第5章 软件测试过程

CHAPTER

本章学习目标

- 了解软件测试策略的内容
- 掌握传统的软件测试流程以及流程中的每一个测试步骤
- 学习与了解传统的软件测试模型以及几种简单的软件测试改进模型的特点
- 了解软件敏捷测试—Scrum 流程
- 学习与了解一些软件敏捷测试案例

软件测试在软件开发过程中占有重要的地位。国外测试机构的研究数据显示，在软件开发的工作量中，软件测试工作量占总工作量的 40%以上，软件开发总费用的 30%～50%将用于软件测试活动。对于一些高科技开发系统，软件测试占有的时间和费用可能更多更高。

然而，在传统的软件瀑布开发模型中，软件测试只是作为软件生命周期的一个阶段，即对已开发完毕的软件源代码进行测试。这与现代软件工程思想相违背，因为软件测试活动需要贯穿整个软件生命周期，是软件质量保障的重要手段之一。

现代软件工程提倡“测试先行”，即要求软件开发与软件测试活动交互且并行于整个软件生命周期中。在软件的需求分析阶段，从软件开发者的角度看，主要对拟开发软件提出完整、清晰、具体的要求，明确软件必须为用户实现的任务（功能性需求、非功能性需求、设计约束等）以及完成软件需求建模与评审等工作。从软件测试者的角度看，此时测试的对象是相关文档资料，如用户需求规格说明书等。测试人员需要制订测试计划，和开发人员一起从需求定义的正确性、完整性、可验证性、可行性等方面进行评审。

在软件概要设计和详细设计阶段（概要设计描述拟开发软件总体系统架构中各个模块的划分及相互之间的关系，详细设计描述各个模块具体的算法和数据结构），测试人员一方面需要对软件设计阶段产生的文字、相关图表进行评审，另一方面需要设计出软件主要功能模块的测试用例。

在编码阶段，开发人员主要采用高级程序语言，对已完成详细设计的模块进行编程实现。与此同时，开发人员还要对已有的程序代码进行单元测试，可以是静态分析和动态测试两种白盒测试类型相结合的方式。需要注意的是，单元测试一定是由程序员来完成的。可以根据实际情况，选择桌面检查、代码走查、代码审查等形式进行。在软件测试阶段，测试人员对软件系统进行集成测试、确认测试、系统测试，主要测试系统的功能、性能等方面是否与用户需求相一致。最后，在软件检验交付与维护阶段，测试人员通常会模拟用户使用场景，或在实际用户环境下对系统进行验收测试（通常采用自动化测试工具进行测试验收），包括对软件产品的功能测试、性能测试、回归测试、发布测试等。

5.1 软件生命周期中的测试策略

现代 IT 行业软件测试活动通常由软件开发人员与专门的测试人员（机构、组织）共同策划和管理。也就是说，软件测试是一系列事先需要计划、事中需要管理的活动及过程。所以，开展软件测试活动前需要制定软件测试策略。软件测试策略是指测试将按什么样思路和方式设计、制定以及实施。从宏观上看，测试策略是从原则性、框架性层面来考虑测试设计及测试实施过程，而测试方法则是一些具体性的、方法性的测试技术方面的运用。

通常，软件测试策略的内容主要由部署测试、制订测试计划、执行测试计划、分析测试结果、编写测试报告 5 部分内容组成，如图 5.1 所示。表 5.1 给出了每一部分内容的具体工作。

