人员编写和测试,检查代码是否按照设计要求正常运行,是进行正确性检验的测试工作。

(2) 集成测试是将各个单元模块组合在一起,测试它们之间的交互和协作是否正确。

(3) 系统测试是对整个系统进行端到端的测试,验证系统是否符合用户需求和规格说明书的所有功能和性能要求。

(4) 验收测试是由项目管理者或客户代表验证系统是否符合业务需求和用户期望。

3. 按照是否关注程序内部结构分类

(1) 黑盒测试是通过输入数据检查实际输出结果的正确性,不关心或很少关心系统内部的实现细节,常用于功能测试和验证需求文档。

(2) 白盒测试是面向代码的测试,在了解代码结构、算法、数据等方面信息后,检查代码质量和逻辑错误,通常由开发人员进行。

(3) 灰盒测试是综合黑盒测试和白盒测试的优点,既关注外部也关注内部的测试技术。

4. 按照测试属性分类

(1) 功能测试是测试系统是否符合需求文档,包括界面测试、逻辑测试、安全测试等。

(2) 性能测试是测试系统在正常负载和压力下的表现,如并发用户数、响应时间等。

(3) 兼容性测试是测试系统在不同硬件、操作系统、数据库等环境下的兼容性。

(4) 安全性测试是测试系统的漏洞和风险,如数据加密、网络安全等。

(5) 易用性测试是测试系统的易学性、易操作性、可理解性等。

从不同的角度,对测试有不同的分类。图 3-1 总结了从不同角度进行的软件测试分类。很多测试可以见名知义。

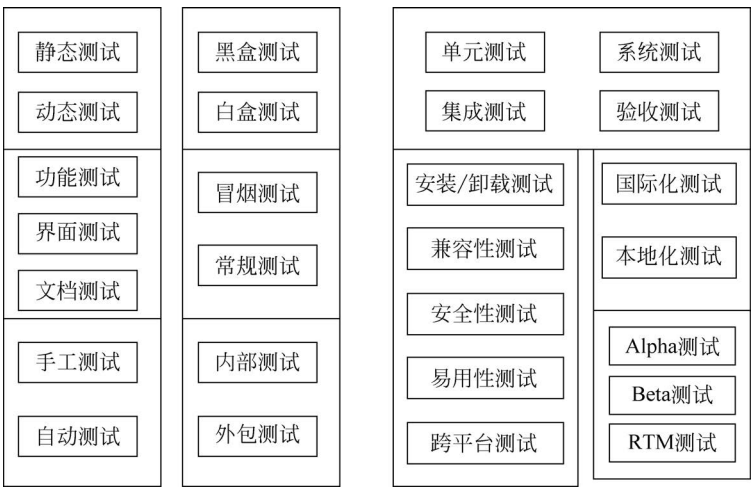

图 3-1　从不同角度进行的软件测试分类

其中,冒烟测试是软件测试中的一种功能测试,旨在验证应用程序的基本功能是否能够正常工作而没有明显缺陷。在冒烟测试中,测试人员会进行一系列简单的测试,以确保软件能够顺利启动、基本功能可用,并且没有严重的问题阻碍测试的继续进行。冒烟测试通常在每次构建后或大型系统集成前进行,是一项快速、高级别的测试方式,有助于尽早发现潜在的严重问题。

3.2 软件测试策略与方法

软件测试策略是指制定测试整体策略，以及所使用的测试技术和方法。不同软件测试策略的区别在于不同的出发点、不同的思路，以及采用不同的手段和方法。具体地说，软件测试策略包括要使用的测试技术和工具、测试完成标准、影响资源分配的特殊考虑等。在此着重介绍要使用的测试技术。

3.2.1 静态测试和动态测试

按照是否运行程序，软件测试可分为静态测试和动态测试。静态测试通过人工或自动化工具对程序和文档进行分析和检查。动态测试通过人工或自动化工具运行程序进行检查，分析程序的执行状态和外部表现。

1. 静态测试

素质培养	静态测试不运行程序，却能找到30%～70%的逻辑设计和编码错误，不要觉得不运行程序就没法做测试。对事物的认知，需要打破刻板印象，用事实与数据说话。

静态测试(static testing)是一种在软件开发生命周期早期使用的测试方法，主要用于对软件源代码、文档和其他相关工作产品进行检查和评审，以发现其中存在的问题和缺陷。在此过程中，无须执行软件程序或创建测试环境。静态测试可以帮助团队及时发现错误，在早期改进软件质量，并提高开发者的专业技能和代码质量控制水平，缩短开发周期和降低开发成本。

