CHAPTER

第 3 章 软件测试的基本概念

本章学习目标

- 了解软件缺陷与软件测试的基本概念
- 理解测试计划与测试用例的基本内容
- 理解测试策略的概念及基本测试方法
- 理解静态测试与动态测试的区别与联系
- 掌握黑盒测试与白盒测试的应用场景
- 理解人工测试与自动测试的区别与联系
- 了解软件测试规范及专业测试人员要求

本章介绍软件缺陷、软件测试，重点介绍软件测试任务、计划、策略、方法等相关概念。同时，介绍软件的基本测试方法，测试用例的概念及选择，最后分类介绍几种常用的测试方法，及其区别与联系。

3.1 软件缺陷与软件测试的主要任务

软件开发的任何一个环节都可能影响整个软件的开发质量和开发效率，甚至是毁掉整个项目。人总会犯错，在需求阶段、设计阶段、编码阶段，不可避免地会发生一些错误，这些错误会导致软件存在缺陷，所以在软件开发过程中，由于软件本身的特点和目前的开发模式，软件缺陷是不可避免的。软件一旦有了缺陷，软件质量就无法保证，可能会丧失用户满意度，软件测试是发现软件中存在缺陷、保证软件质量的主要手段，因此，软件测试要求测试人员对软件缺陷要有一个深入的认识。

3.1.1 Bug 的由来

软件缺陷(Defect)，又被称为 Bug。Bug 是一个英文单词，本意是臭虫、缺陷、损坏、窃听器、小虫等，现在，人们将在计算机系统和程序中隐藏的未被发现的缺陷或问题统称为 Bug。

在计算机领域，流传着一个有趣的故事。20 世纪 40 年代，电子计算机的体积非常庞

大，数量非常少，造价非常高，主要应用在军事领域。1947 年 9 月 9 日，美国水上研究中心使用马克-Ⅱ型计算机进行数据处理，马克-Ⅱ型计算机在运算时，通过继电器开关执行二进制指令语句，当指令为“1”时，继电器的电磁铁受到激励带电，使得继电器接点闭合接通，电流即可通过；当指令为“0”时，继电器的电磁铁不受激励，继电器中的弹簧使得接点断开，电流不能通过。一位为美国海军工作的电脑专家格蕾丝·赫柏对马克-Ⅱ型计算机设置 170 000 个继电器进行编程后，技术人员进行整机运行，计算机突然停止工作，于是工作人员爬上去查找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾，显然是由于天气炎热加上机房无空调设备，飞蛾在机房中乱飞，受光和热的吸引，飞到正要闭合的继电器触点之间，被继电器触点夹住，且被高电压击死，导致电路中断，造成工作故障。工作人员只需将飞蛾拿掉，计算机便可以恢复正常工作。在报告中，赫柏用胶条将飞蛾贴在记事本中，如图 3-1 所示，在上面加了一行注释——“First actual case of bug being found”。从此，人们把计算机错误称为 Bug，将发现 Bug 并纠正的过程称为 Debug。Bug 这个说法一直沿用至今。

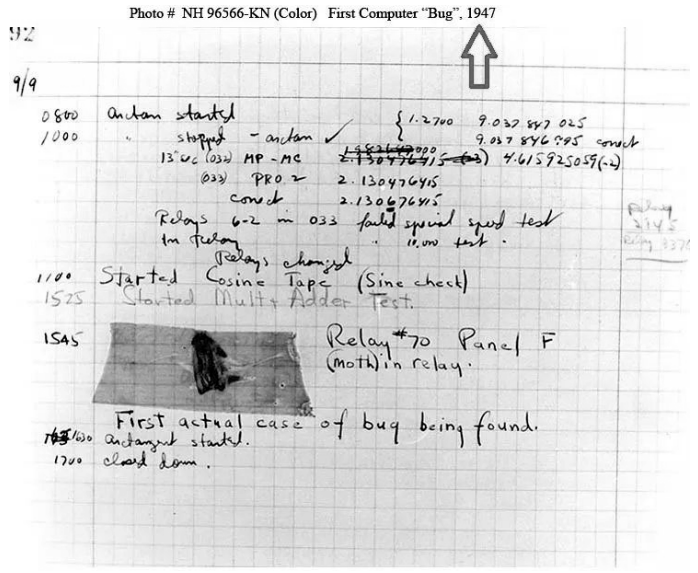

图 3-1　Bug 的由来

3.1.2　软件缺陷概述

在日常的软件测试过程中，经常会遇到与软件问题相关的概念，不同的人对问题的称呼也不相同。比如错误(error)、缺陷(defect)、故障(fault)、失效(failure)等。这些术语都是指软件中存在的一些问题，但是它们的具体含义和定义却有很大不同。

在 ANSI 982.2 标准中对软件的错误、缺陷、故障和失效进行了定义。

错误：是指编写错误的代码，通常是无意中造成的。错误可分为两类，一类是语法错误(syntax error)。在编译语言中，语法错误在代码编译阶段出现，因为该类错误，代码不能正常编译通过。另一类是逻辑错误(logical error)，指代码的运行逻辑与预期不同，它

与代码的实际执行密切相关，所以不易被发现。可见，软件错误是人为原因所导致的不正确结果的过程。它可能是程序的内部错误，也可能是文档内的错误，甚至是环境方面的问题。

缺陷：是指软件与用户需求不一致，无法正确完成软件需求所要求的功能。

故障：是指由于不当使用计算机软件而引起的故障，以及因系统或系统参数设置不当而出现的故障。软件故障一般是可以恢复的。例如，软件处于执行一个多余循环过程时，就可以说该软件出现了故障。此时，若无适当的措施（容错）及时处理，便会导致软件失效。

失效：ISO/CD 10303-226 将失效定义为一个组件、设备、子系统或者系统无法（不具备相应的能力）完成为它设计的任务。

软件错误是一种人为疏忽而造成的错误，最初的软件错误来自软件文档、软件代码等，如代码中加法写成减法；一个软件错误必定产生一个或多个软件缺陷，如设计缺陷、代码缺陷等，也就是软件功能与需求不一致；当程序的某些代码存在错误导致程序执行失败时，也就是一个软件缺陷被激活，便产生一个软件故障，故障分为内部故障和外部故障；同一个软件缺陷在不同条件下被激活，可能产生不同的软件故障。软件故障如果没有及时的容错措施加以处理，或者当程序错误太多时，便不可避免地导致软件在功能操作等方面失效，同一个软件故障在不同条件下也可能产生不同的软件失效。

假如有一段求最大值的函数程序代码，该程序要求输入两个整数，经过程序运算，可以输出两个整数中最大的一个整数。例如，当输入两个整数 3 和 5 时，应该输出 5，但是实际执行代码后，发现程序输出了 3，而不是 5，与预期结果不同。所以，可以断定该程序中存在一个缺陷。经过代码检查发现，程序中错误地将 X＞Y，写成了 X＜Y，这里好像存在一个错误。经过将其修正，重新运行，发现输出结果为 5，此时可以断定，这段代码里确实存在一个错误。当这个含有错误的求最大值的函数被一个排序函数调用时，就会导致排序结果错误，此时，可以说包含这段排序函数代码的软件存在软件故障，整个排序算法都失效了。