当然，对于一些大型软件测试项目，还包含对测试活动的总体设计与部署、测试成本与效率分析、测试度量及过程管理等内容。

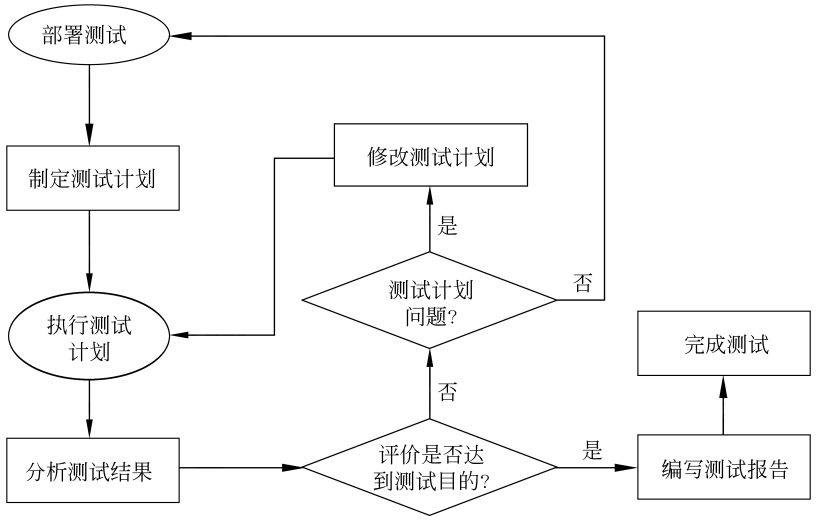

图 5.1 软件测试策略

表 5.1　软件测试策略的主要内容

测 试 策 略	内 容 说 明
部署测试	主要是搭建测试环境。例如，在本地测试环境下部署各类测试服务器、Web 服务器、文件管理服务器、数据库服务器等，安装测试程序运行环境（如 Java 程序的 JDK 运行环境）、相应的测试工具及管理平台，服务器的安全策略设置等
制订测试计划	规定测试活动的范围、方法、资源和进度。明确要执行的测试任务有哪些，即明确需要测试和不需要测试的内容（对于不需要测试的内容，需要给出原因）。确定每个测试任务的责任人、完成时间以及与测试计划相关的风险因素等
执行测试计划	执行测试用例，记录原始测试数据，记录缺陷，对缺陷进行跟踪、管理和监控等
分析测试结果	对测试结果（所发现的缺陷）进行整理、归纳和分析。一般借助于 Excel 文件、数据库和一些直方图、圆饼图、趋势图等进行分析和表示，通过发现缺陷数量或在模块中的分布情况掌握程序代码的质量。通过缺陷趋势分析开发团队解决缺陷的能力或状态
编写测试报告	把测试的过程和结果写成文档，分析发现的问题和缺陷，为纠正软件存在的质量问题提供依据，同时为软件验收和交付打下基础

5.2　传统的软件测试流程

软件测试具有阶段性。按照测试的前后次序，从是否需要执行被测软件的角度，传统的软件测试流程一般按 5 个步骤进行，即单元测试、集成测试、确认测试、系统测试与验收测试，如图 5.2 所示。

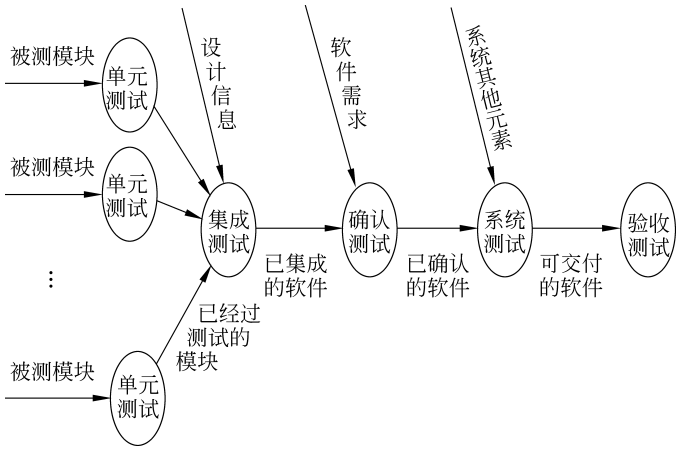

图 5.2　传统的软件测试流程

5.2.1　单元测试

单元测试对用源代码实现的每一个被测模块（程序单元）进行测试，检查各模块内部程序是否正确实现了规定的功能。通常，单元测试在编码阶段进行，由软件开发人员完

成。编写完源程序代码，经过评审和验证，确认没有错误后，就可以着手准备单元测试活动了。从程序的内部逻辑结构出发，利用设计文档，设计出可以验证程序功能、找出程序错误的单元测试的测试用例，从而指导单元测试活动。当然，多个模块也可以平行地独立进行单元测试。