静态测试对程序的数据流和控制流等信息进行分析，找出系统缺陷。在不运行被测试软件的情况下，静态测试可找出30%～70%的逻辑设计问题和编码错误。静态测试包括代码检查、代码审查、编码风格与规范、静态结构分析等，也包括对文档和软件原型的一些审查。静态测试可以人工进行，也可以借助测试工具自动进行。

1) 代码检查

代码检查也称为代码走查，主要检查代码和设计的一致性、代码对标准的遵从性、逻辑表达的正确性、代码结构的合理性，以及代码的可读性等方面，通常由开发人员进行代码检查。代码检查的具体内容包括变量、数据类型、程序逻辑、程序结构、语法等内容。代码检查可以发现程序里不明确和模糊的部分，找出程序中不可移植部分，以及其他违背编程风格的问题。代码检查能够快速找到缺陷，但人工的代码检查工作非常耗时，并且要求人员经验丰富。

2) 代码审查

代码审查是对代码的正式审查，与代码检查的目标一样，都是为了使代码符合标准规范、无逻辑错误。代码审查通常由包括测试人员在内的项目组成员以限时的正式会议的方式进行审查，并形成静态分析错误报告。在代码审查前需要准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表等。

表3-1对比了代码检查和代码查审，以说明其中的区别与联系。

表 3-1　代码检查与代码审查的对比

项 目	代 码 检 查	代 码 审 查
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	以开发人员为主	项目组成员，包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时，不要现场修改代码	限时，不要现场修改代码
生成文档	会议记录	静态分析错误报告
目标	代码标准规范，无逻辑错误	代码标准规范，无逻辑错误

代码审查中通常要检查如下几方面的错误。

(1) 数据引用错误。指使用未经正确初始化和引用的变量、常量、数组、字符串或记录而导致的软件缺陷。例如变量未初始化、数组和字符串下标越界、对数组的下标操作遗漏、变量与赋值类型不一致、引用的指针未分配内存等。

(2) 数据声明错误。未正确地声明或使用变量和常量。

(3) 计算错误。计算无法得到预期结果。例如，不同数据类型或数据类型相同但长度不同的变量计算，计算过程中或计算结果溢出，赋值的变量上界小于赋值表达式的值，除数/模为零，变量的值超过有意义的范围(如概率的计算结果不在 0～1 范围)，等等。

(4) 比较错误。比较和判断错误很可能是边界条件的问题。例如，混淆小于和小于或等于、逻辑表达式中的操作数不是逻辑值，等等。

(5) 控制流程错误。编程语言中循环等控制结构未按预期方式工作。通常由计算或者比较的错误直接或间接造成。例如，死循环，存在从未执行的代码，由于变量赋值错误而意外进入循环，等等。

(6) 子程序参数错误。子程序不正确地传递数据。例如，实际传送的参数类型或次序与定义不一致，更改了仅作为输入值的参数，等等。

(7) 输出错误。包括文件读取、接收键盘或者鼠标输入，以及向打印机或者屏幕等输出设备写入错误等。例如，软件没有严格遵守外部设备读写数据的专用格式，文件或外设不存在，错误情况发生时没有相应处理，未以预期的方式处理预计错误，错误提示信息不正确或不准确，等等。

3) 编码风格与规范审查

编码风格与规范审查主要检查代码是否符合团队或行业规定的编码标准。程序设计时要使程序结构合理、清晰。程序不仅要能可执行并得出正确结果，而且要便于调试和维护，具有良好的可读性。程序不仅要程序员自己能看懂，也要让别人看懂。好的程序设计和编码风格有助于提高程序的正确性、可读性、可维护性和可用性。

4) 静态结构分析

静态结构分析主要是以图形的方式表现程序的内部结构，如函数调用关系图、函数内部控制流图等。函数调用关系图以直观的图形方式描述程序中各个函数的调用和被调用关系；控制流图用于描述函数的逻辑结构。

5）文档检查

文档检查主要是检查各种文档是否符合规范和要求，内容明晰、逻辑正确等。审查的文档包括需求文档、设计文档、用户手册等。

6）原型检查

原型检查用于检查软件系统的原型是否满足用户需求和需求规格说明书的要求。

2. 动态测试

动态测试(dynamic testing)是一种在软件开发生命周期后期使用的测试方法，通过运行软件，检验软件的动态行为和运行结果的正确性，通过验证程序行为与规格说明书是否相符来发现问题。

