2

系统生命周期中的测试策略

现在软件测试已经贯穿于软件开发生命周期的整个过程中，是软件质量控制体系中的重要环节，在整个系统生命周期中有哪些测试策略呢？本章我们将详细介绍系统生命周期中常用的测试策略。

本章主要包括以下内容：

- 测试级别
- 测试在质量体系中的位置
- 软件测试模型
- 系统生命周期中的测试策略

2.1 测试级别

在软件开发生命周期过程中，有很多种软件测试模型（关于软件测试模型在 2.3 节中会详细介绍），不同的测试模型其对应的测试级别也有不同，以最典型的 V 模型为例，软件测试级别可以分为组件测试、集成测试、系统测试和验收测试。

2.1.1 组件测试

组件测试在实际的测试过程中所对应的阶段是我们的单元测试阶段，组件测试是最低层的一个级别的测试。在整个开发过程中，单元模块是开发工程师最先开发出来的，单元模型是最小的组件，通常单元模块是我们说的函数或类，对于第二代语言，单元模块就是函数，对于面向对象的语言，单元模块就是类。把对于这类单元模块的测试称为组件测试。

组件测试的具体目的如下：

（1）验证代码是与设计相符合的。

（2）发现设计和需求中存在的错误。

（3）发现在编码过程中引入的错误。

关于组件测试需要关注两个维度的内容：功能方面和非功能方面。功能方面主要是内部的逻辑结构，验证程序内部逻辑结构是否满足设计要求，当然除此之外还包括内部数据结构、独立路径集等方面的内容。非功能测试主要是其他逻辑设计之外的内容，主要包括代码性能、内存泄漏、代码健壮性、代码可靠性等。

单元测试需要一个环境，也就是如果要测试一个函数或类，它不可能独立运行，它需要存在一定的条件下才能运行，在这个环境中会涉及两个重要的组成部分：桩模块和驱动模块。

桩模块（Stub）是指模拟被测试的模块所调用的模块，而不是软件产品的组成部分。主模块作为驱动模块，与之直接相连的模块用桩模块代替。

驱动单元（Driver）：所测函数的主程序，它接收测试数据，并把数据传送给所测试单元，最后再输出实测结果。

当然如果每测试一个单元模块都需要写这么多桩模块和驱动模块，那么工作效率就会很低，所以现在有很多的一些专业的工具来实现，如 Gtest、Junit 等。

关于更详细的单元测试的内容，在第 10 章都会有详细的介绍。

2.1.2 集成测试

集成测试，也叫组装测试或联合测试。集成测试是 V 模型中第二个级别的测试，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（如根据结构图）组装成为子系统或系统，进行集成测试。因为一些模块虽然可以单独地工作，但并不能保证这些单独的模块连接在一起时也能正常工作，一些局部反映不出来的问题，在全局上很可能暴露出来。

集成测试（也叫组装测试、联合测试）是单元测试的逻辑扩展。它最简单的形式是，把两个已经测试过的单元组合成一个组件，测试这些组件之间接口数据传递是否有问题。从这个角度来说，集成测试是将多个单元模块进行聚合。在实际测试过程中有很多单元，在对单元组合测试之前必须先分清楚各单元模块之间的关系，然后有序地对绝大部分的单元模块进行组合，直到最后成一个系统。

集成测试的前提条件是在进行集成测试之前一定要保证每个单元模块测试完成，并且每个单元模型要满足设计要求，不能存在问题，如果在组合测试过程中存在问题，这说明单元模块与单元模块之间的接口存在问题。

在测试过程中，集成测试的层次包括两种：一是系统内部集成；二是系统间的集成。系统内部集成是指在整个系统内部功能模块与功能模块集成，一个系统可能由很多不同的功能模块组成，对于这种在一个系统内部将功能与功能进行集成的测试称为系统内的集成。除了系统内的集成外，还有另一类集成是系统间的集成，系统间的集成是指两个独立的系统之间，通过某种方式进行传递数据。举一个简单的例子，现在网购是很平常的事情，在某个电子商务平台上下单买一件产品，如在易迅下单买一部手机，当我们下单后，这个订单号就会从易迅官方平台发送到库存的管理系统中，

之后对产品进行包装和发货，发货必须使用到物流，此时这个订单号又会发送到相关的物流中心。像这种情况就是典型的系统间的集成，这个订单号在系统间传递，那么测试时就要测试这个订单号能不能正常地传递到其他的子系统中。

集成测试的策略有很多种，主要包括自底向上集成测试、自顶向下集成测试、三明治集成测试、核心集成测试、分层集成测试、基于使用的集成测试等。

关于更详细的集成测试的内容，在第 11 章都会有详细的介绍。

2.1.3 系统测试

系统测试（System Testing）是将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的测试活动。系统可能包含硬件，但不一定包含硬件，可能就是纯软件。

在系统测试概念中详细地描述了三个维度的内容：系统测试对象、系统测试是一个过程、系统应该有一个流程。

（1）系统测试对象

系统测试的对象是软硬件集合在一起的系统，不应是独立的软件与硬件环境。当然具体操作、执行时可根据实际情况来组织。也就是说，我们通常说的系统测试不一定只有软件，还可能包含硬件、电源和结构等，手机产品就是典型的这类系统，不仅有软件，还有硬件、电源、结构等。验证时应尽可能模拟实际的运行环境与条件。在测试过程中系统测试应该尽量模拟实际的运行环境，这样可以尽最大可能保证系统上线后不出问题。

（2）系统测试是一个过程

为了验证系统是否满足客户需求，需要一系列的测试活动来保证，即系统测试并不是一个简单的步骤，所以为了让系统测试更全面，就需要对系统测试的活动进行管理。

（3）系统测试应该有一个流程

为了更好地管理这些活动，我们制定了一个标准的测试流程，包括五个步骤：计划与控制、分析与设计、实现与执行、评估与报告、结束活动。关于系统测试的流程在第 4 章中会详细地介绍。

在系统测试过程中有一个很重要的环节就是测试设计，这也是我们常说的系统测试方法，系统测试方法即测试用例设计方法，常见的系统测试用例设计方法包括：等价类、边界值、因果图、判定表、正交试验、场景分析法、状态迁移等，关于用例设计方法在第 8 章中会有详细的介绍。