综上所述，所谓软件缺陷就是计算机软件中存在某种影响软件正常运行的问题、错误或功能缺陷，也就是不能满足用户需求。随着软件技术的不断发展，软件从业人员对软件缺陷的认识逐渐深刻，IEEE 729-1983 给出软件缺陷的标准定义。

从产品内部看，软件缺陷是软件产品开发或维护过程中存在的错误、毛病等各种问题；从产品外部看，软件缺陷是系统所需要实现的某种功能的失效或违背。

软件缺陷的含义相对比较广泛，其表现形式多种多样，如功能特性没有实现或部分实现；设计不合理存在缺陷；实际结果和预期结果不一致；运行出错，包括运行中断、系统崩溃、界面混乱等；数据结果不正确、精度不够；用户不能接受的其他问题，如存取时间过长、界面不美观等。总之，一般把符合下列 5 种情况之一的问题都定义为软件缺陷。

- 软件没有实现软件需求规格说明书中的要求。
- 软件中出现软件需求规格说明书中指明不应该出现的错误。
- 软件功能超出软件需求规格说明书的范围。
- 软件未达到软件需求规格说明书虽未指明但应达到的要求。

- 软件测试人员认为难以理解、不易使用、运行速度缓慢或者最终用户认为不好的问题。

软件缺陷不光存在于代码中，需求规格说明书和设计文档中也存在大量缺陷，经过大量实践统计发现，在整个软件工程的软件缺陷中，软件规格说明书缺陷占 54%，软件设计缺陷占 25%，代码缺陷仅占 15%，其他缺陷占 6%，如图 3-2 所示。

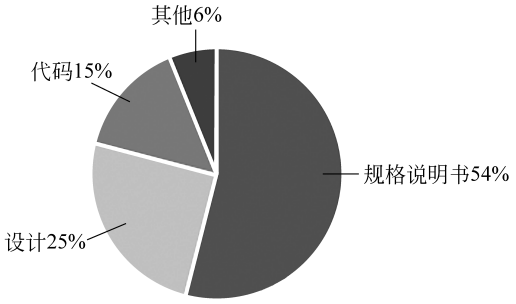

图 3-2　软件缺陷和错误的分布

软件缺陷不仅表现在软件功能方面，也可能表现在软件性能方面，例如，要建设一个网站，客户需求是网站必须满足 200 万人并发访问，但是开发人员所提供的软件设计架构只能满足 20 万人并发访问，那么这就是软件设计在性能方面存在缺陷。软件发布时也可能存在缺陷，只是此时，软件中存在缺陷的级别不太严重，不影响整个软件的使用效果。例如，移动研发平台 EMAS-专有云 20180808 发布的 Release Notes，如图 3-3 所示，可以看出该版本修复了 4 个问题，即高可用分辨率信息问题，Cache 配置请求异常问题，鉴权参数带空格(机型等)导致鉴权失败问题，以及适配部分 7.0/8.0 机型无法唤起系统安装页面问题。

RELEASE NOTES

EMAS_20180808

1、Android SDK 版本信息

SOPHIX SDK: 不能与 ATLAS 并存，接入 ATLAS 后默认使用 Dexpatch 方式热修复，无需再接入 Sophix

网关 SDK 在基础库中

变更信息

高可用 SDK: compile('com.alibaba.ha:alihatbadapter:1.1.0.7-open@aar') 修复高可用分辨率信息没有问题

网关 SDK：compile('com.taobao.android:mtopsdk_allinone:3.0.8.2-open@jar'){transitive true} 修复 ZCache 配置请求异常问题

WEEX SDK: 0.20.0.1, emoji 标签在 Android 9.0 适配，启动性能优化，富文本组件优化。

通道服务 SDK: compile('com.taobao.android:accs_sdk_taobao:3.3.6.8-open'){transitive true} 修复鉴权参数带空格(机型等)导致鉴权失败问题

更新 SDK: 1.0.3-open， 适配部分 7.0/8.0 机型无法唤起系统安装页面问题，升级改动

图 3-3　EMAS Release Notes

通常人们习惯将软件缺陷称为 Bug。它不仅包括代码级别的错误，也包括在设计和测试阶段发现的缺陷。一般将软件整个生命周期中存在的或者发现的各种问题和有助于改善产品质量的提议、需要引起注意的方面、值得跟踪的问题都列入缺陷范围，因此，可以在缺陷和 Bug 之间画上一个等号，在后续章节中将不再区分缺陷和 Bug。

3.1.3 软件缺陷的级别

在软件开发和维护过程中，一旦发现软件缺陷，开发人员就需要想方设法找到引起缺陷的原因，分析其对软件产品质量的影响。然而，项目开发时间有限，开发人员不可能将软件中所有缺陷都发现并处理解决，这就需要确定软件缺陷的严重性和处理缺陷的优先级。一般来说，缺陷引起的问题越严重，开发人员处理该缺陷的紧迫性越高，该缺陷的优先级也就越高。现有的缺陷分类大多是针对编码阶段的，因为该阶段产生的 Bug 最多。

软件缺陷的严重性是指软件缺陷对软件质量的破坏程度，也就是软件缺陷的存在对软件功能和性能产生的影响程度。软件缺陷的严重性大多是从最终用户角度做出的判断，根据 CMM5，软件缺陷的严重性可分为 3～5 个等级，根据软件项目的实际情况来决定级别的划分。本书将缺陷划分为 4 级，即致命、严重、一般和轻微。

致命缺陷，指出现致命错误，导致用户数据受到破坏，系统崩溃、悬挂、死机，甚至系统运行环境破坏，造成事故，引起生命财产损失等，如系统无法安装、登录或其他主要功能不可用，死循环或内存不足等原因导致程序无法运行等。

严重缺陷，指系统的主要功能部分失效，数据不能保存，系统的次要功能完全丧失，系统所提供的功能或服务受到明显的影响。例如，软件的某个菜单不起作用，软件部分功能运行后产生错误的结果等。

一般缺陷，指虽然存在一些错误，但不影响用户正常使用，表现不严重。例如，界面存在明显缺陷，设计不友好，提示信息不准确，操作时间过长等。

轻微缺陷，指存在无关紧要的小问题，影响操作者操作的方便性或影响软件界面的美观，但不影响整个软件功能的操作和执行。例如，某个控件没有对齐、用户界面色调或格式不规范等。

软件缺陷优先级是表示处理和修正软件缺陷的先后顺序的指标，即哪些软件缺陷先修正，哪些软件缺陷稍后修正或者不修正。从软件开发工程角度考虑，软件缺陷的优先级可以分为最高优先级、较高优先级、中等优先级和一般优先级。

