软件测试的核心目的：要度量和提高软件质量

软件质量：是一个包含多个属性的，可度量的多维度的量
根据软件质量的度量方式可以分为静态质量属性和动态质量属性
- 静态质量属性是指实际代码和相关文档的质量，包括代码的结构化、可维护性、可测试性，文档的完整性和可读性
- 动态质量属性是指软件使用过程中质量属性，包括软件的可靠性、正确性、完备性、可用性、可维护性、可信性和性能等。

软件缺陷的定义
（1）软件未达到产品说明书中已经标明的功能；
（2）软件出现了产品说明书中指明不会出现的错误；
（3）软件未达到产品说明书中虽未指出但应当达到的目标；
（4）软件功能超出了产品说明书中指明的范围；
（5）软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

软件缺陷管理
- 缺陷预防
- 缺陷发现（软件测试的目的）
- 缺陷记录和报告
- 缺陷分类和跟踪
- 缺陷处理
- 缺陷预测

软件测试是为了发现错误而执行程序的过程。
软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计的一批测试用例，并利用这些测试用例运行程序以及发现错误的过程，即执行测试步骤。
使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果和实际结果之间的差别。
测试是以评价一个程序或者系统的属性为目标的任何一种活动，测试是对软件质量的度量。
软件测试是一种作为主体的人通过各种手段对客体软件的某种固有属性进行的一种以认知和改造为目的的活动。
测试是为了度量和提高被测试软件的质量，对测试软件进行工程设计，使用和维护的并发生命周期活动。

软件生命周期：一个软件生命周期包括制定计划、需求分析定义、软件设计、程序编码、软件测试、软件运行、软件维护、软件停用等 8 个阶段。

软件测试的对象：软件开发过程中所产生的需求规格说明、概要设计规格说明、详细设计规格说明以及源程序都是软件测试的对象。

软件测试的目的
测试是程序的执行过程，目的在于发现错误；不能证明程序的正确性，除非仅处理有限种情况。

检查系统是否满足需求也是测试的期望目标。
一个好的测试用例在于发现了还未曾发现的错误；一次成功的测试则是发现了错误的测试。

软件测试的原则

（1）尽早地和及时地测试；
（2）测试用例应当由测试数据和与之对应的预期结果这两部分组成；
（3）在程序提交测试后，应当由专门的测试人员进行测试；
（4）测试用例应包括合理的输入条件和不合理的输入条件；
（5）严格执行测试计划，排除测试的随意性；
（6）充分注意测试当中的群体现象；
（7）应对每一个测试结果做全面的检查；
（8）保存测试计划、测试用例、出错统计和最终分析报告，为维护工作提供充分的资料。
（9）进行风险分析，确定计划风险
（10）制定测试策略
（11）使用测试清单
（12）使用测试工具
（13）度量测试有效性
（14）不断进行培训
（15）宣传测试思想
（16）软件测试是一种服务
（17）软件测试是一项极富创造性、极具智力挑战性的工作
（18）选择一种合适的方法
（19）增量和分级测试
（20）分析缺陷趋势和模式

