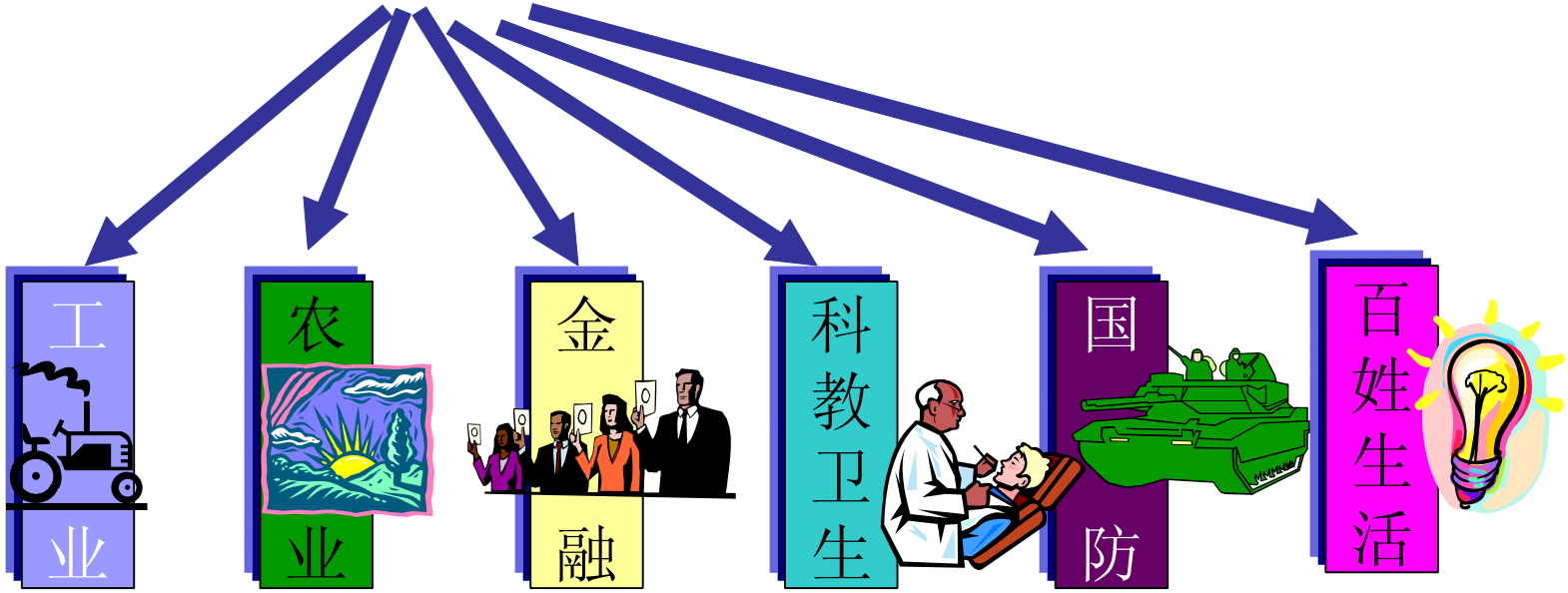
软件测试与质量保证

张鹤飞

软件测试

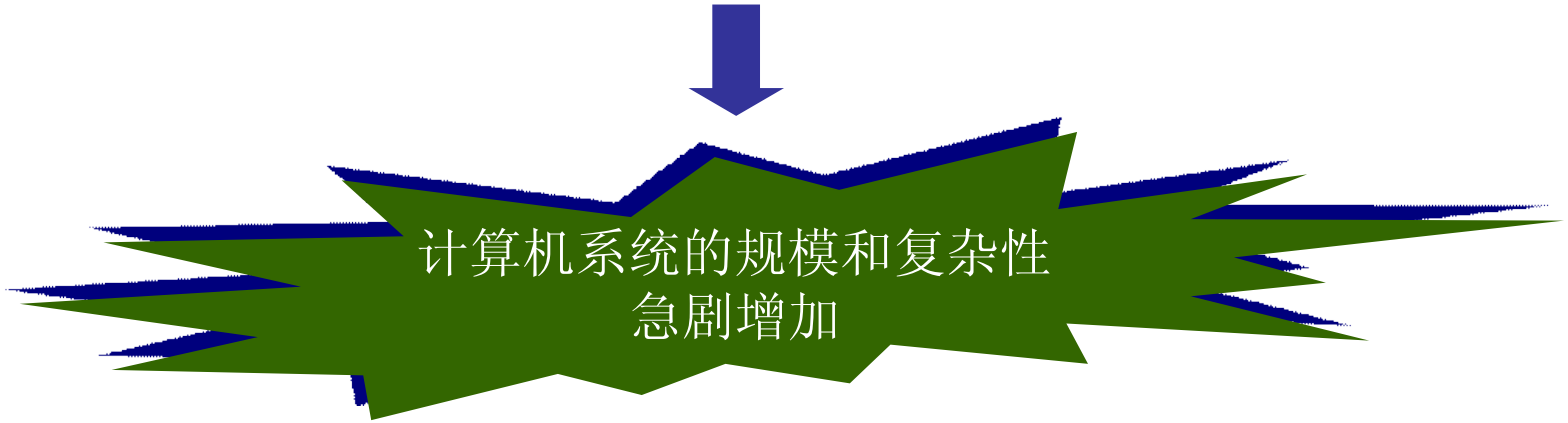
从计算机诞生至今，计算机无疑成为当代发展最为迅猛的科学技术。

今天，计算机已渗透到人们生活的各个方面。



软件测试

随着对计算机需求和依赖的与日俱增



其软件开发成本以及由于软件故障而造成的经济损失也正在增加

软件质量问题已成为人们共同关注的焦点。

软件测试

软件开发商为了占有市场，

把软件质量作为企业的重要目标之一，

以免在激烈的竞争中被淘汰出局。

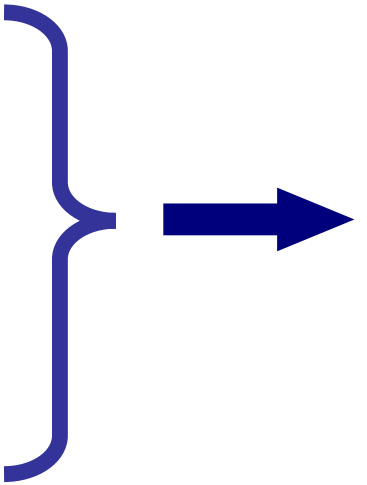
用户为了保证自己业务的顺利完成，

当然也希望选用优质的软件。

软件测试

一些关键应用，如

- 民航订票系统、
- 银行结算系统、
- 证券交易系统、
- 自动飞行控制软件、
- 军事防御、
- 核电站安全控制系统



对软件质量提出了更高的要求。

软件测试

使用质量欠佳的软件，还可能造成灾难性的后果。

- 2003年，软件问题造成美国东北部及加拿大停电，导致5000万人受影响，3人丧生，各种损失估计约为60亿美元
- 2004年，北美银行由于一个新安装的软件的缺陷，使得数以百万计的客户受到影响，该缺陷的修复花费了整整两个星期的时间，造成的损失以亿元计
- 2000年美国海军飞机坠落，导致4人丧生（控制软件问题）
- 1997年韩国空难，导致225人丧生（雷达控制软件问题）
- 2003年4月，美国一个专门为学生提供贷款的公司由于软件出错，错误计算80万宗学生贷款利率，导致了800万美元的利率损失
- 千年虫问题
-

因此，许多科学家在展望21世纪计算机科学发展方向和策略时，把软件质量放在优先于提高软件功能和性能的地位。

软件测试

软件测试是对软件需求分析、设计规格说明和编码的最终复审

是软件质量保证的关键步骤

是为了发现故障而执行程序的过程。

软件测试

随着软件系统规模和复杂性的增加，

进行专业化高效软件测试的要求越来越严格，

软件测试职业的价值逐步得到了认可，

软件测试从业人员急剧增加，

软件测试

软件测试无疑是面向未来信息化社会的热门专业之一

在未来3~5年内，软件测试技术将作为一门新兴产业而快速发展起来。

内容提要：

第一章　软件测试概述
第二章　软件测试实质
第三章　软件测试策略
第四章　黑盒测试
第五章　白盒测试
第六章　集成测试与系统测试
第七章　验证测试和确认测试
第八章　测试计划与测试文档
第九章　面向对象的软件测试
第十章　软件测试自动化和测试工具
第十一章　软件质量保证
第十二章　软件测试职业指导

第一章 软件测试概述

1.1 计算机系统的软件可靠性问题
1.2 软件测试与软件可靠性
1.3 软件测试的发展历史、现状和展望

人们对计算机依赖的程度越高，对其可靠性的要求就越高。

IEEE(Institute of Electrical and Electronics Engineers，电气与电子工程师学会)定义软件可靠性为：
系统在特定的环境下，在给定的时间内，无故障地运行的概率。
用来评价软件按照用户的要求和设计目标，完成规定功能的能力，
涉及软件的性能、功能性、可用性、可服务性、可安装性、可维护性
以及文挡等多方面特性。
软件测试是保证软件可靠性的一个关键因素。

1.软件测试的重要性

计算机和程序是一对孪生兄弟,自从计算机诞生之日就必须要有程序在其上运行,从而得到问题的正确解,必须对程序的功能性进行测试.所以,软件测试工作是在软件工程诞生之前就客观存在了,一直延用至今,且其测试的内容和技术也有了较大的发展。

无论从何种角度讲，软件测试都是软件开发过程一个必不可少的活动，是对软件需求分析、设计规约和编码的最终复审；是软件质量保证的关键步骤。软件测试是根据软件开发各阶段的规约和软件的内部结构，精心设计一批测试用例（包括输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现软件中不符合质量特性要求（即缺陷或错误）的过程。目前，许多软件开发机构将研制力量的40%以上投入到软件测试之中，体现了充分重视软件质量要求。可见软件测试的重要性。

软件测试不等同于程序测试。软件测试应当贯穿在软件生存周期全过程。因此，需求描述、需求规约、设计规约、模块设书以及程序等都应成为软件测试的对象。

在进行软件产品或软件系统开发时，主要有3类人员必须参与，他们是项目经理、开发人员和测试人员。一般来说，大家都会十分重视开发工作，因此在一个项目组中，会有很多的开发人员，而测试人员比较少。经过多次实践后，才会增加测试人员，如微软公司就是这种情况。目前软件测试人员就比较多了，如Exchange 2000，项目经理23人，开发人员140人，测试人员350人；再如Windows 2000，项目经理250人，开发人员1700人，测试人员3200人，可以看出测试人员和开发人员之比，竟达5：3。