系统测试的类型也很多，常见的系统测试类型包括：功能测试、性能测试、兼容性测试、易用性测试、安全性测试等。系统测试可以分为多种类型取决于软件质量模型，关于软件质量模型在第 12 章中会详细地介绍。

系统测试的目的主要包括以下两个方面：

（1）通过与系统的需求定义做比较，发现软件与系统定义不符合或与之矛盾的地方。

（2）系统测试的测试用例应根据需求分析说明书来设计，并在实际使用环境下运行。

关于系统测试更详细的内容见第 13 章。

2.1.4 验收测试

很多的公司在系统测试完成后就将产品发布了，其实系统测试之后还有一个测试阶段就是验收测试，当然并不是所有的公司都会进行验收测试，一般外包项目会有验收测试，即客户会对产品进行验收，以评估产品质量是否满足要求。

验收测试是软件发布之前最后一个测试阶段，是在单元测试、集成测试和系统测试完成之后的一个测试阶段，也称之为交付测试。验收测试是向最终用户表明系统能够像预定要求那样正确地工作，验收测试的策略通常包括四种：正式验收、非正式验收、Alpha 测试和 Beta 测试。

正式验收测试是一项管理严格的过程，它通常是系统测试的延续。计划和设计这些测试的周密和详细程度不亚于系统测试。选择的测试用例应该是系统测试中所执行测试用例的子集。不要偏离所选择的测试用例方向，这一点很重要。在很多组织中，正式验收测试是完全自动执行的。

非正式验收测试执行测试过程的限定不像正式验收测试中那样严格。在此测试中，确定并记录要研究的功能和业务任务，但没有可以遵循的特定测试用例。测试内容由各测试员决定。这种验收测试方法不像正式验收测试那样组织有序，而且更为主观。

Alpha 测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。Alpha 测试的目的是评价软件产品的功能、局域化、可使用性、可靠性、性能和支持等特性是否满足用户要求。

Beta 测试是一种验收测试。它与 Alpha 测试有很多相似之处，都是关注产品功能、性能、可靠性等特性，但与 Alpha 测试也有一些不同之处，如 Beta 测试是由最终用户或潜在用户来执行。

关于验收测试更详细的内容见第 20 章。

2.2 测试在质量体系中的位置

测试不仅仅是找出软件中的缺陷，它在软件质量体系中占有重要的位置，下面我们来讨论在能力成熟度模型和基于过程的质量模型中，软件测试所处的位置。

2.2.1 能力成熟度模型集成

CMMI（Capability Maturity Model Integration，能力成熟度模型集成）认证评估在过去的十几年中，对全球的软件产业产生了非常深远的影响。CMMI 共有五个等级，分别标志着软件企业能力成熟度的五个层次，如图 2-1 所示。从低到高，软件开发生产计划精度逐级上升，单位工程生产周期逐级缩短，单位工程成本逐级降低。据 SEI 统计，通过评估的软件公司对项目的估计与控制能力提升约 40%～50%，生产率提高 10%～20%，软件产品出错率下降超过 1/3。

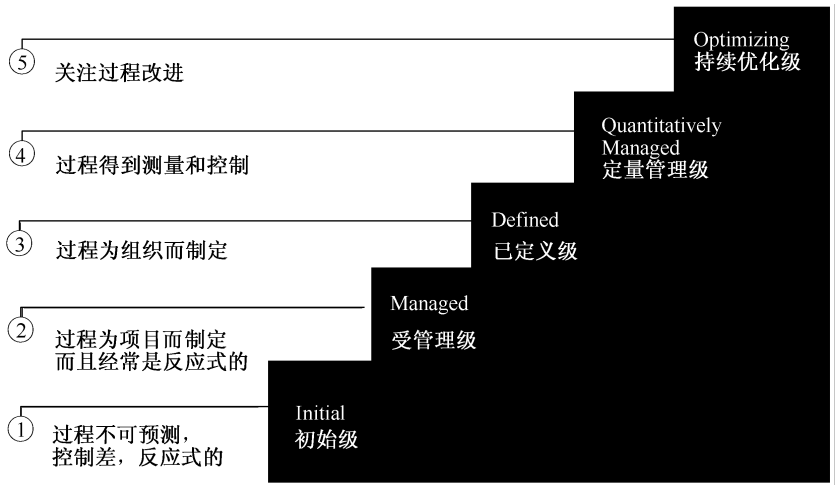

图 2-1　CMMI 模型

第一级：初始级

初始级的软件过程是未加定义的随意过程，项目的执行也是随意的，甚至是混乱的。当然有些企业可能已经制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，并且在执行过程中没有政策、资源等方面的支持，它仍然被视为初始级。

第二级：受管理级

根据多年的经验和教训，企业总结出软件开发的首要问题不是技术问题，而是管理问题。因此，CMMI 发展到了第二级，更强调软件管理过程，建立一个可管理的过程是很重要的，它可以将开发的过程重复，只有可重复的过程才能逐渐改进并使其成熟。受管理级的管理过程主要包括五个方面：需求管理、项目管理、质量管理、配置管理和子合同管理；其中项目管理过程又分为计划过程和跟踪与监控过程。通过实施这些过程，从管理角度可以看到一个按计划执行且阶段可控的软件开发过程。

第三级：已定义级

在受管理级定义了管理的基本过程，但并没有定义执行的步骤标准。在第三级则要求制定企业范围的工程化标准，并将这些标准集成到企业软件开发标准过程中。规定所有开发的项目或产品都必须遵守该标准过程，并且按照过程执行，当然在实际过程中，可以根据具体的项目对该过程进行适当的裁剪，但过程的裁剪不是随意的，在使用前必须经过企业有关人员的批准。

第四级：定量管理级

第四级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品（包括工作产品和提交给用户的最终产品）的质量需要有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品，量化控制将使软件开发真正成为一种工业生产活动。

第五级：持续优化级

持续优化级的目标是达到一个持续改善的境界。所谓持续改进，是指根据过程执行的反馈信息

来改善当前已定义的开发过程，即优化已定义的执行步骤。如果企业达到了第五级，就表明该企业能够根据实际的项目性质、技术等因素，不断调整软件开发过程使开发过程达到最优。