最高优先级，是指该软件缺陷需要被立即修复，例如，软件的主要功能出现严重错误或系统崩溃等。

较高优先级，是指该软件缺陷需要在产品发布之前必须被修正，如软件功能和性能的一般缺陷。

中等优先级，是指该软件缺陷需要正常排队等待修复或列入软件发布清单，例如，本地化软件的某些字符没有翻译或者翻译不准确。

一般优先级，是指该软件缺陷可以在方便时被修正。通常是一些对软件质量影响非常轻微或者出现概率很低的缺陷。

软件缺陷的严重性和优先级是从不同角度定义软件缺陷对软件本身的影响程度和处

理方式。一般来说,严重性程度高的软件缺陷具有较高优先级,因为严重性高说明缺陷对软件质量的影响大。严重性低的软件缺陷的处理结果可能只是使软件更加完美,可以稍后处理,其优先级较低。但是,这种严重性与处理时效的对应关系不是一成不变的,有时严重性高的软件缺陷优先级不一定高,一些严重性低的缺陷则需要及时处理,具有较高优先级。所以,修正软件缺陷需要综合考虑各种因素,如市场发布、质量风险等问题。比如,由于整个系统软件架构设计所造成的,在软件开发中后期发现的 Bug,影响面很大,其优先级比较高。但是修复该 Bug 需要大量人力和时间,甚至很容易引入新 Bug。此时项目经理需要和其他部门人员联系,如市场部门、技术支持部门,甚至客户,经过沟通,如果确认该 Bug 对大多数用户没有影响,则降低该 Bug 的优先级,推迟解决或不解决。

3.1.4 软件测试任务

在日常生活中,软件已经无处不在,吃饭、购物、出行、住宿等方面的各种软件已经悄然地改变着人们的生活、学习方式,人们对软件的依赖性越来越大。但是实践证明,尽管人们在开发软件的过程中使用了许多保证软件质量的方法和技术,但开发出的软件中还会隐藏着许多错误和缺陷。尤其是在规模大、复杂性高的软件中更是如此。如何才能让人们使用到更多高质量、高可靠性和高安全性的软件产品已经成为迫在眉睫、亟待解决的问题。为了解决这个问题,首先必须承认,由于开发人员思维的局限性和软件系统的复杂性,软件缺陷在软件开发过程中是不可避免的,开发人员可以做的事情,只能是通过努力尽可能多地、尽可能早地寻找、定位和解决隐藏在软件中的缺陷,避免付出高昂代价,大大提高系统开发过程的效率,而寻找软件中的缺陷就是软件测试的主要目的。所以,严格的软件测试对于保证软件质量具有重要作用。

软件测试的根本目的是发现尽可能多的缺陷。为了更好地实现这一目的,软件测试阶段包括以下几个方面的任务,如图 3-4 所示。

1. 测试需求

测试人员仔细阅读软件需求规格说明书、设计文档和其他必要文档,熟悉项目基本需求,分析确定软件测试需求。

2. 测试设计

首先,测试组负责人依据软件需求文档、项目周期、项目特点、工具、人员安排等制订测试计划,确定测试所需资源、测试对象、测试步骤和方法,以及测试将得到的输出或提交的产物。

其次,测试人员根据测试需求,提取测试功能点,利用各种测试方法编写测试用例(冒烟测试用例和普通测试用例)。如果测试需求中存在不清楚、模糊、有二义性的需求,需要及时和开发人员确认沟通,明确每一个测试需求,尽可能将测试用例细化到输入框、按钮等小功能需求中。

最后,测试人员完成测试用例的编写后,需要组织开展测试用例评审。测试用例评审参与人员有开发人员、测试人员和产品人员,通过测试用例的评审,减少测试人员执行阶段的无效工作,避免出现三方对同一需求的理解不一致现象。

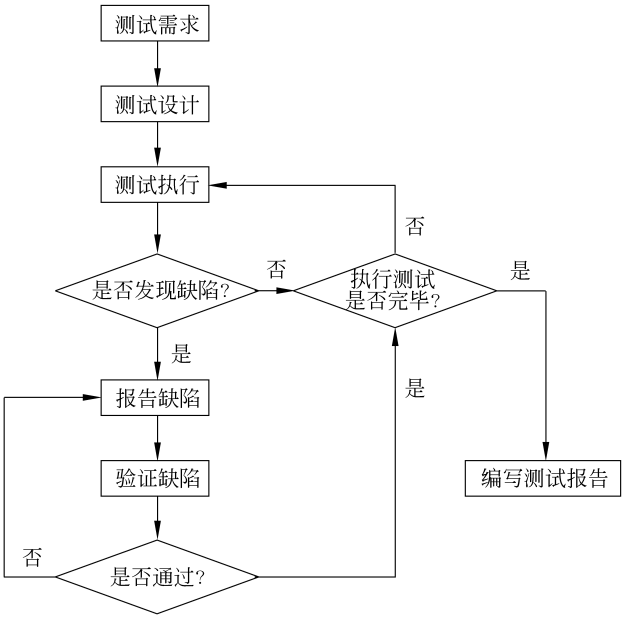

图 3-4　软件测试任务流程图

3. 测试执行

首先，开发人员提交软件测试版本，测试人员优先执行冒烟测试，通过冒烟测试尽量消除表面明显错误，确认软件基本功能正常，减少后期测试负担。冒烟测试结果需要通过邮件通知相关人员，包括开发人员、测试人员和产品人员。如果待测试软件达到冒烟测试要求，则可以安排根据正式测试用例对该软件版本进行测试。否则，需要重新编译版本，再次执行冒烟测试，直到测试通过。

其次，开发人员提交正式软件测试版本，测试人员针对该版本中已经修改的问题进行回归测试。同时，测试人员根据设计好的测试用例，分模块测试整个软件系统，主要测试各模块功能是否完全实现，是否存在软件缺陷。在测试过程中可能会发现一些问题，比如需求定义不明确，业务逻辑有冲突等，需要及时与相关人员沟通并给出清晰明确的定义，得到测试结论后必须要求产品人员更新文档，在测试中发现的问题要及时在缺陷追踪系统中提交给相关开发人员进行修正。

最后，模块测试完成后，小组内部进行交叉测试，测试各模块的交互是否正常实现，并关注界面和用户体验等问题。同时，对系统进行一些性能测试、兼容性测试等。

每一轮功能测试结束后，测试人员均需及时提交对应的测试日志，对测试情况进行总结。

4. 编写测试报告

软件测试过程结束后，测试人员需要及时编写测试报告，将测试情况、过程和结果整理成文档，对在测试过程中发现的问题和缺陷进行初步分析，为纠正软件存在的质量问题提供依据，同时为软件验收、发布和交付奠定基础。

