软件测试工程师认证

高级课程大纲

2007版

国际软件测试认证委员会

高级课程大纲工作组：Bernard Homès （主席），Graham Bath，Rex Black，Sigrid Eldh，Jayapradeep Jiothis，Paul Jorgensen，Vipul Kocher，Judy McKay，Klaus Olsen，Randy Rice，Jürgen Richter，Eric Riou Du Cosquer，Mike Smith，Geoff Thompson，Erik Van Veenendaal，2006-2007 年。

版本历史

版本	日期	说明
ISEB V1.1	2001 年 9 月 4 日	ISEB 从业人员大纲
ISTQB 1.2E	2003 年 9 月	ISTQB 高级课程大纲（EOQ-SG）
2007 版	2007 年 10 月 12 日	认证测试工程师高级课程大纲 2007 版
2007 中文版	2009 年 4 月 15 日	认证测试工程师高级课程大纲 2007 中文版
2007 中文修订版	2010 年 8 月 20 日	软件测试工程师认证高级课程大纲 2007 中文修订版

目录

致谢

CSTQB 致谢

ISTQB 软件测试工程师认证高级课程大纲中文修订版（以下简称“中文修订版”）是在 2007 中文版（2009 年 4 月 15 日）的基础上修订而成的，修订过程中以 2007 英文版（2007 年 10 月 12 日）的内容为依据。在此，CSTQB 向参加中文修订版修订工作的各位 CSTQB 测试专家表示衷心的感谢，对他们为 CSTQB 和中国软件测试业的奉献表示敬意。

中文修订版的策划负责人：

崔启亮

中文修订版的修订人员（按姓氏笔划排序）：

马均飞、李华北、陈耿、宣以广、贺炘、崔启亮、曹静、熊晓虹

中文修订版的终审人员（按姓氏笔划排序）：

马均飞、崔启亮

中文修订版完成于 2010 年 8 月。

ISTQB 致谢

2007 英文版由国际软件测试认证委员会高级工作组的核心团队编制，他们包括: Bernard Homès（主席）, Graham Bath, Rex Black, Sigrid Eldh, Jayapradeep Jiothis, Paul Jorgensen, Vipul Kocher, Judy McKay, Thomas Mueller, Klaus Olsen, Randy Rice, Jürgen Richter, Eric Riou Du Cosquer, Mike Smith, Geoff Thompson, Erik Van Veenendaal.

核心团队向评审团队和所有测试委员会成员提出的建议和帮助表示感谢。

直至高级课程大纲截稿时，其工作组成员如下（按字母排序）：

Graham Bath, Rex Black, Robert Bender, Chris Carter, Maria Clara Choucair, Sigrid Eldh, Dorothy Graham, Bernard Homès （chair）, Jayapradeep Jiothis, Vipul Kocher, Anastasios Kyriakopoulos, Judy McKay, Thomas Mueller, Klaus Olsen, Avinoam Porat, Meile Posthuma, Erkki Pöyhönen, Jürgen Richter, Eric Riou Du Cosquer, Jan Sabak, Hans Schaefer, Maud Schlich, Rajesh Sivaraman, Mike Smith, Michael Stahl, Geoff Thompson, Erik Van Veenendaal.

下列成员参与了评审、评论和大纲表决工作：

Bernard Homès （主席）

Reto Armuzzi
Sue Atkins
Graham Bath
Paul Beekman
Armin Beer
Rex Black
Francisca Blunschi
Armin Born
Con Bracke
Chris Carter
Maria Clara Choucair
Robert Dankanin
Piet de Roo
Sigrid Eldh
Tim Edmonds
Erwin Engelsma
Graham Freeburn
Dorothy Graham
Brian Hambling
Jeff B Higgott
Bernard Homès
Rob Hendriks
Dr Suhaimi Ibrahim
Phillip Isles
Pr. Paul C. Jorgensen
Vipul Kocher
Anastasios Kyriakopoulos
Junfei Ma
Fergus McClachlan
Judy McKay

