软件集成、确认和系统测试方法

引言

软件测试按测试用例设计(TEST CASE DESIGN)方法分为白盒测试(WHITE-BOX TESTING)和黑盒测试(BLACK-BOX TESTING)。 按测试过程或测试策略，软件测试分为单元测试(UNIT TESTING)，集成测试（INTEGRATION TESTING〕，确认测试（VALIDATION TESTING〕和系统测试（SYSTEM TESTING〕。在以前的有关文档中，我们已经对白盒和黑盒测试中的测试用例设计方法进行了详细的讲解。同时也对单元测试进行了讲解和 培训。本文将从测试策略方面讲解软件测试中的集成测试，确认测试和系统测试策略。

软件测试的 策略方法

测试是一系列有计划地系统性的活动。为了实行测试活动，人们提出了许多测试策略方法。一个软件测试策略不仅要包括低层测试（LOW LEVEL TESTING〕而且要包含高层测试〔HIGH LEVEL TESTING〕。低层测试是为了验证原代码的正确性。高层测试是为了证实主要系统功能满足用户需求性。

验证与确认（VERFICATION AND VALIDATION〕

在广义上，软件测试是验证和确认VERFICATION AND VALIDATION （V & V〕。验证指保证软件正确地实现了一特定功能的一系列活动。确认是指保证所生产的软件可追溯到用户需求的一系列活动。BOEHM对V & V的解释是：

VEIFICATION：“Are we building the product right?”

VALIDATION: “ Are we building the right product?”

V&V的定义包含了许多活动，即软件质量保证SQA。图2-1 示出实现软件质量的这些活动。

软件工程方法提供了质量建立的基础。分析、设计和编码方法通过提供统一的技术和可预测的结果来提高质量。正规检视和评审有助于保证软件工程各个阶段产品的质量。度量和控制被应用到软件配置的每一个部件中。标准和过程有助于保证开发的一致性。一个正规的 SQA过程加强整体质量。测试是保证质量的最后一道措施。但是不能把测试看作一个安全网。质量是贯穿于软件过程的每一个阶段。因此尽管测试在V&V中起着非常重要的作用，但是许多其它活动也是必要的。为了提高软件的全员质量，应该重视V&V中的每一个活动.