单元测试的具体内容主要有 5 个方面，如表 5.2 所示。

表 5.2 单元测试内容的 5 个方面

测试内容	测试说明
模块接口	包括对调用子模块的参数、参数表、全程数据、文件输入/输出进行测试
局部数据结构	模块内部源代码的数据类型说明、初始化操作、默认值等方面测试
独立路径	对模块中程序执行流程历经的重要路径进行测试
边界条件	选择对模块中的数据流、控制流中刚好等于、小于或大于规定的比较值进行测试
错误处理	测试模块中所含的错误处理程序是否会对一些输入错误或缺陷进行判断，是否会拒绝不合理的输入数据等

单元测试可以以静态测试或动态测试的方式进行。静态单元测试主要指代码走查这类检查性测试，针对代码文本检查，不需编译和运行代码。动态单元测试要通过编写测试代码（或设计测试用例）测试，需编译和运行代码，调用被测代码运行。当然，不论是静态测试还是动态测试的方式，单元测试都可采用人工方式或自动化方式进行。

目前单元测试的模式主要有两种，即代码先行模式与测试驱动模式。前者是一种传统的单元测试方式，即先编码后测试。这种方式易于实施与控制，一般可选择重要模块的代码进行测试。测试驱动模式是指在实现模块代码之前就完成对其测试用例的设计，要求程序员对即将编写的代码进行需求上的细节分析和代码设计方案的考虑。这种单元测试模式通常用于敏捷软件项目的测试活动，会改变程序员的编码习惯。

5.2.2 集成测试

集成测试也称联合测试或组装测试，是在模块完成单元测试之后，把所有模块按照之前软件概要设计阶段规定好的一些设计信息（如模块结构图）的要求组装成相应的子系统或系统的测试活动，目的是检测模块结构组装的正确性。在实际测试中，尽管一些模块通过了单元测试，能够单独工作，但是并不能保证多个模块组合起来也能正常工作。同样，被测程序在某些局部方面反映不出来的问题，有可能会在全局中暴露出来，影响整体功能的实现。

开展集成测试的过程中，需要根据被测模块在模块结构图中的地位设计相应的桩模块与驱动模块。桩模块也称为桩程序，用来模拟被测对象所调用的下一层模块（程序）。驱动模块也称为驱动程序，是指对下层模块进行测试时，用来模拟调用该被测对象的上一层模块（程序）。无论是桩模块还是驱动模块，都不是真正意义上的被测系统模块，都是“假”的模块，只是用于模拟其调用（下一层模块）与被调用（上一层模块）的功能。

例如，在集成测试中，若要测试图 5.3 中模块 B 的功能是否正确，首先需要了解模块 B 在模块结构图中的位置。即模块 B 的上层是模块 A(模块 A 调用模块 B)，模块 B 的下层是模块 E 与模块 F(模块 B 调用模块 E、模块 F)。也就是说，需要为模块 B 设计(构造)一个驱动模块与两个桩模块，用于模拟模块 A 及模块 E、模块 F 的功能，如图 5.4 所示。当图 5.4 所示的模块调用图能够运行并测试通过，才能证明模块 B 通过了集成测试，这时才能使用真正的模块 A 及模块 E、F 替代相应的驱动模块及桩模块。当然，图 5.3 中的其余模块若想通过集成测试，仍然需要为它们设计(构造)相应的驱动模块或桩模块。