CMMI 模型中包括验证（VER）和确认（VAL）两大过程域，这两大过程域与软件测试有着紧密的联系，也是规范软件测试的两大过程域。

（1）验证（VER）过程域的目的是确保所选定的工作产品符合其指定的需求。验证过程域包括验证准备、验证执行和纠正措施识别。

验证的对象包括产品和中间工作产品，验证方法是将待验证的对象与选定的客户需求、产品需求和产品组件需求加以比较。验证是渐进的过程，因为它发生在产品和工作产品的整个开发过程中，从需求开始验证，历经工作产品的验证，最终为已完成产品的验证。

（2）确认（VAL）过程域的目的是展示完全置于预期环境中的产品或产品组件是否满足预期的使用需求。

所有的产品都可在其预期环境中实施确认活动，例如：操作、培训、制造、维护及支持服务。所有用于工作产品的确认方法，也能使用在对产品和产品组件的确定过程中（在所有过程域中，产品和产品组件的含义包括服务和其组件）。工作产品（例如需求、设计、原型）存在于整个产品生命周期，应及早并逐步实施确认。

确认环境必须可代表产品和产品组件的预期环境，同时该确认环境也适用于工作产品确认活动的预期环境。

确认证明是指所提供的产品是否符合预期的使用需求，验证与确认很容易被混淆，验证是确定每个工作产品是否正确反映了特定需求，即验证确保“你正确地做了”；而确认是指“你做了正确的事”。确认活动使用与验证类似的方法，例如测试、分析、检查、示范或模拟。通常，确认活动包含最终用户和其他相关人员。确认与验证活动经常同时实施，且可能使用部分相同的环境。若有可能，实施确认应将产品或产品组件置于其预期环境中运行。确认可能使用全部或部分的预期环境，使用工作产品实施确认，可让问题在项目生命周期中通过相关人员的参与及早被发现。服务的确认活动可应用于工作产品，例如建议书、服务目录、工作描述和服务记录。

当在确认过程中问题被识别出来时，需要参考需求开发、技术解决方案或项目监控过程域的实践来解决。

2.2.2 基于过程中的质量

目前，软件项目需求正飞速增长，相应的软件开发活动也随之急剧增长，这样使得软件过程（即用于开发和维护软件及其相关产品的一组活动、方法、实践及转换）得到更多的关注。软件过程在成本估算、项目进度和软件质量等方面必须把握准确，同时产品必须满足用户对其功能和质量的要求，所以深入研究软件度量模型、建立基于度量的量化管理是控制软件过程、提高软件质量的有效保证。基于过程的质量控制如图 2-2 所示。

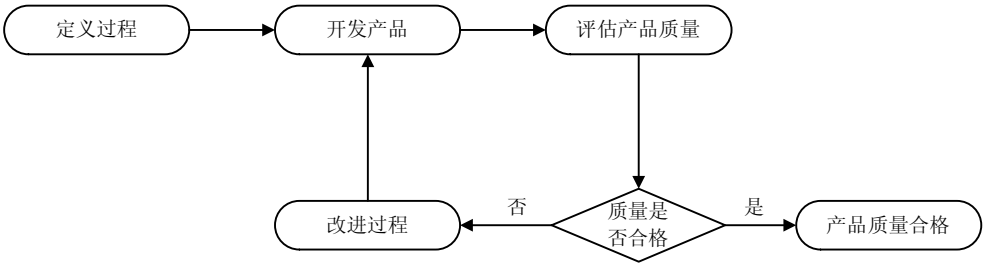

图 2-2　基于过程的质量控制

而软件测试是评估产品质量的重要手段，软件测试贯穿产品开发的始终，那么在整个软件测试过程中，应该如何来度量软件测试的质量呢？在整个测试过程中，质量度量主要包括以下几个方面：

- 测试覆盖率；
- 测试执行的质量和效率；
- 测试用例深度、质量和有效性；
- 缺陷分布分析。

（1）测试覆盖率。

测试覆盖率是指在测试过程中对被测试对象的需求、功能、代码测试的程度。主要包括对需求和代码两个方面的覆盖评估，但其实这两个方面的评估本质是一致的，都是通过测试用例来评估覆盖率。

1）基于需求的测试覆盖评估依赖于对已执行/运行的测试用例的核实和分析，其主要是通过评估测试用例覆盖率来评估，在测试过程中的目标是要求需求的覆盖率达到 100%。在实际测试过程中，可以通过统计已执行的覆盖率和执行成功的覆盖率来评估需求覆盖率的值。

已经执行的测试用例覆盖率指所有测试用例中被执行用例所占百分比，公式如下：

已执行的测试覆盖率＝Tx/Rft

其中，Tx 表示已执行的测试用例数，Rft 是测试需求的总数。

成功执行的覆盖率指测试过程中执行成功的测试用例所占百分比，公式如下：

成功的测试覆盖率＝Ts/Rft

其中，Ts 表示已执行并且执行状态为成功的测试用例，Rft 是测试需求的总数。

2）基于代码的测试覆盖率是对被测试的程序代码语句、路径或条件的覆盖率分析。代码覆盖可以建立在控制流（语句、分支或路径）或数据流的基础上，主要用于白盒测试阶段。控制流覆盖的目的是测试代码行、分支条件、代码中的路径或软件控制流的其他元素；数据流覆盖的目的是通过软件操作测试数据状态是否有效，例如，数据元素在使用之前是否已经定义。

基于代码的测试覆盖通过以下公式计算：

已执行的测试覆盖率＝Tc/Tnc

其中，Tc 是指使用代码语句、条件分支、代码路径、数据状态判定点方法设计的并被执行的用例数，Tnc（Total number of items in the code）是指项目中总的代码数。

（2）测试执行的质量和效率。

测试执行的效率是指测试工程师每天执行的测试用例数，一般每天执行 50 条测试用例。

测试执行的质量包括两个方面：一方面是指每个测试用例发现的缺陷数；另一方面是指软件发布后遗留的软件缺陷数占总缺陷数的百分比，一般要求低于 0.5%。

故测试执行的质量和效率一般使用以下指标来统计：

- 每人每天所执行的测试用例数；
- 每人每天发现的缺陷数；
- 缺陷遗留率。

（3）测试用例深度、质量和有效性。

测试用例是所有测试活动的基础，测试用例质量的好坏直接影响软件测试的质量。

测试用例的度量主要从测试用例深度（也叫测试用例密度）、质量和有效性三个方面来实现。当然如果开展了自动化测试，还可以从测试用例自动化的程度这一维度来度量。

测试用例深度（Test Case Depth，TCD）指每 KLOC（千行代码）设计的测试用例数或每个功能点所设计的测试用例数，一般情况下认为每 KLOC 设计的用例数越多，表示测试的质量越高。当然必须考虑冗余或重复的用例数，在设计用例时应该尽量避免出现冗余或重复。