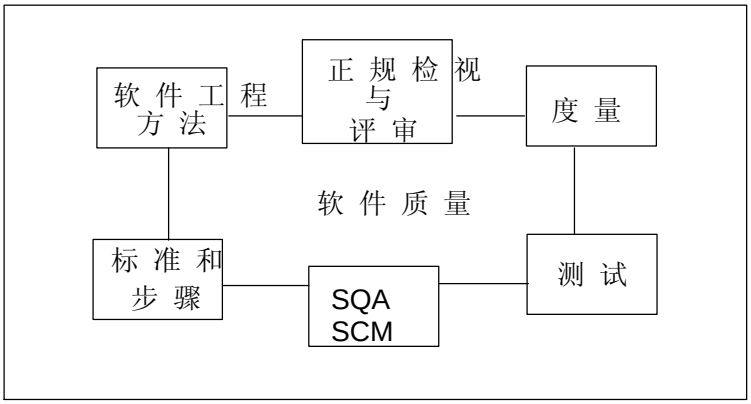

图2-1 实现软件质量的活动

软件测试策略

软件工程过程可以看成一螺旋状。软件测试策略也可以看成一螺旋状。如图2-2示测试策略图。

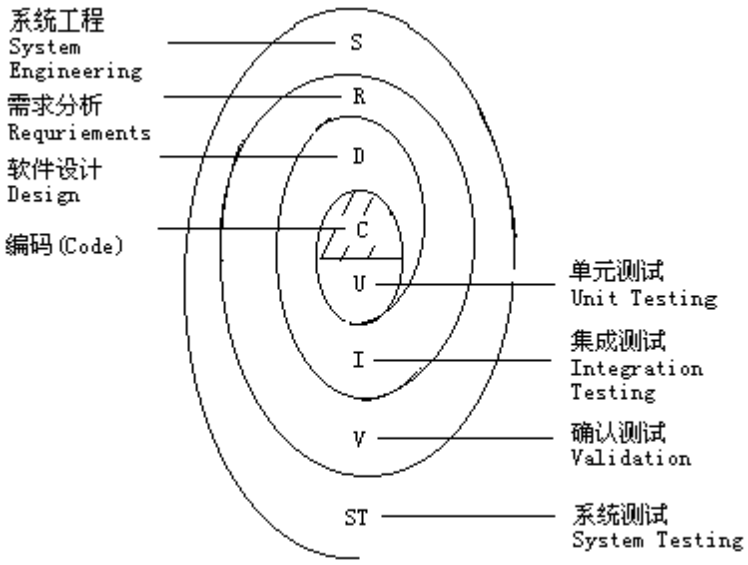

图2-2 软件测试策略图

从图2-2可看出软件测试策略: 起始于代码阶段的单元测试，然后是向外延伸到设计阶段的集成测试，再扩展到需求分析阶段的确认测试，最后是系统工程阶段的系统测试。从系统过程的角度看，测试策略有四个步骤：单元测试，集成测试,确认测试和系统测试。图2-3示软件测试步骤。

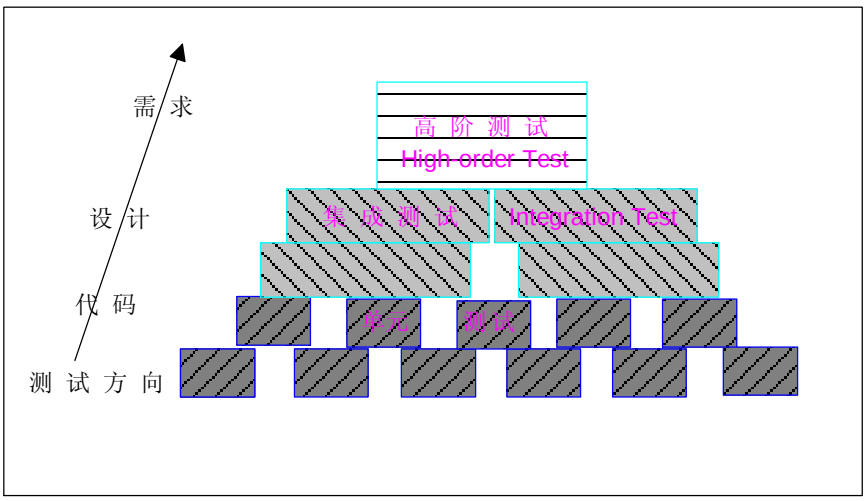

图2-3 软件测试步骤

从图2-3可看出，初始测试集中在每个模块上，由于每个模块完成一个功能单元，因此该阶段的测试称为单元测试。单元测试主要应用白盒测试方法。接下来是模块组装和集成以便组成完整的软件包。集成测试集中在证实和程序构成问题上，集成测试主要采用黑盒测试方法，附之以白盒测试方法。软件集成后，需要完成一系列高阶测试（确认和系统测试）。确认标准必须被测试。确认测试提供软件满足所有功能、性能需求的最后保证。确认测试仅仅应用黑盒测试用例设计方法。

测试完成的标准

软件测试中，人们经常会提出这样的问题：“什么时候测试完成？——怎样知道测试足够充分？”。很遗憾，对该问题还没有一个确定的答案，但是可提供一些统计和经验上的答案或指导。为了确定何时算测试进行的足够充分，软件工程师需要更严格的标准。MUSA和ACKERMAN建议基于统计准则来回答这些问题。应用统计模型和软件可靠性理论，软件失效/故障（测试中未发现的）的模型可以以执行时间函数形式建立。例如，一个软件故障模型，称为对数泊松执行时间模型。有关软件可靠性和软件测试强度方面的知识和模型建立将在以后相关文档中介绍。