3.2 测试计划与测试用例

3.2.1 测试计划

俗话说："凡事预则立，不预则废！"在日常工作和生活中，做任何事情之前都需要先制订计划，比如，学习计划、出行计划、生产计划等。同样，在测试开始之前，测试人员也需要制订相应的测试计划。测试计划是为测试过程中各项活动制订一个可行的、综合的计划，其内容通常包括测试对象、测试范围、测试方法、测试进度和预期结果等。详细的测试计划还需要为测试项目中的每个角色定义职责和工作内容，识别测试活动中可能存在的各种风险，并预防和消除可能的风险，降低损失，从而保证正确地、有效地验证正在开发的软件系统。

软件测试计划是软件项目计划的子计划，在项目启动初期就必须规划制订。《ANSI/IEEE 软件测试文档标准 829-1983》将测试计划定义为："一个叙述了预定的测试活动的范围、途径、资源及进度安排的文档。它确认了测试项、被测特征、测试任务、人员安排，以及任何偶发事件的风险。"通过软件测试计划，测试人员可以明确测试任务和测试过程中使用的测试方法，保持测试实施过程中各方参与人员顺畅沟通，项目管理人员和测试管理人员实时跟踪和控制测试进度，应对测试过程中的各种变更。

软件测试计划是软件测试的第一个阶段。一般需要解决的是 5W1H（Why、What、When、Where、Who、How）的问题，即：Why——为什么要执行这些测试；What——需要测试哪些方面，不同阶段的工作内容有哪些；When——测试不同阶段的起止时间；Where——相应文档、缺陷的存放位置、测试环境等；Who——项目有关人员的组成，安排哪些测试人员进行测试；How——如何去做，使用哪些测试工具以及测试方法进行测试。所以，要编制一个测试计划，首先需要确定测试目标，也就是回答 Why 的问题。通过对用户需求文档和设计规格文档的分析，确定被测试软件的质量要求和通过测试所需要达到的目标。其次，通过理解需求、分析需求，确定需求规格说明文档中的功能测试点，明确测试人员需要测试什么，也就是分析 What。接着，分析和规划哪些资源是可以使用的，如何合理分配和利用这些资源，分析出 Who、Where 和 When，所以在测试计划中需要包含测试时间进度的安排，以及人力资源的分配和测试环境的安排；接着，确定测试过程中需要使用的方法和测试策略，包括要进行的测试阶段（单元测试、集成测试和系统测试）以及要执行的测试类型（功能测试、性能测试、负载测试、强度测试等），也就是 How。最后，做任何一件事情都会存在风险，所以在制订测试计划时，还需要预先考虑测试过程中可能存在的风险，并对风险进行分析与评估，给出预防措施和应对手段。

测试计划的内容主要包括引言、测试内容、测试规则、测试环境、项目任务、实施计划和风险管理七大部分，一个典型的测试计划目录如图 3-5 所示。

- 引言部分主要介绍项目名称、开发背景、开发情况，以及要完成的功能和测试计划的具体目的，指明预期读者范围，给出文档中的术语定义和参考文献等资料。

图 3-5 软件测试计划目录

- 测试内容部分主要制订测试功能清单，罗列每一项测试内容的名称标识符、测试进度安排以及测试内容和目的。
- 测试规则部分需要制订进入准则、暂停/退出准则、测试方法、测试手段、测试要点和测试工具。进入准则是指进入测试的前提条件，例如，正确安装软件包后就可以进行使用；暂停/退出准则是指测试暂停或者退出的条件或标准，例如，软件系统在进行单元测试、集成测试、系统测试和验收测试时，发现一级错误、二级错误，暂停测试返回开发。软件系统经过单元测试、集成测试、系统测试和验收测试，分别达到单元测试、集成测试、系统测试和验收测试停止标准；测试方法是指在测试过程中需要使用哪些测试方法设计测试用例，完成测试，如黑盒测试、白盒测试等；测试手段是指在测试过程中所采用的技术手段，如路径测试、语句与分支覆盖、基于需求的测试、组合测试等；测试要点是指测试的重点以及测试目标，如系统功能是否符合客户要求，各个模块之间的衔接程度是否顺畅，测试软件是否存在缺陷和漏洞；测试工具是指测试过程中所使用的辅助工具，如负载压力测试工具、功能测试工具、测试管理工具等。
- 测试环境主要描述软硬件环境和特定测试环境等要求。软硬件环境是指软件运行的软硬件环境要求，如需要安装系统智能机，操作系统为安卓 4.0 版本以上；特定测试环境一般是指安全性环境，如操作系统的安全性。
- 项目任务主要包括测试规划、测试设计、测试执行准备、测试执行和测试总结。测试规划是对整个测试过程做出比较全面的计划和规定，例如，将响应时间定义为对请求做出响应所需要的时间，把响应时间划分为“呈现时间”和“系统响应时间”两部分；测试设计就是有目的、有计划地设计整个测试活动，例如，将系统安装/升级测试设计为在正常情况下，第一次安装/升级或完整的/自定义的安装都能顺利完成；在异常情况下包括磁盘空间不足、缺少目录创建权限等情况进行安装时，应有相应的提示；测试总结是总结测试活动的结果，并根据这些结果进行评价。
- 实施计划部分主要完成工作量估计、人员需求及安排、进度安排、其他资源需求及安排和可交付工件。工作量的评估是指根据工作内容和项目任务对包括测试设计、测试执行和测试总结的工作量进行评测，以“人·月”或“人·日”计算，并详细注释测试设计、测试执行和测试总结工作所占的比重。需要说明的一点是，软件测试工作量应为开发工作量的 30%～40%为宜；人员需求及安排是指给出具体

参与人员、人员在项目中扮演的角色以及具体职责；进度安排是指给出项目里程碑测试规划、测试设计、测试设计实施、测试执行和测试总结等工作的开始时间、结束时间和输出要求等；可交付工件需要列出将要创建的各种文档、工具和报告，以及其创建人员、交付对象和交付时间。

- 风险管理部分需要描述测试各方面存在的风险情况，并预估风险可能引发的后果，以及项目各参与方对风险的承受情况。测试阶段的风险管理主要针对项目计划的变更、需求的变更以及测试产品版本变更的不确定性，做出有效的应对措施。对于项目计划的变更，测试人员需要及时跟踪项目，项目经理需要把这些变更信息及时通知项目组，使得整个项目组工作处于同一个阶段。

测试计划不一定要尽善尽美，但一定要切合实际，要根据项目特点、公司实际情况来编制。测试计划制订后，不是一成不变的，软件需求、软件开发、人员流动等都在时刻发生着变化，测试计划也需要根据实际情况的变化而不断进行调整，以满足实际测试要求。

3.2.2 测试用例

软件测试阶段的基本任务是根据软件开发各阶段的文档资料和程序的内部结构，精心设计一组“高产”的测试用例。测试用例是为发现软件中存在的问题而编写的一组包含测试输入、执行条件以及预期结果的文档，用来判断软件程序是否工作正确，软件产品是否满足需求。它是有效地发现软件缺陷的最小测试执行单元。测试用例指导测试人员如何去测试，如何找出软件中潜在的缺陷。一个好的测试用例很有可能发现尚未被发现的缺陷。