测试用例质量（Test Case Quality，TCQ）其实是一个很复杂的指标，它包括两个方面：一方面指如何设计一个好的测试用例；另一方面指测试发现缺陷的数量。

一般情况下，一个好的测试用例应该考虑以下几个方面：

- 测试用例覆盖程度；
- 测试用例是否已达到工作量最小化；
- 测试用例的分类、描述是否清晰；
- 测试用例是否表明目的；
- 测试用例的易维护性；
- 有充分的负面测试；
- 测试用例没有重复、没有冗余。

发现缺陷方面主要是指测试用例发现的缺陷数量，公式如下：

TCQ＝测试用例发现的缺陷数量/总的缺陷数量

总的缺陷数量除了测试用例发现的缺陷数外，还包括通过 ad-hoc 测试（随机、自由的测试）、集体走查（Work-through）和 Fire-drill 测试（类似消防训练的用户压力/验收测试）等其他手段发现的缺陷。

企业开展自动化测试，可以计算可自动化测试用例的数量，这也是衡量测试用例质量的一个方面，将手动测试用例转换为自动化测试用例可以节约写自动化测试用例的时间，可转换的越多，节约的成本就越多。

（4）缺陷分布分析。

缺陷是测试过程中体现工作效率和价值的重要指标之一，也是分析系统质量的重要指标。在

测试过程中除了要提交缺陷外，还需要对缺陷的分布情况进行分析，这样可以作为改进系统质量的依据。

在提交缺陷时，需要注意一些必需的元素项，即一个好的缺陷通常需要包含的内容，现在一些企业通常会使用缺陷管理工具来管理测试过程中所发现的缺陷。

对提交的缺陷需要进行分析，这样可以进一步改进系统质量，并且可以改进测试方法和测试策略，常用的缺陷分析方法有：ODC 正交缺陷分析法、Gompertz 缺陷分析法、Rayleigh 缺陷分析法、四象限缺陷分析法和根源缺陷分析法，具体的缺陷分析法在第 7 章详细介绍。

2.3 软件测试模型

在软件质量体系中，为了更好地指导软件开发的全部过程、活动和任务，人们提出了软件开发模型。典型的开发模型有：边做边改模型（Build-and-Fix Model）、瀑布模型（Waterfall Model）、快速原型模型（Rapid Prototype Model）、增量模型（Incremental Model）、螺旋模型（Spiral Model）、演化模型（Incremental Model）、喷泉模型（Fountain Model）、智能模型（四代技术（4GL））、混合模型（Hybrid Model）。但是所有的开发模型都没有把软件测试列进去，这样就无法对软件测试过程进行很好的指导，而随着软件测试的发展，软件测试成为软件质量保证的重要手段之一，软件测试也慢慢地受到公司的重视，于是人们就希望软件测试也像软件开发一样，由一个模型来指导整个软件测试过程。当前最常见的软件测试模型有瀑布模型、V 模型、W 模型、H 模型和 X 模型，下面详细介绍。

2.3.1 瀑布模型

1970 年温斯顿·罗伊斯（Winston Royce）提出了著名的“瀑布模型”，直到 20 世纪 80 年代早期，它一直是唯一被广泛采用的软件开发模型，瀑布模型是由瀑布开发模型演变而来的。

瀑布模型将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护六项基本活动，其过程是将上一项活动接收的工作对象作为输入，当该项活动完成后会输出该项活动的工作成果，并将该项成果作为下一项活动的输入。该模型规定这六项基本活动自上而下、固定相互衔接的次序，如同瀑布流水，逐级下落。从本质上讲，它是一个软件开发架构，开发过程是通过一系列阶段顺序展开的，从需求分析直到产品发布和维护。如果在其中某个阶段有信息未被覆盖或有问题，那么就得返回到上一个阶段，并对这些阶段进行适当的修改才能进入下一个阶段，这样每个阶段都会产生循环反馈，开发过程从一个阶段“流动”到下一个阶段，这也是瀑布模型名称的由来，瀑布模型如图 2-3 所示。