2.软件测试的目的和原则

基于不同的立场，存在着不同的测试目的。从用户的角度看，一般希望通过软件测试暴露软件中隐藏的缺陷，以此来决定是否可以接受该软件产品和软件系统。从软件开发者的角度看，希望通过测试说明软件中不存在缺陷，表明该软件已正确地实现了用户的要求，从而确立用户对该软件的质量信任。从软件管理者角度看，希望花费有限的资源达到该软件用户的质量要求，经费和进度是其首要考虑的焦点。因此，就软件测试的目的可提出如下的观点：

（1）测试是程序的执行过程，目的在于发现缺陷；

（2）一个好的测试用例在于能发现至今未发现的缺陷；

（3）一个成功的测试是发现了至今未发现的多个缺陷的测试。

所以，测试的目的是以最少的资源和时间，找出软件中隐藏的各种缺陷甚至错误。测试的成功与否就是以发现软件中潜在的缺陷多少来衡量。

根据这些测试目的和目标，软件测试应该注意些什么呢？郑人杰等人编著的《实用软件工程》第2版（清华大学出版社，1999年）中有一段描述对此问题进行了回答。

（1）应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。

由于原始问题的复杂性，软件的复杂性和抽象性，软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。所以不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到 软件开发的各个阶段中。坚持在软件开发的各个阶段的技术评审，这样才能在开发过程中尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患，提高软件质量。

（2）测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。

在做测试之前，应当根据测试的要求选择在测试过程中使用的测试用例。测试用例主要用来检验程序员编制的程序，因此不但需要测试的输入数据，而且需要针对这些输入数据的预期输出结果。如果对测试输入数据没有给出预期的程序输出结果，那么就缺少了检验实测结果的基准，就有可能把一个似是而非的错误结果当成正确结果。

（3）程序员应避免检查自己的程序.

测试工作需要严格的作风，客观的态度和冷静的情绪。人们常常由于各种原因，具有一种不愿否定自己工作的心里，认为揭露自己程序中的问题总不是一件愉快的事，这一心里状态就成为测试自己程序的障碍。另外，程序员对规约理解错误而引入的错误更难发现。如果由别人来测试程序员编写的程序，可能会更客观，更有效，并更容易取得成功。要注意的是，这点不能与程序的排错（调试）相混淆。排错由程序员自己来做可能更有效。

（4）在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。

合理的输入条件是指能验证程序正确的输入条件，而不合理的输入条件是指异常的、临界的及可能引起问题变异的输入条件。在测试程序时，人们常常倾向于过多地考虑合法的和期望的输入条件，以检查程序是否做了它应该做的事情，而忽视了不合法的和预想不到的输入条件。事实上，软件在投入运行之后，用户的使用往往不遵循事先的约定，使用了一些意外的输入，如用户在键盘上按错了键或输入了非法的命令。如果开发的软件遇到这种情况时不能作出适当的反应，给出相应的信息，那么就容易产生故障，轻则给出错误结果，重则导致软件失效。因此，软件系统处理非法命令的能力也必须在测试时受到检验。用不合理的输入条件测试程序时，往往比用合理的输入条件进行测试能发现更多的错误。

（5）充分注意测试中的群集现象。

测试时不要以为找到了几个错误问题就已解决，不须继续测试了。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目或检错率成正比。

在所测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集性现象，已为许多程序的测试实践所证实。例如IBM公司的OS/370操作系统，47%的错误仅与该系统的4%的程序模块有关。

（6）严格执行测试计划，排除测试的随意性。

测试计划应包括：所测试软件的功能，输入和输出，测试内容，各项测试的进度安排，资源要求，测试资料，测试工具，测试用例的选择，测试的控制方式和过程，系统组装方式，跟踪规程，排错规程，回归测试的规定以及评价标准等。

对于测试计划，要明确规定，不要随意解释。

（7）应当对每一个测试结果做全面检查

这是一条最明显的原则，但常常被忽视。有些错误的征兆在输出实测结果时已经明显地出现了，但是如果不仔细全面地检查测试结果，就会使这些缺陷或错误被遗漏掉。所以必须对预期的输出结果明确定义，对实测的结果仔细分析检查，抓住症候，暴露错误。

（8）妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便。

3.软件测试与相关的几个概念

软件测试不仅仅是软件质量保证体系中的重要一环,而且也是保证质量的重要技术手段。

（1）排错（debugging）是查找、分析和纠正错误的过程。而测试（testing）是由人工或自动方法来执行或评价软件系统或软件子系统的过程，以验证是否满足规定的需求，或识别出期望的结果和实际结果之间有无差别。因此，测试的任务是尽可能多地发现软件中的缺陷和错误，而排错的任务是进一步诊断和改正程序中潜在的缺陷和错误。一般来说，排错有两类活动：其一是确定程序中可疑缺陷或错误的确切性质和位置；其二，是对程序（设计、编码）进行修改，从而排除这个潜在的缺陷或错误。

（2）验证（verification）有3个含义：其一，确定软件生存周期中的一个给定阶段的产品是否达到前阶段确立的需求的过程；其二，程序正确性的形式证明，即采用形式理论证明程序符合设计规约规定的过程；其三，评审、审查、测试、检查、审计等各类活动，或对某些项处理、服务或文件等是否和规定的需求相一致进行判断和提出报告。

（3）确认（validation）是在软件开发过程结束时，对软件进行评价，以确认它和软件需求是否相一致的过程，其目的是想证实在一个给定的外部环境中软件的正确性。确认一般包括需求规约的确认和程序的确认，而程序的确认又分为静态确认和动态确认。静态确认不在计算机上实际执行程序，通过人工分析或程序正确性证明来证实程序的正确性。动态确认主要通过动态执行程序作分析，或者测试程序来检查其动态行为，以证实程序是否存在问题，这通常就叫确认测试。

4.软件测试方法分类

软件测试有各种分类方法，按照不同的分类原则有不同的分类结果。软件测试的分类原则可以有一下几种：

（1）按照软件测试的动、静态来分，有静态测试和动态测试。

在软件开发过程中，每产生一个文档，或每一个活动结束时产生的文档，都必须对文档进行测试，静态测试通过了，则该过程或活动才结束，开发工作取得了进展，可以进入下一个阶段或活动，对这种静态测试也叫做评审。

动态测试就是通过运行程序来检验程序的动态行为和运行结果的正确性。运行程序并非动态测试的目的，通过运行来检验程序是否正确才是动态测试的目的。动态测试必须具备测试用例，有时还需要具备驱动程序、桩模块和测试监视代码。

（2）按照软件层面来分，美国国家宇航局建议将软件评审的内容分两个层面来进行，即技术评审和管理评审。

技术评审的任务：建立软件配置管理基线；提出并解决技术问题，审查技术工作；评价项目的状态，判明有关技术问题的近期、长期风险，并加以讨论；在技术代表的权限内，达成已判明风险的转移策略；标识呈交给管理人员讨论的风险要素和有关问题；确保用户和软件开发技术人员之间的交流畅通。

管理评审的任务：报告上级管理部门该项目的状态、所采取的方针、所达成的技术协议，以及软件产品进展的总体情况；解决技术评审不能解决的问题；就技术评审不能解决的近期、长期风险可达成的转移战略；鉴别并解决管理方面的问题以及技术评审没有提出的风险；征得用户的同意和各方认可以便及时完成。

（3）按照软件开发过程的内、外进行分类

①软件开发过程中的测试。按软件开发过程中所处的阶段（或活动）及其作用来分，有单元测试、集成测试、系统测试、验收测试。软件开发过程中的测试，大部分是开发单位自行完成的。当然，也可交第三方软件测试机构执行，但往往是系统测试和验收测试。有时，这种测试，因为不是由用户进行的，又称α测试。

②软件产品测试。其测试对象是产品化或正在产品化的软件。这种测试的内容包含范围很广，通常由第三方软件测试机构完成。

通常的软件产品测试：功能测试、性能测试、β测试（用户测试）

专门的软件产品测试：可靠性测试、标准符合性测试、互操作性测试、安全性测试、强度测试。

（4）按照测试用例所依据的信息来源，测试方法可以分为以下几种：

①以程序为基础的测试。通过对程序的分析形成测试用例，并以程序被执行的程度来判断测试是否充分，这就是“白盒法”。

②以需求规约和需求描述为基础的测试。通过分析软件的需求描述和需求规约形成测试用例，并根据需求描述和需求规约所约定的功能和性能是否得到了充分的检验来判断测试是否充分。这就是“黑盒法”。

③程序和需求相结合的测试。综合考虑需求和实现形成测试用例。

④以接口为基础的测试，仅仅依靠软件与其运行环境之间的接口形成测试用例，随机测试就是一种以接口为基础的测试方法。

（5）按照判断测试的充分性，测试方法可分为以下几种。

①结构性测试，旨在充分覆盖程序结构，并以程序中的某类成分是否都已得到测试为依据，来判断测试的充分性，如语句覆盖是一种结构性测试。

②排错性测试，旨在排除程序中潜在缺陷的可能性，并根据测试用例集成排除软件潜在缺陷可能性的能力判断测试的充分性。

③分域测试，通过对软件的需求和实现进行分析，将软件的输入空间划分成一系列子空间，然后在每一个子空间内选择一个或多个测试数据。

④功能测试，根据软件所需的功能和所实现的功能，形成测试用例，分析测试的充分性。

（6）按照软件测试的完整性，Shooman从程序结构和测试覆盖程度分为如下几种。

①完全性和连续性测试：要求程序中的所有指令至少执行一次（100%的语句覆盖）

②图路径测试：所有图路径至少执行一次（100%的图路径覆盖）

③程序路径测试：所有程序路径至少执行一次（100%的程序路径覆盖）

④穷举测试：对输入参数的所有值执行所有程序路径，或对输入参数的所有值和所有输入序列，以及初始条件的所有组合，执行所有程序路径。

5.软件错误的分级

在软件产品开发中,不论哪一阶段产生的缺陷和错误,其最后的表现就是:软件产品达不到用户的要求,由于存在软件错误,使之不能有效地完成规定的任务.