测试用例的内容包括 8 个基本项，分别是测试用例编号、测试项目、用例标题、重要级别、预置条件、输入数据、操作步骤和预期输出。需要强调的是，不同公司的测试用例所包含的内容也不尽相同。

用例编号是由字符和数字组合而成的字符串，是用来作唯一标识的。例如，采用产品编号-ST-系统测试项名-系统测试子项名-XXX 进行编号。

测试项目是当前测试用例所在的测试大类或被测试需求、被测试模块等，如订单管理。

用例标题需清楚、简洁地描述每个用例的关注点，每个用例标题是不允许重复的。

重要级别，即用例优先级，一般分为高、中、低三种，测试人员可以参照此级别安排测试用例的执行时间。

预置条件是执行当前测试用例时需要满足的条件。如果该预置条件无法满足，则不能执行测试步骤。预置条件不是必需的，根据测试用例的实际情况决定。

输入数据是测试用例在执行过程中，软件系统需要接收的数据，如登录时需要测试人员输入的用户名和密码。

操作步骤是执行当前测试用例需要执行的操作步骤，需要明确地给出每个步骤的详细描述，测试人员根据该步骤逐个完成测试用例的执行。

预期输出是执行当前测试用例后的正常输出数据或状态，包括返回值内容、界面响应结果、输出结果的规则符合度、数据库等存储表中的操作状态等。

测试用例是为特定的目的而设计的一组测试输入、执行条件和预期结果，以便测试某个程序路径和结果是否满足某个特定需求。

测试用例的编写可遵循以下步骤完成：

首先，对各个功能模块进行测试点的分析提取，图 3-6 为 PC 端 QQ 账号的登录模块，在该模块中，可以提取正常登录、账号为空时单击登录、密码为空时单击登录、账号密码都为空时单击登录、密码错误时单击登录、找回密码功能是否有效、记住密码功能是否有效、自动登录功能是否有效、二维码登录是否有效、注册功能是否有效等测试点。

图 3-6　PC 端 QQ 账号的登录模块图

其次，需要对各功能点细致地设计测试用例。划分好功能点后，就可以利用等价类划分法、边界值分析法等测试方法编写测试用例，并进行标注。这样，对于后期的测试用例的整理有很大的帮助。以 PC 端 QQ 账号正常登录为例，其测试用例设计如表 3-1 所示。

最后，在测试过程中不断完善测试用例。测试人员每完成一轮测试，都会对所测试的内容有进一步的了解。开发人员在实际开发过程中，可能会对某些功能的细节部分做出修改，测试人员根据变更和对软件功能的熟悉程度进一步完善测试用例，主要是对测试步骤的修改和异常情况的补充，提高测试用例对需求的覆盖率，以便发现更多的 Bug。

表 3-1　PC 端 QQ 账号正常登录测试用例表

用例编号	测试项目	用例标题	重要级别	预置条件	输入数据	操作步骤	预期输出
USC1	登录	正常登录	高	注册账号	用户名：234532 密码：12345678	1. 双击 QQ 快捷图标，打开登录界面； 2. 输入正确账号及密码； 3. 单击“登录”按钮	登录成功并跳转至已登录状态的 QQ 页面

GB/T 15532-2008《计算机软件测试规范》标准规定设计测试用例时，应遵循以下原则。

- 基于测试需求的原则，即测试用例应按照测试类别的不同，进行不同的设计。例如，单元测试依据详细设计说明书设计，集成测试依据概要设计说明书设计，系统

测试依据用户需求设计；

- 基于测试方法的原则，即设计测试用例时，应明确所采用的测试用例设计方法。要达到不同的测试充分性要求，必须采用相应的测试方法；
- 兼顾测试充分性和效率原则，即测试用例集应兼顾测试充分性和测试效率；每个测试用例的内容也应完整，具有可操作性；
- 测试执行的可再现性原则，即应保证测试用例执行的可再现性。

3.3 软件测试策略

假如某公司开发一款基于 Internet 的即时通信软件，该软件在正式发布之前，需要经过无数次的软件测试，测试涉及的范围很广，包括功能性测试、性能测试等。即使只考虑功能性测试，涉及的范围也是非常广泛的，不仅有登录、聊天、联系人、看点、动态、打卡、钱包、个性装扮……而且还需要在不同型号的手机、计算机上进行测试，测试工作量非常庞大，进行完全测试或者穷举测试都是不可能的。那么，如何进行软件测试才可以在有限时间内最大程度地发现可能存在的问题？是将整个程序作为一个整体来测试，还是只测试其中的一部分？什么时候客户需要参与到测试工作中？这就是测试策略需要回答的问题。

策略，一般是指可以实现目标的方案集合。软件测试策略是在定义软件测试标准和测试规范标准指导下，依据测试项目特定环境约束，规定软件测试原则、方式、思路的集合。一般来说，测试策略描述了在软件开发过程中测试所使用的方法，指导测试人员测试活动应该如何进行，描述测试过程中存在哪些风险，以及如何规避或者降低风险。更加通俗地讲，测试策略就是回答测试什么和如何进行测试。

创建软件测试策略时可以参考各种需求文档和设计文档，收集开发软件系统所需要的各种软硬件资源的详细说明，了解针对测试和进度约束而需要的人力资源以及他们的角色和职责，理解软件测试的思路方法、测试标准、完成标准等。材料收集完毕后，测试人员就可以开始正式地制定测试策略了。

1. 测试策略制定依据

一个好的测试策略应该包括实施测试的类型和目标、实施测试的阶段和技术、用于评估的测试结果和测试是否完成评测的标准，以及影响测试策略工作的特殊事项等内容。那么，应该如何选择测试策略呢？

1）基于测试技术的测试策略

测试专家提出使用各种测试方法的综合策略：无论在任何情况下，都可以使用边界值测试方法；可以在必要时使用等价类划分法补充测试用例；对已设计出的测试用例通过逻辑覆盖程度检查其是否达到要求；如果功能规格说明书中含有输入条件的组合情况，则可以选择因果图测试方法。

2）基于测试方案的测试策略

一般来说，对于基于测试方案的测试策略应该考虑以下方面：根据功能的重要性和

可能发生故障将造成的损失来确定测试等级和测试重点；使用尽可能少的测试用例发现尽可能多的软件缺陷，避免测试过度和测试不足的情况发生。

2. 测试策略内容

软件测试策略描述了需要进行的测试步骤，包括这些步骤的计划和执行时机，以及所需要的工作量、时间和资源。任何软件测试策略，都需要包含测试计划、测试用例、测试执行以及测试结果数据的收集和评价。一般来说，测试策略在内容上需要包含以下要点，但也不局限于此。

