第 3 章 软件测试技术体系

在第 2 章中，带领读者站在用户角度，通过使用软件，找到软件功能上的缺陷。而在实际测试过程中，通常会从多个角度测试软件产品。

3.1 软件测试类型

按照项目的实际测试活动，下面给出最常见的测试类型：功能测试、接口测试及性能测试，如图 3-1 所示。

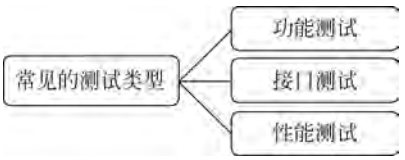

图 3-1 常见的测试类型

3.1.1 功能测试

软件产品必须具备一定的功能，借助这些功能为用户服务，一个公司如果只做一种类型的测试，那一定是功能测试。功能测试一般是在整个软件产品开发完成后，通过直接运行软件的方式，对前端（用户界面）的输入与输出功能进行测试，检验软件能否正常使用各项功能、业务逻辑是否清楚、是否满足用户需求。功能测试所涉及的软件产品可能是 Web 程序、手机 APP，也可能是微信小程序。在前面章节中所列举的对火柴人打羽毛球游戏的测试就属于功能测试，通过直接运行软件，检验游戏中的各项功能是否正确实现，以及是否满足游戏用户的需求。

3.1.2 接口测试

很多软件公司在初期仅进行功能测试，功能测试有一个很大的限制条件，即只有系统（前端界面与后端业务逻辑）开发完成后，测试人员才可以进行功能测试，这时留给测试人员的工作时间往往比较少。

接口测试最重要的一个意义就是可以使得测试提前切入。测试人员可以在界面没有开发完成之前就开始测试，以便提早发现问题。一般来说，软件后台接口开发基本完成之后，就需要开始接口测试。

接口其实就是前端与后端进行沟通交互的桥梁。接口测试除了可以将测试工作前置

外，还可以解决下面的一些问题。例如，在用户注册功能中，需求规定用户名为 6～18 个字符，可以包含字母(区分大小写)、数字和下画线。在对用户名规则进行功能测试时，若输入 20 个字符或输入特殊字符，这些测试用例都可以对软件前端进行功能校验，却可能没有对软件后端进行功能校验，假如此时有人通过抓包绕过前端校验直接将非法数据发送到后端，软件会如何处理？答案是：如果用户名和密码未在后端做校验，而又有人绕过前端校验的话，那用户名和密码就可以随便输入了。

如果是登录功能出现类似的问题，那别有用心的人还可能会通过 SQL 注入、拖库等手段盗取数据信息。拖库原本是数据库领域的术语，指从数据库中导出数据。到了黑客攻击泛滥的今天，它被用来指网站遭到入侵后，黑客窃取其数据库，甚至有可能窃取管理员权限。

所以，接口测试的必要性就体现出来了，通过接口测试，可以达到一些功能测试完成不了的测试效果：

- 可以发现很多在前端页面上操作时发现不了的 Bug；
- 可以检查系统的异常处理能力。

接口测试相对 UI 测试也比较稳定，其更容易实现自动化持续集成，降低人工回归测试的人力成本与时间成本，缩短测试周期，支持软件系统后端的快速发版需求。

下面给出一个注册接口示例，帮助读者认识接口测试。注册接口信息如图 3-2 所示。

接口名称	注册第三步
接口地址	/appapi/registerRealName
接口方法	Post，编码 utf-8;userName 用 encodeURI 转码

● 输入参数定义

参数	类型	是否必须	描　述
userId	string	是	用户 ID
userName	string	是	真实姓名
idNumber	string	是	身份证

● 返回数据说明

```
{
    "resultCode":0,
    "resultMsg":"",
}
```

字段名称	类型	描　述
resultCode	int	成功返回 0 ，失败返回>0 413：请您填写有效的姓名 414：请您填写有效的身份证号码 420：身份证号已被认证 421：开户失败，请联系客服 418：身份证号与姓名不匹配，身份认证已达今日最大次数 419：身份证号与姓名不匹配，身份认证还可提交(4)次
resultMsg	string	失败时返回的错误信息

图 3-2　注册接口文档

从图 3-2 可以看出，这个接口是一个注册接口，接口文档给出了接口的地址、方法、参数及返回值。该接口有 3 个参数，分别是用户 ID、真实姓名及身份证号，返回信息为 JSON 数据。测试时，需要通过超文本传输协议（HyperText Transfer Protocol，HTTP）将设计好的测试数据发送至接口，验证返回的数据内容是否符合预期。一般使用 Postman、JMeter 等工具进行接口测试。