Don Mills
Gary Mogyorodi
Richard Morgan
Silvio Moser
Ernst Müller
Reto Müller
Thomas Müller
Peter Mullins
Beat Nagel
Richard Neeve
Klaus Olsen
Dale Perry
Helmut Pichler
Jörg Pietzsch
Avionam Porat
Iris Pinkster
Horst Pohlmann
Meile Posthuma
Eric Riou Du Cosquer
Stefan Ruff
Hans Schaefer
Maud Schlich
Rajesh Sivaraman
Mike Smith
Katja Stalder
Neil Thompson
Benjamin Timmermans
Chris van Bael
Jurian van de Laar
Marnix van den Ent
Mark van der Zwan
Stephanie van Dijck
Jan van Moll
Erik Van Veenendaal
Roland Weber
Phillip Whettlock
Derek Young
Mike Young
Wenqiang Zheng.

本文英文版由 ISTQB®大会于 2007 年 10 月 12 日正式发布。

0. 课程大纲引言

0.1 国际软件测试认证委员会

国际软件测试认证委员会（International Software Testing Qualifications Board，简称 ISTQB®）由多个国家和地区的委员会组成。截止本版本大纲发布时，ISTQB®已经拥有 33 个委员会成员。关于 ISTQB 组织和成员的更多信息，请访问www.istqb.org。

本文档的编写目的

本课程大纲是 ISTQB 高级软件测试认证的基础。ISTQB®课程大纲开发的目标如下：

1. 委员会成员可以将大纲翻译成本国语言，可用于对培训机构的认证。委员会成员可按照本国语言需求对大纲进行修订，以适应当地出版要求。
2. 考试组可以依据课程大纲开发符合每个模块学习目标的本地化试题。
3. 培训机构可以根据课程大纲开发培训课件，并使用适合的培训方法进行教学。
4. 认证备考者，可以依据大纲进行考试准备（单独进行考试准备或者作为培训过程的一部分）
5. 对于全球软件工程和信息系统工程领域的组织、软件和系统测试的从业者，该大纲可以作为编写相关书籍和文章的参考材料。

有关组织以其它目的使用本大纲，需要事先取得 ISTQB®的书面许可。

认证高级软件测试工程师

高级认证证书适合于在软件测试职业生涯中希望取得领先优势的从业者。其目标群体涵盖：测试员、测试分析员、测试工程师、测试咨询人员、测试经理、用户验收测试人员以及软件开发人员。该高级认证也适合于希望深入理解软件测试的人员，如：项目经理、质量经理、软件开发经理、业务分析人员、IT 主管和管理顾问。希望参加高级认证考试的人员必须先取得初级认证考试证书，并需向考试委员会证明其拥有足够的实践经验，具备参加高级认证考试的资格。具体实践经验标准可向当地的考试委员会咨询。

知识水平

每一章都划分了学习目标，以便在独立的模块中清晰识别。关于学习目标的更多细节和实例请参见 0.3 节。

本课程大纲的内容，术语、主要原则（目标）和所有标准，如果没有特别标识，都应至少达到牢记（K1）的要求。

考试

所有高级认证考试都必须以本课程大纲和初级课程大纲为基础。答题时，可能需要使用本课程大纲和初级课程大纲多章的内容作为基础进行解答。这意味着本课程大纲和初级课程大纲的各章都在考试范围之内。

ISTQB®高级考试指南中定义了考试的形式。如有必要，独立的成员单位可以选择其它考试形式。

考试可以作为授权认证培训课程的一部分，也可以独立进行（例如，在授权的考试中心）。考试既可以书面进行，也可以以电子方式进行，但是所有的考试必须在考试认证委员会或者考试委员会授权的人员监督下进行。

授权

ISTQB®的成员委员会可以对课程材料符合本大纲要求的培训机构进行授权。授权指南可以从当地委员会或者其实体获取。获得认证的培训课程必须符合本大纲的要求，并将 ISTQB®的认证考试作为课程的一部分。

