集成测试

学习目标：

- 了解集成测试的概念。
- 理解集成测试的价值。
- 了解集成测试与单元测试的关系。
- 掌握集成测试的测试内容、方法和过程。

本章介绍了集成测试的基本概念、关注的主要内容、测试目的、环境、策略等，并对各种测试策略进行优缺点及适用场合范围分析，结合项目的实际环境以及各测试方案适用的范围进行合理的选择。

5.1 集成测试概述

在测试过程中经常遇到的情况是：单元测试中的每个模块都能单独工作，但是将这些模块集成到一起后，某些模块就不能正常工作了，例如，接口数据丢失、模块之间的不良影响、误差积累等。因此，单元测试无法代替集成测试，每个模块的性能最优并不能保证集成之后的指标达到最优。

集成测试由专门的测试小组来进行，测试小组是由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。

5.1.1 集成测试的定义

集成测试(Intergration Testing，IT)又称组装测试或联合测试。集成测试就是在单元测试的基础上，将所有已通过单元测试的模块按照概要设计的要求组装为子系统或系统，并进行测试的过程。其目的是确保各个单元模块组合在一起后能够按照既定意图协作运行，并确保增量的行为正确。

需要再次强调的是，第一，不经过单元测试的模块是不应该进行集成测试的，否则会给集成测试的效果和效率带来巨大的不利影响；第二，集成测试的主要工作是测试模块之间

的接口，但是接口测试不等于集成测试。

5.1.2 集成测试关注的主要内容

1. 功能性测试

(1) 程序的功能测试。检查各个子功能组合起来能否满足设计所要求的功能。

(2) 模块间是否有不利影响。一个程序单元或模块的功能是否会对另一个程序单元或模块的功能产生不利影响。

(3) 单个模块的误差是否会累积放大。根据计算精度的要求，单个程序模块的误差积累起来，是否仍能够达到要求的技术指标。

(4) 程序单元或模块之间的接口测试。把各个程序单元或模块连接起来时，数据在通过其接口时是否会出现不一致的情况，是否会出现数据丢失。

(5) 全局数据结构是否有问题。检查各个程序单元或模块所用到的全局变量是否一致、合理。

(6) 对程序中可能有的特殊安全性要求进行测试。

2. 可靠性测试

根据软件需求和设计中提出的要求，对软件的容错性、易恢复性、错误处理能力进行测试。

3. 易用性测试

根据软件设计中提出的要求，对软件的易理解性、易学性和易操作性进行检查和测试。

4. 性能测试

根据软件需求和设计中提出的要求，对软件的时间特性和资源特性进行测试。

5. 维护性测试

根据软件需求和设计中提出的要求，对软件的易修改性进行测试。

6. 可移植性测试

根据软件需求和设计中提出的要求，对软件在不同操作系统环境下被使用的正确性进行测试。

5.1.3 集成测试的目的

集成测试的目的是确保各单元组合在一起后能够按既定意图协作运行，并确保增量的行为正确，所测试的内容包括单元间的接口以及集成后的功能。

在现实世界中，当开发应用程序时，它被分解为更小的模块，并且为每个开发人员分配一个模块。一个开发人员实现的逻辑与另一个开发人员完全不同，因此检查开发人员实现的逻辑是否符合预期并根据规定的标准呈现正确的结果是十分重要的。

很多时候，当数据从一个模块移动到另一个模块时，数据层面或结构会发生变化。附加或删除某些值，这会导致后续模块出现问题。

例如，有这样一个集成测试场景，如图 5-1 所示，在银行应用程序中，客户正在使用“查询余额”模块，查询到他的当前余额是 1000 元，当他跳转到“转账”模块，并将 500 元转出到另一个账户，接下来再回到“查询余额”模块，那么现在他的账户的最新余额应为 500 元。此测试涉及了多个模块，需要通过集成测试来完成。