单元测试（UNIT TESTING)

单元测试是基于程序模块进行正确性验证的测试。这方面的内容，我们已经准备出了充分的资料，并进行了多期培训。因此这里将不在讲述。

集成测试（INTEGRATION TESTING〕

集成测试，也叫组装测试或联合测试。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。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。也就是应该考虑以下问题：

（1〕在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；

（2）各个子功能组合起来，能否达到预期要求的父功能；

（3）一个模块的功能是否会对另一个模块的功能产生不利的影响；

（4）全局数据结构是否有问题；

（5）单个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

因此，单元测试后，有必要进行集成测试，发现并排除在模块连接中可能发生的上述问题，最终构成要求的软件子系统或系统。对子系统，集成测试也叫部件测试。

任何合理地组织集成测试，即选择什么方式把模块组装起来形成一个可运行的系统,直接影响到模块测试用例的形式、所用测试工具的类型、模块编号和测试的次序、生成测试用例和调试的费用。通常，有两种不同的组装方式：一次性组装方式和增值式组装方式。

一次性组装方式（BIG BANG）

一次性组装方式是一种非增值组装方式（NON－INCREMENTAL INTEGRATION），也叫整体拼装。按这种组装方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。例如，有一块系统结构，如图4-1(a)所示。其单元测试和组装顺序如图4-1(b)所示。

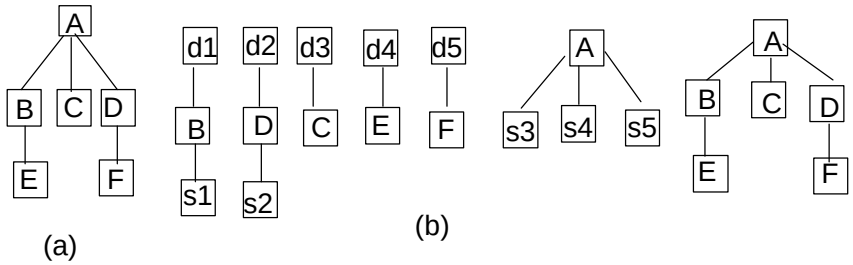

图4-1 一次性组装方式

在图中，模块d1,d2,d3,d4,d5是对各个模块做单元测试时建立的驱动模块，s1,s2,s3,s4,s5是为单元测试而建立的桩模块。这种一次性组装方式试图在辅助模块的协助下，在模块单元测试的基础上，将所测模块连接起来进行测试。但是由于程序中不可避免地存在模块间接口、全局数据结构等方面的问题，所以一次试运行成功的可能性并不很大。其结果发现有错误，但茫然找不到原因。查错和改错都会遇到困难。

增殖式组装方式(Incremental Integration)

增值式组装方式又称渐增式组装。首先是对一个个模块进行模块单元测试，然后将这些模块组装成较大系统，在组装的过程中边连接边测试，以发现连接过程中产生的问题。最后增殖逐步组装成为要求的软件系统。

自顶向下的增殖方式(Top-Down Integration)

这种组装方式是将模块按系统程序结构，沿控制层次自顶向下进行组装。其步骤如下：

(1)以主模块为所测模块兼驱动模块，所有直属于主模块的下属模块全部用桩模块对主模块进行测试。

(2)采用深度优先(depth-first)(如图4-2）或宽度优先(breadth-first)的策略，用实际模块替换相应桩模块，再用桩代替它们的直接下属模块，与已测试的模块或子系统组装成新的子系统。