瀑布模型的核心思想是按工序将问题简化，将功能的实现与设计分开，便于分工协作，即采用结构化的分析与设计方法将逻辑实现与物理实现分开。

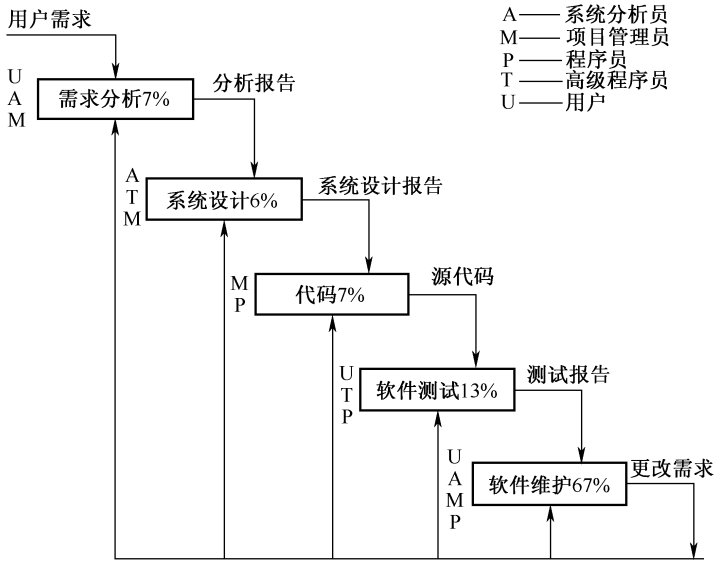

图 2-3　瀑布模型

瀑布模型的优点如下：

- 为项目提供了按阶段划分的检查点；
- 当前一阶段完成后，只需要关注后续阶段；
- 可在迭代模型中应用瀑布模型，如图 2-4 所示。

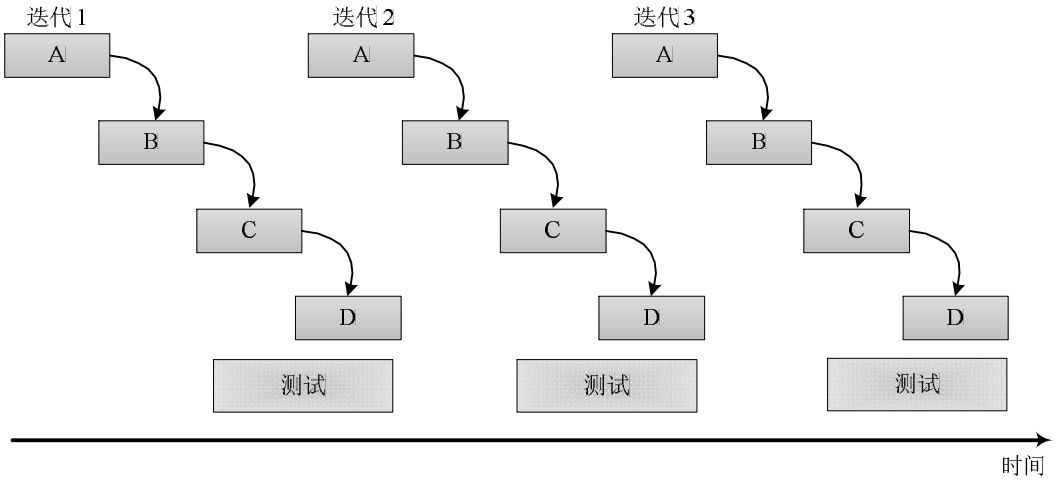

图 2-4　迭代中的瀑布模型

增量迭代应用于瀑布模型，迭代 1 解决最大的问题，每次迭代产生一个可运行的版本，同时增加更多的功能，但每次迭代必须经过严格的质量和集成测试。

瀑布模型有以下缺点：

- 项目中各个阶段之间极少有反馈；
- 只有在项目生命周期的后期才能看到结果；
- 通过过多的强制完成日期和里程碑来跟踪各个项目阶段。

2.3.2 V 模型

V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，V 模型在英国国家计算中心文献中发布，目的是改进软件开发的效率和效果。它是软件测试最具代表性的测试模型之一。

在传统的开发模型中，如瀑布模型，通常把软件测试过程作为在需求分析、概要设计、详细设计和编码全部完成之后的一个阶段，尽管有时软件测试工作会占整个项目周期一半的时间，但是仍然被认为软件测试只是一个收尾工作，而不是主要的工程。故对以前的测试模型进行了一定程度的改进，V 模型其实是软件开发瀑布模型的变种，反映了软件测试活动与软件开发过程（从分析到设计）的关系，如图 2-5 所示。

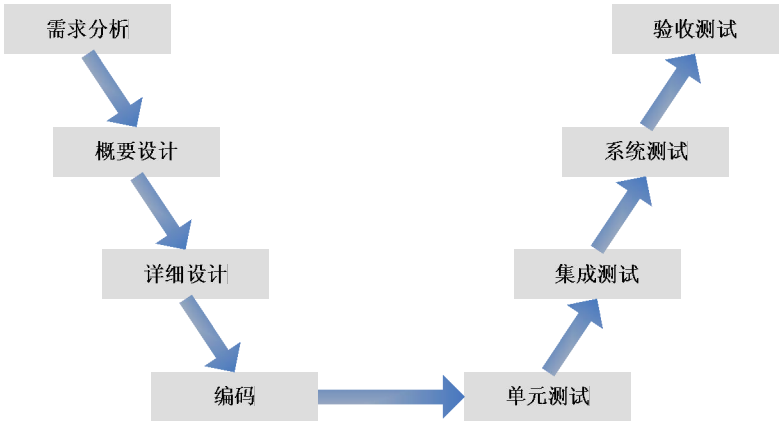

图 2-5 V 模型

V 模型从左到右，描述了基本的开发过程和测试行为，明确地标明了测试工程中存在的不同级别以及测试阶段和开发过程各阶段的对应关系。图中箭头代表了时间方向，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程各阶段。

V 模型指出，单元和集成测试是验证程序设计，单元测试主要由白盒测试工程对代码进行测试，但目前国内真正做白盒测试的企业不多。这主要有两大原因：第一，白盒测试投入的成本很高，并且产出不明显，很多企业不希望投入更多的资源去做这项工作；第二，白盒测试对测试工程师的要求较高，在目前系统测试还没有完全成熟的情况下很难真正地开展白盒测试。而集成测试是介于白盒测试与系统测试之间的一种测试，也叫灰盒测试，由于它与白盒测试和系统测试之间没有明显的界限，所以在实际的测试过程中，即使开展集成测试也是由系统测试工程师来完成。

系统测试主要验证系统设计，检测系统功能、性能的质量特性是否达到系统设计的指标，由测试人员和用户进行软件的确认测试和验收测试，以及对需求说明书进行测试，以确定软件的实现是否满足用户需求或合同要求。

V 模型存在一定的局限性，它把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段。如果不做白盒测试，那么其实都是在系统完成集成后才开始系统测试的，这样需求分析阶段隐藏的问题一直到后期的验收测试才被发现，因此修改缺陷的成本就高了很多。

V 模型详细的描述了每个测试阶段所对应验证的对象，单元测试验收的对象是详细测试说明书、集成测试验证的对象是概要设计说明书，系统测试验证的对象是需求说明书。在测试过程中，我们经常说测试的目的就是验证产品是否满足客户的需求，那么如何确保我们的产品是满足客户需求的呢？换一个角度来理解，其实结果是靠过程来保证的，也就是说，如果我们可以确保开发每个阶段的工作是正确的，那么就说明开发出来的产品肯定是满足客户需求的，因为开发每个阶段的工作都是以需求说明书为依据的，所以 V 模型有一个优点是其详细地介绍了测试每个阶段所测试验证的依据。

由于 V 模型是软件开发中瀑布模型的变种，所以它存在和瀑布模型相似的一些问题。由于测试阶段处于软件实现后，这意味着在代码完成后必须有足够的时间预留给测试活动；否则将导致测试不充分，开发前期未发现的错误会传递并扩散到后面的阶段，而在后面发现这些错误时，可能已经很难再修正，从而导致项目的失败。

