ISTQB 测试人员认证
初级（基础级）大纲

2011 版

中文修订版本 1（2015 年 5 月 6 日）

国际软件测试认证委员会

中文版的翻译编辑和出版统一由 ISTQB 授权的 CSTQB 负责

版权标志

ISTQB 测试人员认证初级大纲修订历史:

版本	日期	备注
ISTQB 2011	自2011年4月1日起生效	ISTQB测试人员认证初级大纲 修订版-见附录E-版本说明
ISTQB 2010	自2010年3月30日起生效	ISTQB测试人员认证初级大纲 修订版-见附录E-版本说明
ISTQB 2007	2007年5月1日	ISTQB测试人员认证初级大纲 修订版
ISTQB 2005	2005年7月1日	ISTQB测试人员认证初级大纲
ASQF V2.2	2003年7月	ASQF初级大纲 V2.2
ISEB V2.0	1999年2月25日	ISEB软件测试初级大纲 V2.0 1999年2月25日

ISTQB 测试人员认证初级大纲中文版本的修订历史:

版本	日期	备注
ISTQB2011中文版修订版1	2015年5月6日	根据“软件测试专业术语中英文对照表 v2.4 修订版本 1”修订本文档中的部分术语，以保持一致性 负责人：沈建雄
ISTQB2011 中 文 版 V13	2011年12月15日	2011 年基础级大纲和术语工作组根据软件测试专业术语中英文对照表 v2.1,修订本文档中的部分术语，以保持一致性
ISTQB2011 中 文 版 v12	2011年7月1日	按照 ISTQB 2011 版进行修订 工作组组长：沈建雄
ISTQB2010中文版评审版 V01	2010年8月12日	按照 ISTQB 2010 版进行修订 负责人：周震漪
ISTQB2007中文版	2008年8月28日	按照英文 2007 版翻译，力求与原版保持一致。 责任负责人：周震漪
CSTQB 2007版	2007年7月1日	
CSTQB 2005版	2005年7月1日	

目　录

致谢

国际软件测试认证委员会初级大纲工作组（2011 版）有：Thomas Müller（chair），Debra Friedenberg。编撰本大纲的核心团队感谢评审团队：Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Rioudu Cosquier HansSchaefer, StephanieUlrich, ErikvanVeenendaal, 以及给于意见和建议的所有国家委员会。参加 2011 中文版修订的专家有（按姓氏拼音排序）静国玥、刘晓更、沈建雄、徐文叶、周震漪等。

国际软件测试认证委员会初级课程大纲工作组（2010 版）有：Thomas Müller（chair），Rahul Verma, Martin Klonk and Armin Beer。编撰本书的核心团队感谢评审团队：Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, TuulaPääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Wiliams, Erik van Veendendaal 以及给于意见和建议的所有国家委员会。中国编写组合评审组是由 CSTQB 的专家组成员组成的工作组：周震漪、沈建雄、崔启亮等。

国际软件测试认证委员会初级课程大纲工作组（2007 版）有：Thomas Müller（chair），Dorothy Graham, Debra Friedenberg, and Erik van Veendendaal 和评审组成员 Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, Wonil Kwon 以及给于意见和建议的所有国家委员会。参加中文版的专家有（按姓氏拼音排序）：曹静、杜庆峰、刘琴、刘小茵、马均飞、吴晓臻、郑文强、周震漪等。

国际软件测试认证委员会初级课程大纲工作组（2005 版）有：Thomas Müller（chair），Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veendendal. 编撰本书的核心团队感谢评审团队以及对当前课程大纲提供建议的所有国家委员会。

大纲简介

本文档的目的

本大纲是国际软件测试认证初级水平的中文版课程大纲。国际软件测试认证委员会（以下简称ISTQB）提供标准的课程大纲给各个国家考试委员会。各国委员可以在规定的权限内将大纲授权给培训机构以及以当地的语言组织认证考试的考题并提供给考试机构。培训机构借助于本大纲自行负责编写课件并采取适当的授课方法。同时，本课程大纲能为报考者备考提供帮助。

关于本课程大纲的修订历史和背景知识信息，可以参考附录 A。

软件测试人员初级认证

初级资质认证可以针对和软件测试工作相关的任何角色，包括测试人员、测试分析员、测试工程师、测试顾问、测试经理、用户验收测试人员和软件开发人员等。同时本初级资质认证也适合希望对软件测试有所了解的人，比如项目经理、质量经理、软件开发经理、业务分析师、IT 主管和管理顾问等。拥有初级资质证书后，可以继续向高级软件测试资质认证努力。

学习目标和认知水平

在课程大纲中，每个章节都会提供相应的认知水平要求：

- K1：牢记
- K2：理解
- K3：应用
- K4：分析

更多的细节和学习目标的例子可以参考附录 B。

需要牢记章节标题下面列出的所有条目（K1），即使在学习目标中没有非常明显的涉及。

关于考试

初级认证考试的内容将基于本课程大纲的内容。但是考试中涉及到的问题，可能需要用到课程大纲的一个甚至多个章节的知识。考试的范围覆盖本课程大纲的所有章节。

考题的形式是多项选择题。

考试可以作为认证培训课程的一部分，也可以单独参加考试（例如：在授权的考试中心）。

认证培训课程的完成不作为考试的先决条件。

授权

ISTQB 国家委员会（在中国是 CSTQB）可以授权那些遵照大纲编写课程材料的培训机构。授权指南可以从国家委员会（在中国是 CSTQB）获取，或者从开展授权的机构或团体获取。被授权课程需要遵照本大纲，并且被允许将 ISTQB 考试作为课程的一部分。

更多关于培训机构的指南可以参考附录 D。

细节

本大纲的详细等级目标是全球一致性的教学形式和考核。为了达到这个目标，本课程大纲由下面几部分组成：

- 总体教学目标，描述了初级水平资质认证的目的；
- 培训的系列知识，包括详细的描述并且如有需要可参考额外的资料；
- 各个知识领域的学习目标，描述知识产出和将要达到的认知水平；
- 列出学员必须能够理解和掌握的知识条目；
- 描述主要的教学理念，包括已经被接受的文献或标准等资源。

本课程大纲并没有包含软件测试的整个知识领域，只是提供了初级课程需要覆盖的具体方面。

课程大纲的结构

本课程大纲主要由 6 大章组成。每一章的第一级标题明确了本章最终的学习目标和建议授课的时间。比如：

2. 软件生命周期中的测试（K2）（115 分钟）

显示了第二章的学习目标是 K1（当更高的级别已经显示时可以假设也应达到该级别）和 K2（但不是 K3），建议花费 115 分钟来进行本部分的教学。课程大纲的每一章都由几节组成。每节同样会由相应的学习目标和所需的教学时间组成。没有规定具体时间的子章节，其教学的时间包含在了整个章节之中。

1. 软件测试基础（K2）（155 分钟）

软件测试基础的学习目标

本章的学习目标将明确完成每个模块的学习后，学员能做什么。

1.1 为什么需要软件测试？（K2）

LO-1.1.1 通过具体的例子，描述软件中的缺陷会以什么样的方式损害个人、损害环境或者损害公司利益（K2）。

LO-1.1.2 区分引起缺陷的根本原因及其影响（K2）。

LO-1.1.3 通过举例的方式说明为什么需要测试（K2）。

LO-1.1.4 描述为什么测试是质量保证的一部分，通过举例说明测试是如何提高软件质量的（K2）。

LO-1.1.5 通过举例来理解和比较术语错误、缺陷、故障、失效的概念以及相应的定义（K2）。

1.2 什么是测试（K2）

LO-1.2.1 认识测试的总体目标（K1）。

LO-1.2.2 举例说明软件生命周期中不同阶段的测试目标（K2）。

LO-1.2.3 区分测试与调试的不同（K2）。

1.3 软件测试的基本原则（K2）

LO-1.3.1 说明测试的七个基本原则（K2）。

1.4 基本的测试过程（K1）

LO-1.4.1 认识从计划开始到结束过程的五个基本测试活动和各自的任务（K1）。

1.5 测试的心理学（K2）

LO-1.5.1 认识影响测试成功与否的心理因素（K1）。

LO-1.5.2 对比测试人员和开发人员的思维方式的差异（K2）。

1.1. 为什么需要测试（K2）（20 分钟）

术语

缺陷（bug）、缺陷（defect）、错误（error）、失效（failure）、故障（fault）、错误（mistake）、质量（quality）、风险（risk）。

1.1.1. 软件系统的重要性（K1）

在当今社会，软件系统越来越成为生活中不可或缺的一部分，包括从商业应用（比如银行系统）到消费产品（比如汽车）各个领域。然而，很多人都有这样的经历：软件并没有按照预期进行工作。软件的不正确执行可能会导致许多问题，包括资金、时间和商业信誉等的损失，甚至导致人员的伤亡。

1.1.2. 引起软件缺陷的原因（K2）

所有的人都会犯错误（error, mistake），因此在由人设计的程序代码或文档中也会引入缺陷（defect, fault, bug）。当存在缺陷的代码被执行时，系统就可能无法实现期望的功能（或者实现了未期望的功能），从而引起软件失效（failure）。虽然在软件、系统或文档中的缺陷可能会引起失效，但并非所有的缺陷都是如此。

产生缺陷的原因是多种多样的：人们本身容易犯错误、时间的压力、复杂的代码、复杂的系统架构、技术的革新、以及/或者许多系统之间的交互等。

失效也可能是由于环境条件引起的：例如：辐射、电磁场和污染等都有可能引起固件中的故障，或者由于硬件环境的改变而影响软件的执行。

1.1.3. 测试在软件开发，维护和运行中所担当的角色（K2）

对软件系统和文档进行严格的测试，可以减少软件系统在运行环境中的风险，假如在软件正式发布之前发现和修正了缺陷，可以提高软件系统的质量。

进行软件测试也可能是为了满足合同或法律法规的要求，或者是为了满足行业标准的要求。

1.1.4. 测试和质量（K2）

可以根据测试中所发现的缺陷，对软件功能和非功能性需求以及特性（例如：可靠性、可用性、效率、可维护性和可移植性）进行度量，从而评估软件质量。更多关于非功能测试方面的信息，可以参考第二章。更多关于软件特征的信息，可以参考“软件工程－软件产品质量（ISO 9126）”。

当测试发现很少或者没有发现缺陷的时候，测试就会帮助树立对于软件质量的信心。一个设计合理的测试过程完成并顺利通过，可以降低整个系统存在问题的风险。而对测试过程中发现的缺陷

进行了修正，则软件系统的质量就会提高。

我们应该从以前的项目中吸取经验教训。通过分析在其他项目中发现的缺陷和引起缺陷的根本原因，可以改进软件开发过程。过程的改进反过来可以预防相同的缺陷再次发生，从而提高以后系统的质量。这是质量保证工作的一方面。

测试应该作为开发过程中质量保证工作的不可或缺的一部分（与开发标准、培训和缺陷分析一样）。

1.1.5. 测试是否充分（K2）

在判断测试是否足够时，需要考虑下面的因素：风险（包括技术风险、商业产品风险和项目风险等）以及项目在时间和预算上的限制等（有关风险的详细内容参见第五章）。

为了进入下一个开发过程，或将系统交付给用户，测试需要给利益相关者提供足够的信息，帮助他们决定是否发布被测软件或系统。

1.2. 什么是测试（K2）（30 分钟）

术语

调试(debugging)、需求(requirement)、评审(review)、测试用例(test case)、测试(testing)、测试目标（test objective)。

背景

在一般人的理解当中，测试活动只包含了运行测试，也就是执行软件。但实际上这只是测试的一部分，而不是测试的所有活动。

测试活动包含了测试执行之前和之后的一些活动，包括计划和控制、选择测试条件、设计和执行测试用例、检查测试结果、评估出口准则、报告测试过程及被测系统、在一个测试阶段完成后要进行测试结束或总结工作。测试同时也包括文档的评审（包括源代码）和执行静态分析。

动态测试和静态测试这两种手段都可以达到相似的目标，即以提供信息来改进被测试软件系统的质量，以及改善开发和测试的过程。

测试可以有如下目标：

- 发现缺陷；
- 增加对质量的信心；
- 为决策提供信息；
- 预防缺陷。

在软件生命周期的早期进行测试设计的思维过程和活动（通过测试设计来检验测试依据），可以避免将缺陷引入代码。对文档的评审（例如需求文档）并识别和解决问题也有助于防止在代码中出现缺陷。

不同的测试阶段，需要考虑不同的测试目标。比如，在开发测试中，如组件测试、集成测试和系统测试等，测试的主要目标是尽可能的发现失效，从而识别和修正尽可能多的缺陷。在验收测试中，测试的主要目标是确认系统是否按照预期工作，是建立满足了需求的信心。而在有些情况下，测试的主要目标是对软件的质量进行评估(不是为了修正缺陷)，从而为利益相关者提供这样的信息：在给定的时间点发布系统版本可能存在的风险。而维护测试通常是为了验证在开发过程中的软件变更是否引入新的缺陷。在运行测试阶段，测试的主要目标是为了评估系统的特征，比如可靠性或可用性等。

调试和测试是两个不同的概念。动态测试可以发现由于软件缺陷引起的失效。而调试是一种发现，分析，清除引起失效原因的开发活动，随后由测试员进行的再测试是为了确认修改的代码已经解决了失效问题。每个活动的职责是截然不同的，即测试员进行测试，开发人员进行调试。

测试的过程和测试活动将在 1.4 节讲述。

1.3.测试的基本原则（K2）（35 分钟）

术语

穷尽测试（exhaustive testing）。

基本原则

在过去的 40 年中，软件测试界提出了很多测试原则，并且提供了适合所有测试的一些通用的测试指南。

原则 1 — 测试显示存在缺陷

测试可以显示存在缺陷，但不能证明系统不存在缺陷。测试可以减少软件中存在未被发现缺陷的可能性，但即使测试没有发现任何缺陷，也不能证明软件或系统是完全正确的。

原则 2 — 穷尽测试是不可行的

除了小型项目，进行完全（各种输入和前提条件的组合）的测试是不可行的。通过运用风险分析和不同系统功能的测试优先级，来确定测试的关注点，从而替代穷尽测试。

原则 3 — 测试尽早介入

为了尽早发现缺陷，在软件或系统开发生命周期中，测试活动应该尽可能早的介入，并且应该将关注点放在已经定义的测试目标上。

原则 4 — 缺陷集群性

测试工作的分配比例应该与预期的和后期观察到的缺陷分布模块相适应。少数模块通常包含大部分在测试版本中发现的缺陷或失效。

原则 5 — 杀虫剂悖论

采用同样的测试用例多次重复进行测试，最后将不再能够发现新的缺陷。为了克服这种“杀虫剂悖论”，测试用例需要进行定期评审和修改，同时需要不断增加新的不同的测试用例来测试软件或系统的不同部分，从而发现潜在的更多的缺陷。