在动态测试中，计算机必须真正运行被测试的程序，通过输入测试用例，对其运行情况即输入与输出的对应关系进行分析，以达到检测的目的。可以看出，动态测试的实现需要两个基本要素：一是可运行的被测试程序，二是测试数据(测试用例)。

动态测试可通过以下几个步骤来实现。

(1) 选取程序输入定义域的有效值，以及定义域外的无效值，作为测试输入数据；

(2) 根据测试输入数据确定预期的输出结果；

(3) 用选取的输入数据执行被测程序；

(4) 程序运行结果与预期结果相比较，如果结果不一致，说明找到了软件缺陷或故障。

3.2.2 黑盒测试与白盒测试

动态测试一般又分为黑盒测试和白盒测试。此外，还有一种介于白盒测试与黑盒测试之间的灰盒测试。

1. 黑盒测试

黑盒测试(black box testing)的基本观点是：任何程序都可被看作从输入定义域映射到输出值域的函数的过程。黑盒测试不关心内部代码结构。测试人员在不了解内部代码和结构的情况下，仅通过输入、输出和系统对各种输入的响应等“黑盒子视角”来评估软件的正确性、可靠性和安全性。这种测试方法通常用于评估整个软件系统，特别是在用户界面层次上进行的功能测试。黑盒测试强调测试应该关注期望的行为而不是实现细节，以确保软件可以按预期运行。

黑盒测试不仅能够找到大多数其他测试方法无法发现的错误，而且对于外购软件、参数化软件包以及某些生成的软件，由于无法得到源程序，用其他方法进行测试是完全无法完成的，只能进行黑盒测试。

黑盒测试主要针对软件界面、功能、外部数据库访问以及软件初始化等方面进行测试，试图发现功能错误或遗漏、界面错误或不美观、外部信息访问错误、性能错误、初始化和终止错误以及接口错误。

黑盒测试是功能测试方法，主要根据软件规格说明书设计测试用例。因此，黑盒测试方法具有以下特征。

(1) 测试用例可复用。黑盒测试与软件的具体实现过程无关。当软件的具体实现过程发生变化时，黑盒测试用例仍可复用。

(2) 时间优势。黑盒测试用例的设计可以与软件开发同时进行,可以压缩总的开发时间。

(3) 测试人员独立于开发人员。黑盒测试是基于系统需求和功能进行的测试,测试人员不需要了解系统的内部设计和实现细节,可独立于开发人员进行测试。

(4) 可自动化。黑盒测试方法可以被自动化执行,测试脚本在运行时模拟用户操作,检测系统的反应并记录结果。自动化测试能够提高测试效率。

(5) 难以找到隐藏的缺陷。黑盒测试方法仅基于需求和规范测试软件系统,很难发现系统中的隐藏缺陷或异常情况。而且,由于无法查看软件的内部结构和代码,测试人员有可能会遗漏一些重要的测试场景。

(6) 穷举输入测试方法。只有把所有可能的输入都进行测试,才能以这种方法查出程序中所有的功能错误。

黑盒测试的具体技术方法主要包括等价类划分法、边界值分析法、因果图法、决策表法等。

2. 白盒测试

白盒测试(white box testing)是一种着重于测试程序内部结构和逻辑关系的软件测试方法,又称结构测试或基于程序的测试、逻辑测试。白盒测试将被测程序看作一个打开的盒子,测试者能够看到被测源程序,分析程序的内部结构,包括设计、代码和算法等方面,根据其内部结构设计测试用例。

白盒测试可以帮助测试人员发现代码中隐藏的缺陷和错误,提高软件质量。与黑盒测试不同,白盒测试需要测试人员具备一定的编程知识和技能,以便理解和分析程序源代码。

也许有人认为,只要确保程序中所有的路径都执行一次,就能实现全面的白盒测试,进而产生“百分之百正确的程序”。但在实际中这是不可能实现的,即使是一个非常小的控制流程,进行穷举测试都需要花费巨大的时间代价。

在白盒测试时,通常的做法是对程序的结构特性做到一定程度的覆盖,也就是“基于覆盖率的测试”。测试人员可以严格定义要测试的确切内容,明确要达到的测试覆盖率,引导测试者朝着提高覆盖率的方向努力,找出那些可能已被忽视的程序错误。