关于培训机构指导的更多信息，请参见附录 C – 培训机构注意事项。

细节

为使教学和考试在国际范围内保持统一标准，本课程大纲由下面几部分组成：

- 高级课程大纲总体目标。
- 各个领域所需要达到的学习目标和认知水平
- 教学中需要的资料和必须的附加信息列表
- 学员必须理解和牢记的知识列表。
- ，已经被接受的文献和标准资源的教学理念。

本大纲并不是对软件测试领域知识的全面介绍；它只是涵盖了认证测试工程师高级课程中需要涉及的内容。

大纲是如何组织的

本大纲包含 10 个主要章节，每章都包含简介部分，其每个简介部分都对如何应用不同测试专业（或测试模块）提出了深刻见解。

为了方便培训，0.3 至 0.6 节针对每章的不同模块，提供了特定的学习目标。章内还包括了本模块培训的最低时间要求。

我们强烈建议在学习每章时，同步阅读大纲以确定学习目标。这将使读者可以充分理解各章所必需的基本内容。

术语和定义

在软件著述中所使用的众多专业术语，往往都可以交替的使用。本高级课程大纲中的术语定义，来源于 ISTQB®发布的软件测试标准术语表。

方式方法

实施软件测试活动可以有多种途径，例如可以基于规格说明，代码结构，数据，风险，过程，标准以及基于类似分类等实施测试。不同的流程和工具对测试流程进行支持；不同的方法可以对现有流程进行改进。

本高级课程大纲根据 ISO 9126 推荐的方法进行组织，按照功能性、非功能性和辅助方法等进行划分。支持流程和改进方法也被提及。为高级测试人员和测试管理者改进组织和流程提供了依据。

0.2 预期

本高级课程大纲将从三个主要方面对应试者的思路进行考察，每一方面都代表了组织对应试者的职责和期望。在任何组织中，职责和相关的任务都会由不同的人员或独立一个人承担完成。工作职责描述如下：

0.2.1 高级测试经理

高级测试管理专业人员应有能力完成下列工作：

- 定义被测系统的总体测试目标和测试策略。
- 计划任务、进度控制和任务跟踪。
- 描述和组织必要的活动。
- 选择、获取和分派任务所需的足够资源。
- 选择、组织和领导测试团队。
- 组织测试团队成员之间、测试团队和其它项目干系人之间的沟通。
- 进行决策并提供决策所依据的足够信息。

0.2.2 高级测试分析员

高级测试分析员应有能力完成如下工作：

- 按照测试策略中的业务需求结构化测试任务。
- 详细分析系统以满足用户对质量的期望。
- 评估系统需求以确定有效测试范围。
- 准备和执行足够的测试活动并报告进度。
- 提供必要的信息以支撑评估结论。
- 部署必要的工具和技术，实现预先定义的测试目标。

0.2.3 高级测试技术分析员

高级测试技术分析员应有能力完成如下工作：

- 按照测试策略中的技术需求结构化测试任务。
- 详细分析系统内部架构以满足期望的质量要求。
- 从技术质量特性上评估系统，如性能、安全性等。
- 准备和执行足够的测试活动并报告项目进度。
- 领导技术测试活动。
- 提供必要的信息以支撑评估结论。
- 部署必要的工具和技术，实现预先定义的测试目标。

0.3 学习目标/知识等级

本大纲定义和应用下列学习目标。本大纲中的每个主题都应根据其定义的学习目标进行考核。

等级 1：牢记（K1）

考生可以分辨、记住和复述术语或者概念的内容。

关键词：牢记、回忆、识别、认知

例子

可以识别“失效”的定义：

- “不能向最终用户或其他利益共享者提供服务”或....
- “组件或者系统的实际运行情况是否符合预期标准”。

等级 2：理解（K2）

考生应可以对给出的相关主题选择原因或解释，可以举出实例（例如对比术语）、测试概念、测试过程（解释任务流程）进行总结、区分、分类并给出相应实例。

关键词：总结、归类、比较、映射、对比、举例说明、解释、翻译、描述、推断、结论、分类。

举例
请解释尽早开始测试设计的原因：
- 在清除缺陷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：需测试多个组件，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的各个方面，如端到端的流程。