V 模型最大的缺陷就是只把程序作为被测试对象，而需求、说明书等其他规格说明书都未被列为测试对象。

总之 V 模型具有以下特征：

（1）测试阶段划分得很清楚。

（2）每个开发阶段都有相应的测试对其进行验证。

（3）测试与开发是串行进行的而不是并行，也就是测试需要等开发完成后再开始。

（4）测试对象只有程序，而不包括需求等其他的说明书。

（5）V 模型是瀑布模型的变种，瀑布模型存在的问题 V 模型也存在。

2.3.3 W 模型

由于 V 模型存在一些明显的缺陷，人们就在实际测试过程中对 V 模型进行了改进，将 V 模型演变为 W 模型。W 模型由 Evolutif 公司提出，由两个 V 字型模型组成，相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动，如图 2-6 所示。

W 模型也称之为双 V 模型，一个 V 是开发的生命同期，另一个 V 是测试的生命周期，W 模型与 V 模型有一个很大的不同，就是 W 模型是一个并行的模型，V 模型是一个串行的模型，W 模型开始是从需求分析开始就开始了，而不是等到编码完成后才开始。并且测试阶段的划分更清楚，而不仅仅是单元测试、集成测试、系统测试，还包括前期的测试计划、测试方案等内容，这更符合现在企业测试的流程。

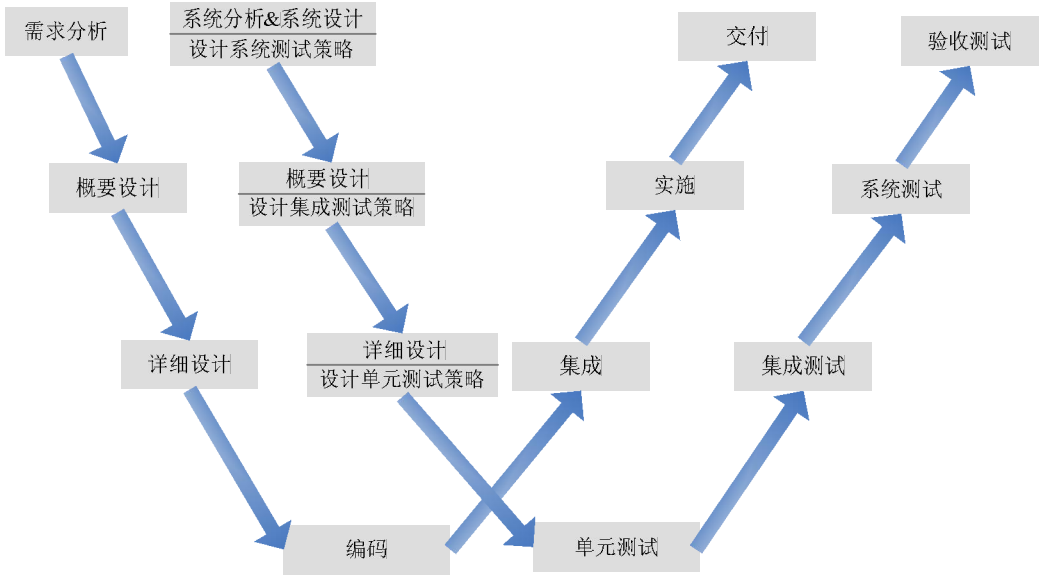

图 2-6　W 模型

W 模型强调测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。

W 模型有利于尽早全面地发现问题。从需求分析开始测试工程师就参与到项目的测试中，当需求分析完成后，测试工程师就需要参与到需求的验证和确认活动中，并需要提供可测试性需求分析说明书，这样可以尽早地发现需求阶段的缺陷。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。但 W 模型也存在局限性，需求、设计、编码等活动被视为是串行的，同时，测试和开发活动也保持着一种线性的前后关系，上一阶段完全结束，才可正式开始下一阶段工作，这样就无法支持迭代的开发模型。对于当前软件开发复杂多变的情况，W 模型并不能解除测试管理面临的困惑。

总之 W 模型具有以下特征：

（1）测试阶段划分得更全面，不仅仅是单元测试、集成测试和系统测试；

（2）测试与开发是并行的，从需求测试就应该开始介入；

（3）提出尽早测试的概念，这样可以降低缺陷修复成本；

（4）测试对象不仅仅是程序，还包括需求或其他的相关文档。

2.3.4　H 模型

H 模型将测试活动分离出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来，如图 2-7 所示。

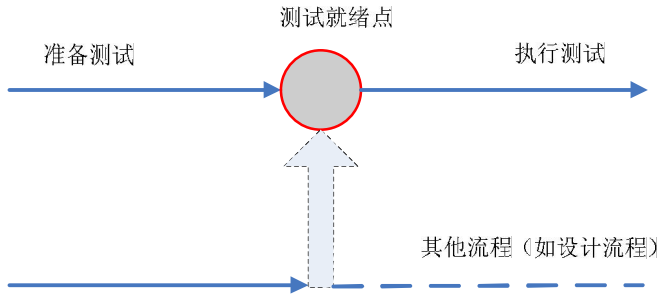

图 2-7　H 模型

H 模型提倡者认为测试是一个独立的过程中，所以在 H 模型中并没有看到关于开发的过程，而是测试的一个流程，当然这个测试的流程并不像 V 模型和 W 模型那样有明确的测试区分。

H 模型演示了在整个生命周期中某个层次上一次软件测试的“微循环”。当测试条件准备完成，进入测试就绪状态后，所在测试 H 模型中有一个测试就绪点，也就是测试有一个准入条件。通常情况下判断测试是否达到准入条件，应该检查以下几部分内容是否已经完成：

- 该开发流程对应的测试策略是否完成；
- 测试方案是否完成；
- 测试用例是否完成；
- 测试环境是否搭建好；
- 相关输入件、输出件是否明确。

也就是说，通常我们要检查上面一些内容是否完成，再确定我们是否需要进入下一个阶段的测试。当测试条件成熟，并且测试准备工作已经完成，进入了测试就绪点，测试执行活动才可以进行。

H 模型中还有一个“其他流程”的测试，这个观点强调了测试其他不一定要是常见的应用程序也可以其他的内容，这可以理解为整个产品包中所有的对象，包括开发阶段的一些设计流程，这样将测试的范围直接扩展到整个产品包，而非 W 模型中提到的代码、需求或其他相关说明书。

与 V 模型和 W 模型不同的是，H 模型的核心是将软件测试过程独立出来，并贯穿产品的整个生命周期，与开发流程并行进行，不需要等到程序全部开发完成才开始执行测试，这充分体现了软件测试要尽早准备、尽早执行的原则。不同的测试活动可以按照某个次序先后进行，当一次测试工作后产品质量无法达到要求时，可以反复进行多次测试。