(3)进行回归测试（即重新执行以前做过的全部测试或部分测试），排除组装过程中引起的错误的可能。

(4)判断是否所有的模块都已组装到系统中？是则结束测试，否则转到(2)去执行。

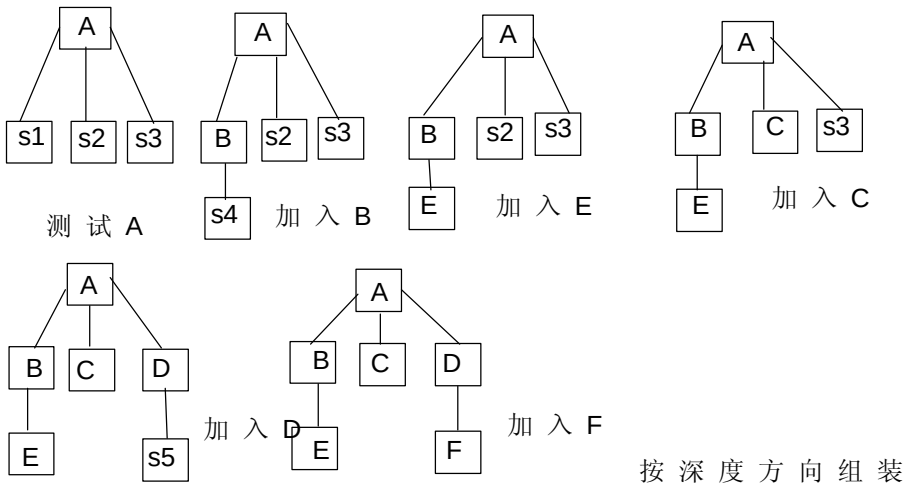

图4-2 自顶向下增殖方式（按深度方向组装)

自顶向下的增殖方式在测试过程中较早地验证了主要的控制和判断点。在一个功能划分合理的程序模块结构中，判断常常出现在较高的层次里，因而较早就能遇到。如果这主要控制有问题，尽早发现它能够减少以后的返工，所以这是十分必要的。如果选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能，可先对逻辑输入的分支进行组装和测试，检查和克服潜藏 的错误和缺陷，验证其功能的正确性，就为其后对主要加工分支的组装和测试提供了保证。此外，功能可行性较早得到证实，还能够给开发者和用户带来成功的信心。

自顶各下的组装和测试存在一个逻辑次序问题。在为了充分测试较高层的处理而需要较低层处理的信息时，就会出现这类问题。在自顶向下组装阶段，还需要用桩模块代替较低层的模块，

所以关于桩模块的编写，根据情况可能不同有如下图4-3所示的几种选择。

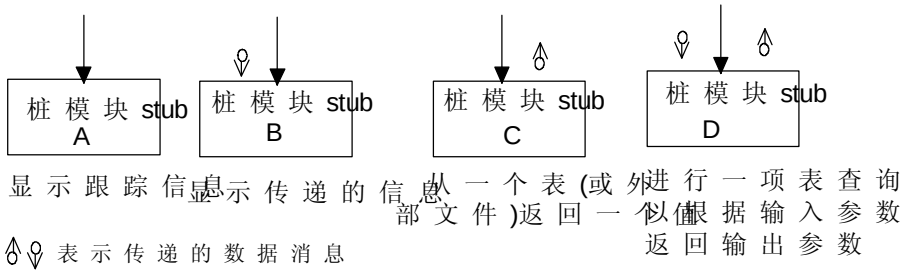

图4-3 桩模块的几种选择

为了能够准确地实施测试，应当让桩模块正确而有效地模拟子模块的功能和合理的接口，不能是只包含返回语句或只显示该模块已调用信息，不执行任何功能的哑模块。如果不能使桩模块正确地向上传递有用的信息，可以采用以下解决办法。

(1)将很多测试推迟到桩模块用实际模块替代了之后进行；

(2)进一步开发能模拟实际模块功能的桩模块；

(3)自底向上组装和测试软件；

自底向上的增殖方式

这种组装的方式是从程序模块结构的最底层的模块开始组装和测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块(包括子模块的所有下属模块)已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以直接运行子模块得到。自底向上增殖的步骤如下：