等级 3：应用（K3）

考生应可以选择正确的测试思想或者技术，并应用于给定场景中。K3 一般适用于过程性知识。没有太多创造性活动，如评估一个应用软件，或者为特定软件创建测试模型。给定一个测试模型，覆盖了教学大纲中的程序步骤，根据这个模型生成测试用例，就是类似于 K3 的工作。

关键词：实施、执行、运用、遵循流程、使用流程。

举例
- 可以识别有效边界值和无效边界值
- 使用通用的测试用例生成流程，从给定的状态转换图（或一个测试用例集）中派生出测试用例并覆盖所有的状态转换。

等级 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。典型的应用是为解决某些问题或任务而进行的文档、软件、项目情况分析，并提出合适的建议。

关键词：分析、区分、选择、结构、定位、特性、解构、评估、判定、监控、协调、创建、合成、产生、假定、计划、设计、构建、生产。

举例
- 分析产品风险，进行风险预防和制定容灾方案。
- 描述事件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（关于学习目标的认知等级）

Bloom, B. S. （1956）. *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain,* David McKay, Co. Inc.
Anderson, L. W. and Krathwohl, D. R. （eds） （2001）. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon.

0.4 测试经理的学习目标

本节列举了测试经理模块的详细学习目标。

本大纲的所有内容至少要在 K1 级别上进行考察。考生应可以识别、牢记和回忆这些内容。

因此，下面的学习目标仅包含 K2、K3 和 K4 的内容。

测试经理大纲简介 – [60 分钟]
（包括ISTQB®初级大纲的修订内容）

第 1 章：软件测试基础 – [150 分钟]
1.2 软件生命周期中的测试
- （K2）描述如何进行测试才算是某些软件开发与维护活动的一部分。

- （K4）分析软件生命周期模型并勾勒出最适当的任务/测试活动去实施（区分测试活动和开发活动）

1.3 特定系统

- （K2）举例说明测试综合系统的详细过程。
- （K2）解释为什么系统安全测试的三个主要输出必须满足相关的法规

1.4 测度和度量

- （K2）描述和比较所有标准化的测试方法
- （K3）通过度量测试对象和测试过程来监控测试活动

第 2 章：测试过程 – [120 分钟]

2.3 测试计划与控制

- （K2）举例描述测试策略如何影响测试计划。
- （K2）比较测试阶段工作的产物，并通过举例解释开发阶段的产物与测试阶段的产物之间的关系。
- （K2）对测试控制活动进行分类以便确定测试任务、策略和目标是否已完成。

2.5 测试实现与执行

- （K2）说明测试执行的前提条件。
- （K2）通过例子说明不同测试技术在早期测试中的优点和缺点。
- （K2）解释为什么用户和（或）客户需要参与测试执行。
- （K2）描述如何根据不同的测试级别，确定测试日志的级别。

2.6 评估出口准则与报告

- （K2）总结测试过程中需要搜集的信息，用来完成准确的测试报告和评估是否满足测试标准。

2.7 测试结束活动

- （K2）总结四组完成的测试活动。
- （K3）总结测试结束阶段获得的经验教训，发现改进点或者可重用点。

第 3 章：测试管理 – [1120 分钟]

3.2 测试管理文档

- （K4）列出满足 IEEE 829 标准的测试管理文档，例如测试计划、测试设计规格说明和测试规程规格说明。
- （K2）描述测试策略/测试方法中至少 4 个重要因素，并说明哪些满足 IEEE 829 标准的测试文档包含了这些因素。
- （K2）说明如何在测试管理文档中对测试策略的偏差进行管理，并简述原因

3.3 测试计划文档模板

- （K2）总结 IEEE 829 中主要测试计划的结构。
- （K2）根据各个组织结构、产品风险，以及项目的风险、规模和类型等情形裁剪 IEEE 829 标准中建议的测试计划结构。