3.1.3 性能测试

当功能测试通过后，软件系统上线运行前，还需要对其进行性能测试。想象一个场景，淘宝双十一活动时，大量用户在同一时间段访问系统，系统是否可以正常运行？系统是否能够及时反应？图 3-3 所示的是某软件在大量用户同时访问时不能再成功对外提供服务的情况。

图 3-3 系统崩溃

性能测试是指通过模拟生产运行的业务压力或用户使用场景，测试系统的性能是否满足生产性能的要求，目的是为软件产品的使用者提供高质量、高效率的软件产品。

3.2 软件测试级别

针对不同开发阶段的测试目的，测试活动分为单元测试、集成测试、确认测试、系统测试及验收测试等级别。

3.2.1 单元测试

单元测试是对已实现的软件的最小单元进行测试，以保证构成软件的各个单元的质量。单元测试中的“单元”是软件系统或产品中可以被分离但又能被测试的最小单元。这些最小单元可以是一个类，一个子程序或一个函数，也可以是这些很小的单元构成的更大单元，如一个模块或一个组件。

在单元测试活动中，强调被测试对象的独立性。软件的独立单元与程序的其他部分被隔离开，以避免测试时其他单元对该单元的影响。这样，在测试时，既可以缩小问题分析范围，又可以比较彻底地消除各个单元中可能存在的问题，降低后期在实施功能测试和系统测试时可能带来的问题查找的困难级别。

单元测试应从各个单元层次对单元内部算法、外部功能实现等进行检验，包括对程序代码的评审和通过运行单元程序验证其功能特性等内容。单元测试的目标不仅包括测试单元代码的功能性，还需确保程序代码在结构上的安全和可靠。执行完全的单元测试，可以减少应用级别测试所需的测试工作量，从根本上减少缺陷发生的可能性。通过单元测试，希望达到下列目标。

- 单元体现了其特定的功能，如果需要，返回正确的值。
- 单元的运行能够覆盖预先设定好的各种逻辑。
- 在单元工作过程中，其内部数据能够保持完整性，包括全局变量的处理、内部数据的形式、内容及相互关系等不发生错误。
- 可以接收正确数据，也能处理非法数据，在数据边界条件上，单元也能够正确工作。
- 该单元的算法合理，性能良好。
- 该单元代码经过扫描，没有发现任何安全性问题。

在实际测试工作中，测试人员发现，如果仅对软件进行功能测试和验收测试，似乎缺陷总是找不完，不是这边出现缺陷，就是那个角落发现问题，每天报告的缺陷虽不多，但总能发现新的且比较严重的缺陷，测试没有尽头。为什么会出现这种情况？

产生这种现象的主要原因就是在功能测试之前没有进行充分的单元测试。虽然测试时不可能穷尽所有程序路径，但整个软件的基础构成单元如果没有进行单元测试，则软件基础就不稳，仅靠功能测试和验收测试根本不能彻底解决问题。单元的质量是整个软件质量的基础，所以充分的单元测试是非常必要的。

通过单元测试可以更早地发现缺陷，缩短开发周期，降低软件成本。多数缺陷在单元测试中很容易被发现，但如果没有进行单元测试，那么这些缺陷在后期测试时就会隐藏得很深而难以发现，最终导致测试周期延长，开发成本急剧增加。

3.2.2 集成测试

实际工作中常会遇到这样的情况，每个模块的单元测试已经通过，但把这些模块集成在一起之后，软件却不能正常工作。出现这种情况的原因，往往是模块之间的接口出现问题，如模块之间参数传递不匹配、全局变量被误用或误差不断积累达到不可接受的程度等。

1. 集成测试模式

集成测试模式是软件集成测试中的策略体现，直接关系到开发和测试的效率。集成测试模式可以分为两种基本模式。

- 非渐增式模式：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，也常被称为大棒模式。
- 渐增式模式：把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。

采用大棒模式，设计人员习惯于把所有模块按照要求一次全部组装起来，然后进行整体测试。而在测试之前，系统集成方面的问题不断积累，问题越来越多，所以在测试时会发现一大堆错误。同时，由于一次性集成，模块数量多，模块之间的关系比较复杂，这些错误交织在一起，很难确定问题出现在哪里，定位和纠正每个错误就变得非常困难。开发者不得不耗费大量的时间和精力来寻找这些缺陷的根源，造成很大的开发成本。