(1)由驱动模块控制最底层模块的并行测试；也可以把最底层模块组合成实现某一特定软件功能的簇，由驱动模块控制它进行测试。

(2)用实际模块代替驱动模块，与它已测试的直属子模块组装成为子系统。

(3)为子系统配备驱动模块，进行新的测试。

(4)判断是否已组装到达主模块。是则结束测试，否则执行(2)。

以图4-1(a)所示的系统结构为例，用下图4-4来说明自底向上组装和测试的顺序。

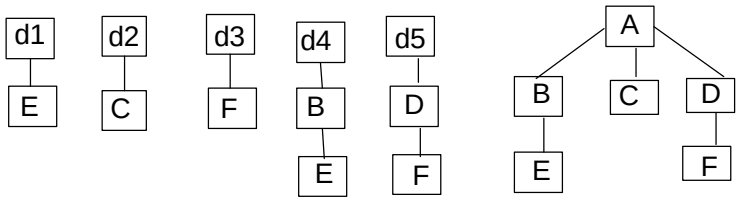

图 4-4 自 底 向 上 增 值 组 装 方 式

自底向上进行组装和测试时，需要为所测模块或子系统编制相应的驱动模块。常见的几种类型的驱动模块如图4-5所示：

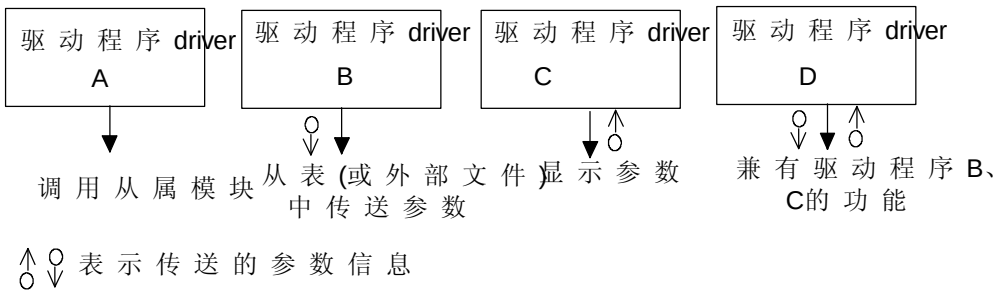

图 4-5 驱 动 模 块 的 几 种 选 择

随着组装层次的向上移动，驱动模块将大为减少。如果对程序模块结构的最上面两层模块采用自顶向下进行组装和测试，可以明显地减少驱动模块的数目，而且可以大大减少把几个系统组装起来所需要做的工作。

混合增殖式测试

自顶向下增殖的方式和自底向上增殖的方式各有优缺点。一般来讲，一种方式的优点是另一种方式的缺点。

自顶向下增殖方式的缺点是需要建立桩模块。要使桩模块能够模拟实际子模块的功能十分困难，因为桩模块在接收了所测模块发送的信息后需要按照它所代替的实际子模块功能返回应该回送的信息，这必将增加建立桩模块的复杂度，而且导致增加一些附加的测试。同时涉及复杂算法和真正输入/输出的模块一般在底层，它们是最容易出问题的模块，到组装和测试的后期才遇到这些模块，一旦发现问题，导致过多的回归测试，而自顶向下增殖方式的优点能够较早地发现在主要控制方面的问题。

自底向上增殖方式的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”。就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。但这种方式的优点是不需要桩模块，而建立驱动模块一般比建立桩模块容易，同时由于涉及

到复杂算法和真正输入/输出的模块最先得到组装和测试，可以把最容易出问题的部分在早期解决。此外自底向上增殖的方式可以实施多个模块的并行测试，提高测试效率。因此，通常是把以上两种方式结合起来进行组装和测试。下面简单介绍三种常见的综合的增殖方式。

(1)衍变的自顶向下的增殖测试：它的基本思想是强化对输入/输出模块的和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增殖测试。