常见的白盒测试方法包括语句覆盖、分支覆盖、条件覆盖、判定覆盖、判定/条件覆盖、路径覆盖。

虽然白盒测试提供了评价测试的逻辑覆盖准则,但如果程序结构本身存在问题,例如程序逻辑错误或者遗漏了软件规格说明书中已规定的功能,那么无论采用哪种结构测试,即使其覆盖率达到了百分之百,也检查不出来。因此,提高结构测试的覆盖率,可以增强对被测软件的信度,但也不能做到万无一失。

3. 灰盒测试

灰盒测试(gray box testing)是黑盒测试和白盒测试的结合。灰盒测试关注输出对于输入的正确性,也关注内部表现。

灰盒测试不像白盒测试那样详细完整,只通过一些表征性现象、事件和标志判断程序内部的运行状态。这是由于在实际测试工作中,有时候输出结果正确,但内部逻辑其实是错的。此时,如果每次都通过白盒测试寻找软件缺陷,效率会很低。为此,测试人员可以有限制地分析程序结构和功能,部分了解被测试系统的内部设计和实现细节,测试其内部状

态和行为。

灰盒测试通常用于测试基于 Web 或客户-服务器架构的应用程序，并在数据交换、暴露逻辑错误以及尚未被覆盖的路径方面提供更好的测试覆盖率和有效性。它可以帮助测试人员发现潜在的性能问题、安全漏洞和其他缺陷，并帮助开发人员诊断和修复这些问题。

3.3 小　　结

本章介绍了软件测试策略与方法。首先简要介绍了不同分类标准下的测试方法，然后重点讲解静态测试与动态测试、黑盒测试与白盒测试，分别说明这些测试方法的概念、特点、囊括的范畴及相关步骤。

3.4 习　　题

1. 选择题

(1) 测试时采用人工检查或计算机辅助静态分析的手段检查程序。这种测试称为(　　)。

A. 白盒测试　　B. 黑盒测试　　C. 静态测试　　D. 动态测试

(2) 下列说法中正确的是(　　)。

A. 程序测试无法确认程序没有错误

B. 黑盒测试是逻辑驱动的测试

C. 穷举测试一定可以暴露数据敏感错误

D. 白盒测试是一种输入输出驱动的测试

(3) 软件测试中常用的静态分析方法是(　　)。

① 引用分析；② 算法分析；③ 可靠性分析；④ 效率分析；⑤ 代码走查

A. ①③　　B. ④③　　C. ②⑤　　D. ①⑤

(4) 代码走查和代码审查的主要区别是(　　)。

A. 代码审查由程序员组织讨论，代码走查由高级管理人员领导评审活动

B. 代码审查只检查代码是否错误，代码走查还要检查程序与设计文档的一致性

C. 代码走查只检查程序的正确性，代码审查还要评审程序员的编程和工作业绩

D. 代码审查是一种正式的评审活动，而代码走查的讨论过程是非正式的

(5) 下列中不属于静态分析的是(　　)。

A. 代码规则检查　　B. 程序结构分析

C. 程序复杂度分析　　D. 内存泄露

(6) 下列中不属于动态分析的是(　　)。

A. 代码覆盖率　　B. 程序数据流分析

C. 系统压力测试　　D. 模块功能检查

(7) 黑盒测试一般从(　　)的观点来执行测试。

A. 最终用户　　B. 设计人员　　C. 程序员　　D. 软件购买者

(8) 黑盒测试是一种重要的测试策略,又称数据驱动的测试,其测试数据来源于(　　)。

A. 软件规格说明书　　B. 软件设计说明书

C. 概要设计说明书　　D. 详细设计说明书

2. 判断题

(1) 动态测试有黑盒测试和白盒测试两种测试方法。　　(　　)

(2) 测试是调试的一部分。　　(　　)

(3) 在不了解软件功能、没有产品说明书和需求文档的条件下可进行动态黑盒测试。　　(　　)

(4) 软件测试按照测试过程分为黑盒测试和白盒测试。　　(　　)

3. 简答题

(1) 动态测试和静态测试的区别是什么?

(2) 划分软件测试属于白盒测试还是黑盒测试的依据是什么?