与之相反的是渐增式模式，程序一段一段地扩展，测试的范围一步一步地增大，错误易于定位和纠正。虽然渐增式模式需要编写的代码偏多，工作量较大，但它有明显的优势：能更早地发现模块间接口错误，使测试更彻底。同时，渐增式模式发现错误后，由于短时间内（如一天之中）代码发生变动较小，更容易判断问题出现在什么地方，因此可以很快找到出错的位置，方便修正问题。所以，业界普遍采用渐增式模式，也就是持续集成的策略。使用持续集成，绝大多数模块之间的接口缺陷，在其引入的第一天可能就会被发现。软件开发中各个模块可能不是同时完成的，测试人员可以尽可能早地集成已完成的模块，有利于尽早发现缺陷，避免像大棒模式那样一下子出现大量的缺陷。

2. 自顶向下集成测试

自顶向下集成测试是从主控模块开始，沿着软件的控制层次向下移动，逐渐把各个模块结合起来。在自顶向下组装过程中，可以使用深度优先策略或宽度优先策略。

如图 3-4 所示，自顶向下集成测试的具体步骤如下。

（1）对主控模式进行测试，测试时用桩程序代替所有直接附属于主控模块的模块。

（2）根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代替一个桩程序（新结合进来的模块往往又需要新的桩程序）。

（3）在加入每一个新模块的时候，完成其集成测试。

（4）为了保证新加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。

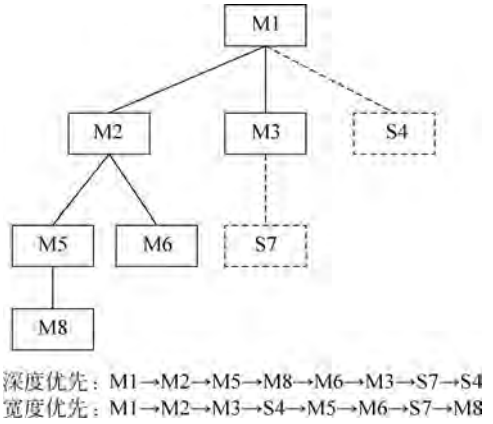

图 3-4　自顶向下集成方法

从步骤（2）开始不断地重复进行步骤（2）（3）（4）过程，直至完成所有模块的集成。自顶向下集成模块时，一般需要开发桩程序，不需要开发驱动程序。模块层次越高，其影响面越广，重要性也越高。自顶向下集成测试能够在测试阶段的早期验证系统的主要功能逻辑，越重要的控制模块，越能优先得到测试。但软件中使用频繁的基础函数一般处在模块结构图的底层，由于这些模块集成的时间比较晚，因此这些基础函数中的错误也会发现得比较晚。另外，该方法需要编写大量的桩程序，因此在具体实施时可能会遇到比较大的阻力。

3. 自底向上集成测试

自底向上集成测试是指从底层模块（即软件模块结构图中最底层的模块）开始，逐步向上不断集成模块进行测试的方法，以图 3-5 所示的模块结构为例，自底向上集成测试的具体策略如下。