原则 6 — 测试活动依赖于测试背景

针对不同的测试背景，进行不同的的测试活动。比如，对安全关键的软件进行测试，与对一般的电子商务软件的测试是不一样的。

原则 7 — 不存在缺陷（就是有用系统）的谬论

假如系统无法使用，或者系统不能完成客户的需求和期望，发现和修改缺陷是没有任何意义的。

1.4. 基本的测试过程（K1）（35 分钟）

术语

确认测试（confirmation testing）、再测试（retesting）、出口准则（exit criteria）、事件（incident）、回归测试（regression testing）、测试依据（test basis）、测试条件（test condition）、测试覆盖（test coverage）、测试数据（test data）、测试执行（test execution）、测试日志（test log）、测试计划（test plan）、测试规程（test procedure）、测试方针（test policy）、测试套件（test suite）、测试总结报告（test summary report）、测试件（testware）。

背景

测试最显而易见的活动是测试的执行。但是为了提高效率和有效性，在测试计划中，同样需要花费比较多的时间用于计划测试活动、设计测试用例、准备测试的执行和评估测试的状态。

基本的测试过程主要由下面一些活动组成：

- 测试计划和控制；
- 测试分析和设计；
- 测试实现和执行；
- 评估出口准则和报告；
- 测试结束活动。

虽然上面这些活动在逻辑上是有连续的，但在整个测试过程中它们可能会重叠或同时进行。通常在相应的系统或项目环境下剪裁这些主要活动行为是必需的。

1.4.1. 测试计划和控制阶段（K1）

测试计划的主要活动是：识别测试任务、定义测试目标以及为了实现测试目标和任务确定必要的测试活动。

测试控制是持续进行的活动：通过对测试实际进度和测试计划之间的比较，报告测试的状态，包括与计划之间存在的偏差。测试控制包括在必要的时候采取必要的措施来满足测试的任务和目标。需要在项目的整个生命周期中对测试活动进行监督，以达到控制测试过程的目的。同时，测试计划的制定也需要考虑测试监控活动的反馈信息。

测试计划和控制阶段的任务将在第五章讲述。

1.4.2. 测试分析和设计阶段（K1）

测试分析和设计是将概括的测试目标转化为具体的测试条件和测试用例的一系列活动。

测试分析和设计阶段的主要任务：

- 评审测试依据（比如需求、软件完整性级别[1]（风险等级）、风险分析报告、系统架构、设计和接口说明）；
- 评估测试依据和测试对象的可测性；
- 通过对测试项、规格说明、测试对象行为和结构的分析，识别测试条件并确定其优先级；
- 设计测试用例并确定优先级；
- 确定测试条件和测试用例所需要的测试数据；
- 规划测试环境的搭建和确定测试需要的基础设施和工具；
- 创建测试依据和测试用例间的双向可追溯性。

1.4.3. 测试实现和执行阶段（K1）

测试实现和执行阶段的主要活动包括：通过特定的顺序组织测试用例来完成测试规程和脚本的设计，并且包括测试执行所需的其他任何信息，以及测试环境的搭建和运行测试。

测试实现和执行阶段的主要任务：

- 测试用例的开发、实现并确定它们的优先级。（包括识别测试数据）；
- 开发测试规程并确定优先级，创建测试数据，同时也可以准备测试用具和设计自动化测试脚本；
- 根据测试规程创建测试套件，以提高测试执行的效率；
- 确认已经正确搭建了测试环境；
- 确认并更新测试依据和测试用例间的双向可追溯性；
- 根据计划的执行顺序，通过手工或使用测试执行工具来执行测试规程；
- 记录测试执行的结果，以及被测软件、测试工具和测试件的标识和版本；
- 将实际结果和预期结果进行比较；
- 对实际结果和预期结果之间的差异，作为事件上报，并且进行分析以确定引起差异的原因（例如：代码缺陷、具体测试数据缺陷、测试文档缺陷、或测试执行的方法有误等）；
- 缺陷修正后，重新进行测试活动。比如通过再次执行上次执行失败的用例来确认缺陷是否已经被修正（确认测试）。执行修正后的测试用例或执行一些测试用例来确保缺陷的修正没有对软件未修改的部分造成不良影响或对于缺陷的修正没有引发其他的缺陷（回归测试）。

1.4.4. 评估出口准则和报告（K1）

评估出口准则是将测试的执行结果和已经定义的测试目标进行比较的活动。这个活动在各个测试级别上都需要进行。（参见章节 2.2）

评估测试出口准则的主要任务：

- 按照测试计划中定义的测试出口准则检查测试日志；
- 评估是否需要进行更多的测试，或是否需要更改测试的出口准则；

[1]软件完整性级别：为了向软件利益相关者反映软件重要性所定义的软件遵守的或必须遵守的一系列利益相关者选定的软件和/或软件系统特性的级别。（例如，软件复杂度、风险评估、产品安全性级别、产品防止对程序和数据未授权访问的能力级别、预期性能、可靠性、成本）

- 为利益相关者提供一个测试总结报告。

1.4.5. 测试结束活动（K1）

测试结束活动就是从已完成的测试活动中收集和整合有用的数据，这些数据可以是测试经验、测试件、影响测试的因素和其他数据。在以下几种情况下需要执行测试结束活动，例如：当软件系统正式发布、当一个测试项目完成（或取消）、当达到一个里程碑或当一个维护版本完成时。

测试结束活动的主要任务：

- 检查提交了哪些计划的可交付产品；
- 事件报告是否关闭、或对未关闭的事件报告提交变更需求；
- 记录系统的验收；
- 记录和归档测试件、测试环境和测试基础设备，以便以后的重复使用；
- 移交测试件到维护部门；
- 分析和记录所获得的经验教训，用于以后的项目和测试成熟度改进；
- 使用为测试成熟度的提高所收集的信息。

1.5. 测试的心理学（K2）（25 分钟）

术语

错误推测（error guessing）、独立（independence）。

背景

在测试和评审中使用的思维方式，与在项目分析和开发中使用的不同。具有正确思维方式的开发人员可以测试他们自己写的代码。但通常将此职责从开发人员分离给测试人员，有助于开发人员集中精力，并且具有以下额外优势，例如：通过培训和使用专业的测试资源获得的独立的观点。独立测试可以应用于任何测试级别。

一定程度的独立（可以避免开发人员对自己代码的偏爱），通常可以更加高效地发现软件缺陷和软件存在的失效。但独立不能替代对软件的熟悉和经验，开发人员同样也可以高效的在他们自己的代码中找出很多缺陷。

在这可以从低到高定义不同级别的独立：

- 测试由软件本身编写的人员来执行（低级别的独立）；
- 测试由一个其他开发人员（如来自同一开发小组）来执行；
- 测试由组织内的一个或多个其他小组成员（如独立的测试小组）或测试专家（如可用性或性能测试专家）来执行；
- 测试由来自其他组织或其他公司的成员来执行（如测试外包或其他外部组织的鉴定）。

测试的目标驱使着小组成员和项目的活动。小组成员将根据管理层或其他利益相关者设定的目标对他们的计划进行调整，比如需要发现更多的缺陷，或确认软件是否满足其目标。因此，对测试的目标进行清晰的设定是非常重要的。

测试过程中发现的失效，可能会被看成是测试员对产品和作者的指责。因此，测试通常被认为是破坏性的活动，即使它对于管理产品风险非常有建设性作用。在系统中发现失效需要测试员具有一颗好奇心、专业的怀疑态度、一双挑剔的眼睛、对细节的关注、与开发人员良好的沟通能力以及对常见的错误进行判断的经验。

假如可以用建设性的态度对发现的缺陷或失效进行沟通，就可以避免测试员、分析人员、设计人员和开发人员之间的不愉快。这个道理不仅适用于文档的评审过程，同样也适用于测试过程。

在以建设性的方式讨论缺陷、进度和风险时，测试员和测试的负责人都需要具有良好的人与人之间沟通的能力。对于软件代码或文档的作者，缺陷的信息可以帮助他们来提高他们的技术水平。如果在测试阶段发现和修复缺陷，就可以为在后期（例如在正式的使用时）节省时间和金钱，而且可以降低风险。

沟通方面的问题经常会发生，特别是当测试员只是被视为不受欢迎的缺陷消息的传递者的时候。然而可以使用下面的一些方法来改善测试员和其他小组成员之间的沟通和相互关系：

- 以合作而不是争斗的方式开始项目，时时提醒项目的每位成员：共同目标是追求高质量的产品；
- 对产品中发现的问题以中性的和以事实为依据的方式来沟通，而不要指责引入这个问题的

小组成员或个人。比如，客观而实际地编写缺陷报告和评审发现的问题；

- 尽量理解其他成员的感受，以及他们为什么会有这种反应；
- 确信其他成员已经理解你的描述，反之亦然。

1.6.职业道德（10 分钟）

在软件测试中包含了使个人可以获取保密的、授权的信息。为保证信息规范化使用，需要遵循必要的职业道德。ISTQB 借鉴、引用了 ACM 和 IEEE 对于工程师道德规范，陈述职业道德规范如下：

公共 – 认证测试工程师应当以公众利益为目标。

客户和雇主 – 在保持与公众利益一致的原则下，认证测试工程师应注意满足客户和雇主的最高利益。

产品 – 认证测试工程师应当确保他们提供的（在产品和系统中由他们测试的）发布版本符合最高的专业标准。

判断 – 认证测试工程师应当维护他们职业判断的完整性和独立性。

管理 – 认证软件测试管理人员和测试领导人员应赞成和促进对软件测试合乎道德规范的管理。

专业 – 在与公众利益一致的原则下，认证测试工程师应当推进其专业的完整性和声誉。

同事 – 认证测试工程师对其同事应持平等和互助和支持的态度，并促进与软件开发人员的合作。

自我 – 认证测试工程师应当参与终生职业实践的学习，并促进合乎道德的职业实践方法。

参考文献：

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1998, Myers, 1979
1.4 Hetzel, 1998
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1988

2. 软件生命周期中的测试（K2）（115 分钟）

软件生命周期中的测试的学习目标

学习目标将明确完成下面模块的学习后，学员能做什么。

2.1 软件开发模型（K2）

LO-2.1.1 应用具体项目和产品类型的例子解释在开发生命周期中开发、测试活动与工作产品之间的关系（K2）。

LO-2.1.2 知道必须根据项目背景和产品特征来选择软件开发的模型（K1）。

LO-2.1.3 理解在任何生命周期模型中良好的测试应具备的特征（K1）。

2.2 测试级别（K2）

LO-2.2.1 比较不同测试级别之间的区别：测试的主要目的、典型的测试对象、典型的测试目标（功能性的或结构性的）、相关的工作产品、测试的人员、识别缺陷和失效的种类（K2）。

2.3 测试类型（K2）

LO-2.3.1 通过举例比较四种不同的软件测试类型（功能测试、非功能测试、结构测试和与变更相关的测试）（K2）。

LO-2.3.2 明白功能测试和结构测试可以应用在任何测试级别（K1）。

LO-2.3.3 根据非功能需求来识别和描述非功能测试的类型。（K2）。

LO-2.3.4 根据对软件系统结构或构架的分析来识别和描述测试的类型（K2）。

LO-2.3.5 描述确认测试和回归测试的目的（K2）。

2.4 维护测试（K2）

LO-2.4.1 比较维护测试（一个现存系统的测试）与一个新的应用软件的测试在测试类型、测试的触发和测试规模等方面的区别（K2）。

LO-2.4.2 识别维护测试的原因（由于修改、移植或退役等因素）（K1）。

LO-2.4.3 描述回归测试和变更的影响分析在软件维护中的作用（K2）。

2.1. 软件开发模型（K2）（20 分钟）

术语

商业现货软件（commercial off the shelf（COTS））、迭代-增量开发模型（iterative-incremental development model）、确认（validation）、验证（verification）、V-模型（V-model）。

背景

测试不是孤立存在的，测试活动与开发活动息息相关。不同的开发生命周期模型需要不同的测试方法。

2.1.1. V 模型（顺序开发模型）（K2）

虽然存在多种多样的 V-模型，但典型的 V-模型一般有四种测试级别，分别与四种开发级别相对应。

在本课程大纲中，这四种测试级别是：

- 组件/单元测试；
- 集成测试；
- 系统测试；
- 验收测试。

实际上，V-模型的测试级别可能会比上面提到的 4 种多，也可能少，或者有不同的测试级别，这取决于不同的项目和软件产品。比如，在组件测试后，可能有组件集成测试，在系统测试后有系统集成测试。

在开发过程中生成的软件工作产品（比如业务场景、用例、需求规格说明、设计文档和代码）常常作为一种或多种测试级别的测试基础。通用的工作产品可以参考能力成熟度模型集成 CMMI 或软件生命周期过程（IEEE/IEC 12207）。验证和确认（早期的测试设计）可以在软件工作产品的开发过程中进行。

2.1.2. 迭代-增量开发模型（K2）

迭代-增量开发模型由需求建立、设计、构建和测试等一系列相对较短的开发周期构成。比如：原型开发、快速应用开发（RAD）、统一软件开发过程（RUP）和敏捷开发模型等。在每次迭代过程中，对迭代产生的系统可能需要在不同的测试级别上进行测试。通过将增量模块加入到以前开发的模块中，形成一个逐渐增大的系统，这个系统同样需要进行测试。在完成第一次迭代后，对所有的迭代进行回归测试会变得越来越重要。验证和确认可以在每个增量模块中进行。

2.1.3. 生命周期模型中的测试（K2）

在任何生命周期模型中，良好的测试都应该具有下面几个特点：

- 每个开发活动都有相对应的测试活动；
- 每个测试级别都有其特有的测试目标；
- 对于每个测试级别，需要在相应的开发活动过程中进行相应的测试分析和设计；
- 在开发生命周期中，测试员在文档初稿阶段就应该参与文档的评审。

根据项目的特征或系统的架构，可以对测试级别进行合并或重新进行组合。比如，对于商业现货软件（COTS）产品集成到某个系统，购买者可以在系统级别（例如：与基础设施集成、与其他系统的集成或与系统应用的集成）进行集成测试和验收测试（功能的和/或非功能的测试，用户和/或运行测试等）。

2.2. 测试级别（K2）（60 分钟）

术语

Alpha 测试（alpha testing）、Beta 测试（beta testing）、组件测试（componenttesting）（也称为单元测试、模块测试或程序测试）、驱动器（driver）、现场测试（field testing）、功能需求（functional requirement）、集成（integration）、集成测试（integration testing）、非功能需求（non-functional requirement）、健壮性测试（robustness testing）、桩（stub）、系统测试（system testing）、测试环境（testenvironment）、测试级别（test level）、测试驱动开发（test-driven development）、用户验收测试（user acceptance testing）。

背景

对于每个测试级别，都需要明确下面的内容：测试的总体目标、测试用例设计需要参考的工作产品（即测试的依据）、测试的对象（即测试什么）、发现的典型缺陷和失效、对测试用具的需求、测试工具的支持、专门的方法和职责等。

在测试计划中应当考虑是否对系统配置数据进行测试。

2.2.1. 组件测试/单元测试（K2）

测试依据：

- 组件需求说明；
- 详细设计文档；
- 代码。

典型测试对象：

- 组件；
- 程序；
- 数据转换/移植程序；
- 数据库模型。

在独立可测试的软件中（模块、程序、对象和类等），可以通过组件测试发现缺陷，以及验证软件功能。根据开发生命周期和系统的背景，组件测试可以和系统的其他部分分开，单独进行测试。在组件测试过程中，会使用到桩、驱动器和模拟器。