1）测试级别

和软件开发过程相对应，测试过程也可以分为单元测试、集成测试、系统测试和验收测试 4 个主要阶段。

- 单元测试：单元测试是针对软件中的最小可测试单元进行检查和验证，是软件开发过程中最低级别的测试活动，通常由开发人员完成。
- 集成测试：集成测试是在单元测试的基础上，将所有模块按照设计要求组装起来进行测试，主要目的是发现与接口有关的问题。为了提高集成测试的效果，集成测试最好由不属于该软件开发组的软件设计人员来完成。集成测试的测试策略有大爆炸集成、自顶向下集成、自底向上集成和三明治集成等。
- 系统测试：系统测试是在集成测试通过后进行的，是对整个系统的测试，将硬件、软件和操作人员看作一个整体，检验其是否有不符合系统说明书的地方。这种测试可以发现系统分析和设计中的错误。它主要由测试部门进行，是测试部门最大且最重要的一个测试，对产品的质量有很大的影响。
- 验收测试：验收测试是在软件产品完成单元测试、集成测试和系统测试之后，产品发布之前所进行的最后一个测试操作。验收测试以需求阶段的“需求规格说明书”为验收标准，测试时要求模拟用户的真实运行环境。

2）角色与职责

在测试策略中需要明确定义各个角色，以及该角色的职责。只有分工明确，工作才能高效，不会出现相互推卸责任的情况。比如，项目经理负责统筹项目的规划，测试经理负责统筹测试工作和制订测试计划等，测试工程师负责测试用例的设计、开发及实施等。

3）环境需求

环境需求用于描述软件测试时所需要搭建的模拟用户真实环境的需求，包括软硬件环境和网络服务环境等。模拟环境越真实，软件上线后的风险就越小。

4）风险分析

软件测试的风险是不可避免的，作为测试管理人员，必须在平时工作中，分析这些风险的类别、风险来源，以及风险可能引发的后果，将风险尽早地识别出来，并且想出对策，最大程度地降低甚至消除这些风险，包括软件需求风险、人员风险、代码质量风险和测试环境风险等。充分分析风险后，软件测试管理人员需要针对风险制定相应的对策。例如，测试人员预留时间熟悉测试工具，对于核心测试人员配置一些候补测试人员，避免这些测试人员因为不熟悉工具或离职、请假等延误测试的情况。在项目开发过程中的每个阶段，尽早让用户参与进来，加强各阶段产出物的评审，减少需求变更的风险。

5）测试范围

测试范围就是指测试的内容，比如功能，性能，易用性……

6）测试进度

测试进度就是评估完成测试所需要的时间。要制定合适的测试进度，首先需要明确测试范围，然后根据测试资源的多少来制订能被各方面认可的测试进度计划。制订测试进度计划时，可以尽量参考历史数据，与以前完成的项目进行类比，确定质量和进度所存在的某种数量关系。但因为测试过程中存在各种风险和不确定性，所以通常在制订测试进度计划时，会添加一段时间作为缓冲，尤其是一个全新的项目，一般需要将初始的计划时间翻倍。

7）回归测试

回归测试就是检验已经被发现的缺陷有没有在新的版本中被正确地修正，且修改过程中是否引入新的缺陷。在做回归测试时采用不同的测试策略，一种是完全重复测试，就是把所有的测试用例全部执行一遍，确定问题已经被修正，且周边的功能没有受到影响。另一种是选择性重复测试，就是选择一部分测试用例进行测试，比如，选择发生错误的模块用例进行测试，或者除了出错模块用例外，还可以选择一些与出错模块有联系的模块用例，或者测试完出错模块后，检查软件是否达到指标，然后根据达标情况不断扩大测试范围。

8）测试优先级

因为软件测试时间和测试资源是有限的，所以软件系统不可能尽善尽美，但是必须满足用户的需求和期望，所以必须确定哪些功能模块最重要，以便在测试工作中成功地权衡资源约束和风险等因素，确定测试的优先级。新浪科技曾发表过一篇文章《浅析软件测试用例的优先级》，文中提到如何划分测试用例的优先级，将所有功能性验证的测试都标注为高优先级别，把所有错误和边界值或确认测试标注为中优先级，把所有的非功能性的测试都标注为低优先级；然后把功能性验证分为重要和不重要两种，不重要的测试优先级降为中优先级，把错误和边界测试也分为重要和不重要两种，将重要的测试优先级提升为高优先级，把非功能性测试也分为重要和不重要两种，将重要的测试优先级提升为中优先级，按照重要和不重要，重复划分高、中、低 3 个基本的测试内容，直到可以应用的测试用例数量最小为止。

3.4 软件测试方法概述

软件测试是使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验其是否满足规定的需求或弄清预期结果与实际结果之间的差别。软件测试方法多种多样，从是否需要执行被测试软件的角度看，可分为静态测试和动态测试；从测试是否针对系统内部结构和具体实现算法的角度看，可分为白盒测试和黑盒测试；从测试执行时使用的工具角度看，可分为人工测试和自动化测试。

所谓静态测试是指不运行被测试程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等方面来检查程序的正确性。静态测试包括代码检查、静态结构分析和代码质

量度量等。静态测试可以由人工进行，也可以借助软件工具自动完成。常用的一种静态测试是找出程序中违背程序编程风格的问题。动态测试是指通过运行软件来检验软件的动态行为和运行结果的正确性。动态测试方法由三部分组成，即构造测试用例、执行程序和分析程序输出结果。

黑盒测试是测试人员将测试对象看作一个黑盒子，完全不考虑程序内部的逻辑结构和内部特性，只依据程序的“需求规格说明书”，检查程序功能是否符合功能说明。具体的黑盒测试用例设计方法包括等价类划分法、边界值分析法、错误推断法、因果图法、判定表驱动法、正交试验设计法、场景法等。白盒测试，从字面意思理解，就是清楚盒子内部的东西，具体讲就是在了解程序内部逻辑结构基础上进行的一种设计测试用例的方法。白盒测试的方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法等。

自动化测试，顾名思义就是软件测试的自动化，即在预先设定的条件下运行被测试程序，分析运行结果。这种测试方法是将人工驱动的测试行为转化为机器执行的一种过程。相反，人工测试是测试人员根据设计的测试用例一步一步地执行测试，得到实际结果，并将其与期望结果进行比对的过程。

3.5 静态测试与动态测试

狭义的软件测试思想是只对可运行的软件进行测试，广义的软件测试思想是将测试遍布于软件开发生命周期的各个阶段，包括软件需求、软件设计、软件编码、软件测试以及软件维护等阶段。软件测试方法一般分为两大类：静态测试和动态测试，静态测试比动态测试具有更长的生命周期，如图 3-7 所示。静态测试贯穿于软件开发的整个生命周期，动态测试只存在于软件编码和软件测试阶段。相比于动态测试，静态测试能更早地发现软件缺陷，更能体现测试的经济学原则。