软件测试的任务,就在于找出或发现软件中存在的错误(假如错误存在的话).由于错误的性质不同,危害性也不同.软件测试如果能发现软件中危害性大的错误(如果存在的话),那么该软件测试的价值就越高.一般将软件错误分为5级。

（1）第1级错误：不能满足软件需求，基本功能未完全实现，危及人员或设备安全的错误。

（2）第2级错误：不利于完全满足软件需求或基本功能的实现，并且不存在可以变通的解决办法（重新装入或重新启动该软件不属于变通解决办法）。

（3）第3级错误：不利于完全满足软件需求或基本功能的实现，但却存在合理的、可以变通的解决办法（重新装入或重新启动该软件不属于变通解决办法）。

（4）第4级错误：不影响完全满足软件需求或基本功能的实现，但有不便于操作员操作的错误。

（5）第5级错误：不属于第1~4级错误的其他错误。

第二章 软件测试实质

2.1 软件测试的基本概念
2.2 软件故障
2.3 测试的复杂性与经济性
2.4 测试的充分性问题
2.5 测试原则
2.6 停止测试的标准

第二章 软件测试实质

针对测试人员，测试的最好定义是：
测试是为了发现故障而执行程序的过程。

这一定义非常明确地提出了
软件测试以发现故障为目的。

2.1 软件测试的基本概念
2.2 软件故障
2.3 测试的复杂性与经济性
2.4 测试的充分性问题
2.5 测试原则
2.6 停止测试的标准

IEEE提出的软件工程标准术语中给软件测试下的定义是：
“使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别”。

该定义包含了两方面的含义：
1）是否满足规定的需求
2）是否有差别

这一定义非常明确地提出了软件测试
以检验软件是否满足需求为目标。

第二章 软件测试实质

一般来说，
“成功”表示达到了某种目的，而“失败”则表示令人失望或事不如愿。

但在测试中，由于查不出故障的测试浪费了大量的时间和人力，
因此使用“成功”这个词就显得不够恰当了。

事实证明，发现故障是一个有价值的工作。

发现故障的测试是成功的测试，
而使程序产生正确结果的测试称为失败的测试。

第二章 软件测试实质

软件测试主要涉及5方面的问题：

- 谁来执行测试?
- 测试什么?
- 什么时候测试?
- 怎样进行测试?
- 测试完成的标准是什么?

第二章 软件测试

故障(fault)、失效(failure)、
错误(error)、缺陷(defect)、隐错(bug)、
过失(mistake)、异常（anomaly），
这些术语常用来描述软件失败时的现象。

2.1 软件测试的基本概念
2.2 软件故障
2.3 测试的复杂性与经济性
2.4 测试的充分性问题
2.5 测试原则
2.6 停止测试的标准

我们采用IEEE制定的标准术语。

- 错误(error)——人是会犯错误的。一个很接近的同义词是过失（mistake）。过失是人犯下的，是人做的一件错事或人为产生的一个不正确结果。
- 故障(fault)——故障是错误的结果(可能导致失效)。更精确地说，故障是错误的表现。与故障很接近的一个同义词是缺陷(defect)，
- 失效(failure)——故障(例如崩溃)引起的结果(表现)。

所有的错误都是要付出代价的。

统计资料表明，
在软件开发的不同阶段进行改动需要付出的代价完全不同。
后期改动的代价比前期进行相应修改要高出2~3个数量级。

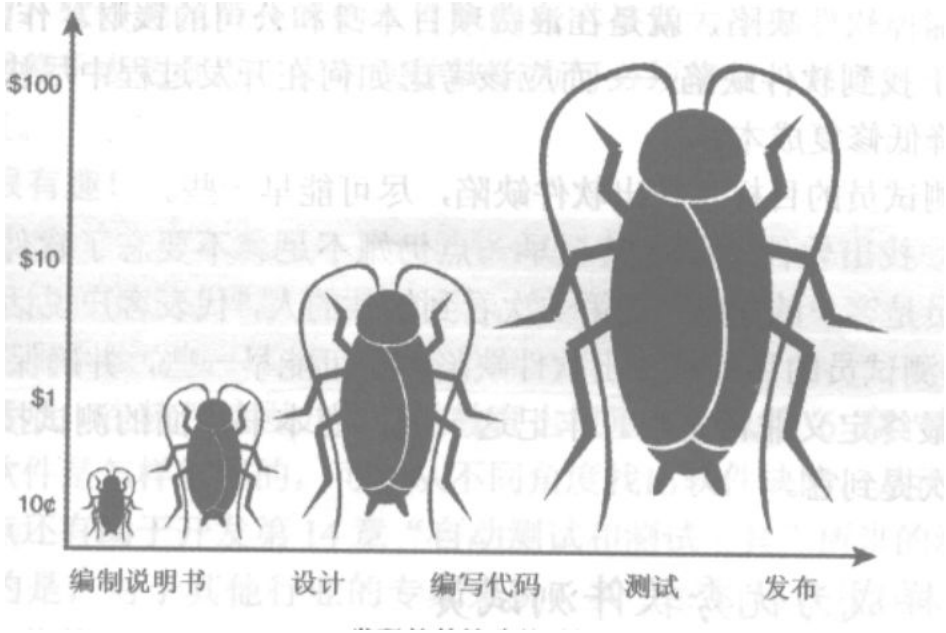


图2-4 软件故障的修复费用

第二章 软件测试实质

2.1 软件测试的基本概念

2.2 软件故障

2.3 测试的复杂性与经济性

2.4 测试的充分性问题

2.5 测试原则

2.6 停止测试的标准

穷举测试工作量太大，实际上是行不通的，
这就注定了一切实际测试都是不彻底的。
因此，软件测试的一个基本问题是经济学问题。
软件测试的总目标是充分利用有限的人力和物力资源，
高效率、高质量地完成测试。

第二章 软件测试实质

2.1 软件测试的基本概念
2.2 软件故障
2.3 测试的复杂性与经济性
2.4 测试的充分性问题
2.5 测试原则
2.6 停止测试的标准

不充分的测试是愚蠢的，而过度的测试则是一种罪孽。

停止测试的标准？

第二章 软件测试实质

2.1 软件测试的基本概念
2.2 软件故障
2.3 测试的复杂性与经济性
2.4 测试的充分性问题
2.5 测试原则
2.6 停止测试的标准

一些至关重要的测试原则或方针，可以视为软件测试和软件开发的“交通规则”或者“生活法则”。

例如：
- 完全测试程序是不可能的
- 软件测试是有风险的
- 存在的故障数量与发现的故障数成正比
- 杀虫剂现象
- 应避免测试自己编写的程序

第三章 软件测试策略

3.1 软件开发模型

3.2 软件测试过程

3.3 黑盒测试与白盒测试

3.4 静态测试与动态测试

3.5 验证测试与确认测试

第三章 软件测试策略

3.1 软件开发模型
3.2 软件测试过程
3.3 黑盒测试与白盒测试
3.4 静态测试与动态测试
3.5 验证测试与确认测试

另一种常用的软件开发模型是螺旋模型
该模型加入了风险分析在内。

瀑布模型多年来广泛流行，
它在支持结构化软件开发、控制软件开发的复杂性、
促进软件开发工程化等方面起着显著作用。

软件开发模型是软件开发过程、活动和任务的结构框架，
能够清晰、直观地表达软件开发的全部过程，
明确规定要完成的主要活动和任务，
是软件项目开发的基础。

第三章 软件测试策略

3.1 软件开发模型

3.2 软件测试过程

3.3 黑盒测试与白盒测试

3.4 静态测试与动态测试

3.5 验证测试与确认测试

软件开发是一个自顶向下，逐步细化的过程。

软件测试则是依相反顺序的自底向上，逐步集成的过程。

低一级的测试为上一级的测试准备条件。

软件测试过程包括:测试资料的收集与整理；熟悉所要测试的软件；测试方案的制定；测试计划的编写；测试实例的设计与编写；测试准备工作；测试操作；软件缺陷记录及报告；修改充实测试实例及测试计划书；测试自动化程序的编写；修改、增加测试程序

第三章 软件测试策略

黑盒测试和白盒测试是
两类广泛使用的软件测试方法。

3.1 软件开发模型
3.2 软件测试过程
3.3 黑盒测试与白盒测试
3.4 静态测试与动态测试
3.5 验证测试与确认测试

黑盒测试的基本观点是
将被测程序看作一个打不开的黑盒，
测试人员在完全不考虑程序内部结构和内部特性的情况下，
只依靠被测程序输入和输出之间的关系，或程序的功能来设计测试用例。

白盒测试将被测程序看作一个打开的盒子，
测试人员根据其内部构造设计测试用例。

第三章 软件测试策略

3.1 软件开发模型
3.2 软件测试过程
3.3 黑盒测试与白盒测试
3.4 静态测试与动态测试
3.5 验证测试与确认测试

静态测试是指不利用计算机运行被测试的程序，通过其它手段达到检测的目的，是对被测程序进行特性分析的一些方法的总称。

动态测试则是指通常意义上的测试——通过运行和使用被测程序，
发现软件故障，以达到检测的目

经验表明，使用静态测试可以发现大约**30%**到**70%**的逻辑设计和编码错误。

第三章 软件测试策略

3.1 软件开发模型
3.2 软件测试过程
3.3 黑盒测试与白盒测试
3.4 静态测试与动态测试
3.5 验证测试与确认测试

确认测试则通过运行代码来完成，是在开发过程中或结束时，对系统或部件进行评估以确定其是否满足需求规格的过程。

验证测试针对开发过程中的任何中间产品进行，是为确定某一开发阶段的产品是否满足在该阶段开始时提出的要求而对系统或部件进行评估的过程。

验证就是对诸如需求规格说明，设计规格说明和代码之类的产品进行评估、审查和检查的过程，属于静态测试。

确认是“基于计算机的测试”过程，属于动态测试。

第四章 黑盒测试

4.1 三个被测程序

4.2 等价类划分测试

4.3 边界值分析

4.4 决策表测试

4.5 其它黑盒测试方法

第四章 黑盒测试

4.1 三个被测程序

4.2 等价类划分测试

4.3 边界值分析

4.4 决策表测试

4.5 其它黑盒测试方法

在第四章和第五章，我们将采用三个例子来说明各种单元测试方法。
这三个例子分别是：

- 一个是三角形问题；
- 一个是逻辑比较复杂的NextDate函数；
- 一个是有代表性的雇佣金问题。

这三个例子合在一起，
足可以说明软件测试人员在单元级别上可能会遇到的大多数问题。

第四章 黑盒测试

等价类划分把程序的输入域划分成若干个互不相交的一组子集，即等价类。
等价类由等价关系决定，
因此等价类中的元素有一些共同的特点：
如果用等价类中的一个元素作为测试数据进行测试不能发现程序中的故障，
那么使用集合中的其它元素进行测试也不可能发现错误。
也就是说，对揭露程序中的故障来说，等价类中的每个元素是等效的。

例如，在三角形问题中，
如果我们选择三元组（5，5，5）作为测试用例的输入，
可以判定这是一个等边三角形的测试用例。
在程序测试中能暴露某个程序故障，
那么以(6，6，6)或（100，100，100）作为测试数据也能发现这个故障。
因此，这些测试用例是冗余的。