(2)自底向上-自顶向下的增殖测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

(3)回归测试：这种方式采取自顶向下的方式测试所修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

在组装测试时，测试者应当确定关键模块，对这些关键模块及早进行测试。关键模块至少应具有以下几种特征之一：(1)满足某些软件需求；(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)有明确定义的性能要求。

在做回归测试时，也应该集中测试关键模块的功能。

集成测试的组织和实施

集成测试是一种正规测试过程，必须精心计划，并与单元测试的完成时间协调起来。在制定测试计划时，应考虑如下因素：

1)是采用何种系统组装方法来进行组装测试；

2)组装测试过程中连接各个模块的顺序；

3)模块代码编制和测试进度是否与组装测试的顺序一致

4)测试过程中是否需要专门的硬件设备；

解决了上述问题之后，就可以列出各个模块的编制、测试计划表，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期、以及需要的测试用例和所期望的测试结果。

在缺少软件测试所需要的硬件设备时，应检查该硬件的交付日期是否与集成测试计划一致。例如，若测试需要数字化仪和绘图仪，则相应测试应安排在这些设备能够投入使用之时，并需要为硬件的安装和交付使用保留一段时间，以留下时间余量。此外，在测试计划中需要考虑测试所需软件（驱动模块、桩模块、测试用例生成程序等）的准备情况。

集成测试完成的标志

怎样判定集成测试过程完成了, 可按以下几个方面检查:

1)成功地执行了测试计划中规定的所有集成测试；

2)修正了所发现的错误；

3)测试结果通过了专门小组的评审。

集成测试应由专门的测试小组来进行，测试小组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。

在完成预定的组装测试工作之后，测试小组应负责对测试结果进行整理、分析，形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。此外还应提出目前不能解决、还需要管理人员和开发人员注意的一些问题，提供测试评审和最终决策，以提出处理意见。

集成测试需要提交的文档有：集成测试计划、集成测试规格说明、集成测试分析报告。

确认测试(Validation Testing)

确认测试又称为效性测试。它的任务是验证软件的功能和性能及其特性是否与用户的要求一致。对软件的功能和性能要求在软件需求规格说明中已经明确规定。在软件需求规格说明中描述了全部用户可见的软件属性，其中有一节叫做有效性准则，它包含的信息就是软件确认测试的基础。 集成测试完成以后，分散开发的模块被联接起来，构成完整的程序。其中各模块之间接口存在的种种问题都已消除。于是测试工作进入最后阶段--确认测试(Validation testing)。什么是确认测试，说法众多，其中最简明、最严格的解释是检验所开发的软件是否能按顾客提出的要求运行。若能达到这一要求，则认为开发的软件是合格的。因而有的软件开发部门把确认测试称为合格性测试(qualification testing)。这里所说的顾客要求通常指的是在软件规格说明书中确定的软件功能和技术指标，或是专门为测试所规定的确认准则。