组件测试可能包括功能测试和特定的非功能特征测试，比如资源行为测试（如内存泄漏）或健壮性测试和结构测试（比如分支覆盖）。根据工作产品，例如组件规格说明、软件设计或数据模型等设计测试用例。

通常，通过开发环境的支持，比如组件测试框架或调试工具，组件测试会深入到代码中，而且实际上设计代码的开发人员通常也会参与其中。在这种情况下，一旦发现缺陷，就可以立即进行修改，而不需要正式的缺陷管理过程。

组件测试的一个方法是在编写代码之前就完成编写和自动化测试用例，这称之为测试优先的方法或测试驱动开发。这是高度迭代的方法，并且取决于如下的循环周期：测试用例的开发、构建和集成小块的代码，执行组件测试，修正任何问题并反复循环，直到它们全部通过测试。

2.2.2. 集成测试（K2）

测试依据：

- 软件和系统设计文档；
- 系统架构；
- 工作流；
- 用例。

典型测试对象：

- 子系统；
- 数据库实现；
- 基础结构；
- 接口；
- 系统配置和配置数据。

集成测试是对组件之间的接口进行测试，以及测试一个系统内不同部分的相互作用，比如操作系统、文件系统、硬件或系统之间的接口。

对于集成测试，可以应用多种集成级别，也可以根据不同的测试对象规模采用不同的级别，比如：

1. 组件集成测试对不同的软件组件之间的相互作用进行测试，一般在组件测试之后进行。
2. 系统集成测试对不同系统或软硬件之间的相互作用进行测试，一般在系统测试之后进行。在这种情况下，开发组织/团体通常可能只控制自己这边的接口，这就可能存在风险。按照工作流执行的业务操作可能包含了一系列系统，因此跨平台的问题可能至关重要。

集成的规模越大，就越难在某一特定的组件或系统中定位缺陷，从而增加了风险并会花费额外的更多时间去发现和修理这些故障。

系统化集成的策略可以根据系统结构（例如自顶向下或自底向上）、功能任务集、事务处理顺序或系统和组件的其他方面等来制定。为了能方便快速地隔离故障和定位缺陷，集成程度应该逐步增加，而不是采用“大爆炸”式的集成。

测试特定的非功能特征（比如性能）也可以包含在系统集成测试中。

在集成的每个阶段，测试员只是把精力集中在集成本身。举例来说，假如集成模块 A 和模块 B，测试人员是应该关注两个模块之间的交互，而不是每个模块的功能。功能测试和结构测试方法都可以应用在集成测试。

在理想情况下，测试员应该理解系统的架构，从而可以影响相应的集成计划。假如集成测试计划是在组件或系统生成之前制定，则可以根据对集成最有效率的顺序来进行设计。

2.2.3. 系统测试（K2）

测试依据：

- 系统和软件需求规格说明；
- 用例；
- 功能规格说明；
- 风险分析报告。

典型测试对象：

- 系统、用户手册和操作手册；
- 系统配置和配置数据。

系统测试关注的是在开发项目或程序中定义的一个完整的系统/产品的行为。在主测试计划和/或在其所处的测试级别的级别测试计划内应该明确测试范围。

在系统测试中，测试环境应该尽量和最终的目标或生产环境相一致，从而减少不能发现和环境相关的失效的风险。

系统测试可能包含基于不同方面的测试：基于风险评估的、基于需求规格说明的、基于业务过程的、基于用例的、或基于其他对系统行为的更高级别描述或模型的、基于与操作系统的相互作用的、基于系统资源等的测试。

系统测试应该对系统功能和非功能需求进行研究。需求可以以文本形式或模型方式描述。同时测试员也需要面对需求不完全或需求没有文档化的情况。针对功能需求的系统测试开始时可以选择最适合的基于规格说明的测试即黑盒技术来对系统进行测试。比如：可以根据业务准则描述的因果组合来生成决策表。基于结构的技术即白盒测试技术，可以评估测试的覆盖率，可以基于评估覆盖一个结构元素，如菜单结构或者页面的导航等的完整性。（参见第 4 章）

系统测试通常由独立的测试团队进行。

2.2.4. 验收测试（K2）

测试依据：

- 用户需求；
- 系统需求；
- 用例；
- 业务流程；
- 风险分析报告。

典型测试对象：

- 基于完全集成系统的业务流程；
- 操作与维护流程；
- 用户处理过程；
- 结构；
- 报告；
- 配置数据。

验收测试通常是由使用系统的用户或客户来进行，同时系统的其他利益相关者也可能参与其中。

验收测试的目的是建立对系统、系统的某部分或特定的系统非功能特征建立信心。发现缺陷不是验收测试的主要目标。验收测试可以用来评估系统对于部署和使用的准备情况，但是验收测试不一定是最后级别的测试。比如，可能会在进行某个系统验收测试之后，进行大规模的系统集成测试。

验收测试可以在多个测试级别上进行，比如：

- 商业现货软件（COTS）产品可以在安装或集成时进行验收测试；
- 组件的可用性验收测试可以在组件测试中进行；
- 增加新功能的验收测试可以在系统测试之前进行。

验收测试有下面几种典型的类型：

用户验收测试

通常由商业用户验证系统的可用性。

操作（验收）测试

系统操作验收测试由系统管理员来进行，测试内容主要包括：

- 系统备份/恢复测试；
- 灾难恢复测试；
- 用户管理测试；
- 维护任务测试；
- 数据加载和移植活动；
- 安全漏洞阶段性检查。

合同和法规性验收测试

合同验收测试根据合同中规定的生产客户定制软件的验收准则，对软件进行测试。应该在合同拟定时定义验收准则。法规性验收测试根据必须要遵守的法律法规来进行测试，比如政府、法律和安全方面的法律法规。

Alpha 和 Beta（或现场）测试

在软件产品正式商业销售之前，市场或商业现货软件开发人员希望从市场中潜在的或已经存在的客户中得到关于软件的反馈信息。Alpha 测试通常在开发组织现场进行，但测试并非由开发团队执行。Beta 测试或实地测试，是在客户或潜在客户现场进行并由他们执行。

有些组织也可能使用不同的术语，比如在系统正式移交给客户之前或之后进行的测试分别称为工厂验收测试和现场验收测试等。

2.3. 测试类型（K2）40 分钟

术语

黑盒测试(black-box testing)、代码覆盖(code coverage)、功能测试(functional testing)、互操作性测试（interoperability testing）、负载测试（load testing）、可维护性测试（maintainability testing)、性能测试（performance testing)、可移植性测试（portability testing)、可靠性测试(reliability testing)、安全性测试(security testing)、压力测试(stress testing)、结构测试(structural testing)、可用性测试(usability testing)、白盒测试(white-box testing)。

背景

根据特定的测试目标或测试原因，一系列测试活动可以旨在来对软件系统（或系统的一部分）进行测试。

每种测试类型都会针对特定的测试目标：

- 可能是测试软件所实现的功能；
- 也可能是非功能的质量特征，比如可靠性或可用性；
- 软件或系统的结构或架构；
- 相关变更，如确认缺陷已被修改（确认测试）以及更改后是否引入新的缺陷（回归测试）。

一个软件的模型可以用来开发和/或应用在基于结构的测试(例如，控制流模型或菜单结构模型)、非功能测试（性能模型、可用性模型、安全威胁建模）和功能测试（过程流模型、 状态转换模型或简明语言规范)。

2.3.1. 功能测试（K2）

系统、子系统或组件要实现的功能可以在工作产品中，如需求规格说明书、用户用例或功能规格说明书予以描述，不过也可能没有相应的文档。功能指的是系统能做什么。

功能测试基于功能和特征（在文档中描述的内容或测试员自己的理解）以及专门的系统之间的交互，可以在各个级别的测试中进行（例如组件测试可以基于组件的规格说明书)。

可以采用基于规格说明的技术，根据软件或系统的功能来设计测试条件和测试用例（参见第 4 章)。功能测试主要是考虑软件的外部表现行为（黑盒测试)。

安全性测试是功能测试的一种，它会对安全性相关的功能（比如防火墙）进行测试，从而检测系统和数据是否能抵御外部恶意的威胁，如病毒等。互操作性测试是另一种功能性测试，评估软件产品与其他一个或多个组件或系统交互的能力。

2.3.2. 软件非功能特征测试（非功能测试）（K2）

非功能测试包括但不限于：性能测试、负载测试、压力测试、可用性测试、可维护性测试、可靠性测试和可移植性测试。非功能性测试就是测试系统运行的表现如何。

非功能测试可以在任何测试级别上执行。术语“非功能测试”是指：为了测量系统和软件的特征，需要进行的测试。这些特征可以用不同尺度予以量化，比如进行性能测试来检验响应时间。这些非功能测试可以参考“软件工程－软件产品质量（ISO 9126）”中定义的质量模型。非功能测试关注的是软件的外部行为表现，通常采用黑盒测试设计技术来实现测试用例。

2.3.3. 软件结构/架构测试（结构测试）（K2）

可以在任何测试级别上进行结构测试（白盒测试）。结构测试技术最好在进行基于规格说明的测试之后使用，以便通过评估结构类型的覆盖来测量测试的完整性。

覆盖是指结构通过测试套件检验的程度，以项被覆盖的百分比来表示。假如覆盖率不是 100%，可能需要设计更多的测试用例，来测试被遗漏的项，从而提高测试的覆盖。有关覆盖技术参见第 4 章。

在所有的测试级别，特别是在组件测试和组件集成测试中，可以利用工具来测量代码内某些元素的覆盖率，比如语句覆盖和判定覆盖。结构测试也可以基于系统的结构，比如调用层次结构。

结构测试方法也同样可以运用到系统、系统集成或验收测试级别（比如业务模型或菜单结构）。

2.3.4. 与变更相关的测试（再测试和回归测试）（K2）

当发现和修改了一个缺陷后，应进行再测试以确定已经成功的修改了原来的缺陷，这称之为确认。调试（定位并修复缺陷）是一种开发活动，不是一种测试活动。

回归测试是对已被测过的程序在修改缺陷后进行的重复测试，以发现在这些变更后是否有新的缺陷引入或被屏蔽。这些缺陷可能存在于被测试的软件中，也可能在与之相关或不相关的其他软件组件中。当软件发生变更或者应用软件的环境发生变化时，需要进行回归测试。回归测试的规模可以根据在以前正常运行的软件中发现新的缺陷的风险大小来决定。

确认测试和回归测试应该可以重复进行。

回归测试可以在所有的测试级别上进行，同时适用于功能测试、非功能测试和结构测试。回归测试套件一般都会执行多次，而且通常很少有变动，因此将回归测试自动化是很好的选择。

2.4.维护测试（K2）（15 分钟）

术语

影响分析（impact analysis）、维护测试（maintenance testing）。

背景

软件系统一旦部署，通常会服务几年甚至几十年。在这期间，经常需要对软件系统、它的配置数据或它的环境进行修正、改变或扩展。软件版本提升计划对成功的维护测试至关重要。这里必须区分计划中的版本与补丁。维护测试是在一个现有的运行系统上进行，且一旦对软件或系统进行修改、移植或退役处理时，就需要进行维护测试。

修改可以是计划中的功能增加（例如：根据版本发布的计划）、修正和应急变更、环境的变化，比如计划中的操作系统或数据库升级、为商业现货软件计划升级或由于新发现或暴露的操作系统漏洞而打的补丁等。

为移植（如从一个平台移植到另外一个平台）而进行的维护测试应该包括新环境的运行测试，以及对变更以后的软件的运行测试。当数据从另一个应用程序移植到正在维护的系统时，需要移植测试（转换测试）。

为系统退役而进行的维护测试应该包括数据移植测试，或当数据要长时间的保存时还须存档测试。

除了对已变更的部分进行测试外，维护测试还包括对系统没有发生变更的其他部分进行回归测试。维护测试的范围取决于变更的风险、现有系统的规模和变更的大小。维护测试根据变更情况的不同，可以在某一测试级别或所有测试级别和测试类型上进行。确定变更如何影响现有系统的过程，称之为影响分析，它有助于决定实施回归测试的广度和深度。

如果规格说明遗失、过时或测试人员没有具备领域知识，进行维护测试将是一件困难的事情。

参考文献：

2.1.3 CMMI,Craig,2002,Hetzel,1998,IEEE 12207
2.2 Hetzel,1998
2.2.4 Copeland,2004,Myers,1979
2.3.1 Beizer,1990,Black,2001,Copeland,2004
2.3.2 Black,2001,ISO 9126
2.3.3 Beizer,1990,Copeland,2004,Hetzel,1988
2.3.4 Hetzel,1988,IEEE 829
2.4 Black,2001,Craig,2002,Hetzel,1988,IEEE 829

3. 静态技术（K2）60 分钟

静态技术的学习目标

学习目标将明确完成下面模块的学习后，学员能做什么。

3.1 静态技术和测试过程（K2）

LO-3.1.1 了解可以通过不同的静态技术来检查软件工作产品的质量（K1）。

LO-3.1.2 描述了在评估软件工作产品中运用静态技术的重要性和它的价值（K2）。

LO-3.1.3 结合测试对象、缺陷类型来说明静态测试技术与动态测试技术之间的不同，以及这些技术在软件生命周期中的作用。（K2）

3.2 评审过程（K2）

LO-3.2.1 理解典型的正式评审过程中的阶段、角色和职责定义（K1）。

LO-3.2.2 解释不同类型评审的区别：非正式评审、技术评审、走查和审查（K2）。

LO-3.2.3 解释影响评审成功的主要因素（K2）。

3.3 静态分析的工具支持（K2）

LO-3.3.1 理解通过静态分析能够识别的典型缺陷和错误，并与评审和动态测试之间进行比较（K1）。

LO-3.3.2 举例描述静态分析的主要优点（K2）。

LO-3.3.3 列出通过静态分析工具识别的典型的代码缺陷和设计缺陷（K1）。

3.1. 静态技术和测试过程（K2）15 分钟

术语

动态测试（dynamic testing）、静态测试（static testing）

背景

与要求运行软件的动态测试技术不同，静态测试技术通过手工检查（评审）或自动化分析（静态分析）的方式对代码或者其他的项目文档进行检查而不需要执行代码。

评审是对软件工作产品（包括代码）进行测试的一种方式，可以在动态测试执行之前进行。在生命周期早期的评审过程中发现并修改缺陷（例如发现需求中的缺陷）的成本会比在动态测试中才发现并修改这些缺陷的成本低的多。

评审可以完全以人工的方式进行，也可以通过工具的支持来进行。人工进行评审的主要活动是检查工作产品，并对工作产品做出评估。可以对任何软件工作产品进行评审，包括需求规格说明、设计规格说明、代码、测试计划、测试说明、测试用例、测试脚本、用户指南或 web 页面等。

软件评审的主要好处有：尽早发现和修改缺陷、改善开发能力、缩短开发时间、缩减测试成本和时间、减少产品生命周期成本、减少缺陷以及改善沟通等。评审也可以在工作产品中发现一些遗漏的内容，例如发现需求有遗漏，而这在动态测试中是很难被发现的。

评审、静态分析和动态测试具有共同的目标：识别缺陷。它们之间是互补的：不同的技术可以有效和高效地发现不同类型的缺陷。与动态测试相比，静态技术发现的是软件失效的原因（缺陷）而不是失效本身。