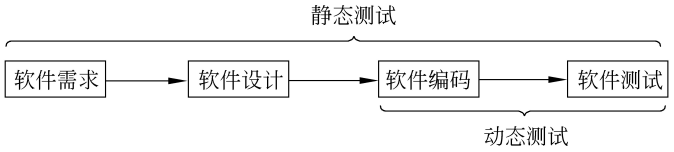

图 3-7　软件测试阶段图

3.5.1 静态测试

静态测试不实际运行软件，主要是对软件编程格式和结构等方面进行评估。它通过程序静态特性的分析，找出欠缺和可疑之处，如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑计算等。可以对需求规格说明书、软件设计说明书、源程序做结构分析、流程图分析、符号执行来找出其中的错误。静态测试的结果可用于进一步的查错，并为测试用例选取提供指导。总之，静态测试就是指

不运行被测试程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序正确性的一种测试方法。

静态测试的内容包括代码检查、静态结构分析和代码质量度量等。它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具自动进行，提升工作效率。

1. 代码检查

代码检查包括代码走查、桌面检查和代码审查等，主要检查代码和设计的一致性、代码对标准的遵循和可读性、代码逻辑表达的正确性和代码结构的合理性等方面；它可以发现违背程序编写标准的问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分和违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

代码走查(code walkthrough)是开发人员与架构师集中讨论代码的过程，其目的是交换有关代码的书写思路，建立对代码标准的集体阐述。代码走查的形式有很多种，包括每日走查、结对互查和专项走查。

每日走查的时间和地点都比较灵活，主要针对每日提交的新代码进行评审。

结对互查是提交代码前指定某位同事进行线上评审，评审通过后方可合入代码。

专项走查是针对某个具体问题或专题进行走查。如果项目较小，参加项目走查的人员可以是整个项目的参与者；如果项目较大，则可以按功能模块分组进行走查。代码走查会议之前，主持人将程序清单和设计规范分发给小组成员，所有成员在进行检查之前需要先对其熟悉，测试人员提前准备书面的测试用例。会议中首先由测试组成员将准备的测试用例提交给走查小组，然后程序员向其他人朗读程序并逐条阐述代码，其他人充当计算机，让测试用例沿程序的逻辑运行一遍，随时记录程序踪迹，然后展开讨论，发现更多问题。

桌面检查是由程序员自己检查程序，一个人阅读程序，对照错误列表检查程序，对照程序推演测试数据，以便自我发现代码中存在的问题。

代码审查(code review)是由若干程序员和测试人员组成一个评审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。其目的是找出及修正在软件开发初期未发现的错误，提升软件质量及开发者的技术。代码审查一般分为 3 类：正式的代码审查、结对编程以及非正式代码审查。

正式的代码审查有审慎及仔细的流程，例如，范根检查法，应由多位参与者分阶段进行。小组负责人提前把设计规格说明书、控制流程图、程序文本及相关要求、规范等提前分发给小组成员，作为评审依据。小组成员阅读材料后，召开审查会议。在会上由软件开发者一行一行地解释程序的逻辑，其他小组成员可以提出问题，展开讨论，审查错误是否存在。实践表明，程序员在讲解的过程中可能自己发现许多错误，讨论和争议可以促进问题的暴露。虽然正式的代码审查可以彻底找到程序中的缺陷，但需要投入许多资源。

结对编程(pair programming)是一种敏捷软件开发方法，两个程序员在一台计算机上共同工作。一个人输入代码，另一个人审查输入的每一行代码。输入代码的人称作驾驶员，审查代码的人称作观察员(或导航员)，两个程序员经常互换角色。

轻量型的非正式代码审查需要投入的资源比正式的代码审查要少，一般在正常软件

开发流程中进行，有时也会将结对编程视为轻量型代码审查的一种。例如，在代码登录后，源代码管理系统自动将代码邮寄给审查者，或者作者及审查者利用配合代码审查的软件进行审查。

在实际使用中，代码检查比动态测试更有效率，能快速找到缺陷，发现30%～70%的逻辑设计和编码缺陷。代码检查应在编译和动态测试之前进行，检查前应准备好需求描述文档、程序设计文档、程序源代码清单、代码编码标准和代码缺陷检查表等。

2. 静态结构分析

静态结构分析是测试者通过使用测试工具分析程序源代码的系统结构、数据结构、数据接口、内部控制逻辑等内部结构，生成函数调用关系图、模块控制流图、内部文件调用关系图等各种图形图表，清晰地标识整个软件的组成结构，便于理解。通过分析这些图表，检查函数的调用关系是否正确、是否存在孤立的函数而没有被调用、明确函数被调用的频繁度，对调用频繁的函数可以重点检查。同时检查编码的规范性、资源是否释放、数据结构是否完整和正确、是否有死代码和死循环、代码本身是否存在明显的效率和性能问题、类和函数的划分是否清晰且易理解、代码是否有完善的异常处理和错误处理机制。

3. 代码质量度量

代码质量包括6个方面，即功能性、可靠性、易用性、效率、可维护性和可移植性。软件质量是软件属性的各种标准度量的组合。针对软件的可维护性，目前业界主要存在3种度量参数：Line复杂度、Halstead复杂度和McCabe复杂度。

1）Line复杂度

Line复杂度是以代码的行数作为计算基准。

2）Halstead复杂度

Halstead复杂度是一种根据程序中运算符和操作数的总数来度量程序复杂度的方法。

例如，任意程序P，总是由操作符和操作数通过有限次的组合连缀而成。P的符号表词汇量 $\eta=\eta_1+\eta_2$（η_1：唯一操作数数量，η_2：唯一操作符数量）。设 N_1 是P中出现的所有操作符，N_2 是程序中出现的所有操作数。度量指标如下：

程序长度 $N=N1+N_2$

程序容量 $V=N\times\log_2\eta$

程序语言等级 $L=V\ ?\ /\ V(V\ ?$ 是程序实现时可能的最小代码容量）

编写程序的效率 $E=V\ /\ L$

编写程序的时间 $T=E/18$

程序错误预测 $B\ =\ E(2/3)\ /\ 1000$

3）McCabe复杂度

McCabe复杂度一般称为圈复杂度，是一种基于程序控制流的复杂性度量方法。它以图论为基础，首先将软件程序转化为流程图，然后将流程图转化为有向图，最后用该有向图的环路数作为程序复杂度的度量值。在程序控制流程图中，节点是程序中代码的最小单元，边代表节点间的程序流。例如，一个有 e 条边和 n 个节点的流程图F，其圈复杂

度为：VF＝$e-n+2$，圈复杂度越高，程序中的控制路径越复杂。

一段求数组中最大值索引的程序代码如图 3-8 所示，其中，list 为存放整型数的数组。

```
public int max(int[] list){
    int max = 0;
    for(int i = 1; i < list.length; i++){
        if(list[i] > list[max]){
            max = i;
        }
    }
    return max;
}
```