在确认测试阶段需要做的工作如下图5-1所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

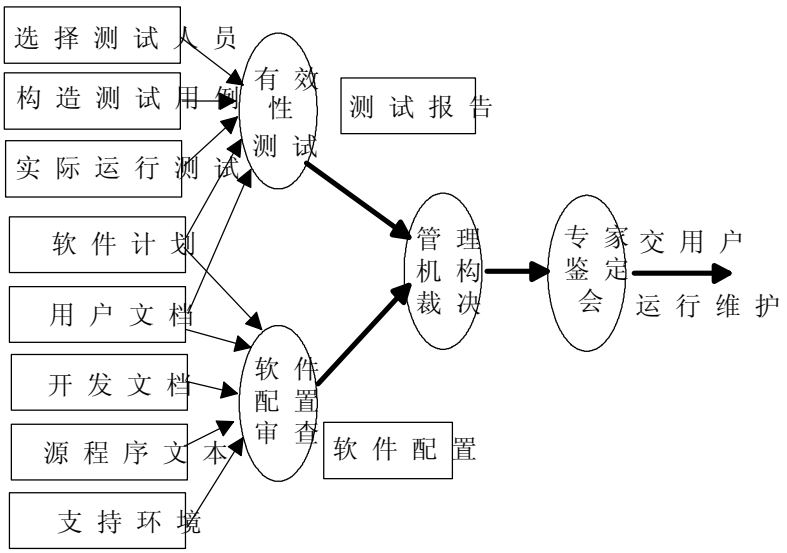

图5-1 确认测试的步骤

确认测试的准则

怎样来判断被开发的软件是成功的？为了确认它的功能、性能以及限制条件是否达到了要求，应进行怎样的测试？在需求规格说明书中可能作了原则性规定，但在测试阶段需要更详细、更具体地在测试规格说明书(Test specification)中作进一步说明。例如，制定测试计划时，要说明确认测试应测试哪些方面，并给出必要的测试用例。除了考虑功能、性能以外，还需检验其它方面的要求。例如，可移植性、兼容性、可维护性、人机接口以及开发的文件资料等是否符合要求。

经过确认测试，应该为已开发的软件作出结论性评价。这也无非是两种情况之中的一个：（1）经过检验的软件功能、性能及其它要求均已满足需求规格说明书的规定，因而可被接受。认为是合格的软件。（2）经过检验发现与需求说明书有相当的偏离，得到一个各项缺陷清单。对于第二种情况，往往很难在交付期以前把发现的问题纠正过来。这就需要开发部门和顾客进行协商，找出解决的办法。

进行有效性测试（黑盒测试）

有效性测试是在模拟的环境（可能是就是开发的环境）下，运用黑盒 测试的方法，验证所测试件是否满足需求规格说明书列出的需求。为此，需要首先制定测试计划，规定要做测试的种类，还需要制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求能达到，所有的文档都是正确且便于使用。同时，对其他软件需求，例如可移植性、兼容性，自动恢复、可维护性等，也都要进行测试，确认是否满足。在全部软件测试的测试用例运行完后，所有的测试结果可以分为两类：

1)测试结果与预期的结果相符。这说明软件的这部分功能或性能特征与需求规格说明一致，因此要为它提交一份问题报告。

2)测试结果与预期的结果不符。这说明软件的这部分功能或性能特征与需求规格说明不一致，因此要为它提交一份问题报告。

软件配置审查

软件配置审查是确认测试过程的重要环节. 其的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，维护阶段所必需的细节，而且已经编排好分类的目录。

除了按合同规定的内容和要求，由工人审查软件配置之外，在确认测试的过程，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误，并且适当地补充和改正。

α 测试和 β 测试

在软件交付使用之后，用户将如何实际使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误 解、异常的数据组合，以及产生对某些用户来说似乎是清晰的但对另一些用户来说却难以理解的输出等等。

当软件是为特定用户开发的时候，需要进行一系列的验收，让用户验证所有的需求是否已经满足。这些测试是以用户为主，而不是以系统开发者为主进行的。验收测试可以是一次简单的非正式的“测试运行”。也可以是一组复杂的有组织有计划的测试活动。事实上，验收测试可能持续几个星期到几个月。

但是如果软件是为多个用户开发的产品的时候，让每个用户逐个执行正式的验收测试是不切实际的。很多软件产品生产者采用一种称之为a测试和b测试的测试方法，以发现可能只有最终用户才能发现的错误。

α 测试是由一个用户在开发环境下进行的测试，也可以是开发机构枘部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用。开发者坐在用户旁边，随时记下错误情况和使用中的问题。这是在受控制的环境下进行的测试， α 测试的目的是班次价软件产品的FLURPS(即功能、局域化、可使用性、可靠性、性能和支持)。尤其注重产品的界面和特色。a测试人员是除开产品开发人员之外首先见到产品的人，他们提出的功能和修改意见是特别有价值的。a测试可以从软件产品编码结束之时开始，或在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程序之后再开始。有关的手册（草稿）等应事先准备好。