与动态测试相比，通过评审更容易发现如下典型缺陷：与标准之间的偏差、需求内的错误、设计错误、可维护性不足和错误的接口规格说明等等。

3.2. 评审过程（K2）25 分钟

术语

入口准则（entry criteria）、正式评审（formal review）、非正式评审（informal review）、审查（inspection）、度量（metric）、主持人/审查负责人（moderator/inspection leader）、同行评审（peer review）、评审员（reviewer）、记录员（scribe）、技术评审（technical review）、走查（walkthrough）。

背景

评审类型是多样化的，可以是非常不正式的评审（例如评审者没有书面指导性资料可参考），也可以是非常正式的评审（有团队参与，书面的审查结果和管理审查的书面步骤）。评审过程的正式性和以下因素相关：开发过程的成熟度、法律法规方面的要求或审核跟踪的需要。

如何开展评审由评审的目标决定（如，发现缺陷、增加理解，培训测试员和团队新成员或对讨论和决定达成共识等）。

3.2.1. 正式评审的阶段（K1）

典型的正式评审由下面几个主要阶段组成：

1. **计划阶段：**
   - 定义评审标准；
   - 选择人员；
   - 分配角色；
   - 为更加正式的评审类型（比如审查）制定入口和出口准则；
   - 选择需要进行评审的文档的内容；
   - 核对入口准则（针对更正式的评审类型）。
2. **预备会阶段：**
   - 分发文档；
   - 向评审参与者解释评审的目标、过程和文档。
3. **个人准备阶段：**
   - 先行评审文档，为评审会议做准备；
   - 标注可能的缺陷、问题和建议；
4. **检查/评价/记录结果（评审会议阶段）：**
   - 讨论和记录，并留下文档化的结果或会议纪要（针对更正式的评审类型）；

- 标注缺陷、 提出处理缺陷的建议、对缺陷作出决策；
- 在任何形式的会议期间或跟踪任何类型的电子通信期间检查/评价和记录问题。

5. **返工阶段：**

- 修改发现的缺陷（通常由作者来进行）；
- 记录缺陷更新的状态（在正式评审中）。

6. **跟踪结果阶段：**

- 检查缺陷是否已得到解决；
- 收集度量数据；
- 核对出口准则（针对更正式的评审类型）。

3.2.2. 角色和职责（K1）

典型的正式评审主要有下面几种角色：

- 经理：决定是否需要进行评审，在项目计划中分派时间，判断是否已达到评审的目标。
- 主持人：主持文档或文档集的评审活动，包括策划评审、召开会议和会议后的跟踪。假如需要，主持人可能还需要进行不同观点之间的协调。主持人通常是评审成功与否的关键。
- 作者：待评审文档的作者或主要责任人。
- 评审员：具有专门技术或业务背景的人员（也称为检查员（checker）或审查员（inspector）），他们在必要的准备后，标识和描述被评审产品存在的问题（如缺陷）。所选择的评审员应该在评审过程中代表不同的观点和角色，并且应该参与各种评审会议。
- 记录员：记录所有的事件、问题，以及在会议过程中识别的未解决的问题。

从不同的角度评审软件和其相关工作产品并使用检查表可以提高评审的效果和效率。例如，从用户、维护人员、测试人员或操作者的角度编写检查表，或从典型需求问题设计检查表都有助于揭示之前未检测到的问题。

3.2.3. 评审类型（K2）

一篇文档可能需要经历多次评审。如果使用了不只一种评审类型，则评审的顺序可能会有所变化。比如，技术评审之前可能会进行非正式评审，或在客户走查之前可能进行需求规格说明审查。常用评审类型的主要特点、选项和目的如下：

非正式评审

- 没有正式的过程；
- 可以是由程序员的同行们或技术负责人对设计和代码进行评审；
- 评审结果可以文档化；
- 根据不同的评审者，评审作用可能会不同；
- 主要目的：以较低的成本获得收益。

走查

- 由作者主持开会；
- 以场景、演示的形式和同行参加的方式进行；
- 开放式模式
  - 评审人员预备会议是可选的；
  - 包含一个发现问题的列表的评审报告是可选的；
- 记录员（不是作者本人）是可选的；
- 在实际情况中可以是非常正式的，也可能是非常不正式的；
- 主要目的：学习、增加理解、发现缺陷。

技术评审

- 文档化和定义的缺陷检测过程，需要包含同行和技术专家；
- 可能是没有管理者参与的同行评审；
- 理想情况下由专门接受过培训的主持人（不是作者本人）来领导；
- 会议之前需要进行准备；
- 使用检查表是可选的；
- 准备评审报告，包括发现问题的列表、软件产品是否符合需求的判断，与发现的问题合适的建议；
- 在实际情况中可以是在不正式的和非常正式的之间；
- 主要目的：讨论、作决策、评估候选方案、发现缺陷、解决技术问题、检查与规格及标准的符合程度。

审查

- 由接受过专门培训的主持人（不是作者本人）来领导；
- 通常是同行检查；
- 定义了不同的角色；
- 引入了度量；
- 根据入口、出口规则的检查列表和规则定义正式的评审过程；
- 会议之前需要进行准备；
- 出具审查报告和发现问题列表；
- 正式的跟踪过程（过程改进部分是可选的）；
- 朗读者是可选的；
- 主要目的：发现缺陷。

走查，技术评审和审查可以是在同行们–即由同一组织级别内的同事们内举行，这种评审类型称为同行评审。

3.2.4. 评审成功的因素（K2）

评审成功的因素包括：

- 每次评审都有预先明确定义的目标；
- 针对评审目标，有合适的评审人员的参与；
- 测试人员参加评审不但有利于提高评审质量，还可以通过评审了解产品，为测试尽早开始

做准备；

- 对发现的缺陷持欢迎态度，并客观地描述缺陷；
- 能够正确处理人员之间的问题以及心理方面的问题（比如对作者而言，能让他觉得有积极正面的体验）；
- 评审应该在一种信任的气氛中进行；并且结果不应用于对参与者的评价；
- 采用的评审技术适合于要达到的目标、软件工作产品的类型和级别以及参与评审的人员；
- 选用合适的检查表或定义合适角色，可以提高缺陷识别的有效性；
- 提供评审技术方面的培训，特别是针对正式的评审技术，比如审查；
- 管理层对良好评审过程的支持（如在项目计划中安排足够的时间来进行评审活动）；
- 强调学习和过程的改进。

3.3. 静态分析的工具支持（K2）20 分钟

术语

编译器（complier）、复杂性（complexity）、控制流（control flow）、数据流（data flow）、静态分析（static analysis）。

背景

静态分析的目的是发现软件源代码和软件模型中的缺陷。静态分析的执行并不需要使用工具去实际运行被测软件。而动态测试是真正运行软件的代码。静态分析可以定位那些在测试过程很难发现的缺陷。与评审一样，静态分析通常发现的是缺陷而不是失效。静态分析工具能够分析程序代码（比如控制流和数据流），以及产生如 HTML 和 XML 的输出。

静态分析的好处：

- 在测试执行之前尽早发现缺陷；
- 通过度量的计算（比如高复杂性测量），早期警示代码和设计可能存在问题的方面；
- 可以发现在动态测试过程不容易发现的一些缺陷；
- 可以发现软件模块之间的相互依赖性和不一致性，例如链接；
- 改进代码和设计的可维护性；
- 在开发过程中学习经验教训，从而预防缺陷。

通过静态分析工具能够发现的典型缺陷如下：

- 引用一个没有定义值的变量；
- 模块和组件之间接口不一致；
- 从未使用的变量；
- 不可达代码或死代码；
- 逻辑上的遗漏与错误（潜在的无限循环）；
- 过于复杂的结构；
- 违背编程规则；
- 安全漏洞；
- 代码和软件模型的语法错误。

开发人员通常在组件测试和集成测试之前或期间，或当代码签入到配置管理工具时使用静态分析工具（按照预先定义的规则或编程规范进行检查），而设计人员在软件建模期间也使用静态分析工具。静态分析工具会产生大量的警告信息，需要很好的管理这些信息，从而可以有效地使用静态分析工具。

编译器也可以为静态分析提供一些帮助，包括度量的计算。

参考文献：

3.2 IEEE 1028

3.2.2 Gilb,1993,van Veenendaal,2004

3.2.4 Gilb,1993,IEEE 1028

3.3 Van Veenendaal,2004

4. 测试设计技术（K3）285 分钟

测试设计技术的学习目标

学习目标将明确完成下面模块的学习后，学员能做什么。

4.1 测试开发过程（K3）

LO-4.1.1 区别：测试设计规格说明、测试用例规格说明和测试规程规格说明（K2）。

LO-4.1.2 比较术语：测试条件、测试用例和测试规程（K2）。

LO-4.1.3 对测试用例本身的质量可以从与需求的可追溯性以及期望结果这两方面来评价（K3）

LO-4.1.4 根据测试人员的理解水平，将测试用例转换为不同详细程度的结构合理的测试规程规格说明（K3）。

4.2 测试设计技术的种类（K2）

LO-4.2.1 复述在测试用例设计中，为什么需要采用基于规格说明的测试（黑盒测试）和基于结构的测试（白盒测试）的方法？列举出各自比较常用的技术（K1）。

LO-4.2.2 解释基于规格说明的测试、基于结构的测试和基于经验的测试三者的特征和区别（K2）。

4.3 基于规格说明的或黑盒测试技术（K3）

LO-4.3.1 使用等价类划分、边界值分析、决策表和状态转换图/表对指定的软件模型编写测试用例：（K3）

LO-4.3.2 解释这四种测试设计技术各自的主要目的，这些技术可以应用于什么测试级别和测试类型，以及如何测量测试覆盖（K2）。

LO-4.3.3 解释用例测试的概念和应用这种技术的优点（K2）。

4.4 基于结构的技术或白盒技术（K4）

LO-4.4.1 描述代码覆盖的概念及其重要性（K2）。

LO-4.4.2 解释语句覆盖和判定覆盖等概念，理解这些概念除了可以应用在组件测试外，还可以应用在其他任何测试级别上（比如系统级别上的业务过程测试）（K2）。

LO-4.4.3 根据给定的控制流，使用语句测试和判定测试设计技术编写测试用例（K3）：

LO-4.4.4 根据已定义的出口准则评估语句覆盖和判定覆盖的完整性（K4）。

4.5 基于经验的技术（K2）

LO-4.5.1 复述在哪些情况下使用基于直觉、基于经验和知识、基于对常见缺陷的认识来编写测试用例（K1）。

LO-4.5.2 比较基于经验的方法和基于规格说明的方法之间的区别（K2）。

4.6 选择测试技术（K2）

LO-4.6.1 根据给定的因素，如：测试依据、各自的模型和软件特性等，选择合适的测试设计技术（K2）。

4.1. 测试开发过程（K2）15 分钟

术语

测试用例说明（test case specification）、测试设计（test design）、测试执行进度表（test execution schedule）、测试规程说明（test procedure specification）、测试脚本（test script）、可追溯性（traceability）。

背景

可以采用不同的方法完成本章节描述的过程，根据具体情况，可以采用很少的或没有文档的非正式方式到采用非常正式的方式（如下所述）。正式的程度是依赖于测试的背景，包括组织的架构、测试及开发过程的成熟度、项目时间的限制、安全或规范需求以及什么样的人员参与等。

在测试分析阶段，要对测试基础文档进行分析，从而决定测试什么，也就是明确测试的条件。将测试条件定义为能通过一个或多个测试用例进行验证的一个条目或事件（比如功能、事务处理、质量特征或结构元素等）。

建立从测试条件到需求的可追溯性，有助于需求变更时的影响分析和测试用例集的需求覆盖率分析。在测试分析阶段，除了考虑一些其它的因素，基于已经识别的风险，实施具体的测试方法从而选择要采用的测试技术（有关风险分析的更多的内容请参见第 5 章）。

在测试设计阶段，要定义和记录测试用例和测试数据。测试用例由：一组输入值、执行的前提条件、预期结果和执行的后置条件等元素组成，以覆盖一定的产生目标或测试条件。测试设计说明（包含测试条件）和测试用例说明的内容在“软件测试文档标准（IEEE Std 829-1998）”中有具体的描述。

预期的测试结果应该作为测试用例说明的一部分，同时包含输出、数据和状态的变化，以及其他的测试结果。假如没有明确预期结果，则一个看似合理却错误的结果可能被视为正确的结果。理想情况下预期结果应该在测试执行之前明确定义。

在测试实现阶段，测试用例的开发、实现、确定优先级和组织都应该包含在测试规程规格说明中（IEEE STD 829-1998）。测试规程（或者手工测试脚本）描述了测试用例执行的顺序。如果使用测试执行工具进行测试，这种测试的动作顺序将在测试脚本中描述（自动化的测试规程）。

不同的测试规程和自动化测试脚本要体现在测试执行进度表中，该计划定义了不同测试规程和可能的自动化测试脚本的执行顺序、执行的时间和执行者。测试执行进度表同时考虑了其他的因素，比如回归测试、测试优先级以及技术和逻辑的依赖等。

4.2. 测试设计技术的种类（K2）15 分钟

术语

黑盒测试设计技术（black-box test design technique）、基于经验的测试设计技术（experience-based test design technique）、测试设计技术（test design technique）、白盒测试设计技术（white-box test design technique）。

背景

使用测试设计技术的目的是为了识别测试条件和开发测试用例。

将测试技术分为黑盒测试技术与白盒测试技术是一种比较传统的分类方法。黑盒测试设计技术（也称为基于规格说明的测试技术）是依据分析测试基础文档来选择测试条件、测试用例或测试数据的技术。它包括了功能和非功能的测试。黑盒测试，顾名思义，不需要使用任何关于被测组件或系统的内部结构信息。白盒测试设计技术（也称为结构化或基于结构的测试技术）是基于分析被测组件或系统的结构的测试技术。黑盒和白盒测试也可以与基于经验的技术结合，以补充开发人员、测试人员和用户的经验，从而决定什么应该被测试。

有些技术可以明确地归为单一的类，而有些可以属于不同的类别。

本大纲涉及基于规格说明的方法归为黑盒测试技术，而基于结构的方法归为白盒测试技术。另外还有基于经验的测试设计技术。

基于规格说明的测试技术具有以下共同特点：

- 使用正式或非正式的模型来描述需要解决的问题、软件或其组件等；
- 根据这些模型，可以系统地导出测试用例。

基于结构的技术的共同特点：

- 根据软件的结构信息设计测试用例，比如软件代码和详细设计信息；
- 可以通过已有的测试用例测量软件的测试覆盖率，并通过系统化的导出设计用例来提高覆盖率。

基于经验的方法具有以下共同特点：

- 测试用例根据参与人员的经验和知识来编写；
- 测试人员、开发人员、用户和其他的利益相关者对软件、软件使用和环境等方面所掌握的知识作为信息来源之一；
- 对可能存在的缺陷及其分布情况的了解作为另一个信息来源。

4.3. 基于规格说明或黑盒测试技术（K3）150 分钟

术语

边界值分析（boundary value analysis）、决策表测试（decision table testing）、等价类划分（equivalence partitioning）、状态转换测试（state transition testing）、用例测试（use case testing）。