在考虑等价类时，应注意区别两种不同的情况：

- 有效等价类：有效等价类是指对程序规格说明，是有意义的，合理的输人数据所构成的集合。

利用有效等价类，可以检验程序是否实现了规格说明预先规定的功能和性能。

- 无效等价类：无效等价类是指对程序规格说明，是不合理或无意义的输入数据所构成的集合。

利用无效等价类，可以检查程序功能和性能的实现是否有不符合规格说明要求的地方。

在设计测试用例时，
应同时考虑有效等价类和无效等价类测试用例的设计。
我们希望用最少的测试用例，覆盖所有的有效等价类。
但对每一个无效等价类，设计一个测试用例来覆盖它。

此外，在设计测试用例时，我们应意识到：
预期结果也是测试用例的一个必要组成部分，
对采用无效输入的测试也是如此。

第四章 黑盒测试

4.1 三个被测程序
4.2 等价类划分测试
4.3 边界值分析
4.4 决策表测试
4.5 其它黑盒测试方法

人们从长期的测试工作经验得知，
大量的故障发生在输入或输出范围的边界上，
而不是在输入范围的内部。

我们知道：当满足a+b>c、a十c>b及b+c>a才能构成三角形。
但如果把三个不等式的任何一个大于号“>”错写成大于等于号“≥”，
那就无法构成三角形了。

使用边界值分析方法设计测试用例，首先应确定边界情况。
输入等价类与输出等价类的边界，就是应着重测试的边界情况。

通过使所有变量取正常值，
只使一个变量分别取最小值、略高于最小值、略低于最大值和最大值，
有两输入变量的程序F的边界值析测试用例是：

{<X_{1nom}，X_{2min}>，<X_{1nom}，X_{2min+}>，<X_{1nom}，X_{2nom}>，<X_{1nom}，X_{2max}>，
<X_{1nom}，X_{2max-}>，<X_{1min}，X_{2nom}>，<X_{1min}，X_{2min+}>，<X_{1min}，X_{2max}>，
<X_{1min}，X_{2max+}>}

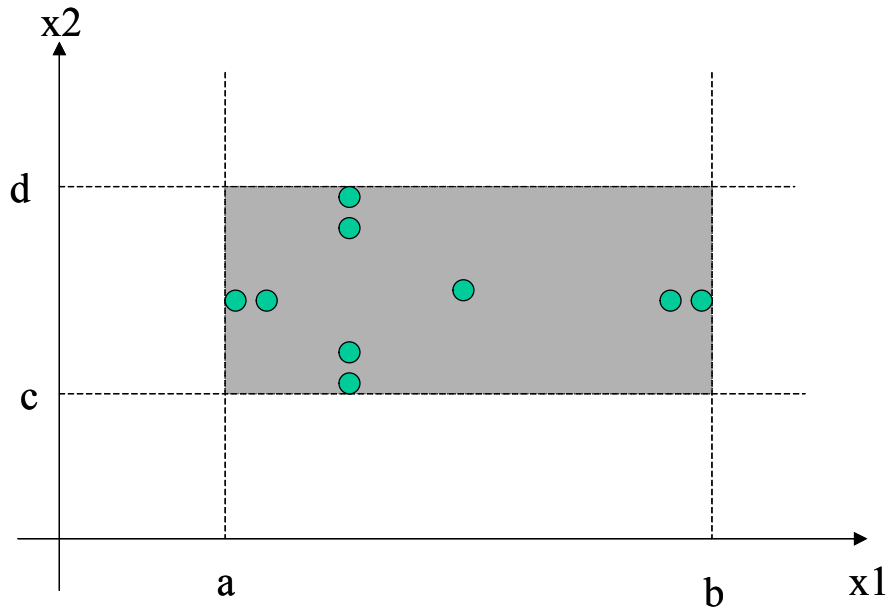


图4-5 有两个变量程序的边界值分析测试用例

次边界条件：

有些边界在软件内部，用户几乎看不到，
但是软件测试仍有必要对这些边界条件进行检查。
如2的幂次方和ASCII码就是这样的两个例子。

例如，假设某种通讯协议支持256条命令，为了提高数据传输效率，
通讯软件总是将常用的信息压缩到一个很小的单元中，
必要时再扩展为大一些的单元。
比如将常用的15条命令压缩为一个半字节数据，
在遇到第16到256之间的命令时，软件转而发送一个一字节的命令。
用户只知道可以执行256条命令，
并不知道软件根据半字节/字节边界执行了不同的计算和操作。
为了覆盖任何可能的2的幂次方次边界，
还要考虑临近半字节边界14，15和16，
以及临近字节边界254，255和256的测试用例。

第四章 黑盒测试

4.1 三个被测程序

4.2 等价类划分测试

4.3 边界值分析

4.4 决策表测试

4.5 其它黑盒测试方法

决策表很适合于处理针对不同逻辑条件组合值，分别执行不同的操作这类问题。

条件桩部分列出了问题的所有条件

动作桩则给出了问题规定的可能采取操作

表4-10 三角问题的决策表

c3: a,b,c 构成一个三角形?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2: a=b?	—	Y	Y	Y	Y	N	N	N	N
c3: a=c?	—	Y	Y	N	N	Y	Y	N	N
c4: b=c?	—	Y	N	Y	N	Y	N	Y	N
a1: 非三角形	X								
a2: 一般三角形									X
a3: 等腰三角形					X		X	X	
a4: 等边三角形		X							
a5: 不可能			X	X		X			

为了使用决策表设计测试用例，可以把条件解释为输入，把行动解释为输出。
决策表最突出的优点是，
它能把复杂的问题按各种可能的情况一一列举出来，
简明而易于理解，也可避免遗漏。
因此利用决策表可以设计出完整的测试用例集合。

第四章 黑盒测试

4.1 三个被测程序
4.2 等价类划分测试
4.3 边界值分析
4.4 决策表测试
4.5 其它黑盒测试方法

因果图方法适合于描述对于多种条件的组合，产生多个相应动作的测试方法，能够帮助我们按一定步骤，高效率地选择测试用例，以检测程序输入条件的各种组合情况。

因果图方法最终生成的是判定表。

1．分析程序规格说明的描述中，哪些是原因，哪些是结果。

原因常常是输入条件或是输入条件的等价类，而结果则是输出条件。

2．找出原因与结果之间，原因与原因之间的对应关系，

并将其表示成连接各个原因与各个结果的“因果图”。

3．把因果图转换成判定表。

第四章 黑盒测试

4.1 三个被测程序

4.2 等价类划分测试

4.3 边界值分析

4.4 决策表测试

4.5 其它黑盒测试方法

特殊值测试：

测试人员使用其领域知识、使用类似程序的测试经验等信息开发测试用例时，常常使用特殊值测试。这种方法不使用测试策略，只根据“最佳工程判断”来设计测试用例。

因此，特殊值测试特别依赖测试人员的能力。

故障猜测法

人们靠经验和直觉猜测程序中可能存在的各种软件故障，从而有针对性地编写检查这些故障的测试用例。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

第五章 白盒测试

5.1 逻辑覆盖

5.2 路径分析

5.3 数据流测试

5.4 符号测试

5.5 域测试策略

5.6 程序变异

5.7 程序插装

逻辑覆盖要求对被测程序的结构作到一定程度的覆盖。

由于覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖及路径覆盖。

使得

断的

假分支

使得程序中每个判断的每个条件的可能取值至少被执行一次。我们来看一段小程序： if (a>0 and b!=a) c=b/a;else c=a;

以此条件覆盖为测试依据，则可建立以下几个测试实例

a>0, b!=a; a<0, b=a; a<0, b!=a; a<0, b=a; a=0, b!=a; a=0, b=a

使得判断中每个条件的所有可能取值至少被执行一次，同时每个判断的所有可能判断结果也至少被执行一次。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

讨论路径数目的计算方法，分析程序中到底有多少条路径?有多少条线形独立路径数

如果程序中出现了多个判断和多个循环，可能的路径数目将会急剧增长，以至实现路径覆盖不可能的。
实际上我们可以做到的只是有选择地测试程序中某些有代表的性路径。
独立路径选择和Z路经覆盖是两种常用的路经覆盖方法。

独立路径是指从程序入口到出口的多次执行中，每次至少有一个语句是新的，未被重复的。

限制循环的次数。
我们只考虑循环执行一次和零次两种情况，这种简化循环意义下的路径覆盖为Z路径覆盖。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试*
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

数据流测试是指关注变量定义点和使用(或引用)点的一种结构测试方式，

数据流测试是指在不运行被测程序的前提下，对变量的定义、引用进行分析，以检测数据的赋值与引用之间是否出现了不合理的现象，
如引用未赋值的变量，对以前未曾引用变量的再次赋值等数据流异常现象。

定义／使用测试是一种常用的数据流测试方法。

- 定义/使用路径，记做du-path

如果对某个变量v ∈V，存在一个定义、使用结点对，
即DEF(v，m)和USE(v，n)，使得变量v在结点m处被定义，
在结点n处被使用，则从m到n的结点序列称为一条定义/使用路径.

- 定义明确路径，记做dc-path

如果对某个变量v ∈V，存在一个定义、使用结点对，
即DEF(v，m)和USE(v，n)，使得变量v在结点m处被定义，
在结点n处被使用，并且从m到n的结点序列中没有其他结点对变量v进行过定义，则从m到n的结点序列称为一条定义明确路径.