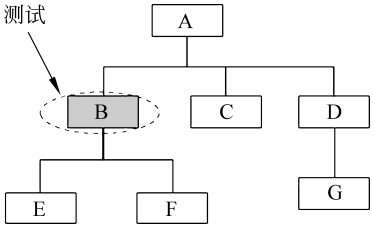

图 5.3　模块结构图

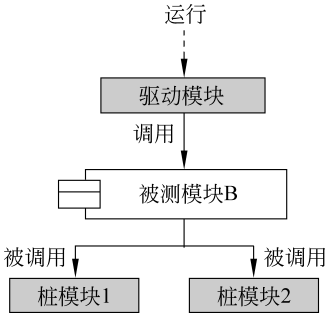

图 5.4　模块调用图

一些常用的集成测试策略如下所示。

1. 非增式集成测试

按照一步到位的方法构造集成测试，通过对每个模块设计相应的桩模块或驱动模块，并完成单元测试，再把所有模块按照模块结构图连接起来，作为一个整体进行测试。

例如，某系统由 5 个模块组成，模块结构如图 5.5 所示。非增式集成测试步骤如下。

(1) 分析模块结构图，确定哪些模块需要设计桩模块，哪些模块需要设计驱动模块，哪些模块既需要设计桩模块，又需要设计驱动模块。

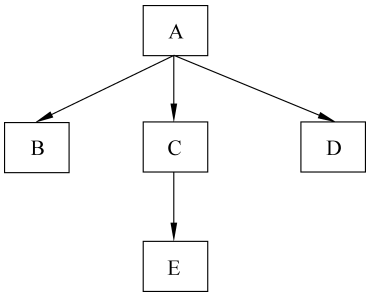

图 5.5　模块结构图

模块 A 同时调用了 3 个模块，所以集成测试时需要为其设计 3 个桩模块，来模拟测试模块 A 调用(B、C、D 3 个模块)功能。但是模块 A 是顶层模块，没有其他模块调用它，所以不需要为其设计驱动模块。同样，由于模块 B、模块 C、模块 D 被模块 A 调用，所以分别要为这 3 个模块设计驱动模块测试它们的被调用功能。由于模块 C 还调用了模块 E，因此在为模块 C 设计驱动模块的同时还要为其设计一个桩模块。模块 E 是最底层模块，只需要为其设计一个驱动模块即可，如图 5.6 所示。

(2) 按图 5.6 所示，为模块 A 至模块 E 依次设计相应的桩模块或驱动模块，完成单元测试后，再按照图 5.5 所示的模块结构图形式连接起来，进行一次集成测试即可。

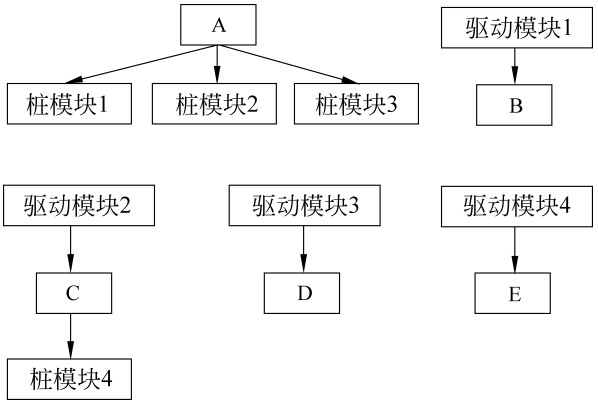

图 5.6　各模块的非增式集成过程图

大棒集成测试方法，即按照模块结构图设计桩模块或驱动模块，完成对每个被测模块的测试活动，然后将所有模块一次性地按照模块结构图连接（组装）起来，进行集成测试。可见，大棒集成测试方法所采用的就是上面提到的非增式集成测试策略。

因为所有的模块都是一次性集成的，所以大棒法集成测试很难确定出错模块的真正位置以及出错的原因。大棒法集成测试只适合规模较小的应用系统，不推荐在规模较大的系统中使用。

2. 增式集成测试

与非增式集成测试不同的是，增式集成测试把单元测试与集成测试结合起来进行，在集成的过程中对每个模块采用边连接边测试的方式，来发现连接过程中产生的问题。这里介绍两种常用的增式集成测试方式。

（1）自顶向下集成。

从顶层模块开始，把顶层模块作为测试驱动模块，通过设计桩模块依次对下层模块进行测试。再逐步把下层的桩模块一次一个地替换为真正的模块进行测试。每替换一个真正的下层模块时，按照同样的方法先设计桩模块，再用该模块的下层真正模块做替换测试，直至整个系统结构被集成完成。