第4章 黑盒测试

黑盒测试不需要了解程序源代码，只需要根据软件需求规格说明书或用户手册的描述，测试人员借助专业知识执行测试，检查程序的功能是否符合功能说明，以测试结果来评估软件的正确性、可用性和健壮性。


黑盒测试——等价类

黑盒测试方法主要用于发现以下几类错误。①是否有不正确或遗漏了的功能；②在接口上，输入能否正确地接收，能否输出正确的结果；③是否有数据结构错误或外部信息（如数据文件）访问错误；④性能上是否能够满足要求；⑤是否有初始化或终止性错误。

在黑盒测试时，必须在所有可能的输入条件和输出条件中确定测试数据，以检查程序是否都能够产生正确的输出。然而，这实际是不可能实现的。因此，黑盒测试必须精心设计测试用例，期待从数量极大的可用测试数据中精选出少量的测试数据，通过少量测试数据高效地把隐藏的错误揭露出来。

本章介绍黑盒测试中常用的方法。通过本章的学习，读者可对黑盒测试技术中的等价类、边界值、决策表、因果图、场景法、正交试验法和错误猜测法有深入的理解和体会，并能够使用具体测试方法设计测试用例。

不同的黑盒测试方法有不同的特点和应用场景。在具体的测试实践中，测试人员应根据具体需求选择合适的黑盒测试方法或方法组合，以提高测试效率和质量。

4.1 等 价 类

4.1.1 等价类划分法的应用场景

软件测试有一个致命的缺陷，就是测试的不彻底性和不完全性。由于穷举测试的测试数量太大，实际中无法实现，需要在大量的可用数据中选择一部分作为测试数据，同时考虑测试效果和测试实际的经济性。这样一来，如何选取合适的测试用例就成为关键问题，由此引入等价类测试的思想。使用等价类划分最主要的目的是在有限的测试资源情况下，用少量有代表性的测试数据得到比较好的测试结果。


等价类

等价类划分法是一种典型的黑盒测试方法，它完全不考虑程序的内部结构，只根据程序规格说明书对输入范围进行划分，把所有可能的输入数据，即程序输入域，划分为若干互不相交的子集，称为等价类，然后从每个等价类中选取少数具有代表性的数据设计测试用例，进行测试。也就是说，等价类划分法的核心思想是“用一组有限的数据代表近似无限的数据”。

等价类划分通过识别许多相等的条件，严格控制了测试用例的数量，并覆盖了大部分其他可能的测试用例，但这种方式不能测试输入条件存在组合的情况。

等价类测试的关键在于等价类的划分（即得出等价类表），以及从等价类中选取测试用例。

4.1.2　等价类的划分原则与方法

等价类是指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的，并合理地假设：测试某等价类的代表值就等价于对这一类其他值的测试。

如果某个等价类中的一个输入条件作为测试数据进行测试时查出了错误，那么使用这一等价类中的其他输入条件进行测试也会查出同样的错误；反之，若没有查出错误，则认为使用该等价类中的其他输入条件也同样查不出错误。

下面以经典的三角形判定问题为例，说明等价类法设计测试用例的思路。

【例 4-1】 在三角形判定程序中，程序接收 3 个整数 a、b、c 作为输入，用作三角形的边。整数 a、b、c 必须满足以下条件：

c_1. $0<a\leqslant 200$

c_2. $0<b\leqslant 200$

c_3. $0<c\leqslant 200$

c_4. $a<b+c$

c_5. $b<a+c$

c_6. $c<a+b$

程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、普通三角形和非三角形。

例 4-1 中，如果已经选择了三元组(5,5,5)作为测试用例的输入，那么再输入(6,6,6)和(100,100,100)时，可以预期不会再发现新问题。这是因为，后两组测试数据将会以与第一个测试用例同样的方式进行“相同处理”。因此，后两组测试用例是冗余的。

可以把全部输入数据合理地划分为若干等价类，在每个等价类中取一个数据作为测试的输入条件，实现用少量代表性测试数据的测试效果。例如在三角形判定问题中，把满足条件 c_1、c_2、c_3 和 $a=b=c$ 的数据归入一个等价类，取其中一组数据(5,5,5)作为测试用例即可，其他同类型数据都不必再选。

1. 等价类的划分原则

等价类法中对类的划分，指将输入域划分为一组互不相交的子集，且这组子集的并集构成全集。这也是等价类划分的原则。

等价类是输入域的某个子集，而所有等价类的并集就是整个输入域。因此，等价类具有完备性、无冗余性和各类中测试用例的等价性。

注意，软件不能只接收有效的、合理的数据，还要经受意外考验，即接收无效的、不合理的数据，这样才能说明软件的可靠性较高。因此，在划分等价类时，需要考虑两种不同情况：有效等价类和无效等价类。

(1) 有效等价类是指由对于程序规格说明书来说是合理的、有意义的输入数据构成的集合。有效等价类可检验程序是否实现了程序需求规格说明书中所规定的功能和性能。