集成测试将单个模块组合在一起并作为一个组进行测试，对模块之间的数据传输进行了全面的测试。

图 5-1　ATM 机取款功能模块

5.1.4　集成测试的环境

随着软件越来越复杂，一个系统往往会分布在不同的软件硬件平台，因此，其集成测试的环境也越来越复杂。在进行集成测试时，对于测试环境主要需要考虑以下几个方面。

1. 硬件环境

在集成测试时，尽可能考虑实际的环境。如果实际环境不可用时，应考虑在模拟环境下进行，并分析模拟环境与实际环境之间可能存在的差异。

2. 软件环境

(1) 操作系统环境，考虑不同机型使用的不同操作系统版本。对于实际环境可能使用的操作系统环境，尽可能都要测试到。

(2) 数据库环境，数据库系统的选择要根据实际的需要，从容量、性能、版本等多方面考虑。

3. 网络环境

一般的网络环境可以使用以太网。

5.2　集成测试的策略和方法

集成测试是在单元测试基础上做的，但不是每个模块写好系统就没问题，将多个单元集成在一起时保证接口间是协调的，这是集成测试的重点。集成测试是一个持续的过程，集成测试的基本策略分为非增量式集成测试策略和增量式集成测试策略。

5.2.1　非增量式集成测试策略

非增量式集成又称为大爆炸集成，也称为一次性集成。该集成就是在最短的时间内把所有通过单元测试的模块一次性地集成到被测系统中进行测试，不考虑组件之间的互相依赖性及可能存在的风险。

如图 5-2 所示，采用大爆炸集成测试方法，是在 A、B、C、D、E、F 各模块分别进行单元测试后，将所有模块组装在一起进行测试。此方法的优点是可以多人并行工作，需要的测试用例数目少，测试方法简单易理解。但当发现错误时，故障定位很困难。使用这种方法需要测试的接口数量众多，很容易会漏掉一些要测试的接口链接。此外，由于集成测试只能在设计完“所有”模块之后才能开始，因此测试团队在测试阶段的执行时间将减少。由于所有模块

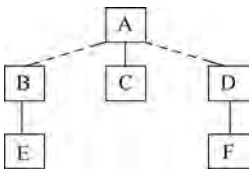

图 5-2　某系统的层次模块图

都被同时测试，因此高风险关键模块不会被隔离并优先进行测试。处理用户界面的外围模块也不是隔离的，并且不会进行优先级测试。

非增量式集成一般适用于维护型项目或一些小型系统，对于维护型项目适合于只有少数模块被增加或修改的情况，非增量集成测试前需要各个组件都要经过充分的单元测试。

5.2.2 增量式集成测试策略

增量式集成测试策略有很多种方法，主要有自顶向下集成、自底向上集成、三明治集成、基于功能的集成、基于风险的集成及分布式集成等。在这种策略下，通过加入两个或多个逻辑相关的模块完成测试，然后添加其他相关模块并测试其功能是否正常。该过程持续进行，直到所有模块都已加入并成功测试。相对于非增量式集成测试策略，该策略的最大特点是支持故障隔离，可以解决故障定位问题。

1. 自顶向下集成测试

在自顶向下集成策略中，按照系统层次结构图，以主程序模块为中心，从顶层控制模块开始，自上而下按照深度优先或者广度优先策略，对各个模块边组装边测试，测试是按照软件系统的控制流程从上到下进行的，可以验证系统的功能性和稳定性。

1）自顶向下集成测试的步骤

（1）对主控模块进行测试。用桩模块代替所有直接附属于主控模块的模块。

（2）根据选定的优先策略（广度优先或深度优先），每次用一个实际模块代替一个桩模块进行测试。

（3）结合下一模块同时进行测试。

（4）为了保证加入的模块没有引入新的错误，需要进行回归测试。

（5）重复（2）～（4）过程，直到所有的模块集成测试完成。

自顶向下集成测试需要借助桩模块进行测试，下面以图 5-2 所示结构为例进行说明。该结构共分为三个层次，上层包括模块 A，中层包括模块 B、C、D，下层包括模块 E、F。如果采用广度优先自顶向下测试方法，首先测试模块 A，其次将模块 A、B、C、D 集成测试，最后将所有模块 A、B、C、D、E、F 集成测试；如果采用深度优先自顶向下测试方法，首先测试模块 A，其次将模块 A、B 集成，然后将模块 A、B、E 集成测试，再将模块 A、B、E、C 集成测试、再将模块 A、B、E、C、D 集成测试，最后将所有模块 A、B、E、C、D、F 集成测试。