同样，对于图 5.7 所示的模块结构图，图 5.7 中的(a)(b)(c)子图分别给出了采用自顶向下集成方法的测试过程。

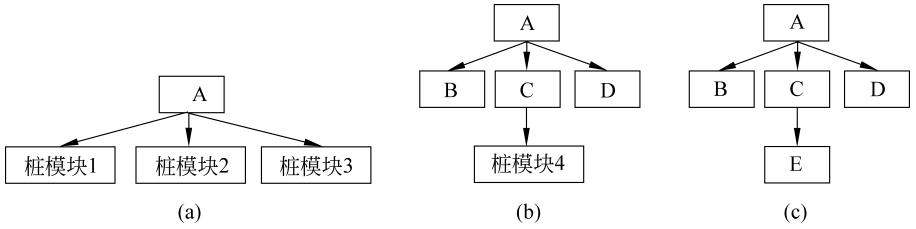

图 5.7　自顶向下集成过程图

(2) 自下向上集成。

模块逐步集成测试工作自最底层的模块逐步向上进行。因为是从最底层开始集成，因而不需要设计桩模块辅助测试。通过设计驱动模块依次对上层模块进行测试，再逐步将上层的驱动模块一次一个地替换为真正的模块进行测试。图 5.8 中的(a)(b)(c)(d)(e)子图分别给出了采用自下向上集成方法的测试过程。

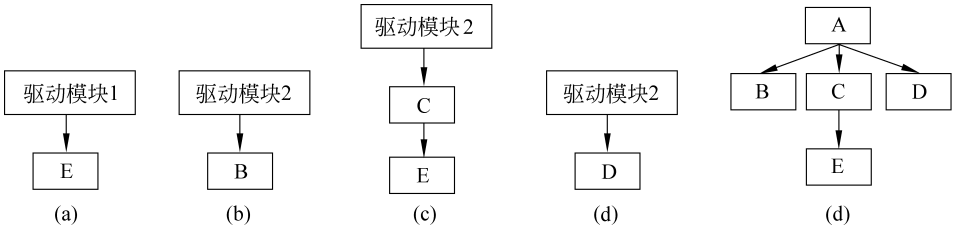

图 5.8　自下向上集成过程图

3. 混合法集成测试

混合法集成测试的策略，是对软件结构中的较上层模块使用“自顶向下”集成测试法，而对于软件结构中的较下层模块使用“自下向上”集成测试法，两者相结合。例如，对于图 5.3 所示的模块结构图，A、B、C、D 4 个模块采用“自顶向下”集成测试策略，E、F、G 3 个模块则采用“自下向上”集成测试策略。

4. 三明治集成测试

三明治集成测试的策略是将“自顶向下”和“自下向上”的集成测试方法有机地结合起来，是一种混合增殖式测试策略。这里仍以图 5.3 所示的模块结构图为例，首先要选择分界模块层。例如，选择模块 B、C、D 所在层次为中间线，模块 B、C、D 所在层次以上采用“自顶向下”的测试方法，模块 B、C、D 所在层次以下采用“自下向上”的测试方法。

三明治集成测试方法的优点是能够适当减少桩模块和驱动模块的开发，缺点是中间层模块(模块 B、C、D 所在层次)不能尽早得到充分测试。

关于软件集成测试，当前 IT 行业采取的持续集成方式是越来越普遍的一种优秀测试实践，即团队开发成员通常每天会对新完成的代码至少开展一次集成测试，也就意味着每天可能发生很多次集成测试活动。

5.2.3　系统测试

系统测试是把已完成集成测试的软件与相关的计算机硬件、网络、外设等其他元素，甚至其他软件系统部署在一起进行的测试活动，从系统的角度来检验和寻找缺陷，主要包含功能测试、性能测试、安全测试、可靠性测试、恢复性测试与兼容性测试等。所以，系统测试的对象不仅包括需要测试的软件产品，也要包含软件依赖的硬件、网络、外设甚至某些支撑类软件、数据及其应用接口等。因此，必须将被测软件与各种依赖的元素充分结合