总之 H 模型具有以下特征：

（1）测试是一个独立的过程；

（2）测试达到准入条件，才可以执行；

（3）测试对象是整个产品包，而不仅仅是程度、需求或相关说明书。

2.3.5　X 模型

X 模型的基本思想是由 Marick 提出的，但是 Marick 不建议建立一个替代模型。Robin F Goldsmith

引用了 Marick 的一些想法，并经过重新组织，形成了“X 模型”。当然这并不是为了和 V 模型相对应而选择这样的名字，是由于 X 通常代表未知，而 Marick 也认为他的观点并不足以支撑一个模型的完整描述，但具备一个模型所需要的主要内容，其中包括了探索性测试（Exploratory Testing）见解，如图 2-8 所示。

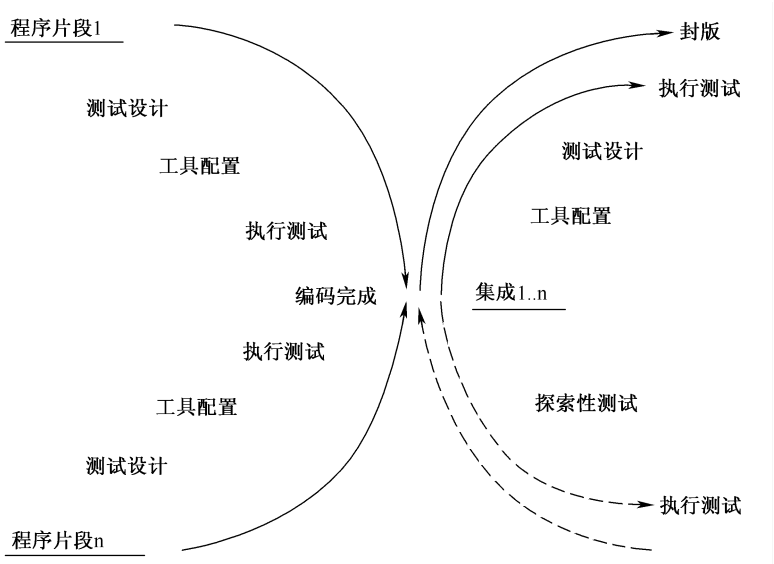

图 2-8　X 模型

Marick 对 V 模型提出质疑，他认为 V 模型必须按照一定顺序严格执行开发步骤，而这样很可能无法反映实际的实践过程。而众所周知很多项目在立项时需求并不完整，但 V 模型还是从需求处理开始，要求对各开发阶段中已经得到的内容进行测试，但它没有规定需要取得多少内容，如果没有任何的需求资料，开发人员知道他们要做什么吗？或者需求不完善，开发工程师做出来的功能就不完善，必须不断地修改。他主张在 X 模型中需要足够的需求，并且需求至少进行一次发布。

Marick 也质疑单元测试和集成测试的区别，目前在国内真正做单元测试的企业不多，很多企业都是跳过单元测试直接进行集成测试，甚至集成测试也被跳过，直接进行系统测试。而 X 模型则没有强制要求在进行集成测试之前，必须对每一个程序片段进行单元测试，但是 X 模型并没有提供是否跳过单元测试的判断准则。

Marick 认为一个模型不应该规定那些和当前所公认的实践不一致的行为。X 模型左边描述的是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁的交接，通过集成最终合成为可执行的程序，这些可执行程序还需要进行测试。已通过集成测试的成品可以进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。

X 模型左边是单元测试和单元模型之间的集成测试，右边是功能的集成测试，通过不断的集成最后成为一个系统，如果整个系统测试没有问题就可以封版发布。这个模型有一个很大的优点是它

呈现了一种动态测试的过程中，也就是测试是一个不断迭代的过程中，这更符合企业实际情况，其他模型更像一个静态的测试过程。

X 模型提倡公司可以根据自身的实际情况确定是否要进行单元测试和集成测试，并不是所有的研发公司都会先做单元测试和集成测试，更多的是直接做系统测试。

在 X 模型中还显示了测试步骤，包括测试设计、工具配置、执行测试三个步骤，虽然这个测试步骤并不很完善，但是毕竟将一些主要的内容表现出来了。

X 模型提倡探索性测试，指不进行事先计划的特殊类型的测试，这样可以帮助有经验的测试工程师发现测试计划之外更多的软件错误，避免把大量时间花费在编写测试文档上，导致真正用于测试的时间减少。