2）自顶向下集成测试的优点

（1）较早地验证主要的控制和判断点。自顶向下这种集成方式，在测试过程中可以较早地验证主要的控制和判断点，如果主要控制有问题，尽早发现它能够减少以后的返工，这是十分必要的。

（2）功能可行性较早得到证实。

（3）最多只需要一个驱动模块。自顶向下这种集成方式最多只需要一个驱动模块，减少了驱动模块的费用开支，也降低了后期对驱动模块的维护成本。

（4）可以与设计并行进行测试。由于这种方式与设计的思路是一样的，所以可以与设计并行开展，如果目标环境或者设计需求改变，这种方式也可以灵活适应。

（5）支持故障隔离。不仅故障定位更容易，还支持故障隔离，比如 A 模块测试正常，但是如果在 B 模块之后出现问题，那么可以确定问题可能出现在 B 模块或者 A 模块和 B 模块

之间的接口上。

3）自顶向下集成测试的缺点

（1）桩模块开发和维护的成本大。这种方式要求在每个测试中都必须提供桩模块，因此需要许多桩模块，桩模块的开发与维护成为了该方式的最大成本。

（2）底层组件的一个需要的修改会导致许多顶层组件的修改。这样就破坏了部分先前构造的测试包。

（3）底层模块较多时，会导致底层模块未得到充分测试。随着底层模块的不断增加，系统越来越复杂，导致底层模块的测试不够充分，尤其是那些被重用的模块。

4）桩模块和驱动模块

桩模块是由被测模块调用的模块。

驱动模块是调用要测试模块的模块。

增量方法使用称为桩模块和驱动模块的虚拟程序来执行。桩模块和驱动模块不实现软件模块的整个编程逻辑，而只是模拟与调用模块的数据通信。

5）自顶向下集成测试的适用范围

（1）产品控制结构比较清晰和稳定。

（2）产品的高层接口比较稳定，底层变化比较频繁。

（3）产品的控制模块可能存在技术风险，需要较早被验证。

（4）希望尽早看到产品的系统功能行为。

2. 自底向上集成测试

在自底向上的策略中，从系统层次结构图的最底层模块开始按照层次结构图逐层向上进行组装和集成测试。这种策略从具有最小依赖性的底层组件开始按照依赖关系树的结构，逐层向上集成，以验证整个系统的稳定性。它需要驱动模块的帮助进行测试。

1）自底向上集成测试的步骤

（1）从最底层的模块开始组装测试。

（2）编写驱动程序，协调测试用例的输入与输出。

（3）测试集成后的构件。

（4）使用实际模块代替驱动程序，按程序结构向上组装测试后的构件。

（5）重复(2)～(4)过程，直到系统的最顶层模块加入到系统中完成测试为止。

自底向上集成测试需要借助驱动模块进行测试，下面以图 5-2 所示结构为例进行说明。对已分别进行了单元测试的各个模块，先分别对集成模块 B、E 和集成模块 D、F 并行进行测试，需要编写各自的驱动模块，最后将模块 A、B、C、D、E、F 集成测试。

2）自底向上集成测试的优点

（1）较早地验证底层模块。

（2）在工作的最初可以采用并行进行集成测试，比自顶向下的测试效率高。

（3）集成策略小。由于驱动模块是额外编写的而不是实际的模块，所以对实际被测模块的可测试性要求比自顶向下的测试策略要小。

（4）减少桩模块编写的工作量，支持故障隔离。

3）自底向上集成测试的缺点

（1）驱动模块开发和维护的成本大。

(2) 对高层的验证被推迟到最后,设计上的错误不能被及时发现,尤其对于那些在整个体系中比较关键的产品。

(3) 随着集成到了顶层,整个系统将变得越来越复杂,很难覆盖底层的一些异常。

4) 自底向上集成测试的适用范围

(1) 底层接口比较稳定的产品。

(2) 高层接口变化比较频繁的产品。

(3) 底层模块较早被完成的产品。

3. 三明治集成测试

三明治集成是一种混合增殖式测试策略,是"自顶向下"和"自底向上"方法的组合。三明治集成就是把系统划分为三层,中间一层为目标层,测试时,对目标层上面的一层使用自顶向下的集成策略,对目标层下面的一层使用自底向上的集成策略,最后测试在目标层会合。