起来，在系统实际运行环境下测试。

系统测试的内容主要包括对系统的功能性测试与非功能性测试两大方面。功能性测试主要包括验证系统输入/输出行为的各种测试，一般采用黑盒测试方法。功能性测试的基础是被测系统的功能需求，其详细描述了系统行为，定义了系统必须完成的功能。此外，系统实际运行时还会体现出一些非功能特性，它不是描述系统需要实现什么功能，而是描述这些功能实现时所需要达到的相关指标、参数、属性等，即系统执行其功能时有多好、多快，或执行程度如何等，这些系统非功能特性的表现会对客户的满意度有重大影响。系统测试的主要内容如表 5.3 所示。

表 5.3 系统测试的主要内容

系统测试	测试内容	说　明
功能性测试内容	逻辑功能测试	根据软件规格说明（达到功能）构造合理的输入（测试用例），并输入至软件，检查是否得到期望结果的输出，即使用有限的输入值来测试和验证软件的逻辑（业务）功能
	界面测试	针对用户界面窗口、下拉式菜单和鼠标操作、各类数据项能否正确地显示并输入系统中等方面进行测试
	易用性测试	指从软件使用的合理与方便程度测评软件，以发现该软件不便于用户使用的某些缺陷
	安装/卸载测试	测试系统能否实现正常安装（典型安装/完全安装/自定义安装/中断安装等）与卸载（从程序组里卸载/从控制面板中卸载/中断卸载过程等）
非功能性测试内容	性能测试	检验软件是否达到性能设计的要求，检验系统的性能运行表现。性能测试可进一步分为一般性能测试、负载测试、压力（强度）测试与稳定性测试等。性能测试通常采用自动化测试方法
	安全性测试	验证系统内的保护机制能否在实际运行中保护系统且不受非法入侵和各种非法干扰。在安全测试中，测试扮演的是试图攻击系统的角色，尝试通过外部手段，利用存在的各种漏洞来获取系统密码或进入系统，使用瓦解和攻破任何防守的安全软件来攻击系统，使其“瘫痪”，使用户无法访问或有目的引发系统出现错误，或在系统恢复过程中侵入系统
	其他测试	包括对系统的恢复性测试（通过各种手段强制性地让软件系统出错，使其不能正常工作，进而检验系统的恢复能力）；兼容性测试（检测软件系统是否能运行在不同的硬件环境和条件下，以及检测各软件之间能否正确地交互及共享信息）；数据转换测试（检测数据转换能否在运行于同一计算机上的两程序间进行，数据能否在通过互联网远距离链接的两程序间正确运行）；文档测试（检测文档和系统行为是否一致）；可维护性测试（检测系统是否具备模块化结构，评估文档的可维护性及是否为最新版本等）

注：关于功能性测试内容中的易用性测试，由于该项的测试主观性较强，实际中不同用户可能会对易用性的理解有所不同。

5.2.4　确认测试

确认测试也称为有效性测试，即验证软件的有效性。测试人员对照软件需求规格说明书，验证软件的功能、性能及其他特性是否完全符合用户需求，软件的相关配置是否完全、正确，所开发的软件能否按照用户提出的要求正常运行等。经过检验的软件功能、性能及其他要求均已满足需求规格说明书的规定，即可被认为是合格的软件。当然，在确认测试中，若发现软件实际运行状况与用户需求有相当程度的偏差，需要得到各项缺陷清单，在交付期之前修改与纠正发现的缺陷与问题。通常确认需经过开发者与用户协商，以共同确定确认测试的准则。

也有一些软件企业的测试部门把确认测试称为合格性测试，若达到上述要求，则认为开发的软件是合格的。

在此简要说明一下回归测试的概念。也就是说，软件或程序被发现有缺陷或发生变更时，程序都将被修改。程序被修改后重新测试的过程，称为回归测试。回归测试也是一种确认测试，具体地说，是程序在被修改后的（重新）确认测试的过程，目的是检查本次修改后有没有引入新的缺陷。回归测试可运用于任何测试阶段：单元测试、集成测试、系统性测试和验收测试阶段。