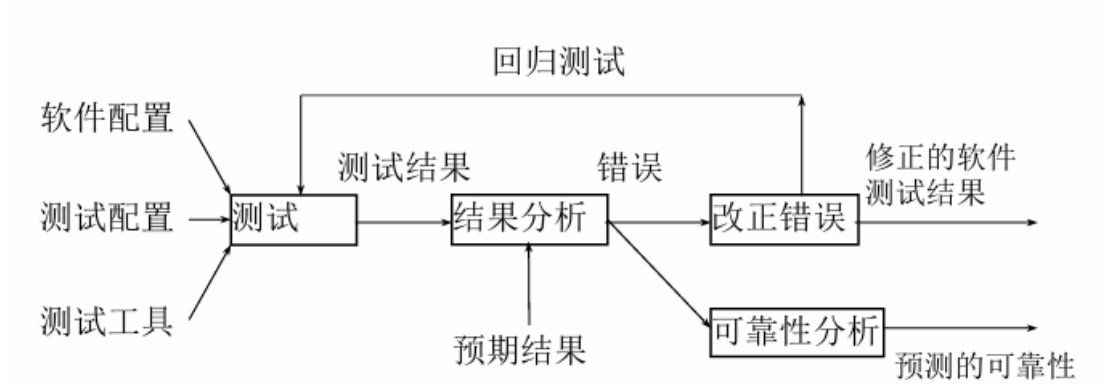

图1-2 测试信息流程

(1) 软件配置：是指测试对象，它包括软件需求规格说明、软件设计说明、和被测试的源程序清单。
(2) 测试配置：包括测试计划，测试用例，测试驱动程序，实际上，在整个软件工程中，测试配置只是软件配置的一个子集。
(3) 测试工具：为提高软件测试效率，可使用测试工具支持测试。其作用就是为测试的实施提供某种服务，以减轻人们完成测试任务中的手工劳动。

软件测试的特性

证错不证对性
综合性或多方求证性
不完全性或抽样性

测试停止的依据（标准）
第一类标准：测试超过了预定时间，则停止测试。
第二类标准：执行了所有的测试用例，但并没有发现故障，则停止测试。
第三类标准：使用特定的测试用例设计方案作为判断测试停止的基础。
第四类标准：正面指出停止测试的具体要求，即停止测试的标准可定义为查出某一预订数目的故障。
第五类标准：根据单位时间内查出故障的数量决定是否停止测试

软件开发模型中的软件测试
1、大棒开发法
2、边写边改法
3、瀑布法
4、快速原型法
5、螺旋模式法
6、敏捷开发模式

测试员应具备的素质
(1) 探索精神
(2) 故障排除能手
(3) 不懈努力
(4) 创造性
(5) 追求完美
(6) 判断准确
(7) 老练稳重
(8) 说服力

软件测试的发展趋势
- 软件测试自动化
- 测试技术智能化
- 测试方法工具化
- 测试工具服务化
- 测试理论系统化
- 测试手段多样化
- 测试过程标准化
- 测试人员专业化
- 测试部门独立化
- 测试管理全面化
- 测试对象精细化

软件测试的工具

测试管理工具：TestDirector 和 TestManager 等
自动化功能测试工具：QTP(Quick Test Professional), Rational Robot, TestCoplete 等
性能测试工具：如 LoadRunner，SilkPerformer 等
单元测试工具：如 XUnit 系列、MSTest 等
白盒测试工具：TEST、AQTime 和 BundsChecker 等
测试用例设计工具：有 CTE XL、AETG 和 PICT 等

软件测试的管理
• 时间维：全过程管理，对软件测试项目的全过程进行控制，具体包括：测试计划管理，测试设计管理，测试执行管理，测试结果管理等。
• 空间维：全方位管理，对与软件质量有关的关键因素实施全方位管理，具体包括：缺陷管理、文档管理、配置管理、评审管理、质量管理和回归管理等。
• 组织维：人员管理，构建从测试人员、测试小组到测试机构的多层次的组织管理模式。

软件测试概念发展简史
• Debugging oriented (1950 年左右)
• Demonstration oriented (1960 年左右)
• Destruction oriented (1970 年左右)
• Evaluation oriented (1980 年左右)
• Prevention oriented (1990 年以后)
• Professional、education and research （2000 年以后）

软件测试的发展趋势
• 软件测试自动化
• 测试技术智能化
• 测试方法工具化
• 测试工具服务化
• 测试理论系统化
• 测试手段多样化
• 测试过程标准化
• 测试人员专业化
• 测试部门独立化
• 测试管理全面化
• 测试对象精细化

软件测试的 PIE 模型
• PIE 模型对于软件测试日常笼统说的“Bug”一词做了细分：
• 1、Fault：软件中存在的静态错误。
• 2、Error：由于 Fault 导致的内部状态的错误。
• 3、Failure：不满于规格说明，用户可见的外部错误。
• 我们要观察到错误的存在，必须经过三个步骤，也就是“PIE”名字的由来：
• 1、Execution（执行）：错误代码必须要被执行到。
• 2、Infection（感染）：触发了错误的中间状态。
• 3、Propagation（传播）：错误的中间状态必须可以传播到最后的输出使得可以被观测到。