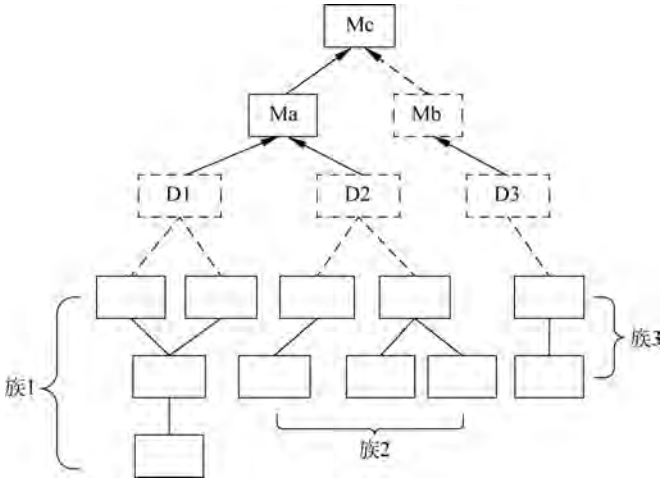

图 3-5　自底向上集成方法

(1) 把底(下)层模块组合成实现某个特定的软件子功能族。

(2) 编写一个驱动程序,调用上述底(下)层模块,并协调测试数据的输入和输出。

(3) 对由驱动程序和子功能族构成的模块集合进行测试。

(4) 去掉驱动程序,沿软件模块结构从下向上移动,加入上层模块形成更大的子功能族。

从步骤(2)开始不断地重复进行步骤(2)(3)(4)过程,直至完成所有模块的集成。自底向上集成测试一般不需要创建桩程序,但需要创建驱动程序。相比桩程序而言,驱动程序比较容易创建。自底向上集成测试能够在最早时间完成对基础函数的测试,其他模块可以更早地调用这些基础函数,有利于提高开发效率,缩短开发周期。但是控制能力强、影响面广的上层模块,其测试时间会靠后,若在测试后期才发现这些模块有问题,修改这些缺陷就会很困难,或者修改的影响面很广,从而存在很大的风险。

4. 混合策略

在实际测试过程中,一般会将自顶向下集成测试和自底向上集成测试有机地结合起来,形成混合测试策略,完成软件系统的集成测试,这种混合测试策略可以发挥自顶向下集成测试和自底向上集成测试的优点,避免其缺点,从而有效地提高测试效率。例如,在测试早期,使用自底向上集成方法测试少数的基础模块(函数),然后再采用自顶向下集成方法完成集成测试。更多的时候,会同时使用自底向上法和自顶向下法进行集成测试,即采用两头向中间推进的策略,配合软件开发的进程,大大降低驱动程序和桩程序的编写工作量,加快开发的整体速度。因为自底向上集成时,先期完成的模块将是后期模块的桩程序,而自顶向下集成时,先期完成的模块将是后期模块的驱动程序,从而使后期模块的单元测试和集成测试出现了部分的交叉,这不仅减少了测试代码的编写,也有利于提高工作效率。这种方法俗称三明治集成测试方法,如图 3-6 所示。

改进的三明治集成测试方法,不仅自两头向中间集成,而且保证每个模块得到单独的测试,使测试进行得更彻底,如图 3-7 所示。

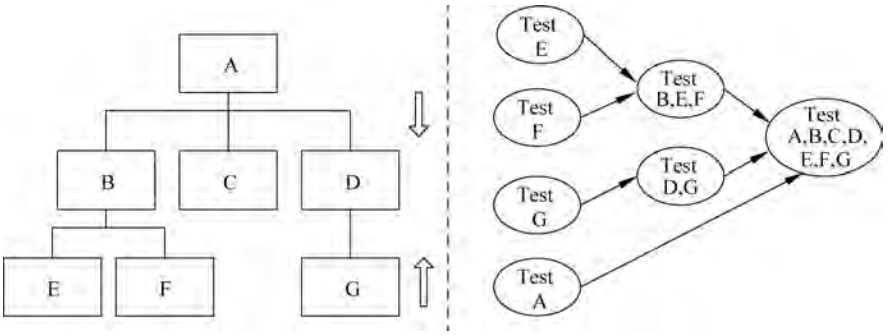

图 3-6　三明治集成测试方法

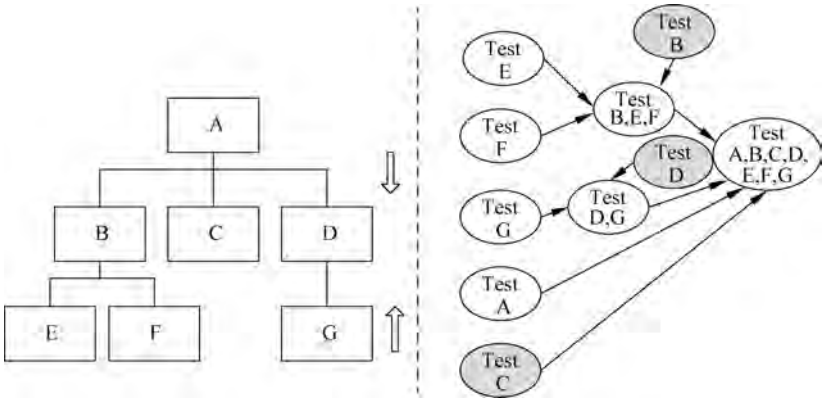

图 3-7　改进的三明治集成测试方法

3.2.3　确认测试

经过集成测试后，已经按照设计要求把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接下来就应该进一步验证软件的有效性，这就是确认测试。测试人员通过确认测试向用户表明软件系统能够按预定要求工作。