1) 三明治集成测试的步骤

(1) 首先对目标层上面的一层采用自顶向下的测试策略,对主模块 A 进行测试,对 A 调用的子模块(目标层)用桩模块代替。

(2) 其次对目标层下面的一层采用自底向上的测试策略。

(3) 最后将三层集成在一起。

三明治集成测试需要利用桩模块与驱动模块进行测试,下面以图 5-2 所示结构为例进行说明。对分别已进行了单元测试的各个模块,在下层分别对集成模块 B、E 和集成模块 D、F 进行测试,在上层对集成模块 A、B、C、D 进行测试,最后将所有模块进行集成测试。

2) 三明治集成测试的优点

三明治集成测试综合了自顶向下集成测试策略和自底向上集成测试策略的优点。不需要大量的桩模块,因为在测试开始的自底向上集成中已经验证了底层模块的正确性。

3) 三明治集成测试的缺点

三明治集成测试中间层在被集成前测试不充分。由于中间层在早期没有得到充分的测试,可能引入缺陷。同时,中间层的选择也很重要,如果中间层选择不当,可能会增加驱动模块和桩模块的设计负担。

4) 三明治集成测试的适用范围

大部分软件开发项目都可以使用这种集成策略。

5.2.3 其他集成测试策略

1. 基于功能的集成测试

基于功能的集成测试是从功能的角度出发,按照功能的关键程度对模块的集成顺序进行组织,目的是尽早地验证系统关键功能。

1) 基于功能的集成测试的步骤

(1) 确定功能的优先级。

(2) 分析优先级最高的功能路径,把该路径上的所有组件都集成到一起,必要时使用驱动模块和桩模块。

(3) 分析下一个关键功能,继续上一步骤,直到针对所有功能都进行了集成。

2）基于功能的集成测试的优点

基于功能的集成测试的优点是可以尽早地看到优先级高的功能（关键功能）被实现，并验证这些优先级高的功能的正确性；由于该方法在验证某个功能时，可能会同时加入多个组件，因此在进度上比自底向上、自顶向下或三明治集成要短；可以减少驱动的开发，原因与自顶向下的集成策略类似。

3）基于功能的集成测试的缺点

基于功能的集成测试的主要缺点是对有些接口的测试不充分，会忽略部分接口错误。如果系统比较复杂，功能之间的相互关联性难以进行分析，可能会产生比较大的冗余测试。

4）基于功能的集成测试的适用范围

此方法适用于关键功能具有较大风险的产品，如：技术探索型的项目，其功能的实现远比质量更关键或一些对于功能实现没有把握的产品。

2. 基于风险的集成测试

基于风险的集成测试是基于一种假设，即系统风险最高的模块间的集成往往是错误集中的地方。因此尽早地对这些高风险模块接口进行重点测试，有助于保证系统的稳定性。

该方法的优点是能够加速系统的稳定性，有利于加强对系统的信心，关键点在于风险的识别和评估，与基于功能的集成测试有一定的相通之处，通常与基于功能的集成测试结合使用。主要适用于系统中风险较大的模块测试。

3. 基于分布式的集成测试

基于分布式的集成测试主要是验证松散耦合的同级模块之间的交互稳定性。在一个分布式系统中，由于没有专门的控制轨迹和服务器层，所以构造测试包比较困难，主要验证远程主机之间的接口是否具有最低限度的可操作。

4. 客户/服务器的集成测试

客户/服务器的集成测试主要针对客户/服务器端系统。对客户端与服务器端的交互进行集成测试，先单独测试每个客户端和服务器端，再将第一个客户端与服务器端集成测试，加入下一个客户端与服务器端集成测试。如此下去，直到所有客户端与服务器端完成集成测试。

5. 持续集成测试

现在的软件开发中采用持续迭代模式，实际工作中将集成测试与单元测试结合的比较紧。持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽早地发现集成错误。

持续集成也叫高频集成测试，通过持续集成进行小范围高频次的测试，测试在研发过程中一直进行。比如白天开发团队进行代码开发，下班前提交代码，已经配置好的测试平台在晚上自动地把新增代码与原有基线集成到一起完成测试，并将测试结果发到各个开发人员的电子邮箱中。