• 错误代码被执行到，未必会触发错误的中间状态；错误的中间状态，未必会导致错误的输出。
• 如果要发现一个 Bug，必须要满足 PIE 模型。PIE 模型对于我们如何提升软件质量有一定的指导借鉴意义。

	静态测试	动态测试
白盒测试	1同行评审 2代码审查 3代码走查 4桌面评审	1语句覆盖；　2分支覆盖； 3条件覆盖；　4分支条件覆盖； 5条件组合；　6修改条件决策； 7 路径覆盖；　8数据流覆盖； 9线性序列跳转覆盖
黑盒测试	1文档审查	1等价类划分；　2边际值分析； 3因果图与决策表；4语法测试； 5故障推测法；　6状态转换测试

白盒测试：把测试对象看着一个打开的盒子，它允许软件测试员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序的状态，取得实际的状态，是否与预期的状态一致，故又称结构测试或逻辑驱动。

一般评审过程的四个特征
发现问题
遵循一定的规则
准备工作
会议报告

同行评审，有时也称好友评审，不是很正式，就相当于“你把你的拿给我看看，我把我的拿给你看看”，通常由软件设计人员和编码人员加上一两个其它程序员或测试人员组成，要提高同行评审的效果，必须加强一般评审过程的四个特征。

代码审查是最正式的一种评审，编写受评审代码的程序员不能参与评审，所以要求每个参与人员都要受过专门的训练，能够学习和理解代码和相关的材料。指定一个记录员和一名协调员负责整个审查过程的有效运行。审查会议结束之后，评审人员与协调员就发现的问题写一个报告，指出解决相关问题的必要性工作。然后交给程序员进行整改，协调员对相关修改进行查证，并根据问题的大小决定是否进行下一轮审查。

代码走查更正式一点，写代码的程序员向由程序员和测试人员组成的五人左右的审查小组走查自己的程序。审查小组成员应该提早收到程序拷贝，认真阅读，写好评语和关心的问题。审查小组中至少要有一个资深的程序员。

桌面检查可视为由单人进行的代码检查或代码走查，由一个人阅读程序，对照错误列表检查程序，对程序推演测试数据。
逻辑覆盖：是对一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。

语句覆盖：选择足够多的测试数据，使被测程序中每个语句至少执行一次。

判定（分支）覆盖：设计若干测试用例，运行所测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。

条件覆盖就是设计若干个测试用例，运行所测程序，使得程序中每个判断条件的可能取值至少执行一次。

判定（分支）-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行一次。

条件组合覆盖就是设计足够的测试用例，运行所测程序，使得每个判断的所有可能的条件取值组合至少执行一次。

修改决策条件：设计测试用例让每个条件变量独立改变判定语句的真假值。

路径覆盖：就是设计足够的测试用例，覆盖程序中所有可能的路径。

线性代码序列跳转测试：测试用例用来执行覆盖每个线性代码序列跳转，每个测试用例可以描述为三个部分：测试输入、被该测试用例执行的线性代码序列跳转、预期输出。

数据流测试：所有定义覆盖（all-definitions）：设计测试用例执行所有变量定义点到某些使用点（计算使用或者谓词使用）的路径。
- 所有计算使用覆盖（all-c-uses）：设计测试用例执行每个变量定义点到其每个计算使用点的路径。
- 所有谓词使用覆盖（all-p-uses）：设计测试用例执行每个变量定义点到其每个谓词使用点的路径。
- 所有使用覆盖（all-uses）：设计测试用例执行每个变量定义点到其每个使用点（计算使用或者谓词使用）的路径。
- 所有定义使用对覆盖(all-du-paths)：设计测试用例执行所有变量定义点到每个使用点(计算使用或者谓词使用）的路径