确认测试又称有效性测试，是在模拟的环境下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书中列出的需求，即验证软件的功能是否与用户的要求一致。软件的功能要求在软件需求规格说明书中已经明确规定，即需求规格说明书包含的用户需求信息就是软件确认测试的基础。

3.2.4　系统测试

当组件、模块构建集成为一个完整的系统之后，接下来实施系统测试。系统测试是对软件整体进行测试，以保证业务、功能和非功能的要求。但是，为了将功能测试、UI 测试等区分开，实际中系统测试特指那些针对软件非功能特性而进行的测试，也就是说，系统测试是验证软件系统的非功能特性。

有时也会针对组件、模块进行性能测试、安全性测试等非功能测试，但最终这些测试必须针对整个软件系统进行，必须将系统作为一个整体进行测试。

要理解系统测试，首先要理解软件系统的非功能特性。用户的需求可以分为功能性需求和非功能性需求，而这些非功能性需求被归纳为软件产品的各种质量特性，如安全性、兼容性和可靠性等。如果系统只是满足了用户的功能需求，而没有满足非功能特性需求，其最终结果是用户对产品还是不满意。例如，某个网站功能齐全，但运行时不够稳定，有时可以访问，有时不能访问，并且每打开一个页面都需要几分钟时间。像这样的网站，即使功能很强，用户也不愿意访问，用户绝不会忍受这种不稳定的、低性能的系统。

综上所述，系统测试就是针对上述非功能特性展开的，是验证软件产品是否符合这些质量特性要求的测试。系统测试包括易用性测试、性能测试、安全测试和兼容性测试等，具体内容如表 3-1 所示。

表 3-1　系统测试类型

测 试 类 型	描　　述
易用性测试	试图发现人为因素或易用性问题
性能测试	用来衡量系统占用资源和系统响应、表现的状态。如果系统用完了所有可用的资源，系统性能就会明显下降，甚至死机。系统操作性能不仅受到系统本身资源的影响，也受到系统内部算法、外部负载等多方面的影响，如内存泄露、缺乏高速缓存机制及大量用户同时发送请求等
安全性测试	测试系统和数据的安全程度，包括功能使用范围、数据存取权限等受保护和受控制的能力。数据与系统的分离、系统权限和数据权限分别设置等都可以提高系统的安全性
兼容性测试	测试软件从一个计算机系统移植到另一个系统或环境的难易程度，或者是一个系统与外部条件共同工作的难易程度。兼容性表现在多个方面，如系统软件与硬件之间的兼容性、软件的不同版本之间的兼容性、不同系统之间的数据相互兼容性等

3.2.5　验收测试

验收测试是在软件产品完成了功能测试和系统测试之后、产品发布之前进行的软件测试活动，它是技术测试的最后一个阶段，也称为交付测试。

验收测试按照项目任务书或合同、供需双方约定的验收依据文档对整个系统进行测试与评审，验收测试决定用户是否接收系统。验收测试结束后，根据验收通过准则分析验收测试结果，做出测试评价及是否通过验收。

验收测试通常会有以下 4 种情况。

- 测试项目通过。
- 测试项目没有通过，并且不存在变通方法，需要做很大的修改。
- 测试项目没有通过，但存在变通方法，在维护后期或下一个版本改进。
- 测试项目无法评估或无法给出完整的评估。此时必须给出原因，如果是因为该测试项目没有说清楚，应该修改测试计划。

3.3 测试方法

常见的测试方法有黑盒测试、白盒测试和灰盒测试。

3.3.1 黑盒测试

黑盒测试通过软件的外部表现发现缺陷和错误。黑盒测试把测试对象看成一个黑盒子，完全不考虑程序内部结构和处理过程，仅针对程序是否能适当地接收输入数据、是否能产生正确的输出信息等进行测试，如图 3-8 所示。

图 3-8　黑盒测试

3.3.2 白盒测试

通过对程序内部结构的分析与检测寻找软件问题的方法称为白盒测试，又称为结构测试。白盒测试可以把程序看成是一个装在透明盒子里的代码，测试人员清楚地了解程序的内部结构和处理过程，通过检查程序的内部结构及逻辑路径是否正确、检查软件内部动作是否符合软件设计说明书的规定发现程序中的缺陷，如图 3-9 所示。