定义/使用路径和定义明确路径描述了
变量从被定义点到被引用点数据流向。
不是定义明确的定义/使用路径，很可能是潜在问题的所在。
所以我们应特别关注这些定义/使用路径。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

符号测试的基本思想是允许程序的输入不仅可以是具体的数值数据，而且还可以包括符号值。

符号测试方法在使用中会遇到一些问题，
比如在遇到循环、过程调用、动态数据结构、
数组和指针处理时，符号执行实现困难。

在执行程序过程中以符号计算代替了普通执行中的数值计算，
所得到的结果自然是符号公式或是符号谓词。
更明确地说，普通测试执行的是算术运算，符号测试执行则是代数运算。
一次符号测试的结果代表了一大类普通测试的运行结果，
实际上是证明了程序接受此类输入，所得输出是正确的，还是错误的。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

域测试的“域”指的是程序的输入空间。
自然，每个被测程序都有一个输入空间，而输入空间又可以分为不同的子空间。
子空间的划分是由程序中分支语句中的谓词决定的。

域测试策略在分析输入域的基础上，
对每一子域的每一谓词边界，
通过选取适当的测试点，
对谓词边界附近的处理进行检测。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

程序变异是一种不同于黑盒测试和白盒测试的测试方法，它属于错误驱动测试，适用于某类特定的程序故障。

假设对程序P进行了微小改动而得到程序MP，则MP也是一个程序，称为P的一个变异体(mutant)。

显而易见，如果程序P是正确的，MP也是一个几乎正确的程序。如果P不正确，则P的某一个变异体MP可能是正确的。

程序变异假定，被测程序由经验丰富的程序人员编写，这样的程序即使不正确，残留在程序中的错误也不是那些重大错误，而是一些难以发现的小错误。
和正确软件相比，表现为一个符号或几个符号的错误。
程序变异的目标就是查出这些简单的错误及其组合。

第五章 白盒测试

5.1 逻辑覆盖
5.2 路径分析
5.3 数据流测试
5.4 符号测试
5.5 域测试策略
5.6 程序变异
5.7 程序插装

程序插装是一种借助于往被测程序中插入操作来实现测试目的方法，在软件测试中有着广泛的应用。

第六章 集成测试与系统测试

6.1 集成测试

6.2 系统测试

第六章 集成测试与系统测试

6.1 集成测试

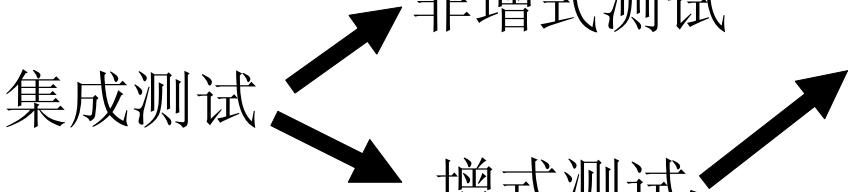
6.2 系统测试

一些模块单独能够工作，并不能保证连接起来也能正常工作。

程序在某些局部反映不出的问题，在全局上很可能暴露出来，影响功能的发挥。

集成测试是将多个模块组合在一起进行测试的过程。

独立地测试程序的每个模块，然后再把它们组合成整个程序的集成测试方法。



自顶向下测试

从主控模块开始，按照软件的控制层次结构，逐步把各个模块集成在一起。

自底向上测试

从最下层的模块开始，按照程序的层次结构，逐渐形成完整的整体。

把下一个待测试的模块组合到已经测试过的那些模块上去，再进行测试。

第六章 集成测试与系统测试

6.1 集成测试

6.2 系统测试

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，很接近我们日常测试实践。

系统测试目标不是找出软件故障，
而是要证明系统的性能。
比如，确定程序是否满足性能需求；
确定系统是否满足可靠性要求等等。

系统测试

- 性能测试
- 强度测试
- 安全性测试
- 恢复测试
- 可靠性测试
- 配置测试
- 可用性测试
- 兼容性测试
- 文档资料测试
- 网站测试

说明在一定工作负荷和格局分配条件下，响应时间及处理速度等特性

检查系统能力的最高实际限度，即软件在一些超负荷情况下的运行情况。

检查系统对非法侵入的防范能力，验证安装在系统内的保护机构是否确实能够对系统进行保护。

恢复测试的主要目的是检查系统的容错能力。

检测系统的平均无故障时间是否超过规定时限；因故障而停机的时间在一年中应不超过多少时间。

配置测试是用各种硬件和软件平台以及不同设置检查软件操作的过程，以保证测试的软件可以使用尽量多样化的硬件组合。

可用性测试检测用户使用软件是否满意

检测软件之间能否正确地交互和共享信息，

软件测试不限于仅测试软件，保证文档的正确性也是软件测试的职责。

第七章 验证测试和确认测试

7.1 验证的基本方法

7.2 验证活动

7.3 通用代码审查单

7.4 确认测试

第七章 验证测试和确认测试

7.1 验证的基本方法

7.2 验证活动

7.3 通用代码审查单

7.4 确认测试

验证是对软件产品进行人工检查或评审。
验证的基本方法有：软件审查、走查、伙伴检查等。

软件审查以会议的形式进行 。
软件审查的目标是：

- 收集数据，发现软件故障。
- 交流，信息。

走查的目标是：
熟悉材料、发现软件故障。

第七章 验证测试和确认测试

7.1 验证的基本方法

7.2 验证活动

7.3 通用代码审查单

7.4 确认测试

验证活动是测试生存周期中的一个阶段，包括需求验证、功能设计验证、详细设计验证和代码验证。

在每一类的验证活动中，都要考虑以下问题：

- 使用的验证方法(审查、走查、伙伴检查等)
- 产品中要验证的和不要验证的范围
- 没有验证的部分所承担的风险。
- 需要优先进行验证的范围
  - 资源、进度、工具和责任等

第七章 验证测试和确认测试

7.1 验证的基本方法

7.2 验证活动

7.3 通用代码审查单

7.4 确认测试

审查单是验证测试的重要工具。

在通用代码审查单的基础上开发定制出适合某种目的或项目的审查单。

第七章 验证测试和确认测试

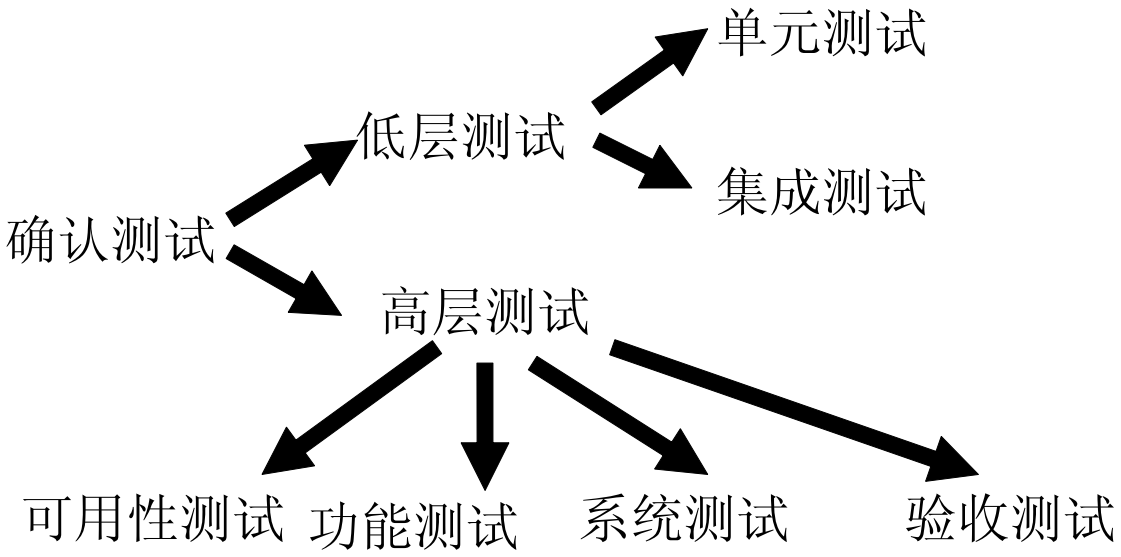
7.1 验证的基本方法

7.2 验证活动

7.3 通用代码审查单

7.4 确认测试

确认测试以需求规格说明中的规定作为检验尺度，在开发过程中或结束时，对系统或组成部分进行评估。



三种确认测试策略：

- 基于需求的测试：基于需求的测试必须采用黑盒测试策略，

在不知道内部设计规格说明或代码的情况下对用户需求进行测试。

- 基于功能的测试：基于功能的测试应该采用黑盒策略。

基于功能的测试常采用等价类划分、边界值分析和故障猜测等方法设计测试用例。

- 基于内部的测试：基于内部的测试只能采用白盒测试策略。

一但采用白盒测试，便可通过一系列的技术确保系统的内部各部分获得充分的测试并且达到足够的逻辑覆盖。

第八章 测试计划与测试文档

8.1 测试计划
8.2 软件测试文档
8.3 主测试计划
8.4 验证测试计划
8.5 确认测试计划
8.6 测试评估
8.7 用户手册
8.8 IEEE/ANSI测试文档概述
8.9软件生存周期各阶段的测试任务与可交付的文档

第八章 测试计划与测试文档

8.1 测试计划

8.2 软件测试文档

8.3 主测试计划

8.4 验证测试计划

8.5 确认测试计划

8.6 测试评估

8.7 用户手册

8.8 IEEE/ANSI测试文档概述

8.9软件生存周期各阶段的测试任务与可交付的文档

高效率的测试是经过计划的。
成功的测试需要有一定的方法，
包括条例、结构、分析和度量。

第八章 测试计划与测试文档

8.1 测试计划

8.2 软件测试文档

8.3 主测试计划

8.4 验证测试计划

8.5 确认测试计划

8.6 测试评估

8.7 用户手册

8.8 IEEE/ANSI测试文档概述

8.9软件生存周期各阶段的测试任务与可交付的文档

测试文档的编制是
测试工作规范化的一个重要组成部分。

包括以下几个内容：

- 测试计划：该计划描述测试活动的范围、方法、资源和进度，其中规定了被测试的对象，被测试的特性、应完成的测试任务、人员职责及风险等。
- 测试设计规格说明：该说明详细描述测试方法，测试用例以及测试通过的准则等。
- 测试用例规格说明：该说明描述测试用例涉及的输入、输出，对环境的要求，对测试规程的要求等。
- 测试步骤规格说明：该说明规定了实施测试的具体步骤。
- 测试日志：该日志是测试小组对测试过程所作的记录。
- 测试事件报告：该报告说明测试中发生的一些重要事件。
- 测试总结报告：对测试活动所作的总结和结论。

第八章 测试计划与测试文档

8.1 测试计划

8.2 软件测试文档

8.3 主测试计划

8.4 验证测试计划

8.5 确认测试计划

8.6 测试评估

8.7 用户手册

8.8 IEEE/ANSI测试文档概述

8.9软件生存周期各阶段的测试任务与可交付的文档

制定主测试计划的目标是：规定测试活动的范围、方法、资源和进度；明确正在测试的项目、要执行的主要测试任务、每个任务的负责人，以及与计划相关的风险。

第八章 测试计划与测试文档

8.1 测试计划
8.2 软件测试文档
8.3 主测试计划
8.4 验证测试计划
8.5 确认测试计划
8.6 测试评估
8.7 用户手册
8.8 IEEE/ANSI测试文档概述
8.9软件生存周期各阶段的测试任务与可交付的文档