黑盒测试：在完全不考虑程序的内部结构和处理过程的前提下，在程序接口进行的测试，它只检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接受输入数据产生正确的输出信息，并且保持外部信息的完整性。因此，又称为功能测试或数据驱动。

等价类划分：把所有可能输入的数据划分成若干等价类，假定每类中的一个典型值在测试中的作用与这类中所有其他值的作用相同。然后可以从每个等价类中只取一组数据作为代表性数据用于测试，以便发现程序中的错误。

边界值分析：确定边界情况。通常输入等价类与输出等价类的边界，就是应该着重测试边界情况。应当选取正好等于、刚刚大于，或刚刚小于边界的值作为测试数据。

错误推测法：列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案。对于程序中容易出错的情况也有一些经验总结出来。

判定表：利用判定表或判定树为工具，列出输入数据各种组合与程序应作的动作（及相应的输出结果）之间的对应关系，然后为判定表的每一列至少设计一个测试用例。

因果图：分析软件规格说明描述中的语义，找出原因和结果之间，原因与原因之间对应的关系。根据这些关系，把因果图转换成判定表。把判定表的每一列拿出来作为依据，设计测试用例。

状态转换：针对有状态软件设计测试用例，核心验证状态间合法/非法切换逻辑是否合规、状态切换后功能与数据是否符合预期。

语法测试：基于输入接口的语法变异生成测试用例。

单元测试：开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。是一种验证行为。是一种设计行为。是一种编写文档的行为。具有回归性。

集成测试：对由经过单元测试的模块组装起来形成的一个子系统进行的测试，这样的测试被称为子系统测试。而对由经过测试的子系统测试组装成的系统进行的测试则称为系统测试。不论是子系统测试还是系统测试都兼有检测和组装的含义，这样的测试通常就称为集成测试。可以分为两类一是非渐增式测试，一是渐增式测试。

渐增式测试是指把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。
非渐增式测试是先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。

集成测试策略：
自顶向下：从主控制模块开始，沿着软件的控制层次向下移动，逐渐把各个模块结合起来(深度优先/ 广度优先)
自底向上：从原子模块开始，逐步向上组装和测试各模块

系统测试：将整个软件系统视为一个整体来进行测试，重点测试软件产品的各项功能是否满足用户的要求，还包括性能、安全性、兼容性、可用性等软件特性及方面的测试。

冒烟测试：一种将代码更改提交到代码库之前的验证过程，在每日 build 建立后，把项目的所有的最新的代码从配置库中取出，然后从头进行编译、链接和运行，对系统的基本功能进行测试，最小化集成风险、简化故障诊断。

验收测试：经过集成测试，已经按照设计把所有模块组装成一个完整的软件系统，接口错误也已经基本排除了，进一步验证软件的有效性的过程就是验收测试，目的是向未来的用户表明系统能够像预定要求那样工作。

回归测试：是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。

α测试：是由用户在开发环境下进行的测试，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。

β 测试：是由软件的多个用户在在开发者无法控制的实际使用环境下进行的测试，这些用户返回有关错误信息给开发者。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。

λ测试：是第三个阶段，此时产品已经相当成熟，只需在个别地方再做进一步的优化处理即可上市发行

压力测试：通过模拟负载，使系统在负载饱和或资源匮乏的状态下运行，检测系统在极限情况下的表现，通过确定一个系统的瓶颈或者不能接收的性能点，来获得系
统能提供的最大的服务级别。

负载测试：通过测试系统在在一定负载条件下的表现，以发现设计上的错误或验证系统的负载能力，确保系统能够在预期的用户负载下正常运行。

软件老化：软件系统长时间运行中出现的可用资源不足、性能下降、失效率增加等现象。

容量测试：通过测试预先分析出反映软件系统应用特征的某项指标的极限值，确保系统在其极限值状态下没有出现任何软件故障或还能保持主要功能正常运行，主要用于评估系统在不同负载情况下的最大处理能力。

性能测试：是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试，验证软件系统是否能够达到用户提出的性能指标，包括负载测试，强度测试，容量测试等。