相对而言，回归测试的工作量较大。当通过执行测试用例来实施回归测试时，所设计的测试用例（集）需要包括以下 3 种不同类型。

① 能测试软件所有功能的代表性测试用例。

② 针对可能会被本次修改所影响的其他模块功能的附加测试用例。

③ 专门针对修改过的软件模块（部分）的测试用例。

5.2.5　验收测试

验收测试是软件交付用户使用之前的最后一项测试活动，也称为交付测试。验收测试是按照软件开发方与用户之前签订的开发合同、任务书或相关验收标准，对整个软件进行最终测试与评审。在确保软件一切准备就绪的前提下，看能否完全可以达到让用户执行软件的既定功能与任务的目标。

α 测试与 β 测试是目前很多软件企业广泛使用的两种验收测试方式。

（1）α 测试。

α 测试是用户在软件开发环境下（现场）进行的测试，也可以是开发机构内部的用户在模拟实际操作环境下进行的测试。即开发者坐在用户旁边，在开发者受控的环境下进行测试。由开发者随时记录错误情况和使用中的问题。当然，也可以让软件企业组织的内部人员（包括测试人员）模拟各类用户，从用户的思维方式与使用角度出发，对即将投入运行的软件产品进行测试，试图发现错误并修复。α 测试安排在软件企业内部（软件开发现场）执行，关键在于尽可能逼真地模拟用户在实际运行环境下对软件的操作，并尽可能多地涵盖未来所有可能的用户操作方式。

（2）β 测试。

β 测试是软件企业让用户在日常工作中实际使用即将投入市场的软件产品，用户试

用软件一段时间后，要报告所发现的异常情况，提出产品改进意见。

β测试是用户在实际使用环境下的测试活动，不受企业开发人员的控制与干预。很多知名软件企业向市场投入正式产品之前，会先推出一个有使用时间限制的产品免费试用版本(β版本)，让用户免费试用一段时间。然后，软件企业针对用户反馈的问题进行修复和完善。

一些大型通用软件正式发布前，通常需要执行α测试和β测试，目的是从实际终端用户的使用角度测试软件的功能和性能，发现可能只有最终用户才能发现的错误。

此外，杨怀洲[①]还提出了验收测试的有关注意事项。

① 验收测试前需要编写正式的验收测试计划，明确通过验收测试的标准，测试通过标准应当由用户确认。

② 验收测试必须在最终用户的实际使用环境中进行，或者尽可能模拟用户的实际运行环境，避免环境差异导致无法发现软件的一些潜在问题。

③ 验收测试应着眼于软件的业务级功能，而不是软件的所有细节功能。必须面向用户，从最终用户使用中的业务场景出发，以用户可以直观感知的方式进行，使用黑盒测试方法，避免涉及过多的开发内部细节。

④ 面向特定工作单位的项目类软件的验收测试，一般由用户和开发方的测试部门共同完成；面向公众、由开发方自行研发的产品类软件的验收测试，应当由开发方的测试部门、产品设计部门、市场部门和产品售后服务部门共同参与完成。

当然，验收测试也决定了用户是否接收或拒收该软件。验收测试安排通常由开发方与用户协商，并在验收测试计划中规定和说明。

5.3 软件测试模型

软件测试模型也称为软件测试过程模型，即在测试实践的基础上有机结合相应的软件开发活动，总结和抽象出的一系列测试活动规律。与软件开发过程一样，软件测试过程同样遵循软件工程原理与管理学思想等。软件测试行业蓬勃发展的同时，对软件质量的关注也接踵而来。目前，软件的质量控制越来越难，软件测试模型作为保证软件测试工作效率和质量的结构框架，需通过强化和改进来适应不断复杂化的软件开发过程。

5.3.1 传统的软件测试模型

传统的软件测试模型主要有V模型、W模型、X模型、H模型与前置模型。

1. V模型

V模型是一个广为人知的软件测试模型，于20世纪90年代提出，如图5.9所示。在V模型中，从左到右描述了基本的开发过程和测试行为，为软件的开发人员和测试管理者提供了一个极为简单的框架。V模型的价值在于它非常明确地标明了测试过程中存

① 杨怀洲.软件测试技术[M].北京：清华大学出版社，2019.