测试评估包括以下几个方面：
- 测试覆盖评估
- 软件故障评估
- 测试有效性评估

第九章 面向对象的软件测试

9.1 面向对象的概念

9.2 面向对象的测试与传统软件测试的区别

9.3 面向对象软件测试

9.4 类测试

9.5 面向对象的集成测试

第九章 面向对象的软件测试

从测试的角度来考虑对象、消息、接口、类、继承、动态绑定等面向对象程序设计的基本概念。

例如：操作和方法(或者成员函数)，对于大多数程序员来说并没有多大的区别，然而对于测试人员来说，区别是显然的。

因为测试一个操作的步骤在某种程度上与测试一个方法的步骤不同，前者是类声明的一部分同时也是操纵对象的一种手段，后者则是实现某个操作的一些代码。

第九章 面向对象的软件测试

9.1 面向对象的概念

9.2 面向对象的测试与传统软件测试的区别

9.3 面向对象软件测试

9.4 类测试

9.5 面向对象的集成测试

面向对象软件特有的继承、封装和动态绑定以及和传统软件之间的差异给面向对象的软件测试提出了一系列新的问题。

例如：在传统的面向过程的程序中，对于函数

y=Function(x);

我们只需要考虑函数Function()的行为特征，

而在面向对象的程序中，我们不得不同时考虑基类函数Base::Function()和继承类函数Derived::Function()的行为特征。

第九章 面向对象的软件测试

面向对象的软件测试主要体现在面向对象单元测试和面向对象集成测试中。

面向对象单元测试针对程序内部具体单一的功能模块进行测试。

面向对象集成测试则针对系统内部的相互服务进行测试，
如成员函数间的相互作用，类间的消息传递等。

第九章 面向对象的软件测试

9.1 面向对象的概念

9.2 面向对象的测试与传统软件测试的区别

9.3 面向对象软件测试

9.4 类测试

9.5 面向对象的集成测试

在测试类的功能实现时，应该首先保证类成员函数的正确性。

面向对象编程的固有特性使得对成员函数的测试，不完全等同于传统的函数或过程测试。还需要考虑：

1．继承的成员函数是否都不需要测试？

对父类中已经测试过的成员函数，两种情况需要在子类中重新进行测试：

- 继承的成员函数在子类中做了改动；
- 成员函数调用了改动过的成员函数。

2. 对父类的测试能否照搬到子类？

第十章 软件测试自动化和测试工具

10.1 测试与测试自动化

10.2 测试工具

10.3 常用测试工具简介

10.4 测试自动化和测试工具的好处

10.5 测试自动化和测试工具存在的问题

第十章 软件测试自动化和测试工具

10.1 测试与测试自动化

10.2 测试工具

10.3 常用测试工具简介

10.4 测试自动化和测试工具的好处

10.5 测试自动化和测试工具存在的问题

测试自动化希望通过自动化测试工具或其他手段，
按照测试工程师的预定计划进行自动的测试，
目的是减轻手工测试的劳动量，从而达到提高软件质量的目的。

测试自动化的目的在于发现老的软件故障，
而手工测试的目的在于发现新故障。

第十章 软件测试自动化和测试工具

10.1 测试与测试自动化

10.2 测试工具

10.3 常用测试工具简介

10.4 测试自动化和测试工具的好处

10.5 测试自动化和测试工具存在的问题

- 白盒测试工具
- 黑盒测试工具
- 测试设计和开发工具
- 测试执行和评估工具
- 测试管理工具
- 测试工具的选择

第十章 软件测试自动化和测试工具

10.1 测试与测试自动化

10.2 测试工具

10.3 常用测试工具简介

10.4 测试自动化和测试工具的好处

10.5 测试自动化和测试工具存在的问题

- Parasoft C++ Test测试工具简介
- 白盒工具--NuMega DecPartner Studio
- 黑盒测试工具—QACenter
- 数据库测试工具——TESTBytes 和ERwin Examiner
- 测试管理工具——TestDirector

第十章 软件测试自动化和测试工具

10.1 测试与测试自动化

10.2 测试工具

10.3 常用测试工具简介

10.4 测试自动化和测试工具的好处

10.5 测试自动化和测试工具存在的问题

测试自动化和测试工具能够通过较少的开销可以获得更彻底的测试，提高软件产品的质量。

但使用自动测试时，也会遇到许多问题，

因为工具毕竟是工具，在处理一些意外事件时，毕竟不如人灵活。

几个较为典型的静态测试工具

QAC

QAC是英国 Programming Reseach 公司开发的针对C语言的软件静态分析工具，通过对C语言程序进行分析找出语法使用中存在的问题，例如：危险的用法、过于复杂、不可移植、不易维护或者不符合本部门要求的编程规范等，它能够对这些编译器和其他开发工具不会察觉的隐藏问题发出警告。使用QAC可以明显地减少代码审查的时间，还可加深程序员对C语言特性的理解。如果在早期的开发阶段就注意到程序所存在的问题，代码的质量可以得到大幅度的提升，测试周期可以缩短。

- 分析C源程序，报告超过1100种潜在的问题，涉及C语言用法、危险结构、有关维护性和移植性的方面；
- 可以分析许多编译器中常见的C语言的扩展结构和非标准结构；
- 非常容易配置警告信息和报告；
- 可计算44种业界接受的度量标准包括 Cyclomatic Complexity（圈复杂度），静态路径统计数和Myer's interval 等，并且可以扩展成为公司特定的度量标准；
- 依据ISO标准产生报告；
- 可以扩展成为实施本公司特定需要的分析检查；
- 可进行多种可视化的输出，包括函数结构图、函数调用树、外部引用、文件包括关系和软件度量分析图等；
- 突出C和C++语言之间的移植性问题；
- 在线HTML帮助连接警告消息，包括替代解决办法；
- Windows和UNIX平台提供易于使用的GUI，并且可以集成到常用开发环境（如VC++、Tornado等）。
- 除了QAC外，Programming Reseach 公司还针对C++语言、Fortran语言等提供了相应的软件静态分析工具QAC++和QA Fortran等。

McCabe

McCabe IQ 是美国McCabe & Association 公司的产品，McCabe IQ 功能强大，包含McCabe Test、McCabe QA、McCabe Reengineering 等多种功能组件。美国国防部、美国海军武器系统和美国通用电子同时对McCabe软件进行了测试，证明McCabe 在软件的质量度量、预见软件错误和规划软件测试方面对软件人员会有非常好的帮助。

- McCabe QA是一个用于软件质量度量的功能组件。通过它可以计算被测软件的McCabe复杂度，并通过一个易理解的可视环境评估整个软件的质量，明确需要改进质量的区域。图形化的显示使得软件QA人员和软件开发人员有了交流的基础。McCabe QA产生程序级结构图（battle maps）和单元级流程图。程序级结构图（battle maps）中的方盒代表模块，不同颜色表示不同质量量度。红色模块大于用户定义的度极限，绿色小于度极限，黄色大于基本度极限而小于第二度极限。这些可视显示可以很容易地帮助用户发现问题。用户还可以利用McCabe QA来追踪软件质量的变化过程。用户在软件开发周期中抓拍软件的特殊点，并且把每个抓拍的点储存起来，这些信息用来绘出在开发周期中软件质量的变化趋势。管理者可以观察和追踪软件质量的变化过程，监督整个系统的复杂性和质量。
- McCabe Reengineering是一个软件再工程（逆向工程）的功能组件。它支持各种软件的再工程，包括对已有软件系统的维护，改变软件特性或移植到新的平台或结构中。利用此软件可以帮助我们识别代码中的冗余代码，进行软件维护和更改时的风险（risk）分析。

PolySpace

PolySpace C Verifier 是法国PolySpace开发的一种比较有特色的静态测试工具。它利用一种基于抽象解释的静态验证技术来静态地发现被测软件运行时的错误（run-time error）。它是一种非侵入式的和基于源程序代码的静态测试工具，可以无需修改和运行被测试软件就发现和检查出那些在未来的运行中可能出错的代码。利用PolySpace Verifier可以在代码审查和静态测试阶段确定被测软件的运行错误，并用不同的颜色将所有可能导致运行错误的软件代码标出来。

PolySpace Verifier可以自动检查下面的错误：

- 企图读未初始化的变量；
- 多线程应用中未保护数据的访问冲突；
- 对空指针和越界指针的引用；
- 对超界数组的访问；
- 非法类型转换（long to short,float to integer）；
- 非法的算术运算（如除零错误，负数开方）；
- 整数和浮点数的上溢出/下溢出；
- 不可达代码。

几个较为典型的动态测试工具

ASMTester

ASMTester是一个支持汇编语言软件的单元测试工具。该工具为软件开发和软件测试人员提供了一个方便、有效的手段，在不需要任何目的机、硬件仿真器或数据发生器的前提下，高效率、低成本地完成对8031/51、8086/88、8096/98、TMSC3x、TMSC6x、1750A等汇编语言软件的单元测试工作。

该工具为用户进行单元测试提供以下功能：

- 被测模块和测试用例的管理与维护
- 提供每个被测单元（模块）动态测试覆盖率指标
- 提供每个被测单元动态测试时的性能指标
- 变量追踪
- 测试结果的自动比对功能
- 完善的嵌入式硬件环境模拟机制
- 自动产生单元测试结果记录单并可打印

Cantata

Cantata/ Cantata++是英国IPL公司开发的一个面向源代码的测试分析工具，可以支持对软件的静态分析和动态测试。

在动态测试方面，该工具为测试的说明、执行、归档、重用和重复动态测试提供一个形式上的框架。它具有以下特点：

- 可以根据用户定义的Test Case Definition 自动生成测试脚本；
- 自动生成桩模块模拟被测模块的函数调用。用户可以传递参数给桩模块，并设置桩模块的返回参数；
- 通过对测试执行过程的监视，得到测试覆盖信息（包括语句、分支、条件、分支/条件、条件组合等不同级别的覆盖信息）和程序执行时间分布情况的信息；
- Cantata支持C语言，Cantata++支持C++语言。

LoadRunner

LoadRunner是美国Mercury Interactive公司开发的一种预测系统行为和性能的负载测试工具。通过以模拟上千万用户，实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner能够对整个企业网络结构进行测试。通过使用LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。

CodeTEST

CodeTEST是美国AMC公司采用专利插桩技术开发出的专为嵌入式软件开发与测试人员使用的性能分析与测试工具。作为专为嵌入式系统软件测试而设计的工具套件，CodeTEST广泛应用于嵌入式软件的在线动态测试。CodeTEST采用硬件辅助软件的系统构架和源代码插桩技术，用适配器或探针，直接连接到被测试系统，从目标板总线获取信号，为跟踪嵌入式应用程序、分析软件性能、统计软件的测试覆盖率以及监测内存的动态分配等提供了一个实时在线的高效率解决方案。

Testbed

Testbed是英国LDRA公司的产品，其动态测试功能通过Tbrun模块实现。Tbrun具有自动生成测试脚本与打桩处理的功能，测试脚本编译后与被测软件连接在一起，从而产生一可执行程序运行于目标系统或主机系统，并产生测试结果文件。