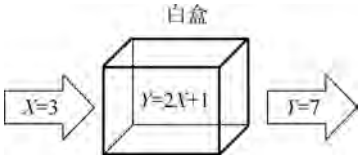

图 3-9　白盒测试

3.3.3 灰盒测试

灰盒测试是介于白盒测试与黑盒测试之间的测试。灰盒测试关注输出对于输入的正确性，同时也关注程序内部表现，但这种关注不像白盒测试那样详细和完整，只是通过一些表面的现象、事件和标志判断程序内部的运行状态。因此，可以这样定义灰盒测试：灰盒测试是基于程序运行时的外部表现，同时又结合程序内部逻辑结构设计用例、执行程序并采集程序路径执行信息和外部用户接口结果的测试技术。

3.4 测试手段

手工测试和自动化测试是很多测试人员争相讨论的两种测试方法。有人对自动化测试趋之若鹜，也有人对自动化测试嗤之以鼻。事实上，这两种测试手段互为补充，合理选择不

同的测试手段能使测试更易组织、更高效。在测试中，很多类似数据的正确性验证、软件界面美观与否、业务逻辑的满足程度等都离不开测试人员的人工判断。因此，自动化测试不可能完全替代手工测试；但若测试时仅依赖手工测试，又会使测试低效。

3.4.1 手工测试

手工测试有其不可替代的地方，因为人具有很强的判断能力，而工具没有。手工测试不可替代的地方至少包括以下 3 点。

- 测试用例的设计：测试人员的经验和对错误的判断能力是工具不可替代的。
- 界面和用户体验测试：人类的审美观和心理体验是工具不可模拟的。
- 正确性检查：人们对是非的判断以及逻辑推理能力是工具不具备的。

3.4.2 自动化测试

在测试执行过程中，经常需要进行多轮测试，而且随着软件版本的不断升级，测试的工作量也会越来越大，很多测试会被不断地重复执行。如果这些测试全靠手工完成，不仅需要占用很多人力资源，而且工作还重复单调。

自动化测试通过编写测试代码代替手工的重复性测试工作，对经常需要多次回归的测试用例进行代码化，可以提高测试效率，解放人力。为了使线上环境更加稳定，可以将软件的核心功能与业务脚本化，进行线上巡检，使线上生产环境更加稳定。实际项目中的自动化测试，一般包括 UI 自动化测试、接口自动化测试及单元自动化测试，如图 3-10 所示。

图 3-10　自动化测试应用领域

1. UI 自动化测试

UI 是指对软件的人机交互、操作逻辑、界面美观等方面的整体设计，是系统与用户之间进行交互和信息交换的媒介，它实现信息的内部形式与人类可以接受形式之间的转换。UI 定义广泛，包含了人机交互与图形用户接口，凡参与人类与机械的信息交流的领域都存在用户界面。

随着互联网的应用和普及，网络产品界面设计(Website User Interface，WebUI)发展迅速，其设计范围包括常见的网站设计(如电商网站、社交网站)、网络软件设计(如邮箱、SaaS 产品)等。

用户界面测试(User Interface Testing)简称 UI 测试，主要测试用户界面功能模块的布局是否合理、整体风格是否一致和各个控件的放置位置是否符合用户使用习惯等，更重要的是测试操作是否便捷、导航是否简单易懂、界面中文字是否正确、命名是否统一、页面是否美观以及文字、图片组合是否完美等。

UI 测试是当前比较耗费人力的环节，大部分专职的测试人员日常工作就是 UI 测试。

“工欲善其事，必先利其器”，测试人员需要自动化工具提升其日常工作效率。例如，需要不断对一个表单提交进行测试，或者需要重复对一个查询结果进行测试，此时就可以通过相应的自动化测试工具模拟这些操作，从而解放重复的劳动。

UI 自动化测试就是用户界面层的自动化测试，通过代码模拟用户在界面上的单击及输入等操作，代替功能测试中需要重复执行的测试用例，提高测试效率。UI 层的自动化测试工具非常多，比较主流的有 QTP、Robot Framework、Watir、Selenium 等，其中，Selenium 是目前比较常用的 UI 自动化测试工具。