综上，X 模型具有以下特征：

（1）公司可以根据自身的情况确定是否要做单元测试，还是直接做系统测试；

（2）测试应该是一个不断迭代的过程，直到封版发布；

（3）提倡探索性测试。

2.4 系统生命周期中的测试策略

软件测试策略是指在软件测试标准、测试规范的指导下，依据测试项目的特定环境约束制定的软件测试原则、策略和方法的集合。系统生命周期测试策略如图 2-9 所示。

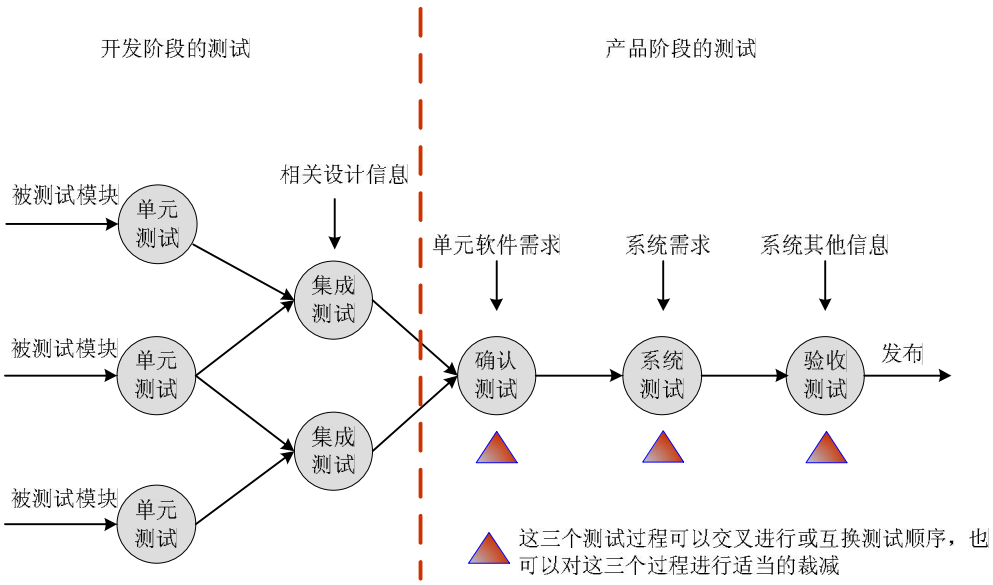

图 2-9 系统生命周期测试策略

软件测试的策略、方法和技术是多种多样的，对于软件测试技术，从是否执行被测试软件的角度划分：可分为静态测试和动态测试两种；从是否针对系统的内部结构和具体实现方法的角度划分：可分为白盒测试、灰盒测试和黑盒测试三种。其中灰盒测试是介于白盒测试与黑盒测试之间的一种测试方法，灰盒测试关注输出对于输入的正确性，同时也关注内部表现，但这种关注不如白盒测试详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，所以不易界定灰盒测试的范围，很多公司不开展灰盒测试。

2.4.1 开发阶段的测试策略

开发阶段是指在整个产品开发过程中我们使用的测试方法，在开发阶段的测试策略和方法主要是白盒测试，当然有一些公司也会开展灰盒测试，但并不多，通常灰盒测试也就是我们说的集成测试。

白盒测试也称结构测试或逻辑驱动测试，主要是测试源程序内部的结构，通过测试来检查程序内部动作是否满足设计规格说明书的要求，检查源程序中路径覆盖情况。白盒测试将被测程序看作一个打开的盒子，即程序内部的逻辑结构可以看的很清楚，如图 2-10 所示。

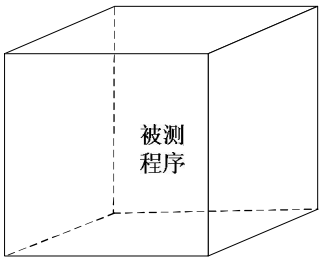

图 2-10 白盒测试

白盒测试要求对被测程序的结构特性做到一定程度的覆盖，逻辑覆盖是衡量白盒测试完整性的一个重要指标，关于逻辑覆盖在 10.3 节中会有详细的介绍。

通常的程序结构覆盖测试方法有：

- 语句覆盖；
- 判定覆盖；
- 条件覆盖；
- 判定/条件覆盖；
- 路径覆盖；
- 基本路径覆盖。

对源程序进行覆盖测试其实是一种动态的测试过程，测试过程中必须输入不同的数据进行测试来达到覆盖测试的目标。一般来说，覆盖率越高说明我们测试设计越全面，但在实际测试过程中我们又很难 100%覆盖，所以我们使用最多的覆盖方法是基本路径覆盖法。

但单元测试时，需要写一些辅助代码，我们把这些辅助代码叫做桩单元和驱动单元，但如果每个单元模块都写很多的辅助代码，这样测试的效率将会大大降低。为了提高测试效率，测试工程师提出了白盒测试框架，希望将这些辅助的代码作为一个固定的框架，这样就可以节约很多的时间，可以将主要的精力放在测试用例的设计上。对于白盒测试框架在第 10 章详细介绍。

2.4.2 产品阶段的测试策略

产品阶段主要采取黑盒测试策略进行测试。黑盒测试也称功能测试，它是通过测试来检测每个

功能是否都能正常使用，是否符合规格说明书。在测试过程中，把程序看作一个不能打开的黑盒子，如图 2-11 所示。在完全不考虑程序内部结构和内部特性的情况下，对程序接口进行测试，它只检查程序功能是否能按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

图 2-11　黑盒测试

黑盒测试是以用户的角度、从输入数据与输出数据的对应关系出发进行测试的，黑盒测试主要发现以下几类错误：

- 验证是否有不正确或遗漏的功能？
- 接口测试方面，验证输入能否正确地接受？输出的结果是否正确？
- 是否有数据结构错误或外部信息访问错误？
- 性能是否能够满足要求？
- 是否有初始化或终止性错误？

从理论上讲，在进行黑盒测试过程中，需要采用穷举法进行测试，需要对全部功能所有可能出现的情况进行覆盖测试，不仅需要测试合法的输入条件，还要测试不合法的输入条件，并且大多数的缺陷是通过输入不合法的测试条件测试出来的，因此测试条件有无穷多种。但在实际测试过程中是不可能这样做的，这样会导致测试成本太高，所以需要制定测试方法和策略来指导测试的实施，保证软件测试有计划地进行。

常用的黑盒测试设计方法有以下几种：

- 等价类划分法；
- 边界值测试法；
- 错误推测法；
- 因果图法；
- 场景法；
- 判定表法；
- 正交实验法。

目前大部分企业将产品阶段的黑盒测试规划到系统测试阶段，但对于有项目外包业务的企业来说，还需要经历验收测试阶段，主要验证产品是否达到需求说明书的要求，而完成的好坏决定着外

包主需要支付的外包成本。在实际的过程中验收测试一般有两种形式：一是企业组织一个团队进行验收测试；二是找第三方测评机构进行评测。

随着软件测试的发展，黑盒测试形成了两个重要的分支：性能测试和自动化测试。在实际工作中，一个好的产品或系统不仅仅功能要正确，其性能也是质量表现的重要一环，所以一些企业根据实际需要开展了性能测试。而自动化测试的目的更多的是为降低手工测试的成本，因为纯粹功能的黑盒测试都是手工测试，是由手工不停地重复测试，这样测试工程师会出现情绪不高的现象，并且激情会逐渐消退，因此一些企业就引进了自动化测试工具，将一些可以使用自动化测试工具进行测试的功能实现自动化测试，降低了测试成本，提高了测试的全面性，这些方法在产品测试阶段经常被用到。

2.5 小结

本章主要讲述了系统生命周期过程中的测试策略，首先介绍了测试级别，对测试过程中测试级别有一个大体的了解，接着介绍了测试在软件质量体系中的位置和作用。重点介绍了当前行业中的几种软件测试模型，主要包括瀑布模型、V 模型、W 模型、H 模型和 X 模型。最后介绍系统生命周期中的测试策略，主要包括白盒测试和黑盒测试两种，在项目开发阶段中，不同阶段采用的测试策略也不同。