第十一章 软件质量保证

11.1 软件质量保证

11.2 软件测试管理技术

11.3 测试的组织方式

11. 4 软件过程成熟度CMM

11.4 ISO 9000

第十一章 软件质量保证

11.1 软件质量保证

11.2 软件测试管理技术

11.3 测试的组织方式

11. 4 软件过程成熟度CMM

11.4 ISO 9000

软件质量保证的主要职责是检查和评价当前软件开发过程，并设法达到防止软件故障出现的目标。

第十一章 软件质量保证

11.1 软件质量保证

11.2 软件测试管理技术

11.3 测试的组织方式

11. 4 软件过程成熟度CMM

11.4 ISO 9000

建立软件测试管理体系的主要目的是
确保软件测试在软件质量保证中发挥应有的关键作用。

第十一章 软件质量保证

11.1 软件质量保证

11.2 软件测试管理技术

11.3 测试的组织方式

11. 4 软件过程成熟度CMM

11.4 ISO 9000

软件能力成熟度CMM现已成为一个行业标准模型，
用来定义和评价软件公司开发过程的成熟度，
为提高软件质量提供指导。

第十一章 软件质量保证

11.1 软件质量保证

11.2 软件测试管理技术

11.3 测试的组织方式

11. 4 软件过程成熟度CMM

11.4 ISO 9000

ISO 9000定义了一套关于质量管理和质量保证的标准，
有助于公司一致地交付符合客户质量要求的产品或服务。

软件质量

- 我们先来看软件质量的定义：反映软件系统或软件产品满足明确或隐含需求的能力有关的特性总和。其含义有四：其一，能满足给定需要的性质和特性的全体；其二，具有所期望的各种属性的组合程度；其三，顾客和用户觉得能满足其综合期望的程度；其四，确定软件在使用中将满足顾客预期要求的程度。简言之，软件质量是软件一些特性的组合，它依赖软件的本身。

- 对于软件质量有多种不同的视面。用户主要感兴趣的是如何使用软件、软件性能和使用软件的效用，特别是在指定的使用环境（context）下获得与有效性、生产率、安全性和满意度有关的规定目标的能力，即使用质量。开发者负责生产出满足质量要求的软件，所以他们对中间制品和最终产品的质量都感兴趣，当然开发者视面也要体现软件维护者所需要的质量特性。对于管理者也许更注重总的质量而不是某一特性，必须服从管理准则，如在进度拖延或成本超支与质量提高之间进行权衡，以达到用有限的人力、成本和时间使软件质量达到优化的目的。

- 保证软件质量基本上可用两种途径来实现，一种是保证生存周期过程，另一种是评价软件最终产品的质量。这两种途径都很重要，且都要求有一系统来管理质量，该系统应确定管理对质量的保证，指明其策略以及恰当的详细执行步骤。前者是采用ISO 9001质量体系—设计、开发、生产、安装和服务的质量保证模式，或者CMM—能力成熟度模型，或者ISO 15504软件过程评估（也称为SPICE，即软件过程改进和能力确定）等方法来取得满足质量要求的软件。后者是把软件产品评价看作软件生存周期的一个过程，目标就是让软件产品在指定的使用环境下具有所需的效用，可以通过测量内部属性，也可以通过测量外部属性，或者通过测量使用质量属性来评价。

软件质量管理

- 目前能被大家接受和公认的是基于软件生存周期过程的质量管理，包括ISO 9001、CMM、ISO 15504等都是属于这种类型，如能力成熟度模型（CMM）较全面地描述和分析软件机构的软件过程能力的发展程度，建立了一个描述软件机构的软件过程成熟度的分级标准和框架。

- 软件过程能力是描述遵循一个软件过程而得到期望结果的程度，软件过程成熟度是指一个具体的软件过程被明确定义、管理、控制其实效的程度。利用能力成熟度模型，软件机构可以评估自己当前的过程成熟度，并通过提出更严格的软件质量标准和过程改进，来选择自己的改进策略，以达到更高一级的成熟程度。软件产品评价需要策划和管理，从而也是管理职能中的一个部分。

软件质量模型

一个框架，它规定了内部和外部质量的质量模型与使用质量的质量模型，以及它们在软件生存周期中的关系。

内部和外部质量的质量模型将软件质量属性分类为6个特性，即功能性、可靠性、易用性、效率、易维护性和易移植性，并进一步细分为27个子特性，从而构成一个有层次的树状结构，结构的最高层由质量特性组成，最低层则由软件质量属性组成。

使用质量的质量模型将软件质量属性分类为4个特性，即有效性、生产率、安全性和满意度。

软件质量特性

- 功能性：在指定条件下使用时，软件产品提供满足明确和隐含需求功能的能力；
- 可靠性：在指定条件下使用时，软件产品维持规定的性能级别的能力；
- 易用性：在指定条件下使用时，软件产品被理解、学习、使用及其吸引用户的能力；
- 效率：在规定条件下，相对于所用资源的数量，软件产品可提供适当性能的能力；
- 易维护性：软件产品可被修改的能力，修改可能包括修正、改进或者适应环境、需求和功能规约的变化；
- 易移植性：软件产品从一种环境迁移到另一种环境的能力；
- 有效性：软件产品在指定使用环境下，使用户准确、完整地获得规定目标的能力；
- 生产率：软件产品在指定使用环境下，使用户花费合适的与有效性相关的资源数量的能力；
- 安全性：软件产品在指定使用环境下，获得可接受的损害人类、商务、软件、财产或环境风险级别的能力；
- 满意度：软件产品在指定使用环境下，使用户满意的能力。

软件质量评价过程

- 3种评价过程：其一：开发者用的过程，计划开发新产品或增强现有产品时为了预测最终产品质量指标；其二，需求方用的过程，计划获取或复用某个已有产品时，为了决定接受产品或者从众多可选产品选择某个产品；其三，评价者用的过程，应开发者、需方或其他机构的请求，对产品进行独立评估，它们通常是第三方机构。不论哪一种评价过程，都是首先要确立评价需求，然后是规定评价、设计评价和执行评价等活动。确立评价需求应确立评价目的，确定产品类型（中间制品、最终产品），规定质量模型；规定评价包括选择度量、建立度量评定等级、确立评估准则；设计评价包括制定评价计划；执行评价包括进行度量、与评估准则相比较，评价结果。

- 早在20世纪60年代末，已经有人讨论大型软件开发项目的组织管理问题。随着软件工程在实践过程中发生的问题，人们认识到软件产品和软件项目的开发，不完全取决于软件开发方法。与程序的复杂性和系统的复杂性相比，前者已得到较大的缓解，更重要的是后者。如进度推迟、经费超支、质量差、软件人员不称职，均与管理有关。自20世纪80年代初，软件界就关注“软件过程”。

- 20世纪80年代中期，IBM在Watts S.Humphrey的指导下，Ron Radice等人将成熟度框架首次应用于软件过程，并由Humphrey于1986年将此成熟度框架带到卡内基·梅隆大学的软件工程研究所（CMU/SEI）。

- 应美国联邦政府要求，为其提供一个评价软件开发商能力的方法，1986年11月，CMU/SEI在MITRE公司的帮助下开始设计过程成熟度框架，以此帮助软件机构改进他们的软件过程。1987年9月，CMU/SEI发表了一个简短的软件过程成熟度框架。其后在Humphrey的《管理软件过程》一书中进行扩充。书中提出了软件过程评估和软件能力评估，和成熟度问卷，用于评估软件过程成熟度。基于这些设想，由Jim Withey, Mark Paulk 和Cynthia Wise 在1990年提出了最早的草案。1991年8月Mary Beth Chrissis 和Bill Curtis 帮助Mark Paulk校订，推出了CMM（成熟度模型）1.1版。

- CMU/SEI原先计划在1997年下半年推出2.0版，1998年推出2.1版。但由于种种原因，未能按计划推出2.0版。

- 由于美国联邦政府的大力推行，CMM模型得到了广泛应用，CMU/SEI于2001年公布了通过CMM评估的软件机构共有964家，并对其作了统计分析。目前CMM本身还在向纵深发展，CMM家族有：
  - 软件能力成熟度模型（SW-CMM）
  - 软件获取能力成熟度模型（SA-CMM）
  - 人员能力成熟度模型（People-CMM）
  - 系统工程能力成熟度模型（SE-CMM）
  - 集成产品开发能力成熟度模型（IPD-CMM）
  - 个体软件工程（PSP）
  - 群组软件工程（TSP）

- 另外，国际标准化组织为组织软件过程评价标准的制订，成立了“软件过程改进和能力确定（Software Process Improvement and Capability Determine, SPICD）”项目，ISO/IEC JTCI的SC 7分技术委员会于1996年9月正式公布了工作草案，代号为15504，即ISO/IEC TR 15504 Information Technology –Software Process Assessment。

- CMM的广泛采纳和成功推广，推动了软件及其他学科的类似模型的开发，从而模型繁衍，导致了过程改善目标和技术的冲突。要求的培训工作有了相当大的增长，部分实践人员在应用各种不同的模型来实现特定的需要时产生了混淆。针对这种情况，美国联邦政府、产业界和CMU/SEI于1998年启动了“能力成熟度模型集成（Capability Maturity Model Integration, CMMI）”项目，于2000年第四季度发布了第一个正式的CMMI，最近的修改是2002年6月10日。CMMI包括了大量的工程改善和过程改善的信息，提供了单一的集成化框架来改善跨越几个学科的机构的工程过程，从而提高机构过程改善的质量和有效性。相信CMMI将会逐步替代CMM。

- 另外我国也非常重视CMM的研究和应用，目前已经参照CMM1.1版制定国家军标“军用软件成熟度模型”，以及参照CMMI制定行业标准“ST/T11234软件过程能力评估模型”和“ST/T11235软件能力成熟度模型”。

软件质量评价简介

- ISO和IEC（国际电工委员会）是世界性的标准化专门机构。在信息技术领域，ISO和IEC建立一个联合的技术委员会，即ISO/IEC JTC1。发布一项国际标准，至少需要有75%的参与表决的国家成员体投票赞成，可见其权威性。

- 一般而言，对软件的质量进行评价要从模型的生成、计分统计及结果分析，到结果报告生成软件自动化，要符合软件质量评价国际标准和国家标准，应较全面地考虑影响软件质量的诸多因素，如各种软件质量特性、评价准则和度量、质量加权系数采用的方法等，能较系统、科学地反映软件的质量。

- 我国执行的《GB/T 16260-1996 信息技术 软件产品评价 质量特性及其使用指南》国家标准（等同于ISO/IEC 9126： 1991），定义了评价软件质量的功能性、可靠性、易用性、效率、维护性、可移植性6个特性和在此之下的21个子特性。