2. 接口自动化测试

接口测试是对系统或组件之间的接口进行测试，主要是校验数据的交换、传递和控制管理过程，以及相互逻辑依赖关系。在 UI 自动化测试中，由于页面的变动，UI 自动化测试并不是很稳定，而接口自动化测试则没有这个问题。在分层测试的“金字塔”模型中（如图 3-11 所示），接口测试属于第二层服务集成测试范畴。相比 UI 层（主要是 Web 或 APP）自动化测试而言，接口自动化测试收益更大。接口自动化可以通过接口测试工具或编写 Python 代码，模拟用户向服务器发送请求报文，判断返回报文是否符合预期来实现。

图 3-11　分层测试的“金字塔”模型

接口自动化测试容易实现，维护成本低，有着更高的投入产出比，是开展自动化测试的首选，目前接口自动化测试在企业中的应用越来越广泛。

3. 单元自动化测试

单元测试是在代码编写阶段针对程序源代码进行的测试。执行单元测试时，需要为被测单元编写相应的驱动程序和桩程序，这些驱动程序和桩程序的编写费时费力，而自动化单元测试可以很好地解决这个问题。

单元自动化测试可以使用单元自动化测试工具或框架实现，不同的语言，其单元测试框架也不同，几乎所有的主流语言都有其对应的自动化测试工具或框架，如 Java 的 JUnit、testNG，C# 的 NUnit，Python 的 Unittest、Pytest 等。不过单元自动化测试对测试工程师的编码能力要求较高，大部分公司在这个层级都无法很好地推行自动化测试。

3.5　本 章 小 结

本章介绍了软件测试的技术体系、常见的测试类型分析、不同级别的测试、测试方法和测试手段等。对于这些测试技术体系，新入门的测试人员不应该追求样样精通，而应该遵循了解、储备、使用的原则和顺序逐步学习，先简单了解各种测试技术的基本原理和方法，储备这些测试技术的相关材料和工具，当开始测试项目时，迅速地找到相关的材料和工具进行快速学习，掌握相关的技术并应用到测试项目中。

3.6 课后习题

1. 不定项选择题

(1) 关于单元测试,下列说法正确的是(　　)。

A. 单元测试是对软件设计的最小模块进行的测试

B. 多个模块不可以进行单元测试

C. 类、文件、窗口都可以作为一个单元进行测试

D. 单元测试以白盒测试为主

(2) 在软件测试中,白盒测试方法通过分析程序的(　　)设计测试用例。

A. 内部逻辑　　B. 功能　　C. 输入数据　　D. 应用范围

(3) 软件兼容性需要测试的要点包括(　　)。

A. 与操作系统的兼容性　　B. 数据兼容性测试

C. 与其他非同类软件的兼容性　　D. 与其他同类软件的兼容性

(4) 从技术角度分析,不是同一类型的测试是(　　)。

A. 黑盒测试　　B. 白盒测试　　C. 单元测试　　D. 灰盒测试

(5) 组装测试又称为(　　)。

A. 集成测试　　B. 系统测试　　C. 回归测试　　D. 确认测试

(6) 自底向上集成测试需要测试人员编写(　　)

A. 驱动程序　　B. 桩程序　　C. 支持程序　　D. 主程序

(7) 对于软件测试分类,下列各项都是按照测试不同阶段进行划分的,除了(　　)。

A. 单元测试　　B. 集成测试　　C. 黑盒测试　　D. 系统测试

2. 问答题

(1) 为什么要进行单元测试? 单元测试的任务和目标是什么?

(2) 比较自顶向下集成测试方法和自底向上集成测试方法各自的优缺点。

(3) 试说明系统测试应包含哪些内容。

(4) 比较手工测试与自动化测试的优缺点。

(5) 试说明你了解的测试类型。

3. 实践题

下面是用C语言编写的三角形形状判断程序,请分别从单元测试检测程序代码的角度、功能测试检测程序功能的角度对此程序进行测试,并按照你的编程经验尝试给出这两种测试思路下的测试用例。

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main()
{
  int a,b,c;
  printf("输入三角形的三个边:");
```

```
    scanf(" %d %d %d",&a,&b,&c);

    if(a<=0||b<=0||c<=0)
        printf("不符合条件,请重新输入 a,b,c\n");
    else if(a+b<=c||abs(a-b)>=c)
        printf("不是三角形\n");
    else if(a==b&&a==c&&b==c)
        printf("这个三角形为等边三角形\n");
    else if(a==b||a==c||b==c)
        printf("这个三角形为等腰三角形\n");
    else
        printf("这个三角形为一般三角形\n");
}
```