β 测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。这些用户是与公司签定了支持产品预发行合同的外部客户，他们要求使用该产品，并愿意返回有关错位错误信息给开发者。与 α 测试不同的是，开发者通常不在测试现场。因而， β 测试是在开发者无法控制的环境下进行的软件现场应用。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告，开发者在综合用户的报告之后，做出修改，最将软件产品交付给全体用户使用。 β 测试主要衡量产品的FLURPS。着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。

由于 β 测试的主要目标是测试可支持性，所以 β 测试应尽可能由主持产品发行的人员来管理。

验收测试(Acceptance Testing)

在通过了系统的有效性测试及软件配置审查之后，就应开始系统的验收测试。验收测试是以用户为主的测试。软件开发人员和QA(质量保证)人员也应参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果，一般使用生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

验收测试实验上是对整个测试计划进行一种“走查(Walkthrough)”。

确认测试的结构

确认测试的结果有两种情况：

1)功能和性能与用户的要求一致，软件可以接受；

2)功能和性能与用户的要求有差距。

出现后一种情况，通常与软件需求分析阶段的差错有关。这时需要开列一张软件各项缺陷表或软件问题报告，通过与用户的协商，解决所发现的缺陷和错误。

确认测试应交付的文档有：确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

系统测试(System Testing)

系统测试是将通过确认测试的软件，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组装测试和确认测试。

系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。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来运行。

由于软件只是计算机系统中的一个组成部分，软件开发完成以后，最终还要与系统中其它部分配套运行。系统在投入运行以前各部分需完成组装和确认测试，以保证各组成部分不仅能单独地受到检验，而且在系统各部分协调工作的环境下也能正常工作。这里所说的系统组成部分除去软件外，还可能包括计算机硬件及其相关的外围设备、数据及其收集和传输机构、掌握计算机系统运行的人员及其操作等，甚至还可能包括受计算控制的执行机构。显然，系统的确认测试已经完全超出了软件工作的范围。然而，软件在系统中毕竟占有相当重要的位置，软件的质量如何，软件的测试工作进行得是否扎实势必与能否顺利、成功地完成系统测试关系极大。另一方面，系统测试实际上是针对系统中各个组成部分进行的综合性检验。尽管每一个检验有着特定的目标，然而所有的检测工作都要验证系统中每个部分均已得到正确的集成，并能完成指定的功能。以下分别简要说明几种系统测试：

恢复测试

恢复测试是要采取各种人工干预方式使软件出错，而不能正常工作，进而检验系统的恢复能力。如果系统本身能够自动地进行恢复，则应检验：重新初始化，检验点设置机构、数据恢复以及重新启动是否正确。如果这一恢复需要人为干预，则应考虑平均修复时间是否在限定的范围以内。

安全测试

安全测试的目的在于验证安装在系统内的保护机构确定能够对系统进行保护，使之不受各种非常的干扰。系统的安全测试要设置一些测试用例谋略实在系统的安全保密措施，检验系统是否有安全保密的漏洞。

强度测试

检验系统的能力最高实际限度。进行强度测试时，让系统的运行处于资源的异常数量、异常频率和异常批量的条件下。例如，如果正常的中断平均频率为每秒一到二次，强度测试设计为每秒10次中断。又如某系统正常运行 可支持10个终端并行工作，强度测试则检验15个终端并行工作的情况。

性能测试

性能测试检验安装在系统内的软件运行性能。这种测试常常与强度测试结合起来进行。为记录性能需要在系统中安装必要的量测仪表或是为度量性能而设置的软件(或程序段)。

参考资料：1.《计算机软件测试技术》郑人杰，清华大学出版社，1992。
2.《Software Engineering--A Practioner’s Approach》,R. S. Pressman, 1998.
3.《实用软件工程》，郑人杰，殷人昆，陶永雷，清华大学出版社，1997。