有关概念

- 保证软件产品质量一般可以通过两种途径实现，一是保证软件开发过程的质量，另外就是评价最终软件产品的质量，这两种途径都非常重要。

- 为了满足软件质量要求而进行的软件产品评价是软件开发生存周期中的一个过程。软件质量可以通过测量内部属性、外部属性和使用质量的属性来评价。目标就是使软件在指定的使用条件下具有所需的效用

- 内部质量：是基于内部观点的软件产品特性的总体。内部质量是针对内部质量需求所被测量和评价的质量。软件产品质量的细节可以在代码实现、评审和测试期间被改进，但是由内部质量表示的软件产品质量的基本性质不会改变，除非进行重新设计。

- 外部质量：是基于外部观点的软件产品特性的总体。即当软件执行时，典型的是使用外部度量在模拟环境中用模拟数据测试时所被测量和评价的质量。在测试期间，大多数错误都应该可以被发现和消除。然而，在测试后仍会存在一些错误。由于难以校正软件的体系结构或软件其他的基础设计，基础设计在整个测试中通常保持不变。

- 使用质量：软件产品对指定用户在特定的使用条件下获得与有效性、生产率、安全性和满意度相关的规定目标的能力。它是基于用户观点的用于指定的使用环境和条件时的质量。它测量用户在特定环境中能达到其目标的程度，而不是测量软件自身的性质。

- 用户的质量要求包括指定的使用条件下对使用质量的需求。当使用软件产品质量特性和子特性来说明外部质量和内部质量的时候，可以使用这些被确定的要求。
- 过程质量有助于提高产品质量，而产品质量又有助于提高使用质量。因此，评估和改进一个过程是提高软件产品质量的一种手段，而评价和改进软件产品质量则又是提高使用质量的一种手段。同样，评价使用质量可以为改进产品提供反馈，而评价产品则可以为改进过程提供反馈。
- 合适的软件内部属性是获得所需外部特性的先决条件，而适当的外部特性则是获得使用质量的先决条件。内部质量、外部质量和使用质量的观点在软件生存周期中是变化的。例如，在生存周期开始阶段作为质量需求而规定的质量大多数是从外部和用户的角度出发的，它与像设计质量这样的中间产品质量不同，后者大多是从内部和开发者的角度来看问题的。

软件质量保证措施

软件质量保证(Software Quality Assurance，通常缩写为SQA)的措施主要有，基于非执行的测试(也称为复审)、基于执行的测试和程序正确性证明。

1. 技术复审的必要性

正式技术复审的明显优点是，能够较早地发现错误，防止错误被传播到软件过程的后续阶段。

正式技术复审实际上是一类复审方法，包括走查(Walkthrough)和审查(Inspection)等具体方法。走查的步骤比审查少，而且没有审查那样正规。

2. 走查

(1) 参与者驱动法

参与者按照事先准备好的列表，提出他们不理解的术语和认为不正确的术语。文档编写组的代表必须对每个质疑做出回答，要么承认确实有错误，要么对质疑做出解释。

(2) 文档驱动法

文档编写者向走查组成员仔细解释文档。走查组成员在此过程中不时针对事先准备好的问题或解释过程中发现的问题提出质疑。这种方法可能比第一种方法更彻底，往往能检测出更多错误。经验表明，采用文档驱动法时许多错误是由文档讲解者自己发现的。

3. 审查

审查的范围要比走查广泛得多，它的步骤也比较多。一般来说，审查有5个基本步骤。

- **综述**：由负责编写文档的一名成员向审查组成员综述该文档。在综述会议结束时把文档分发给每位与会者。
- **准备**：评审员仔细阅读文档。最好列出在审查中发现的错误的类型，并按发生频率把错误类型分级，以辅助审查工作的进行。这些列表有助于评审员们把注意力集中到最常发生错误的区域。

· **审查**：评审组仔细走查整个文档。和走查一样，这一步的目的也是找出文档中的错误，而不是改正它们。审查组组长必须在一天之内写出一份关于审查的报告。通常每次审查会不超过90分钟。

· **返工**：文档的作者负责解决在书面报告中列出的所有错误及问题。

· **跟踪**：组长必须确保所提出的每个问题都得到了圆满的解决(要么修正了文档，要么澄清了被误认为是错误的条目)。必须检查对文档所做的每个修正，以确保没有引入新的错误。如果在审查过程中返工量超过5%，则应该召集审查组再对文档全面地审查一遍。

4. 程序正确性证明

正确性证明的基本思想是证明程序能完成预定的功能。因此，应该提供对程序功能的严格数学说明，然后根据程序代码证明程序确实能实现它的功能说明。　　如果在程序的若干个点上，设计者可以提出关于程序变量及它们的关系的断言，那么在每一点上的断言都应该永远是真的。假设在程序的P_1，P_2，…，P_n等点上的断言分别是a(1)，a(2)，…，a(n)，其中a(1)必须是关于程序输入的断言，a(n)必须是关于程序输出的断言。

为了证明在点P_i和P_{i+1}之间的程序语句是正确的，必须证明执行这些语句之后将使断言a(i)变成a(i+1)。如果对程序内所有相邻点都能完成上述证明过程，则证明了输入断言加上程序可以导出输出断言。如果输入断言和输出断言是正确的，而且程序确实是可以终止的(不包含死循环)，则上述过程就证明了程序的正确性。

第十二章 软件测试职业指导

12.1 软件测试职位

12.2 优秀软件测试工程师应具备的素质

12.3 软件测试信息资源

- 测试时代网：http://www.testage.net/
- 软件工程专家网：http://www.51cmm.com/SoftTesting
- 赛迪网：http://tech.ccidnet.com/pub/column/c319.html
- Software Testing Hotlist (www. io. com/~wazmo/qa)
- Software Testing Online Resources (www. mtsu. edu/~storm)
- Bug Net (www. bugnet. com)

软件测试在美国硅谷地区的软件公司里是十分重要的一项工作。在中等规模的软件公司里，测试人员包括7类：测试部门经理、测试组管理员、测试工程师、测试设施组管理员、测试数据库管理员，软件集成员、自动测试组管理员、自动测试编程员。

（1）测试部门经理。一个软件公司测试部门经理肩负着公司软件产品质量管理的主要责任。他必须有着丰富的科学管理经验，熟悉软件的生命周期全过程，对产品质量的重要性有足够的认识，有很好的组织能力，以及在同一时间里兼管多个项目的能力。其主要责任包括：

①规划领导整个测试部门的运作：包括制定测试工作的架构；确定人员的分配、各组的分工；最后决定采用测试工具、何种版本控制工具及何种缺陷报告工具等。

②有效地控制测试进度：通过定期与不定期检查各组管理人员的工作，掌握各时期的产品质量情况，定期向上级汇报测试进度；随时与开发部门保持沟通，相互了解，相互帮助；在测试发生特殊情况时，及时向开发部门及上级管理层通报。

③定期与编程组负责人沟通，了解他们的进度及产品的成熟程度，以保证开发部门在将其产品交付测试部门之前已按规定完成了单元测试及组装测试。

④协调各部门之间的关系：当测试组认为产品质量未达到要求而编程组又认为问题不大时，测试部门经理则应与开发部门经理坐下来展开分析，协调解决。

⑤评审测试部门的工作，不断改进。包括聘请外部有丰富经验的人对测试架构作出评估，提出改进的意见。组织部门内的各种技术研讨会，让各组之间相互了解，互相学习，取长补短。

⑥监督设施组的工作，保证各版本的控制、测试及缺陷报告系统正常工作。

（2）测试组管理员。测试组管理员应对软件的生命周期及软件测试的重要性有深入的认识，具有很好的组织管理能力及计划能力，并能很好地与测试工程师们沟通以带动他们的积极性。同时，测试管理员应具有使用自动测试工具的经验，他不必具体动手做，但最好是一个懂行的管理员。其主要指责为：

①制定测试计划及进度表，并负责监管执行。

②建立并不断完善测试记录文档制度。

③负责组内分工。

④负责给测试人员做年度工作评审。

⑤与组内人员建立良好的关系，了解并帮助他们解决工作中遇到的问题。

⑥与编程组管理员及组内人员沟通良好，掌握项目的最新状况及软件可以交付测试的日期，以便于随时调整测试进度。

⑦随时与技术支持（售后服务）组沟通，将他们或用户的反馈通报测试组并存入测试档案库。

⑧定时组织组内测试员与编程组召开会议，检讨工作的进程，列出缺陷报告中哪些问题应引起足够的重视，以提供给编程组参考。

⑨随时向测试部门经理报告测试进度及工作中遇到的问题。

（3）测试工程师。测试工程师是整个软件测试中最关键的角色。一个理想的测试工程师应具备以下能力：

①对软件质量的重要性有深入的认知，热爱测试工作。这是最起码的，也是十分重要的条件。

②技术上过硬，并不断追求、更新自己已有的软件及测试方面的知识。

③沟通能力强。必须具有与测试所涉及到的技术（程序员）及非技术人员（管理人员、用户）之间进行交流的能力。在工作中要与其他测试工程师保持良好关系，随时交流测试进展情况及遇到的问题。

④具有敏锐的洞察力。只有对事物（当然包括软件）细节的关注能力强，才会对一些容易忽略的部分或不寻常的状态加以审视，从中捕获缺陷。

⑤有足够的耐心。测试工作常常需要花费不少时间从许多状况或错综复杂的系统中去分离、识别每一个缺陷；当缺陷被修正后，还要重做测试。而且修复一个缺陷之后，很可能因此影响到其他部分，从而引出新的缺陷。

⑥具有较强的独立工作能力。一个好的测试工程师必须习惯独立工作，不依赖于管理人员的帮助与监督。

⑦在遇到特殊情况时，如人手不够、时间有限，应能承担巨大的工作压力，以产品的信誉为目标，尽自己的能力作好一切。

作为一名测试工程师，其主要职责为：

①作好日常的测试工作。熟悉软件规格说明书、设计说明书、各类相关文件及测试计划书，按各项测试要求工作，发现缺陷、作好纪录并及时报告。

②协助测试组管理员完成测试计划书。测试员通常具体负责编写测试实例。

③参与讨论制定软件设计说明书的过程，参与编程评审工作，提出自己的见解。

④在时间进度允许的情况下，参与编写一些自动测试程序，以节省日后重复测试的时间，并将测试的覆盖率提高。

⑤对已有的测试程序进行维护。随着对同一软件反复测试，不断找出新的测试实例，不断充实已有的测试架构。

⑥负责测试操作及自动测试程序的运行。

⑦工作遇到困难或问题时，组织召开小型会议，以寻求解决方法。

⑧作好测试记录，及时报告产品缺陷。

⑨编写软件新版发行说明。