可靠性测试：在给被测试系统加载一定业务压力的情况下，使系统运行一段时间，以此来测试系统是否稳定，旨在评估系统在特定条件下的稳定性和一致性。

安全性测试：是指在测试软件系统中危险防止和危险处理设施进行的测试，以验证其是否有效。

安装测试：确保软件在正常情况和异常情况下都能进行安装的测试，旨在验证软件产品的安装过程是否按照设计要求顺利进行。

可用性测试检查待测试软件的人机界面，通常要检查的部件包括界面布局与色彩、输入输出

格式、程度流程和拼写等，以发现其中的人为因素和易用性的问题。

稳定性测试：主要尝试检测软件在其使用过程中，尤其是在长期使用或持续负荷下，性能是否稳定，以及是否会崩溃。

本地化和国际化测试：检测软件是否针对不同国家或地区进行适配和调整。

可访问性测试：检测软件是否满足政府或软件行业颁布的可访问性规范，为各类用户扫除使用障碍。

授权测试：是一种测试授权过程健壮性的方法，该方法通过理解授权工作原理和过程，利用这些信息来试图规避授权机制。

容错性测试：是检查软件在异常条件下自身是否具有防护性的措施或某种灾难性恢复的手段。当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。

一致性测试或类型测试：是测试一个产品在效率或互通性方面是否符合某个指定的标准。

配置测试：测试目标软件在具体硬件配置情况下，出不出现问题，为的是发现硬件配置可能出现的问题

文档测试：系统测试文档的正确性，对于设计报告，主要是测试程序与设计报告中的设计思想是否一致；对于用户使用说明进行测试时，主要是测试用户使用说明书中对程序操作方法的描述是否正确。

兼容性测试：是指待测试项目在特定的硬件平台上，不同的应用软件之间，不同的操作系统平台上，在不同的网络等环境中能正常的运行的测试。

试玩测试：是在一种游戏设计开发过程中，在游戏投放市场之前，为了检查发现游戏中潜在的错误，进一步改进游戏而采取的一种测试方法。

可恢复性测试：测试软件系统在故障发生后(系统崩溃或其他灾难性出错)，重新建立其性能水平并恢复受影响数据的能力。

卸载测试：是对软件的全部、部分或升级卸载处理过程的测试。主要是测试软件能否卸载，卸载是否干净，对系统有无更改，在系统中的残留与后来的生成文件如何处理等。还有原来更改的系统值是否修改回去。

能力测试：判断目标文档提及的每一项能力是否确实已经实现。

健壮性测试：重点关注应用程序的健壮性，，旨在验证系统在面对各种异常条件、非标准输入和边界情况时的稳定性和正确性。

穿越测试：通过构造测试用例来故意违反各种安全约束，其目的是验证输入检查，检查系统

的鲁棒性和评估系统的安全性。

在线帮助测试：用于验证系统的实时在线帮助的可用性和正确性。

数据转换测试：验证已存在的数据转换是否有效。

备份测试：是可恢复性测试的一个补充，一个部分。目的是验证系统在软件或硬件失效的事件中备份其数据的能力。

接口测试是对软件需求规格说明中的接口需求逐项进行的测试。

人机交互界面测试：对所有人机交互界面提供的操作和显示界面进行测试，以检验是否满足用户要求。

图形用户界面测试：基于软件的人机图形交互界面检测系统是否能正常工作。

余量测试：评估系统在极端条件下的表现和稳定性。该测试的目标是确定系统在负荷、输入范围或其他系统限制接近其边界时的行为。

协议测试：针对网络协议及其实现的正确性和有效性进行验证和评估的测试活动。

内存泄漏测试：专门用于检测和评估软件应用中内存泄漏问题的测试活
动。

存储测试：用于验证和评估数据存储系统或存储设备的性能、可靠性和可扩展性的测试过程。

韧性测试：测试软件系统是否能有效应对各类运行异常和故障。

组合测试：是一种充分考虑各种因素及其相互作用的科学实用的软件测试方法。

蜕变测试：也称为基于性质的软件测试。一般指在预期输出无法给出的情况下，通过设计各种输入关系，观察输出是否满足预期的性质以判断软件的正确性。即测试软件应具有的性质是否存在。