4.3.1. 等价类划分（K3）

可以将软件或系统的输入分成不同的组，对于同一个组的输入，软件或系统应该有相似的表现行为，就好像系统是以相同的方式对这些输入值进行处理。等价类划分（或等价类）可以分为两种类型的数据：有效数据（即应该被系统接受的数据）和无效数据（即应该被系统拒绝的数据）。等价类划分也可以基于输出、内部值、时间相关的值（例如在事件之前或之后）以及接口参数（在集成测试阶段）等进行。可以设计测试用例来覆盖所有有效和无效等价类。等价类划分可以应用在所有测试级别上。

通过应用等价类划分技术，能够实现输入覆盖和输出覆盖。它同样适用于人为的输入、通过系统接口的输入以及集成测试中的接口参数。

4.3.2. 边界值分析（K3）

在各等价类划分的边界通常更可能出现不正确的行为，因此边界就是测试比较可能发现缺陷的区域。每个划分的最大和最小值就是它的边界值。有效部分的边界就是有效边界值，无效部分的边界就是无效边界值。测试的设计应当既覆盖有效边界值又覆盖无效边界值。在设计测试用例时，应该将每个边界值包含在测试用例中。

边界值分析可以应用于所有的测试级别。这种方法的应用相对简单，发现缺陷的能力也比较高，同时，详细的规格说明对边界值分析很有帮助。

边界值分析技术通常被认为是等价类划分技术或其他黑盒测试设计技术的一种拓展。它可以应用在用户从屏幕输入的等价类中，也可以应用在如时间段的范围（如超时，对事务处理速度的需求）或表的边界（如表大小为 256×256）等方面。

4.3.3. 决策表测试（K3）

决策表是一种很好的方法，它可以识别含有逻辑条件的系统需求，还可以将内部系统设计文档化。这种方法可以用来记录一个系统要实施的复杂的业务规则。建立决策表时，要分析规格说明，并识别系统的条件和动作。输入条件和动作通常以“真”或“假”（布尔变量）的方式进行表述。决策表包含了触发条件，通常还有各种输入条件真或假的组合以及各条件组合相应的输出动作。决策表的每一列对应了一个业务规则，该规则定义了各种条件的一个特定组合，以及这个规则相关联的执行动作。决策表测试的常见覆盖标准是每列至少对应一个测试，该测试通常覆盖触发条件的所有

组合。

决策表测试的优点是可以生成测试条件的各种组合，而这些组合可能利用其他方法会无法被测试到。它适用于所有当软件的行为由一些逻辑决策所决定的情况。

4.3.4. 状态转换测试（K3）

根据系统当前的情况或先前的情况（如系统先前的状态），系统可能会产生不同的响应。这种情况下，系统的特征可以通过状态转换图来表示。测试员可以根据软件的状态、状态间的转换、触发状态变化（转换）的输入或事件以及从状态转换导致的可能的行动来进行测试。被测试系统或对象的状态是独立的、可确认的，并且数量是有限的。

一个状态表描绘了状态和输入之间的关系，并能显示可能的无效状态转换。

设计的测试可以覆盖一个典型的状态序列，或覆盖每个状态，或执行每个状态转换，或执行特定顺序的状态转换或测试无效的状态转换。

状态转换测试方法普遍较多的使用在嵌入式软件行业和自动化行业。但是这个技术同样也适用于有特定状态的业务对象的建模或测试具有对话框状态转换流的系统（例如互联网应用或业务场景）。

4.3.5. 用例测试（K2）

可以通过用例来设计测试。用例描述了参与者（用户或系统）之间的相互作用，并从这些交互产生一个从系统用户或客户的角度所期望和能观察到的结果。通常可以在抽象层（业务用例、不受技术限制、业务流程层面）或系统层（系统功能层面的系统用例）来描述用例。每个用例都有测试的前置条件，这是用例成功执行的必要条件。每个用例结束后都存在后置条件，这是在用例执行完成后能观察到的结果和系统的结束状态。用例通常有一个主场景（即最有可能发生的场景）和可选场景。

用例基于系统最可能使用的情况描述了过程流，因此从用例中得到的测试用例，在真实世界中的系统使用过程流中能最有效的发现系统的缺陷。用例非常有助于设计用户/客户参与的验收测试；也可以帮助发现由于不同组件之间的相互作用和相互影响而产生的集成缺陷，这是在单个的组件测试中是无法发现的。从用例中设计测试用例可以和其他基于规格说明的测试技术结合起来使用。

4.4. 基于结构的或白盒技术（K4）60 分钟

术语

代码覆盖（code coverage）、判定覆盖（decision coverage）、语句覆盖（statement coverage）、基于结构的测试（structure-based testing）。

背景

基于结构的测试/白盒测试是根据识别软件或系统的结构，可以从以下内容得到进一步的理解：

- 组件级别：软件组件的结构，比如：语句、判定、分支或每个不同的路径；
- 集成级别：结构可能是调用树（模块调用关系图）；
- 系统级别：结构可能是菜单结构、业务过程或 web 页面结构。

基于语句、分支和判定，本节将讨论三种与代码相关的结构化测试设计技术的代码覆盖。对于判定覆盖，可以使用控制流图来形象表示每个判定之间的转换。

4.4.1. 语句覆盖和覆盖率（K4）

在组件测试中，语句覆盖是指评价一个测试用例套件中已经执行的可执行语句的百分比。语句测试的测试用例用来执行专门的语句，通常用来增加语句的覆盖率。

语句覆盖率取决于被（设计或执行）测试用例覆盖的可执行语句数量除以被测代码中所有可执行语句数量。

4.4.2. 判定覆盖和覆盖率（K4）

判定覆盖，和分支测试相关，是指评价在一个测试用例套中已经执行的判定（例如 if 语句的 true 和 false 选项）输出的百分比。判定测试的测试用例用来执行专门的判定输出。分支起始于代码中的判定点，并表明了在代码中不同位置的控制转移。

判定覆盖率取决于被（设计或执行）的测试用例覆盖的所有判定出口数目除以被测代码中所有可能的判定出口数目。

判定测试是控制流测试技术的一种方式，它在判定点产生一个专门的控制流。判定覆盖比语句覆盖更全面，100％的判定覆盖可以保证 100％的语句覆盖，反之则不行。

4.4.3. 其他的基于结构的技术（K1）

除了判定覆盖，还有程度更高的基于结构的覆盖，如条件覆盖和多重条件覆盖。

覆盖的概念也可以用于其他的测试级别（比如集成测试级别等），在一个测试用例套件中被执行的模块、组件或类覆盖的百分比可以分别称为模块覆盖、组件覆盖或类的覆盖。

在进行代码的结构测试中使用工具支持是非常有帮助的。

4.5. 基于经验的技术（K2）30 分钟

术语

探索性测试（exploratory testing）、缺陷攻击（fault attack）。

背景

基于经验的测试是根据测试人员对相似的应用或技术的经验以及知识和直觉来进行测试的，如果是用来协助系统化的测试方法，这些技术能够识别一些正式技术不能获取的特殊测试，特别是当用在正式技术之后会更有效。但是，这种技术依据测试员的经验，所以产生的效果会有极大的不同。

一个比较常见的基于经验的技术是错误推测法。一般情况下，测试人员是靠经验来预测缺陷。错误推测法的一个结构化方法是列举可能的错误，并设计测试来攻击这些错误，这种系统的方法称之为缺陷攻击。可以根据经验、已有的缺陷和失败数据以及有关软件失败的常识等方面的知识来设计这些缺陷和失效的列表。

探索性测试是指依据包含测试目标的测试章程来同时进行测试设计、测试执行、测试记录和学习，并且是在规定时间内进行的。这种方法在规格说明较少或不完备且时间压力大的情况下使用更有帮助，或者作为对其他更为正式的测试的增加或补充。它可以作为测试过程中的检查，以有助于确保能发现最为严重的缺陷。

4.6. 选择测试技术（K2）15 分钟

术语

无

背景

测试技术的选择基于下面的几个因素，包括：系统类型、法律法规标准、客户或合同的需求、风险的级别、风险的类型、测试目标、文档的可用性、测试员的技能水平、时间和成本预算、开发生命周期、用例模型和以前发现各类缺陷的经验等。

有些测试技术适合于特定的环境和测试级别；而有些则适用于所有的测试级别。

在建立测试用例时，测试人员通常会组合多种测试技术并结合流程、规则和数据驱动技术来保证对测试对象足够的覆盖率。

参考文献：

4.1 Craig, 2002, Hetzel, 1988, IEEE 829
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5. 测试管理（K3）170 分钟

测试管理的学习目标

学习目标将明确完成下面模块的学习后，学员能做什么。

5.1 测试的组织结构（K2）

LO-5.1.1 识别独立测试的重要性（K1）。

LO-5.1.2 阐明在组织内进行独立测试的优点和缺点（K2）。

LO-5.1.3 识别创建测试小组需要考虑不同角色的团队成员（K1）。

LO-5.1.4 牢记测试组长和测试员的主要任务（K1）。

5.2 测试计划和估算（K3）

LO-5.2.1 识别测试计划的不同级别和目标（K1）。

LO-5.2.2 根据“软件测试文档标准”（IEEE Std 829-1998），总结测试计划、测试设计规格说明和测试规程的目的及内容（K2）。

LO-5.2.3 从概念上区分不同的测试方法，例如：分析法、基于模型的方法、系统法、符合过程/标准的、动态/启发式的、咨询式或可重用的方法（K2）。

LO-5.2.4 区分为系统所做的测试计划和测试执行进度安排的不同之处（K2）。

LO-5.2.5 为一组给定的测试用例编写测试执行进度表，需要考虑优先级、技术和逻辑关系等内容（K3）。

LO-5.2.6 列出在测试计划时应该考虑的测试准备和执行活动（K1）。

LO-5.2.7 记忆影响测试成果的主要因素（K1）。

LO-5.2.8 从概念上区分两种不同的估算方法：基于度量的方法和基于专家的方法（K2）。

LO-5.2.9 理解/证明应该针对特定的测试级别和测试用例组定义恰当的入口准则和出口准则（例如集成测试、验收测试或可用性测试的测试用例）（K2）。

5.3 测试过程监控（K2）

LO-5.3.1 记忆用于监督测试准备和执行的常见度量项（K1）。

LO-5.3.2 根据不同的目的和用途对于测试报告和测试控制中用到的测试度量进行说明和比较（例如已发现和已修复的缺陷、通过和失败的测试）（K2）。

LO-5.3.3　根据“软件测试文档标准”(IEEE Std 829-1998)，总结测试总结报告的目的和内容(K2)。

5.4 配置管理（K2）

LO-5.4.1　总结配置管理如何支持测试（K2）。

5.5 风险和测试（K2）

LO-5.5.1　将可能威胁一个或多个利益相关者实现项目目标的潜在问题描述为风险（K2）。

LO-5.5.2　牢记风险的级别是由可能性（发生的可能性）和影响程度（发生后所造成的危害）来决定的（K1）。

LO-5.5.3　区分项目风险和产品风险（K2）。

LO-5.5.4　识别典型的产品风险和项目风险（K1）。

LO-5.5.5　通过例子来描述在测试计划中如何进行风险分析和风险管理（K2）。

5.6 事件管理（K3）

LO-5.6.1　按照“软件测试文档标准（IEEE Std 829-1998)”，总结事件报告的内容（K1）。

LO-5.6.2　针对测试过程中发现的失效编写事件报告（K3）。

5.1. 测试组织（K2）30 分钟

术语

测试员（tester）、测试组长（test leader）、测试经理（test manager）

5.1.1. 测试组织和测试独立性（K2）

通过独立的测试员进行测试和评审，发现缺陷的效率会提高。可能的独立测试如下：

- 不独立的测试员，开发人员测试自己的代码；
- 开发团队内独立的测试员；
- 组织内独立的测试小组或团队，向项目经理或执行经理汇报；
- 来自业务组织、用户团体内的独立测试 e 员；
- 针对特定测试类型的独立测试专家，例如：可用性测试员、安全性测试员或认证测试员（他们根据标准和法律法规对软件产品进行认证）；
- 外包或组织外的独立测试人员。

对于庞大、复杂或安全关键的项目，通常最好有多级别的测试，并让独立的测试员负责某些级别或所有的测试。开发人员也可以参与测试，尤其是一些低级别的测试,但是开发人员往往缺少客观性，会限制他们测试的有效性。独立测试员可以有权要求和定义测试过程及规则，但是测试员应该只有在存在明确管理授权的情况下才能充当这种过程相关的角色。

独立测试的优点：

- 独立的测试员是公正的，可以发现一些其他不同的缺陷。；
- 一个独立的测试员可以验证在系统规格说明和实现阶段所做的一些假设。

独立测试的缺点：

- 与开发小组脱离（如果完全独立）；
- 开发人员可能丧失对软件质量的责任感；
- 独立的测试员可能被视为瓶颈或者成为延时发布而被责备的对象。

测试任务可以由专门的测试员完成，也可以由其他的角色来完成，比如项目经理、质量经理、开发人员、业务和领域内的专家、基础架构或 IT 的运行人员。

5.1.2. 测试组长和测试员的任务（K1）

在本课程大纲中，涉及两个测试角色：测试组长和测试员。这两个角色执行的活动和任务是由项目和产品的背景、人员的角色和组织结构来决定的。

有时候，测试组长也称为测试经理或测试协调人。测试组长的角色也可以由项目经理、开发经理、质量保证经理或测试组的经理来担任。在较大的项目中，常常会有两个职位：测试组长和测试经理。测试组长通常计划、监督和控制 1.4 章节中定义的测试活动和任务。

测试组长可能的主要任务包括：

- 与项目经理以及其他人共同协调测试策略和测试计划；
- 制定或评审项目的测试策略和组织的测试方针；
- 将测试的安排合并到其他项目活动中，比如集成计划；
- 制定测试计划（要考虑背景，了解测试目标和风险），包括选择测试方法，估算测试的时间、工作量和成本，获取资源，定义测试级别、测试周期并规划事件管理；
- 启动测试说明、测试准备、测试实施和测试执行，监督测试结果并检查出口准则；
- 根据测试结果和测试过程（有时记录在状态报告中）调整测试计划，并采取任何必要措施对存在的问题进行补救；
- 对测试件进行配置管理，保证测试件的可追溯性；
- 引入合适的度量项以测量测试进度，评估测试和产品的质量；
- 决定什么应该自动化，自动化的程度，以及如何实现；
- 选择测试工具支持测试，并为测试员组织测试工具使用的培训；
- 决定关于测试环境实施的问题；
- 根据在测试过程中收集的信息编写测试总结报告。

测试员可能的主要任务包括：

- 评审和参与测试计划的制定；
- 分析、评审和评估用户需求、规格说明书及模型的可测试性；
- 创建测试说明；
- 建立测试环境（通常需要系统管理员，网络管理员协同完成）；
- 准备和获取测试数据；
- 进行所有级别的测试，执行并记录测试日志，评估测试结果，记录和预期结果之间的偏差。
- 根据需要使用测试管理工具和测试监控工具；
- 实施自动化测试（可能需要开发人员或测试自动化专家的支持）；
- 在可行的情况下，测量组件和系统的性能；
- 对他人的测试进行评审。