1）什么是持续

“持续”用于描述遵循许多不同流程实践，不是意味着“一直在运行”，而是“随时可运行”。在软件开发领域，它包括以下几个核心概念。

（1）频繁发布：持续实践背后的目标是能够频繁地交付高质量的软件。此处的交付频

率是可变的，可由开发团队或公司定义。对于某些产品，一季度、一个月、一周或一天交付一次可能已经足够频繁了。对于另一些来说，一天可能需要多次交付。所谓持续也有“偶尔”“按需”的方面。最终目标是相同的：在可重复、可靠的过程中为最终用户提供高质量的软件更新。

(2) 自动化流程：实现此频率的关键是用自动化流程处理软件生产中的方方面面，包括构建、测试、分析、版本控制以及在某些情况下的部署。

(3) 可重复：如果使用的自动化流程在给定相同输入的情况下始终具有相同的行为，则这个过程应该是可重复的。也就是说，如果把某个历史版本的代码作为输入，应该得到对应相同的可交付产品。这也假设有相同版本的外部依赖项(即不创建该版本代码使用的其他交付物)。理想情况下，这也意味着可以对管道中的流程进行版本控制和重建。

(4) 快速迭代：“快速”在这里是个相对术语，但无论软件更新/发布的频率如何，预期的持续过程都会以高效的方式将源代码转换为交付物。自动化负责大部分工作，但自动化处理的过程可能仍然很慢。例如，对于每天需要多次发布候选版更新的产品来说，一轮集成测试(integrated testing)耗时就要大半天可能就太慢了。

2) 持续集成(Continuous Integration，CI)平台的组成

(1) 版本控制库。通过受控的访问库管理源代码和其他软件资产(如文档)的变更，提供“单一源代码位置”，让开发人员可从一个可靠渠道取得所有代码，也可以沿时间回溯取得源代码和其他文件的不同版本(数据库集成)。

其中，集中放置软件资产包括：组件(如源文件、库文件等)，第三方组件(取决于语言和使用的平台)、配置文件、初始化应用程序和数据文件、构建脚本和环境设置及组件的安装脚本等。

(2) CI 服务器。变更提交到版本控制库后执行集成构建。可配置定时轮询检查也可按任务手动一键提交新内容，从而变更版本控制库。然后 CI 服务器取出所有源文件运行构建脚本，进行集成构建。通常 CI 服务器会提供一个显示板，构建完成后展示最新构建的结果和构建报告，同时 CI 服务器也可以减少定制脚本的数量。

CI 平台也要提供不同触发类型的触发机制，构建类型常规分为私有构建、集成构建和发布构建等。根据不同的使用情况，会有不同的触发方式需求，如用户驱动触发、定期执行、轮询变更、事件驱动等。

(3) 构建脚本。由于 CI 是一个自动的过程，只使用 IDE 的构建方式不能适应 CI 的需求，所以需要用构建脚本实现过程自动化，包括编译、测试、审查、部署等。

自动化脚本制作使用步骤：确定自动化流程，创建构建脚本，利用版本控制系统将脚本分享到团队使用，利用 CI 使自动化发挥作用。

(4) 反馈机制、文档及反馈。及时提供集成构建的反馈信息，尽快地修复发现的问题。不同业务的持续集成流程不同，在使用 CI 平台时会遇到各种使用问题，平台需提供使用步骤指南和常见问题答疑等文档。

(5) 集成构建计算机。独立的专门用于集成构建的计算机可以确保集成位置不受过去产品的约束。

(6) 对接工具平台插件脚本。研发流程通常包括开发、构建编译、部署、测试、审查及发布，流程很长，不同业务使用的工具平台不同，CI 平台可提供对应接口和插件脚本，使各个