基于规格说明的测试：在已知软件所应具有的功能的基础上，检查程序功能是否按需求规格说明书的规定正常使用，功能是否有遗漏，性能等特性要求是否满足。是验证软件的实现与规格说明的一致性。

基于模型的软件测试：利用模型来描述系统的行为、特性和需求，以便自动生成测试用例、执行测试和验证系统的正确性。

基于错误的软件测试：针对待测试软件中可能存在的某种软件错误，设计相应的测试用例，故意引入或模拟已知的错误、缺陷或不当使用的情况，来评估软件系统的健壮性、可靠性和错误处理能力。

故障驱动软件测试：模拟已知的故障通过精心设计的测试用例，评估系统的功能和可靠性。

失误测试：旨在发现由于人为失误而导致的软件缺陷或错误。

异常条件测试：旨在检验软件在面对各种异常情况时的稳定性和可靠性。

基于搜索的软件测试：利用优化和搜索技术来自动生成测试用例或进行测试评估的测试方法。

统计测试：根据定义好的概率分布来随机选择测试用例的软件测试方法，它的有效性来自获得这种定义好的概率分布，这种定义好的概率分布侧重于对应软件中故障分布概率。

基于操作剖面的测试：利用操作剖面的信息进行软件测试的方法，称之为基于操作剖面的测试。

变异测试：通过在程序中植入一些程序员容易犯的错误，或利用相关的启发式方法植入一些错误，这些错误都是通过对原来的程序做一些句法上的修改，从而形成一组新的程序，称之为变异体。通过运行针对原有程序的测试用例集，如果这些原来的测试例能够发现变异体的错误，称为杀死变异体，否则为变异体存活。最后通过分析被杀死的变异体数量和存活的变异体数量，来推测软件程序中可能仍潜在的错误数量，评估针对原有程序的测试用例集的测试充分性程度。

基于性质的软件测试：根据系统或程序的性质(即预期行为和特性)来生成测试用例的方法。

极限测试：先创建单元测试和验收测试，然后再创建代码库，这种形式的测试叫极限测试，也叫测试驱动开发。

模糊测试：通过向程序输入随机、无效或意外的数据，用于发现程序内部的安全漏洞和其他缺陷。

自适应测试：也称为软件测试的控制论方法，通过在线实时了解被控对象，不断调节控制器，使系统的性能达到技术要求或最优。

导向性随机测试：通过随机生成测试输入，将具体执行与符号执行相结合来执行程序中的所有可行路径，以发现程序缺陷。

随机测试：在待测试软件所有可能的输入中，随机选择和产生测试输入，对待测试软件进行测试，这种测试方法称为随机测试。

自适应随机测试：强调在随机选择测试用例的过程中，要使得测试用例尽可能均与地分散开。

反随机测试：为待测试软件随机生成一个测试用例，随后每一个测试用例的生成都要求与已生成的测试用例保持最大的距离。

结对测试：是一种由两个测试人员结对在一起对待测试软件的方法。

在线测试：也称动态测试，该技术中测试用例根据一个模型程序来产生，测试执行被结合到一个算法中，该方法在待测试软件运行过程中动态产生测试用例，而不是事先产先测试用例。

探索性测试：将对系统的探索和对系统的测试结合起来，强调测试人员个人的自由和责任，可以充分发挥他们的创造性和积极性，把测试过程看成是一种与测试相关的学习，测试设计、测试执行和测试结果的解释同时进行且相互促进的活动，这些活动在并行的过程中不断优化。