从事测试分析、测试设计、特定测试类型或自动化测试方面的工作人员都可以是这些角色的专家。根据测试级别及与产品和项目相关的风险，可以由不同的人员担任测试员的角色，以保持一定程度的独立性。在组件和集成测试的级别，典型的测试员可能是开发人员，进行验收测试的典型测试员可能是业务方面的专家和用户，进行运行验收测试的典型测试员可能是运行操作者。

5.2. 测试计划和估算（K2）40 分钟

术语

测试方法（test approach），测试策略（test strategy）

5.2.1. 测试计划（K2）

本章节将描述在开发和实施项目以及维护过程中，制定测试计划的目的。测试计划可以在项目计划或主测试计划中文档化，也可以在不同的测试级别（如系统测试和验收测试）的测试计划中文档化。测试计划文档的大纲可以参考“软件测试文档标准”（IEEE Std 829-1998）。

测试计划受到很多因素的影响：组织的测试方针、测试范围、测试目标、风险、约束、关键程度、可测试性和资源的可用性等。随着项目和测试计划的不断推进，将有更多的信息和具体细节包含在计划中。

测试计划是个持续的活动，需要在整个生命周期过程和活动中进行。从测试中得到的反馈信息可以识别变化的风险，从而对计划作相应的调整。

5.2.2. 测试计划活动（K3）

对整个系统或部分系统可能的测试计划活动包括：

- 确定测试的范围和风险，明确测试的目标；
- 决定总体测试方法，包括测试级别、入口和出口准则的界定；
- 把测试活动整合和协调到整个软件生命周期活动中去（采购、供应、开发和运维）；
- 决定测试什么？测试由什么角色来执行？如何进行测试？如何评估测试结果？
- 为测试分析和设计活动安排时间进度；
- 为测试实现、执行和评估安排时间进度；
- 为已定义的不同测试活动分配资源；
- 定义测试文档的数量、详细程度、结构和模板；
- 为监控测试准备和执行、缺陷解决和风险问题选择度量项；
- 确定测试规程的详细程度，以提供足够的信息支持可复用的测试准备和执行。

5.2.3. 入口准则（K2）

入口准则定义了什么时候可以开始测试，如某个测试级别的开始，或什么时候一组测试准备就绪可以执行。

入口准则主要包含：

- 测试环境已经准备就绪并可用；
- 测试工具在测试环境中已经准备就绪；
- 可测的代码可用；

- 测试数据可用。

5.2.4. 出口准则（K2）

测试出口准则（exit criteria）的目的是：定义什么时候可以停止测试，比如某个测试级别的结束，或者当测试达到了规定的目标。

出口准则主要包含：

- 完整性测量，比如代码、功能或风险的覆盖率；
- 对缺陷密度或可靠性度量的估算；
- 成本；
- 遗留风险，例如没有被修改的缺陷或在某些部分测试覆盖不足；
- 进度表，例如基于交付到市场的时间。

5.2.5. 测试估算（K2）

在本大纲中，有两种估算测试工作量的方法：

- 基于度量的方法：根据以前或相似项目的度量值来进行测试工作量的估算，或者根据典型的数据来进行估算；
- 基于专家的方法：由任务的责任人或专家来进行测试任务工作量的估算。

一旦估算了测试工作量，就可以识别资源和制定时间进度表。

测试的工作量可能取决于多种因素，包括：

- 产品的特点：规格说明和用于测试模型的其它信息（即测试依据）的质量，产品的规模，问题域的复杂度，可靠性、安全性的需求和文档的需求；
- 开发过程的特点：组织的稳定性、使用的工具、测试过程、参与者的技能水平和时间紧迫程度等；
- 测试的输出：缺陷的数量和需要返工的工作量。

5.2.6. 测试策略，测试方法（K2）

在特定项目中，测试方法是测试策略的具体实现。测试方法是在测试计划和设计阶段中被定义并逐步细化的。它通常取决于（测试）项目目标和风险评估。它是规划测试过程、选择测试设计技术和应用的测试类型以及定义入口和出口准则的起点。

测试方法的选择取决于实际情况，应当考虑风险、危害和安全、可用资源和人员技能、技术、系统的类型（比如客户定制与商业现货软件的比较）、测试对象和相关法规。

典型的测试方法包括：

- 分析的方法，比如基于风险的测试，直接针对风险最高的部分进行测试；
- 基于模型的方法，比如随机测试利用失效率（如:可靠性增长模型）或使用率（如：运行概况）的统计信息；
- 系统的方法，比如基于失效的方法（包括错误推测和故障攻击），基于检查表的方法和基于质量特征的方法；

- 基于与过程或符合标准的方法，比如在行业标准中规定的方法或各类敏捷的方法；
- 动态和启发式的方法，类似于探索性测试，测试很大程度上依赖于事件而非提前计划，而且执行和评估几乎是同时进行的；
- 咨询式的方法，比如测试覆盖率主要是根据测试小组以外的业务领域和/或技术领域专家的建议和指导来推动的；
- 可重用的方法，比如重用已有的测试材料，广泛的功能回归测试的自动化，标准测试套件等。

可以结合使用不同的测试方法，比如基于风险的动态方法。

5.3. 测试过程的监控（K2）20 分钟

术语

缺陷密度（defect density）、失效率（failure rate）、测试控制（test control）、测试监控（test monitoring）、测试总结报告（test summary report）。

5.3.1. 测试过程监控（K1）

测试监控的目的是提供关于测试活动的反馈信息，使测试活动保持可视性。监控的信息可以通过手工或自动的方式进行收集，同时可以用来衡量出口准则，比如测试覆盖率。也可以用度量数据对照原计划的时间进度和预算来评估测试的进度。常用的测试度量项有：

- 测试用例准备工作完成的百分比（或按计划已编写的测试用例的百分比）；
- 测试环境准备工作完成的百分比；
- 测试用例执行情况（例如：执行/没有执行的测试用例数，通过/失败的测试用例数）；
- 缺陷信息（例如：缺陷密度、发现并修改的缺陷、失效率、重新测试的结果）；
- 需求、风险或代码的测试覆盖率；
- 测试员对产品的主观信心；
- 测试里程碑的日期；
- 测试成本，包括寻找下一个缺陷或执行下一轮测试所需成本与收益的比较。

5.3.2. 测试报告（K2）

测试报告是对测试工作和活动等相关信息的总结，主要包括：

- 在测试周期内发生了什么？比如达到测试出口准则的日期；
- 通过分析相关信息和度量可以对下一步的活动提供建议和做出决策，比如对遗留缺陷的评估、继续进行测试的经济效益、未解决的风险以及被测试软件的置信度等。

测试总结报告的大纲可以参考“软件测试文档标准”（IEEE Std 829-1998）。

需要在测试级别的过程中和完成时收集度量信息，来评估：

- 该测试级别的测试目标实现的充分性；
- 采用的测试方法的适当性；
- 针对测试目标的测试的有效性。

5.3.3. 测试控制（K2）

测试控制描述了根据收集和报告的测试信息和度量而采取的指导或纠正措施。措施可能包括任何测试活动，也可能影响其它软件生命周期中的活动或任务。

测试控制措施的例子：

- 基于测试监控信息来做决策；
- 如果一个已识别的风险发生（如软件交付延期），重新确定测试优先级；
- 根据测试环境可用性，改变测试的时间进度表；
- 设定入口准则：规定修改后的缺陷必须经过开发人员再测试（确认测试）后才能将它们集成到版本中去。

5.4. 配置管理（K2）10 分钟

术语

配置管理（configuration management）、版本控制（version control）。

背景

配置管理的目的是在整个项目和产品的生命周期内，建立和维护软件或系统产品（组件、数据和文档）的完整性。

对测试而言，采用配置管理可以确保：

- 测试件的所有相关项都已经被识别，版本受控，相互之间有关联以及和开发项（测试对象）之间有关联的变更可跟踪，从而保证可追溯性；
- 在测试文档中，所有被标识的文档和软件项能被清晰明确的引用。

对于测试员来说，配置管理可以帮助他们唯一地标识（并且复制）测试项、测试文档、测试用例和测试用具。

在测试计划阶段，应该选择配置管理的规程和基础设施（工具），将其文档化并予以实施。

5.5. 风险和测试（K2）30 分钟

术语

产品风险(product risk)、项目风险(project risk)、风险(risk)、基于风险的测试(risk-based testing)。

背景

风险可以定义为事件、危险、威胁或情况等发生的可能性以及由此产生不可预料的后果，即一个潜在的问题。风险级别取决于发生不确定事件的可能性和产生的影响（事件引发的不良后果）。

5.5.1. 项目风险（K2）

项目风险是围绕项目按目标交付的能力的一系列风险，比如：

- 组织因素：
  - 技能、培训和人员的不足；
  - 个人问题；
  - 政策因素，比如：
    - 与测试员进行需求和测试结果沟通方面存在的问题；
    - 测试和评审中发现的信息未能得到进一步跟踪（如未改进开发和测试实践）；
  - 对测试的态度或预期不合理（如：没有意识到在测试中发现缺陷的价值）。
- 技术因素：
  - 不能定义正确的需求；
  - 给定现有限制的情况下，没能满足需求的程度；
  - 测试环境没有及时准备好；
  - 数据转换、迁移计划，开发和测试数据转换/迁移工具造成的延迟；
  - 低质量的设计、编码、配置数据、测试数据和测试。
- 供应商因素：
  - 第三方存在的问题；
  - 合同方面的问题。

在分析、管理和缓解这些风险的时候，测试经理需要遵循完善的项目管理原则。“软件测试文档标准”（IEEE Std 829-1998）中指出，测试计划需要陈述风险和应急措施。

5.5.2. 产品风险（K2）

在软件或系统中的潜在失效部分（即将来可能发生不利事件或危险）称之为产品风险，因为它们对产品质量而言是一个风险，包括：

- 故障频发的软件交付使用；

- 软件/硬件对个人或公司造成潜在损害的可能性；
- 劣质的软件特性（比如功能性、可靠性、易用性和性能等）；
- 低劣的数据完整性和质量（例如：数据迁移问题、数据转换问题、数据传输问题、违反数据标准问题）；
- 软件没有实现既定的功能。

风险通常可以用来决定从什么地方开始测试，什么地方需要更多的测试。测试可以用来降低风险或可以减少负面事件的影响。

产品风险对于项目的成功来讲是一种特殊类型的风险。作为一种风险控制活动，测试通过评估修正严重缺陷的能力和应急计划的有效性来提供关于遗留风险的反馈信息。

在项目初期阶段，使用基于风险的方法进行测试，有利于降低产品风险的级别。它包括对产品风险的识别，并且将这些风险应用到指导测试计划和控制、测试说明、测试准备和执行中。在基于风险的测试方法中，识别出的风险可以用于：

- 决定采用的测试技术；
- 决定要进行测试的范围；
- 确定测试的优先级，尝试尽早的发现严重缺陷；
- 决定是否可以通过一些非测试的活动来降低风险（比如对缺乏经验的设计者进行相应的培训）。

基于风险的测试需要借助于项目利益相关者的集体知识和智慧，从而识别风险以及为了应对这些风险需要采用的测试级别。

为了确保产品失效机会最小化，风险管理活动提供了一些系统化的方法：

- 评估（并定期重新评估）可能出现的错误（风险）；
- 决定哪些风险是重要的需要处理的；
- 处理风险的具体措施。

另外，测试可以帮助识别新的风险，可以有助于确定应该降低哪些风险，以及降低风险的不确定性。

5.6. 事件管理（K3）40 分钟

术语

事件日志（incident logging）、事件管理（incident management）、事件报告（incident report）。

背景

测试的目的之一是发现缺陷，所以实际结果和预期结果之间的差异需要作为一个事件被记录。事件必须进行调查，并且有可能最终被证明是一个缺陷。应当定义合理的措施以便对事件和缺陷进行有效处理。事件和缺陷应该从发现和分类就开始跟踪，直到改正并被确认已经解决。为了完成所有事件的管理，应该在组织内建立一套完整的事件管理过程和分类规则。

在软件产品的开发、评审和测试、以及软件使用的过程中都会产生事件。它们可能是在代码内或在使用的系统内或以任意方式在文档内产生（包括需求文档、开发文档、测试文档和用户文档，如“帮助”或安装手册等）。

事件报告的主要目标如下：

- 为开发人员和其他人员提供问题反馈，在需要的时候可以进行识别、隔离和纠正；
- 为测试组长提供一种有效跟踪被测系统质量和测试进度的方法；
- 为测试过程改进提供资料。

事件报告的具体内容主要包括：

- 提交事件的时间，提交的组织和作者；
- 预期和实际的结果；
- 识别测试项（配置项）和环境；
- 发现事件时软件或系统所处的生命周期阶段；
- 为了能确保重现和解决事件需要描述事件（包括日志、数据库备份或截屏）；
- 对利益相关者的影响范围和程度；
- 对系统影响的严重程度；
- 修复的紧迫性/优先级；
- 事件状态（例如：打开的、延期的、重复的、待修复的、修复后待重测的或关闭的等）；
- 结论、建议和批准；
- 全局的影响，比如事件引起的变更可能会对系统的其他部分产生影响；
- 变更历史记录，比如针对事件的隔离、修改和已修改的确认，项目组成员所采取的行动顺序；
- 参考，包括发现问题所用的测试用例规格说明的标识号。

事件报告的大纲也可以参考“软件测试文档标准”（IEEE Std 829-1998）。

参考文献：

5.1.1 Black,2001,Hetzel,1988
5.1.2 Black,2001,Hetzel,1988
5.2.5 Black,2001,Craig,2002,IEEE Std 829-1998,Kaner 2002
5.3.3 Black,2001,Craig,2002,Hetzel,1988,IEEE Std 829-1998
5.4 Craig,2002
5.5.2 Black,2001,IEEE Std 829-1998
5.6 Black,2001,IEEE Std 829-1998

6. 软件测试工具（K2）80 分钟

软件测试工具的学习目标

学习目标将明确完成下面模块的学习后，学员能做什么。

6.1 测试工具的类型（K2）

LO-6.1.1 根据测试工具的用途和基本的测试过程和软件生命周期活动，对不同类型的测试工具进行分类（K2）。

LO-6.1.2 解释术语“测试工具”，用测试工具支持测试的目的（K2）

6.2 有效使用工具：潜在的利益和风险（K2）

LO-6.2.1 总结测试自动化和使用测试工具的潜在利益和风险（K2）。

LO-6.2.2 记住使用测试执行工具、静态分析工具和测试管理工具时应当考虑的特定因素（K1）。

6.3 组织中工具的引入（K1）

LO-6.3.1 阐述将工具引入组织中的主要原则（K1）。

LO-6.3.2 阐述为评估工具所进行的调查学习验证以及为实施工具所展开的试点阶段的目的（K1）。

LO-6.3.3 识别要获得好的工具支持，仅靠购置工具是不够的，还需要考虑其他因素（K1）。

6.1. 测试工具的类型（K2）45 分钟

术语

配置管理工具（configuration management tool）、覆盖率工具（coverage tool）、调试工具（debugging tool）、动态分析工具（dynamic analysis tool）、事件管理工具（incident management tool）、负载测试工具（load testing tool）、建模工具（modeling tool）、监控工具（monitoring tool）、性能测试工具（performance testing tool）、探测影响（probe effect）、需求管理工具（requirement management tool）、评审工具（review tool）、安全性工具（security tool）、静态分析工具（static analysis tool）、压力测试工具（stress testing tool）、测试比较器（test comparator）、测试数据准备工具（test data preparation tool）、测试设计工具（test design tool）、测试用具（test harness）、测试执行工具（test execution tool）、测试管理工具（test management tool）、单元测试框架工具（unit test framework tool）。