图 3-8　求数组中最大值索引

该代码的流程图如图 3-9 所示，将其转换为有向图如图 3-10 所示。

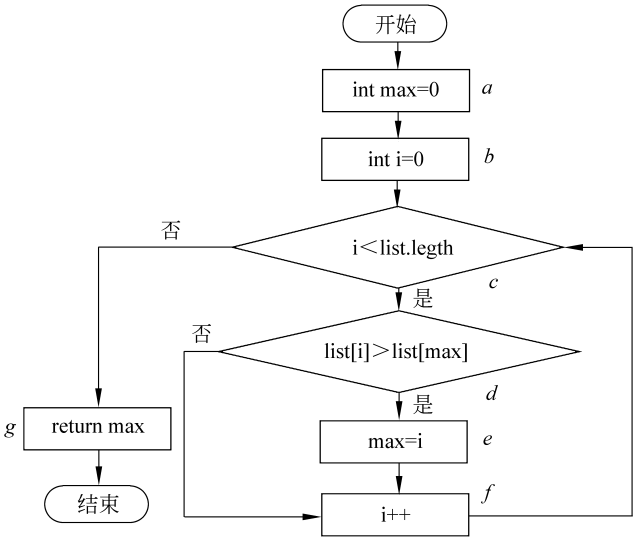

图 3-9　求数组中最大值索引的程序流程图

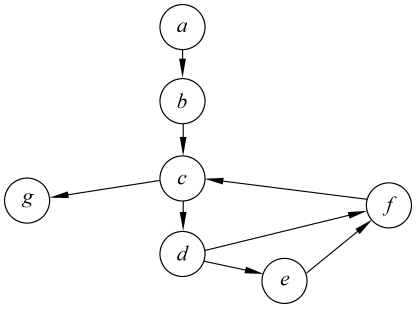

图 3-10　求数组中最大值索引的有向图

图 3-10 中，边数 $e=8$，节点数 $n=7$，所以其圈复杂度为 VF＝8－7＋2＝3。

静态测试在软件测试中起着重要作用，其完成的主要工作包括以下内容。

- 发现程序错误，例如，错用局部变量和全局变量、未定义变量、不匹配参数、不适当循环嵌套或分支嵌套、死循环、不允许的递归、调用不存在的子程序、遗漏标号或代码等。
- 找出问题根源，即找出从未使用过的变量、不会执行到的代码、从未使用过的标号、潜在的死循环等。
- 提供程序缺陷的间接信息，例如，所有变量和常量的交叉应用表、是否违背编码规则、标识符的使用方法和过程的调用层次等。
- 为进一步查找问题做好准备。
- 选择测试用例。
- 进行符号测试。

以使用 Java 语言编写的一段求两个数中较大数的程序为例。图 3-11 为该程序的源代码，该程序通过调用 max()函数求出较大的数。对图 3-11 中程序进行静态分析，发现该程序存在三个问题：一是程序没有注释，二是子函数 max 没有返回值类型，三是存在精度丢失问题。

```
import java.util.Scanner;

public class Test{
    public static max(float x, float y){
        float z;
        z = x > y ? x : y;
        return z;
    }

    public static void main(String[] args){
        float a, b;
        int c;
        Scanner in = new Scanner(System.in);
        a = in.nextFloat();
        b = in.nextFloat();
        c = max(a,b);
        System.out.printf("Max is %d\n",c);
    }
}
```

图 3-11　求两个数中的最大数的程序

因此，静态测试具有发现缺陷早、降低返工成本、覆盖重点和发现缺陷概率高的优点以及耗时长、技术能力要求高的缺点。

3.5.2　动态测试

动态测试方法是指通过运行被测试程序，检查运行结果与预期结果的差异，分析运行效率、正确性和健壮性等性能的测试方法。目前，动态测试是软件测试工作的主要方式。

动态测试的内容包括功能确认与接口测试、覆盖率分析和性能分析等。

1. 功能确认与接口测试

功能确认与接口测试就是在模拟环境下，运用动态测试方法，验证被测试软件是否满足需求规格说明书的需求。本书其他章节内容大都围绕功能测试展开，这里重点介绍接口测试。接口测试主要用于检测外部系统与系统之间、内部各子系统之间的交互点。它是按约定的格式（接口）给待测试的软件传入某种数据，检查接口返回值是否正确。接口测试分为手动接口测试和自动接口测试。现在，很多系统前后端架构是分离的，从安全层面来说，只依赖前端架构进行限制已经完全不能满足系统的安全性要求，还需要后端架构同样进行控制，在这种情况下就需要从接口层面进行验证。

以 HTTP 接口为例。用户登录时，输入正确的用户名和密码，浏览器将输入信息打包发送给服务器，服务器通过数据库校验该用户名和密码是否正确，如果正确，服务器生成一个 session id 和 cookie 以及“success”提示信息。此时，测试人员可以通过抓包工具如 fiddler 或 charles 抓取浏览器和服务器之间传输的数据，如果需要抓取底层数据包，可以使用 Wireshark 工具进行抓取。根据抓取到的数据包准备将要输入接口的数据包，然后用发包工具将数据发送给服务器端的接口，校验其返回值是否符合要求。这是手工接口测试的流程，也可以把接口测试流程编写成脚本，通过人工触发批量执行这些脚本，自动校验返回结果，从而完成接口的自动测试。

2. 覆盖率分析

覆盖率分析是度量测试完整性的一个手段，由测试需求和测试用例的覆盖情况或已执行代码的覆盖情况来表示。覆盖率分析有两种评测手段，一种是基于需求的测试覆盖，另一种是基于代码的测试覆盖。基于需求的测试覆盖是将每一条需求与测试之间建立一种映射关系，最终保证测试可以覆盖每个需求，进而保证软件实现每一个需求。由于基于需求的测试覆盖在流程上是重量级的，现在人们通常所说的测试覆盖率默认指代码覆盖。基于代码的测试覆盖是指至少执行一次的条目数占整个条目数的百分比。这里的“条目数”如果是代码语句，就是代码行覆盖率；如果是函数，就是函数覆盖率；如果是路径，就是路径覆盖率。代码覆盖率分析一般由工具软件完成，目前市场上基于 Java 的主要代码覆盖率工具有 EMMA、JaCoCo 等。

3. 性能分析

性能分析是验证软件在正常环境和系统条件下重复使用时是否能达到某些性能指标。本书第 7 章将详细介绍该方法，此处不再赘述。

3.6 黑盒测试与白盒测试

软件产品的测试可以使用黑盒测试与白盒测试中的一种或两种方法来完成，如图 3-12 所示。黑盒测试是将软件程序当成一个黑盒子，测试人员看不到盒子里面是怎么运行的，只能通过输入数据查看输出结果是否是预期结果。白盒测试是把盒子当成透明的，测试人员可以清楚地看到盒子的内部结构，可以根据内部结构进行测试。