反模型测试：基于构建反向模型，即从预期的功能行为中推导出不正确的或意外的状态和输出，以此来确保系统在异常条件下的鲁棒性和稳定性。

成分测试：对系统的组成部分进行单元测试后进行集成测试，在持续集成环境中反复进行。

有限状态机测试：基于有限状态机理论的测试方法，验证系统在不同输入下的状态转移是否符合预期，确保软件或硬件的功能正确性。

基于 Petri 网的测试：利用 Petri 网理论进行建模和测试的方法，根据 Petri 网的结构生成测试用例，以覆盖不同的执行路径和状态。

基于模型检查的测试：用状态迁移系统（S）表示系统的行为，用模态/ 时序逻辑公式（F）描述系统的性质，S ⊧ F 对有穷状态系统是可判定的。

TTCN（Testing and Test Control Notation）是一种用于测试的编程语言，特别是用于通讯协议测试和 WEB 服务测试。TTCN 测试用例集是一组利用 TTCN 编程语言编写的测试用例组成的集合。

布尔规格测试：在航空、医疗以及一些控制软件等重要的安全关键软件中，布尔表达式可以用作描述规格的条件、模型谓词以及逻辑表达式等，根据这些布尔表达式设计测试用例来测试。处于程序或者规格说明中的布尔谓词 p，如果它包含 n 个布尔变量，需要验证 p 的正确性的测试用例就需要 2n 条。

基于统一建模语言测试：应用待测试软件系统的 UML 模型来获得软件测试的需求和覆盖准则。

差分测试：用于比较两个或多个实现（例如，不同版本的程序或不同的实现算法）在相同输入下的输出结果，常用于确保新版本的软件相对于旧版本或不同实现之间的兼容性和一致性。

故障注入测试：通过故意向系统中注入故障，验证系统在面对错误或异常情况下的反应和处理能力。

面向对象软件测试：是针对使用面向对象编程（OOP）原则开发的软件进行的一系列测试活动。封装性，继承性，多态性，复杂的对象交互，数据和行为的联动，使用模拟和桩，重用性和可扩展性的关注，状态驱动测试，行为驱动开发，异常处理与验证

面向方面的软件测试：是针对使用面向方面编程开发的软件进行的一系列测试活动。包括基于代码的测试即结构化测试或白盒测试、基于规格说明的测试即黑盒测试、基于模型的测试、基于故障和风险的测试、回归测试、变异测试以及测试用例自动生成和执行的自动化测试等。

面向服务的软件测试：原理主要依赖于服务导向架构（SOA）和微服务架构的核心概念。该测试方法关注于确保各个服务之间的正常交互和完整性，以提供高质量的系统。服务接口测试，独立性，服务间的交互测试，契约测试，故障注入，性能测试，安全性测试，API 文档与测试，调试与监控，自动化测试

构件软件测试：是指在构件（Component）基础上进行的测试过程，构件可以是软件系统中的独立模块、服务或功能单元。模块化测试，可重用性，接口验证，自动化，单元测试，集成测试，回归测试，系统测试，性能测试，安全性测试

Web 应用软件测试：针对使用 Web 应用软件进行的一系列测试活动。功能测试，用户界面(UI)测试，兼容性测试，性能测试，安全测试，易用性测试，回归测试，手动测试，自动化测试，持续集成/持续交付

高可信软件：软件的可信性体现的是人们对软件性能的一种新的抽象追求，它其实也包括了正确性、可靠性、安全性、可用性、可控性、完整性、实时性和可预期性等许多传统特性。

嵌入式软件测试：1 硬件依赖性：最重要的目的是保证嵌入式软件能在特定的环境下更可靠地运行。
安全性和可靠性：要保证可靠性。
2 实时性：保证其实时性。
3 有限的资源：需要对内存进行测试。
4 要对相应的嵌入式产品进行测试。
5 不同标准和规范

普适计算环境下的软件测试：上下文背景是挥发性的和瞬态的，由于环境的自发的互操作，背景值可能不断变化，程序的行为不仅依赖于输入，还依赖于环境，普适计算环境中的物理背景本质上是不稳定的，普适计算软件中的各个计算实体（服务或代理）是独立的和自我为中心的
自动化测试，模拟与仿真，用户体验测试，验收测试，持续集成与测试

云测试：是一种软件测试形式，其中网络应用程序使用云计算环境来模拟现实世界的用户流量作为负载和压力测试的一种方式。灵活性和可扩展性，成本效益，快速部署，易于协作，多样化测试

物联网测试：