6.1.1. 理解使用测试工具支持测试的意义和目的（K2）

测试工具可以用于支持一种或多种测试活动。包括：

1. 直接用于测试的工具，如测试执行工具、测试数据生成工具和结果对比工具；
2. 测试过程管理工具，如用于管理测试、测试结果、数据、需求、事件、缺陷等等，并且可以报告和监控测试执行；
3. 用于观测的工具，或简单地说就是探索（例如：监控应用程序文件活动的工具）；
4. 任何对测试有帮助的工具（从这个意义来说，电子表格也是测试工具）。

根据实际情况，工具支持测试可以有如下一个或多个目的：

- 可以通过自动重复任务改进测试活动的效率，或支持手动测试活动，如测试计划、测试设计、测试报告与监控；
- 当手动进行需要大量资源时，可使用自动执行（例如，静态测试）；
- 无法手动完成的测试可以将其自动化（例如：CS 应用程序的大规模性能测试）；
- 增加测试的可靠性（例如：进行大量数据的自动比较或是行为模拟）。

术语“测试框架”也被大量的应用于工业界，它至少有三个含义：

- 可重用、可扩展的测试库，可用于搭建测试工具（也称为测试用具）；
- 设计测试自动化的设计类型（例如：数据驱动、关键字驱动）；
- 测试执行的全过程。

基于本大纲的目的，术语“测试框架”的前两种含义，在章节 6.1.6 有所描述。

6.1.2. 测试工具分类（K2）

有很多工具可以支持不同类型的测试活动。工具可以按照多种规则进行分类，例如：目的、商业/免费/开源/共享、使用的技术等。在本大纲中，是按照测试工具能支持的测试活动来进行分类。

有些工具明确支持一种活动；有些工具可以支持多种活动，但将它们分类到联系最紧密的那一

类活动中。同一家供应商的测试工具，尤其是那些为了协同工作而设计的测试工具，可能会被捆绑到一个软件包中。

某些类型的测试工具本身是植入式的，这意味着测试的实际结果可能会受到影响。比如，由于工具执行了额外的指令，可能导致实际时间的不同，或者你可能得到了不同的代码覆盖率。一个工具的植入式特征也称之为探测影响。

某些测试工具提供的支持可能更适合开发人员（比如在进行组件和组件集成测试时使用的工具）。在以下的分类中将这些工具用“D”来标记。

6.1.3. 测试管理的工具支持（K1）

管理工具适用于整个软件生命周期中的所有测试活动。

测试管理工具

这些工具除了提供测试执行、缺陷跟踪和需求管理的接口外，还提供定量分析和测试对象的报告。它还支持追溯测试对象到需求规格说明并可提供独立的版本控制能力或提供一个外部接口。

需求管理工具

需求管理工具储存了需求描述、需求的一些属性（包括优先级），并提供唯一的标识符以及从需求到相应测试的可追溯性。这些工具也可以帮助识别自相矛盾或遗漏的需求。

事件管理工具（缺陷跟踪工具）

这类工具存储并管理事件报告，即缺陷、失效、变更请求或察觉到的问题和异常，并协助管理事件的整个生命周期，为静态分析提供了可能。

配置管理工具

严格的来说，配置管理工具并不能算是测试工具，但对测试件和相关软件的存储和版本管理时却是必要的，尤其是当配置一个以上硬件/软件环境的时候，比如：操作系统版本、编译器、浏览器等等。

6.1.4. 静态测试的工具支持（K1）

静态测试工具提供了一种在开发过程的早期发现尽可能多的缺陷的高性价比的方法。

评审工具

这类工具可支持评审过程、检查表、评审指导方针，并且能用于存储和交流评审意见、以及缺陷和工作量报告。还能进一步为庞大的或分布于不同地区的团队提供在线评审。

静态分析工具（D）

这类工具通过提供对标准编码规范（包括安全编码）、结构和其相关性分析的支持，从而帮助开发和测试员在动态测试之前就找到缺陷。也可以对代码进行度量（例如：复杂性），在计划或风险分析方面有所帮助。

建模工具（D）

建模工具可以用来确认软件模型（例如：关系型数据库的物理数据模型），可以列举其不一致性并发现缺陷。他们通常也用于生成一些基于模型的测试用例。

6.1.5. 测试说明的工具支持（K1）

测试设计工具

测试设计工具能够根据需求、图形用户界面（GUI）、设计模型（状态、数据或对象）或代码生成测试输入或可执行的测试和/或测试准则。

测试数据准备工具

测试数据准备工具用来处理数据库、文件或数据传输，并且生成可以在测试执行过程中使用的测试数据，并通过数据匿名来确保安全性。

6.1.6. 测试执行和记录工具（K1）

测试执行工具

测试执行工具存储了输入和预期结果并通过脚本语言使测试能够自动或半自动进行并记录每一轮测试。它也可用于录制测试，并支持基于脚本语言或图形用户界面数据的参数化配置和其他自定义设置。

测试用具/单元测试框架工具（D）

单元测试用具或测试框架，通过使用像桩或者驱动这类能够模拟测试对象的装置来模拟将要运行的环境，使组件或子系统的测试变得更容易。

测试比较器

测试比较器能够确定文件、数据库或测试结果之间的差异。测试执行工具通常包括动态比较器，但执行后的比较也许会由一个分离的独立比较工具完成。一个测试比较器可能会使用测试准则，尤其是自动化进行的时候。

覆盖率测量工具（D）

这类工具可以使用植入式或非植入式的方法，通过一组测试来测量已运行的代码中特定代码结构类型（比如语句、分支或判定、以及模块或函数调用）的百分比。

安全性测试工具

这类工具用于评估软件的安全特性，包括评估软件保护数据机密性、完整性、身份验证、权限、可用性和不可否认性的能力。安全性工具通常关注在某一特定的技术、平台和目的。

6.1.7. 性能和监控工具（K1）

动态分析工具（D）

动态分析工具能发现那些只有在软件执行过程中才显现的缺陷，比如与时间依赖有关的问题，或内存泄漏等。它们通常应用在组件和组件集成测试以及中间件的测试。

性能测试/负载测试/压力测试工具

性能测试工具监控和报告系统在模拟大量并发用户使用系统时的性能表现、持续增加的模型、处理各业务的频率和相应的处理百分比。系统的负载只是创造了虚拟用户，用来执行一个所选的业务，这些虚拟用户分布在不同的测试机上，通常理解为负载生成器。

监控工具

监控工具持续地分析、验证和报告特定系统资源的使用情况，对可能存在的服务问题提出警告。

6.1.8. 特定应用领域的测试工具（K1）

数据质量评估

数据是有些项目的核心内容，例如：数据转换/数据迁移项目和应用，像数据仓库，它们会在不同临界和容量时表现出不同的特性。在这种情况下，需要使用工具评估数据质量，来评审和验证数据转换和迁移规则，以确保处理的数据是正确的、完整的并遵循一个预定义的、符合特定背景的标准。

还存在一些针对可用性测试的工具。

6.2. 有效使用工具：潜在的收益与风险（K2）20 分钟

术语

数据驱动测试（data-driven testing）、关键字驱动测试（keyword-driven testing）、脚本语言（scripting language）。

6.2.1. 测试工具的潜在收益和风险（针对所有工具）（K2）

仅仅购买或租用工具并不能保证成功使用工具。任何类型的工具都需要额外努力才能获得真正且持续的成效。工具的支持能给测试带来巨大的潜在（可能）收益和新的机会，但是必须考虑到，工具的使用同样存在风险。

使用工具的潜在收益包括：

- 减少重复性的工作（比如，执行回归测试，重新输入相同测试数据，代码规则检查）；
- 更好的一致性和可重复性（比如，用工具按照相同的顺序和频率执行测试，从需求生成测试）；
- 客观的评估（比如，静态测量、覆盖率）；
- 容易得到测试和测试的相关信息（比如，关于测试进展的统计和图表，事件发生率和性能）。

使用工具存在的风险：

- 对工具抱有不切实际的期望（包括功能性和易用性）；
- 低估首次引入工具所需的时间、成本和工作量（包括培训和额外的专业知识）；
- 低估从工具中获得较大和持续性收益需要付出的时间和工作量（包括使用工具的方式需要更改测试过程，并不断改进）；
- 低估了对工具生成的结果进行维护所需的工作量；
- 对工具过分依赖（替代测试设计或者对一些更适合手工测试的方面却使用自动测试工具）；
- 忽视了在工具中对测试对象的版本控制；
- 忽视了多个重要工具之间的关联和互操作性，例如：需求管理工具、版本控制工具、事件管理工具、缺陷跟踪工具和其他从不同供应商获得的工具；
- 工具供应商破产、停止维护工具或将工具卖给其他供应商的风险；
- 供应商对工具的支持、升级和缺陷修复反应不力；
- 开源/免费工具项目中止的风险；
- 其他不可预知的风险，例如不能支持新平台。

6.2.2. 一些工具类型的特殊考虑（K1）

测试执行工具

测试执行工具使用自动化的测试脚本执行测试对象。为了获得可观收益，经常需要为这类工具投入很多工作量。

通过记录测试员手动操作的捕捉过程往往开始看起来似乎很吸引人，但是这种方法不适合大量的自动化测试。捕获的脚本只是用特定数据和动作来线性表示每个脚本的一部分。当发生意外事件时，这类脚本是不稳定的。

数据驱动的方法是将测试输入（数据）与测试用例分离，并将测试输入存放在一个电子表格中，这样可以使用不同的数据进行相同的测试。不熟悉脚本语言的测试员可以从一个表格内输入测试数据并执行事先定义好的测试脚本。

在数据驱动技术中可以使用一些其它的技术，用一个表格内的元素组合来替代硬编码数据，使用基于可配置参数的算法在运行时生成并提供应用数据。例如，工具可以通过算法生成一个随机的用户 ID，并按可重复的模式，用随机数种子可以控制随机模型。

在关键字驱动的测试方法中，电子表格含有描述系统要采取的行为的关键字（也称为行为字）和测试数据。测试员（即使不熟悉脚本语言）也能针对被测应用，使用这些关键字来定义测试。

所有的测试方法都需要脚本语言方面的技术专家（或测试员或测试自动化专家）。

无论使用什么脚本技术，都需要储存每次测试的预期结果，便于后续比较。

静态分析工具

静态分析工具检查源代码是否遵循编码标准，但是如果应用于已经存在的代码会产生大量警告信息。警告信息不阻碍代码转换成可执行程序，但理想情况是应予以处理，以便将来更容易进行代码的维护。在开始就有计划的逐步使用分析工具来过滤一些警告信息是一种有效的方法。

测试管理工具

测试管理工具需要有与其他工具和表格有接口，以便产生符合组织所需格式的有用信息。

6.3.组织内引入工具（K1）15 分钟

术语

无

背景

为组织选择工具所需要考虑的关键点有：

- 评估组织的成熟度、分析引入工具的优点和缺点和认识引入工具能改善测试过程的可能性；
- 根据清晰的需求和客观的准则进行评估；
- 概念验证，在评估阶段要确认在现有的情况下使用工具对被测软件是否有足够效果，或为了有效使用工具，目前的基础设施需要如何改变；
- 评估供应商（包括培训、提供的支持及其他商业方面考量），如果是非商业性工具要评估提供服务的供应商；
- 为了在工具使用方面得到更好的指导和培训，需要先收集内部需求；
- 评估培训需求时需要考虑现有测试团队的自动化测试技能；
- 根据实际的情况估算成本-收益比。

将选择的工具引入组织要从一个试点项目开始，试点项目有以下目的：

- 对工具有更多的认识；
- 评估工具与现有的过程以及实践的配合程度，确定哪些方面需要作修改；
- 定义一套标准的方法来使用、管理、储存和维护工具及测试资产（比如，定义文件和测试的命名规则、创建库和定义模块化测试套件）；
- 评估在付出合理的成本后能否得到预期的收益。

在组织内成功部署工具的因素包括：

- 逐步在组织的其余部分将工具推广到测试中；
- 调整并改进过程来配合工具的使用；
- 为新使用者提供培训和指导；
- 定义使用指南；
- 实施一种在实际运用中收集工具使用情况的方法；
- 监控工具的使用和收益情况；
- 为测试团队使用工具提供支持；
- 在所有团队内收集经验和教训。

参考文献：

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

7. 参考文献

标准

ISTQB Glossary of terms used in Software Testing Version 2.1
ISTQB软件测试专业术语中英文对照表v2.1（中文译者注）

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S.（2004）*CMMI, Guidelines for Process Integrationand Product Improvement*, Addison Wesley: Reading, MA；
见章节2.1

[IEEE Std 829-1998] IEEE Std 829™（1998）IEEE Standard for Software TesDocumentation,
GB/T 9386-2008《计算机软件测试文档编制规范》（主要参考IEEE 829-1998制定，中文译者注）
见章节2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™（2008）IEEE Standard for Software Reviews and Audits,
GJB 6389-2008《军用软件评审》（主要参考IEEE 1028-1997制定，中文译者注）
见章节3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software life cycle processes,
GB/T 8566-2007《信息技术软件生存周期过程》（ISO/IEC 12207::1995，ISO/IEC 12207:1995/Amd.1:2002, ISO/IEC 12207:1995/Amd.2:2004,MOD，中文译者注），
见章节2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality,
GB/T 16260.1-2006《软件工程产品质量第1部分：质量模型》（等同采用，中文译者注），
见章节2.3

书籍

[Beizer, 1990] Beizer, B.（1990）Software Testing Techniques（2nd edition）, Van Nostrand Reinhold:Boston;
见章节1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R.（2001）Managing the Testing Process（3rdedition）, John Wiley & Sons: NewYork;
见章节1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al.（2001）Integrated Test Design and Automation, Addison Wesley:Reading, MA;
见章节6.2

[Copeland, 2004] Copeland, L.（2004）A Practitioner's Guide to Software Test Design, Artech House:Norwood, MA;
见章节2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P.（2002）Systematic Software Testing, Artech House:Norwood, MA;
见章节1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. and Graham, D.（1999）Software Test Automation, Addison Wesley:Reading, MA;
见章节6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy（1993）Software Inspection, Addison Wesley: Reading,MA；
见章节3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W.（1988）Complete Guide to Software Testing, QED: Wellesley, MA;
见章节1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. and Pettticord, B.（2002）Lessons Learned in Software Testing,John Wiley & Sons:
见章节1.1, 4.5, and 5.2

[Myers 1979] Myers, Glenford J.（1979）The Art of Software Testing, John Wiley & Sons;
软件测试的艺术，机械工业出版社，
见章节1.2, 1.3, 2.2, and 4.3

[Van Veenendaal, 2004] van Veenendaal, E.（ed.）（2004）The Testing Practitioner（Chapters 6, 8, 10）,UTN Publishers: The Netherlands;
见章节3.2, 3.3

8. 附录 A——课程大纲背景

本文档的历史

在 2004-2011 年期间，国际软件测试认证委员会（ISTQB）指定了一些软件方面的专家组成一个工作组，进行课程大纲的编写。课程大纲由挑选的评审成员进行初步评审，然后由国际软件测试联盟的代表们评审。课程大纲设计的具体规则等可以参考附录 C。