模块可连通，数据信息可传递，从而达到自动化、一体化持续集成的效果。

3）持续集成常用工具

（1）AnthillPro：商业的构建管理服务器，提供C功能。

（2）Bamboo：商业的CI服务器，对于开源项目免费。

（3）Build Forge：多功能商业构建管理工具，特点：高性能、分布式构建。

（4）Cruise Control：基于Java实现的持续集成构建工具。

（5）http://CruiseControl.NET：基于C#实现的持续集成构建工具。

（6）Jenkins：基于Java实现的开源持续集成构建工具，现在最流行和知名度最广泛的持续集成工具。

（7）Lunt build：开源的自动化构建工具。

（8）Para Build：商业的自动化软件构建管理服务器。

5.3 案例分析

如图5-1所示，假如想测试转账后的余额，需要用到两个模块，如果这两个模块分配给不同的开发人员，当一个程序员已准备好“余额查询”模块，另一个程序员未准备好“转账”模块，但需要“转账”模块测试集成场景，在这种情况下应该如何做呢？这就用到了上述的集成测试策略。

如果使用大爆炸集成测试，在开始测试之前，需要等待所有模块开发完毕。它的主要缺点是增加了项目执行时间，因为在所有模块开发完成之前，部分测试人员会处于闲置状态，而且追踪缺陷的根本原因变得很困难。

如果使用增量方法，当模块可用时，在其中检查模块是否集成。考虑到“转账”模块尚未开发，但“当前余额”模块已准备就绪，可以创建一个测试桩，该测试桩将接收数据并将其返回给“当前余额”模块，但这不是“转账”模块的完整实现，完整模块将有许多检查，例如，转出账号是否输入正确的账号，转账金额不得超过账户可用余额等，只是模拟了两个模块之间的数据传输，以方便测试。反之，如果“转账”模块已准备就绪，但“当前余额”模块未开发，则将创建一个测试驱动程序模拟模块之间的传输，为了提高集成测试的效率，可以使用自顶向下的方法。首先测试较高级别的模块，此技术将需要创建测试桩。也可以使用自下而上的方法，首先测试较低级别的模块，例如，“存款”“取款”等下层模块，此方法需要创建测试驱动程序。

5.4 集成测试分析及工具

5.4.1 集成测试分析

集成测试分析既包括对被测软件本身的分析，如体系结构分析、模块分析和接口分析等，也包括对测试可行性和测试策略的分析。

1. 体系结构分析

体系结构分析需要从两个角度出发。第一个角度是从实际需求出发，得到系统实现的层次结构图。第二个角度是划分系统模块，得到系统模块之间的依赖关系图。

2. 模块分析

模块分析是集成测试最重要的工作之一。模块划分的好坏直接影响测试的工作量、进度以及质量。因此在集成测试时，应当确定关键模块，对这些关键模块及早进行测试。一个关键模块应具有如下特征。

(1) 一个模块与多个软件需求或者关键功能有关。

(2) 在程序的模块结构中位于较高的层次(高层控制模块)。

(3) 较复杂，较易发生错误。

(4) 含有性能需求的模块。

(5) 被频繁调用的模块。

3. 接口分析

集成测试的重点就是测试接口的功能性、可靠性、安全性、完整性和稳定性等，因此需要对被测对象的接口进行详细的分析。系统内常见的接口有函数接口、类接口、消息接口、其他接口和第三方接口等。

4. 可测试性分析

可测试性分析在项目最开始时作为一项需求提出来，并设计到系统中去。在集成测试阶段，主要是为了平衡随着集成范围的增加而导致的可测试性下降。

5.4.2 集成测试工具

能够直接用于集成测试的测试工具不是很多，而且大部分通用型工具在实际使用时要根据需求进行二次开发。集成测试主要关注接口的测试，常用的接口测试工具有POSTMan、HPPTRequest、JMeter 等。

5.5 集成测试的评价

一般可从如下 4 方面对集成测试进行评价。

(1) 测试用例的规模。测试用例数量越多，设计、执行和分析这些测试用例所花费的工作量越大，因此，测试用例的规模应越小越好。

(2) 驱动模块的设计。受到模块调用关系的影响，参与某次集成测试的模块可能被不包含在本次集成的其他模块调用，为此需要设计驱动模块，驱动模块不包含在产品代码中，因此，驱动模块的数量应越少越好。