本文档是国际软件测试初级认证课程大纲，是 ISTQB（www.istqb.org）批准的第一级别的国际资质。

初级认证资质的目标

- 认识到测试是一门必要而专业的软件工程分支课程；
- 为测试员职业发展提供一个标准化的框架；
- 使具有专业水准的合格测试员能被雇主、客户和同事认可，并且提升测试员的竞争力；
- 在软件工程原则下，促进一致和良好的测试实践；
- 明确测试如何与软件行业相关，并具有何种价值；
- 企业通过宣传测试员招聘政策来雇佣经过认证的测试员，从而获得相对于对手更有竞争力的商业优势；
- 为测试员和对测试感兴趣的人们提供一个机会，来获得国际上认可的测试资质证书。

国际资质认证的目标（采用了 2001 年 11 月在 Sollentuna 召开的 ISTQB 会议精神）

- 在不同国家之间进行测试技能方面的相互比较；
- 测试员在不同的国家之间进行交流更加容易；
- 在跨不同国家项目和国际项目上，不同国家和地区的测试员可以对测试问题有一个共同的理解；
- 增加全球范围内具有资质的测试员数量；
- 通过成立国际组织进行测试资质的认证更具有号召力，这将比某个国家的组织具有更大的影响力和更高价值；
- 通过学习本大纲和相关的术语表，可以对测试有一个国际通用的了解和认识，可以提升所有参与者的测试知识和技能水平；
- 促进更多的国家将测试工作职业化；
- 测试员可以在本国用本国语言获得国际通用的资质证书；
- 在不同的国家之间共享测试知识和资源；
- 通过更多国家的参与，使得测试员和该资质得到国际认可。

初级软件资质的入门要求

参加 ISTQB 软件测试初级资质考试的基本要求是参与者对软件测试有兴趣。然而，我们还是强烈建议参与者同时具有：

- 在软件开发或软件测试领域具有基本的行业背景，比如有 6 个月的系统测试或用户验收测试的经验或者软件开发人员等；
- 参加并且通过了拥有 ISTQB 授权，并符合 ISTQB 标准的学习课程（ISTQB 认可的某个国家委员会）。

软件测试初级认证的背景和历史

独立的软件测试员认证开始于英国计算机协会 ISEB（信息系统考试委员会），其软件测试委员会（www.bcs.org.uk/iseb）成立于 1998 年。在 2002 年德国的 ASQF 开始实施德国的测试员资质认证计划（www.asqf.de）。本大纲基于英国的 ISEB 和德国的 ASQF 的大纲内容，对它们进行了一些更新以及增加了一些新的内容，并且将重点转向了为测试员提供更多实践帮助。

在本国际认证推出以前获得的初级软件测试证书（比如从 ISEB、ASQF 或 ISTQB 认可的国家认证委员会证书）被视为和本国际认证一样有效。初级认证证书不会过期失效，也不需要重新进行更新。获得证书的时间在证书上注明。

在每个参与国，具体的事务由 ISTQB 认可的国家软件测试委员会来管理。国家委员会的具体职责由 ISTQB 制定，由每个国家具体实施。国家委员会的具体职责包括培训机构的授权和认证考试的管理等。

9. 附录 B——学习目标和知识级别

下面定义的学习目标适合于本大纲，大纲中的每个主题都会根据其学习目的进行考试。

级别 1：牢记（K1）

考生可以识别、牢记和回忆术语或者概念的内容。

关键字：牢记（remember）、回忆（retrieve）、记忆（recall）、识别（recognize）、认知（know）。

例子

能够识别“失效”的定义：

- “不能向最终用户或其他干系人提供服务”或
- “组件或者系统的实际运行情况与期望的发布、服务或结果背离”。

级别 2：理解（K2）

考生应能够对大纲主题相关的内容进行解释和分析，同时对测试概念能够进行总结、比较、分类并举例说明。

关键字：总结（summarize）、概括（generalize）、摘要（abstract）、归类（classify）、比较（compare）、映射（map）、对比（contrast）、举例说明（exemplify）、解释（interpret）、翻译（translate）、描述（represent）、推断（infer）、结论（conclude）、分类（categorize）、构建模块（construct models）。

例子

请解释测试应该尽早进行设计的原因：

- 在缺陷移除成本低的时候就发现它们。
- 尽早发现那些最重要的缺陷。

请解释集成测试和系统测试之间的异同：

- 相同：需测试不同组件，可以对非功能性的测试点进行测试
- 不同：集成测试关注于接口和交互，而系统测试则关注于从全系统角度进行测试，如端到端的流程。

级别 3：应用（K3）

考生应可以选择正确的测试概念或者技术，并应用到给定的场景中。

关键字：实施（implement）、执行（execute）、运用（use）、遵循流程（follow a procedure）、使用流程（apply a procedure）。

例子

- 可以为有效和无效等价类识别边界值。
- 能从给定的状态转换图生成所需的测试用例，并能覆盖所有的状态转换。

级别 4：分析（K4）

考生应可以将与流程或技术有关的信息分成不同组成部分，以便更好的理解，可以对现象和结论进行划分。典型的应用是对文档、软件、项目的情况分析，并提出合适的建议来解决问题或任务。

关键字：分析（analyze）、组织（organize）、寻找一致性（find coherence）、整合（integrate）、梗概（outline）、解析（parse）、结构（structure）、特性（attribute）、解构（deconstruct）、区别（differentiate）、辨别（discriminate）、区分（distinguish）、焦点（focus）、选择（select）。

例子

- 分析产品风险，提供风险预防措施和缓解措施。
- 描述缺陷报告中的哪些部分是真实的，哪些内容是从测试结果中推导出来的。

参考文献：

（针对学习目标的级别）

Anderson,L.W.and Krathwonhl,D.R.（eds）（2001）A Taxonomy for Learning,Teaching,and Assessing:A Revision of Bloom’s Taxonomy of Educational Objectives,Allyn & Bacon

10. 附录 C——ISTQB 的规定

初级大纲

本规定用于本课程大纲的开发和评审。(在每条规定的后面都有一个“标注”代表规定的速记缩写。)

10.1.1. 概要规定

SG1：本课程大纲应该对具有 0 到 6 个月（或更长）测试经验的学员而言容易理解和吸收。(6 个月)

SG2：本课程大纲应该是实践性质而不是纯理论。(实践性)

SG3：对于潜在读者来说，本课程大纲清楚而没有异议。(清楚性)

SG4：本课程大纲对于来自不同国家的学员都容易理解，并且可以轻松的将课程大纲翻译成不同的语言。(可翻译性)

SG5：本课程大纲采用美国英语撰写。(美国英语)

10.1.2. 当前的内容

SC1：本课程大纲应该包含最新的测试概念，反映当前软件测试中普遍认可的最佳实践。课程大纲需要每 3 到 5 年左右进行一次评审。(最新的)

SC2：本课程大纲尽量少包含时间相关的问题，比如当前的市场条件，从而可以保证课程大纲有 3 到 5 年的保质期。(保质期)

10.1.3. 学习目标

LO1：学习目标通过不同的级别来进行区分：认知和牢记用认知级别 K1 表示，要求参与者从概念上理解的内容为 K2 级别，而要求参与者能够应用于实践/应用的级别为 K3，要求参与者能够在具体环境中分析文档、软件、项目情况的级别为 K4（知识级别）

LO2：内容的描述应当和学习目的相一致。(学习目的一致性)

LO3：应该按本大纲对每个主要章节给出考题例子，明确解释学习目标和考试。(学习目标-考试)。

10.1.4. 总体结构

ST1：课程大纲的结构应当清楚，并且在不同章节之间可以进行相互的参考和引用，也可以通过考题和相关的文档进行引用。(交叉参考引用)

ST2：尽量避免课程大纲章节之间的内容重叠。（重叠）

ST3：课程大纲的每个章节应该有相同的结构。（结构一致性）

ST4：课程大纲每一页应该包含版本、发布日期以及每页的页码等。（版本）

ST5：课程大纲应该包含每个章节学习需要花费的时间指南（从而来反映相关章节主题的重要性）。（学习时间）

参考资料

SR1：课程大纲中提到的概念会标明给出来源和参考资料，帮助培训机构可以获得某个章节主题的更多相关的信息。（参考资料）

SR2：假如有些来源尚未识别也不清楚参考资料无法清楚表示，本课程大纲中会提供更多的详细信息。比如，术语表对相关的术语进行了定义，所以在课程大纲中只是列出术语，而没有解释。（无参考资料细节）

信息资料来源

本课程大纲中使用的术语，在 ISTQB 软件测试术语表文档中进行详细定义。ISTQB 的当前术语表可从 ISTQB 获得。

与本课程大纲同步，也列出了一些关于软件测试的推荐的书籍。主要的书籍列表也是参考资料的一部分。

11. 附录 D——培训机构注意事项

本大纲的每个主要章节的标题中都标示了以分钟计算的分配的学习时间。这样做的目的一方面是为了提供认可课程中每个章节分配的时间比例，另一方面是给出进行相关章节培训需要花费的最少时间。培训机构可以花费比章节标示的更多时间来教学，而学员也可以花费更多的时间来进行学习和研究。具体的教学课程可以采用和本大纲不一样的顺序进行。

本大纲参考了一些已经建立的标准，在编写培训教材的时候需要用到这些标准。采用的每个标准都必须是本大纲当前版本引用的版本。在本大纲中没有引用的其他出版物、模板或标准，也可以使用和参考，只是考试的时候不会涉及到这些内容。

培训教材中所有 K3 和 K3 学习目标，要求有实践练习。

12. 附录 E——发布备注

2010 版

1. 对学习目标（LO）改动的一些说明
   a) 以下 LO 有用词上的改变（内容和 LO 级别不变）：LO-1.2.2、LO-1.3.1、LO-1.4.1、LO-1.5.1、LO-2.1.1、LO-2.1.3、LO-2.4.2、LO-4.1.3、LO-4.2.1、LO-4.2.2、LO-4.3.1、LO-4.3.2、LO-4.3.3、LO-4.4.1、LO-4.4.2、LO-4.4.3、LO-4.6.1、LO-5.1.2、LO-5.2.2、LO-5.3.2、LO-5.3.3、LO-5.5.2、LO-5.6.1、LO-6.1.1、LO-6.2.2、LO-6.3.2。
   b) LO-1.1.5 已被重述并升级至 K2，因为与缺陷相关的术语之间的比较可以保持一致。
   c) LO-1.2.3（K2）是新增加的，其内容在大纲 2007 中已覆盖到。
   d) LO-3.1.3（K2）现在合并了 LO-3.1.3 和 LO-3.1.4 的内容。
   e) LO-3.1.4 被移除，因为部分内容与 LO-3.1.3 重复。
   f) LO-3.2.1 已被重述，以使其与大纲 2010 内容一致。
   g) LO-3.3.2 升级至 K2 以便与 LO-3.1.2 一致。
   h) LO-4.4.4 被修改的更清晰，从 K3 升级到 K4。因为 LO-4.4.4 已经以 K4 方式表达了。
   i) LO-6.1.2（K1）被从大纲 2010 中移除，并且由 LO-6.1.3（K2）替代。大纲 2010 中已没有 LO-6.1.2。
2. 为保持测试方法(test approach)与术语表中的定义一致。术语“测试策略(test strategy)”将不再作为术语要求了。
3. 章节 1.4 现在包含了测试依据和测试用例之间的可追溯性的定义。
4. 章节 2.x 现在包含了测试对象和测试依据。
5. 再测试（re-testing）现在在术语表中代替确认测试成为主要术语。
6. 数据质量和测试方面在大纲的多处加入：数据质量与风险，章节 2.2、5.5、6.1.8。
7. 章节 5.2.3 入口准则是新加入的子章节，为了与出口准则对应。（入口准则被加入到 LO-5.2.9 中）。
8. 术语“测试策略（test strategy)”和“测试方法（test approach)”的使用与术语表的定义一致。
9. 章节 6.1 被缩短了，因为工具描述对于 45 分钟的课程来说太长了。
10. IEEE Std 829:2008 已经发布。2010 版大纲尚未考虑此新版本。章节 5.2 提到主测试计划的文档。主测试计划的内容被不同测试级别的“测试计划”所覆盖。我们既可以为每个测

试级别制定一个测试计划，也可以制定唯一的一个项目测试计划来覆盖多个测试级别。后一个在本大纲和 ISTQB 术语表中称为“主测试计划（Master Test Plan）”。

11. “道德规范”从高级大纲移至初级大纲。

2011 版

2011“修订版”改动地方如下：

1. 普遍：工作组（Working Group）代替工作组（Working Party）。
2. 用“前置条件”代替“前置-条件”，为了与 ISTQB 词汇表 V2.1 一致。
3. 第一次出现：ISTQB®代替 ISTQB。
4. 大纲简介：因为与附录 B 重复，知识的认知水平描述被去除。
5. 章节 1.6：因为目的不是定义“职业道德”的学习目标，这一节的认知水平被去除。
6. 章节 2.2.1，2.2.2，2.2.3 和 2.2.4，3.2.3：修订列表的格式。
7. 章节 2.2.2，术语失效用在“…某一特定的组件或系统中定位失效…”不是非常正确，因此用术语“缺陷”替换失效。
8. 章节 2.3：修正测试目标的列表的格式，与测试类型（K2）节中的测试术语有关。
9. 章节 2.3.4：更新调试的描述，以便与 ISTQB 软件测试专业术语对照表 V2.1 一致。
10. 章节 2.4，从 “…包括大范围的回归测试…”去除“大范围的”，因为“大范围的”要根据修改（大小、风险、价值等）而定，这在下一句中阐述。
11. 章节 3.2，去除“包括”以使句子更清晰。
12. 章节 3.2.1：因为正式评审的活动被不正确的格式表示，评审活动打算用 12 个主要活动替代 6 个。这已经被改回到 6 个主要活动，使得与本大纲 2007 和 ISTQB 高级大纲 2007 兼容。
13. 章节 4：“开发的”被替换为“定义的”，因为测试用例是被定义的而不是被开发的。
14. 章节 4.2：文字修改以清晰黑盒测试和白盒测试是怎样与基于经验的技术联合使用的。
15. 章节 4.3.5：文字修改，“…参与者之间， 包括用户与系统…”改为“参与者（用户或系统）之间，…”。
16. 章节 4.3.5，“替代路径”被“替代场景”代替。
17. 章节 4.4.2：为了清晰章节 4.4 中的术语“分支测试”，修改了句子使得分支测试更清楚。
18. 章节 4.5 和 5.2.6：术语“基于经验过的”测试被正确的术语“基于经验的”替代。

19. 章节 6.1：标题“6.1.1 理解使用测试工具支持测试的意义和目的（K2）”被“6.1.1 工具支持测试”。
20. 章节 7：文献：[Black, 2001]第 2 版本被第 3 版本替代。
21. 附录 D：所有学习目标 K3 和更高级要求的实践练习已作为一般要求，这是 ISTQB 授权过程（版本 1.26）的特别要求。
22. 附录 E：版本 2007 与 2010 之间改动的学习目标已正确列出。

13. 附录 F——索引