(3) 桩模块的设计。类似地，参与某次集成测试的模块可能调用其他不包含在本次集成中的模块，为此需要设计桩模块，桩模块不应提交给用户，因此，桩模块的数量越少越好。

(4) 缺陷的定位。集成测试的主要任务是检查模块之间的接口，集成测试用例涉及的接口数量越少，越容易定位出错的接口，因此，单个集成测试设计接口的数量越少越好。

5.6 集成测试流程

根据 IEEE 标准，集成测试划分为 5 个阶段，即计划阶段、设计阶段、实施阶段、执行阶段、评估阶段。

1. 集成测试计划阶段

(1) 集成测试准备：相关文档准备，如需求规格说明书、概要设计、产品开发计划等。人员准备，包括测试人员、开发人员、质量控制人员、测试负责人、开发经理等，并明确相关人员的职责。

(2) 集成测试策略与环境：准备开展集成测试的软硬件环境、网络环境，并考虑相应的性能版本等指标。集成测试环境尽量与实际环境相一致。

(3) 集成测试日程计划：根据软件设计文档评估集成测试工作量，合理安排测试日程。一般在概要设计评审后一周开始。

(4) 活动步骤：主要包括确定测试对象与范围，评估集成测试工作量，确定角色与分工，明确测试阶段的时间、任务，制定风险分析与应急计划，准备集成测试工具，定义集成测试完成的标准。

(5) 输出：集成测试计划作为产出物，要通过概要设计阶段基线评审。

2. 集成测试设计阶段

(1) 开始时间：一般地，参照 V-model，集成测试与概要设计相对应，集成测试的设计工作在软件开发的详细设计阶段可以开始。

(2) 集成测试的依据：包括需求规格说明书、概要设计、集成测试计划。

(3) 集成测试设计的入口条件：系统的概要设计基线通过评审。

(4) 活动步骤：主要包括被测试对象体系结构分析、集成测试模块分析、接口分析、集成测试策略、集成测试工具、集成测试环境分析。

(5) 集成测试设计阶段的产物是集成测试设计方案。

(6) 集成测试设计的出口是详细设计通过基线评审。

3. 集成测试实施阶段

集成测试实施阶段的主要工作是根据集成测试计划，建立集成测试环境，完成测试设计任务。

(1) 开始时间：在系统编码阶段开始后可以进行。

(2) 集成测试实施的依据：包括需求规格说明书、概要设计、集成测试计划、集成测试设计方案。

(3) 集成测试实施阶段的入口条件是详细设计通过基线评审。

(4) 活动步骤：集成测试用例设计，根据需要开展集成测试桩模块、驱动模块设计以及代码开发。

(5) 集成测试实施阶段的产物包括集成测试用例、集成测试桩模块和驱动模块代码、集成测试脚本等。

(6) 集成测试设计的出口是集成测试用例通过基线评审。

4. 集成测试执行阶段

(1) 开始时间：集成测试的执行阶段从单元测试完成后开始。

(2) 集成测试执行的输入：需求规格说明书、详细设计、集成测试设计、集成测试用例、集成测试桩模块和驱动模块代码、源代码、单元测试报告。

(3) 集成测试执行阶段的入口条件是单元测试通过基线评审。

(4) 活动步骤：执行集成测试用例，回归测试，撰写集成测试报告。

(5) 集成测试的输出是集成测试报告。

(6) 集成测试执行阶段的出口条件是集成测试阶段基线评审。

5. 集成测试评估阶段

集成测试评估由测试设计人员负责,集成测试人员、编程人员、设计人员对集成测试结果进行统计,生成测试执行报告和缺陷记录,并对集成测试进行评估,对测试结果进行评测,形成结论。

小结

本章重点介绍了集成测试的概念,集成测试与单元测试的区别与联系,集成测试的策略与方法等内容,集成测试的方法有非增量式、增量式、混合/三明治测试等方法,并给出了实际的案例。

习题

1. 判断题

(1) 在 V-model 中,集成测试与概要设计相对应。(　　)

(2) 集成测试一般由专门的测试小组完成。(　　)

(3) 集成测试环境应尽量与实际环境相一致。(　　)

(4) 集成测试的主要工作是测试模块之间的接口。(　　)

(5) 没有经过单元测试的程序也可以进行集成测试。(　　)

2. 简答题

(1) 简述集成测试与单元测试的区别。

(2) 简述自顶向下集成测试的优缺点。

(3) 简述自底向上集成测试的优缺点。

(4) 简述不同集成测试策略的应用场景。

(5) 简述集成测试的流程。