3.4 测试估算

- （K3）考虑影响成本和时间的因素，以一个小型样例项目进行基于度量或基于经验的方法估算工作量。
- （K2）理解并举例说明在大纲中提到的可能引起估算误差的因素。

3.5 测试计划进度

- （K2）描述尽早制定测试计划和进行测试计划迭代的益处。用例子支持解释。

3.6 测试进度监控

- （K2）比较测试过程控制的不同流程。
- （K2）至少给出 5 个不同的概念性例子，解释测试过程中发现的问题如何影响具体的测试过程。
- （K4）利用在监控活动和测量中所观察到的测试进度相关的问题，制订应对计划改进当前的测试进度，并提供改进建议。

- （K4）分析测试结果和确定测试进度，并记录到监控报告中，并从 4 个不同的维度生成最终的测试总结报告。

3.7 测试的商业价值
- （K2）给决定“质量成本”的 4 个类型分别举例（或列出度量）。
- （K3）在给定环境中列出适用的定量值和（或）定性值。

3.8 分布式测试、外包测试与内包测试
- （K2）列出三种测试类型（分布式测试、外包测试和内包测试）的人员编制策略的风险和异同。

3．9 基于风险的测试
3.9.1 基于风险的测试简介
- （K2）解释基于风险的测试如何在多个方面考虑风险因素。
- （K4）区别产品风险和项目风险，并能基于这些风险制定合理的测试策略和测试计划。

3.9.2 风险管理
- （K3）从测试人员的角度，根据 FMEA 方法对一个产品进行风险分析。
- （K4）汇总项目关键成员对项目风险的不同观点，使用汇总后的结果开展减轻风险的测试活动。

3.9.3 软件生命周期中的风险管理
- （K2）描述需要不断迭代的风险管理的特性。
- （K3）将基于风险的测试策略转化为实际的测试活动，并在测试过程中监控其效果。
- （K4）分析和报告测试结果，确定或提出残余的风险以帮助项目管理人员做出正确的决断。

3.10 失效模式与影响分析
- （K2）描述 FMEA 的基本概念，通过例子解释其在项目中的应用和对项目的益处。

3.11 测试管理问题
- （K2）从测试策略、益处、优缺点以及这些因素对测试计划、覆盖、监视、控制等方面，比较探索性测试、综合系统、安全关键系统测试在测试管理中的不同之处。

第 4 章：测试技术 – [0 分钟]
对测试管理者，本部分不做要求。

第 5 章：软件特征测试 – [0 分钟]
对测试管理者，本部分不做要求。

第 6 章：评审 – [120 分钟]
6.2 评审的原则
- （K2）通过动态测试和其它静态测试技术的比较，解释评审的优点。

6.4 评审简介
- （K2）比较不同的评审类型，说明它们各自的优势、劣势以及应用领域。
- （K3）按照规定步骤带领评审团队进行一次正式的评审活动。
- （K4）列出评审计划（作为项目质量/测试计划的一部分），计划中应考虑评审的技术、可能发现的缺陷、测试人员具备的技能，并与其它动态测试方法有机结合在一起。

6.5 评审的成功因素
- （K2）解释在不考虑实施评审的技术、组织因素以及人员因素的情况下，进行评审的潜在风险。

第 7 章：事件管理 – [80 分钟]
- （K3）根据 IEEE 标准 1044-1993 定义的事件管理生命周期进行缺陷管理。
- （K3）根据 IEEE 标准 1044-1993 评估缺陷报告并进行缺陷分类以改进缺陷报告的质量。
- （K4）按时对缺陷报告进行分析并更新缺陷分类。

第 8 章：标准和测试改进过程 – [120 分钟]

- （K2）总结相关标准的出处并解释这些标准在软件测试中的作用。

8.4 改进测试过程

- （K3）与相关人员一起按通用步骤编制测试改进计划。
- （K2）总结 TMM、TPI、CTP、STEP 定义的测试改进过程和 CMMI 过程域的验证和确认过程。
- （K2）解释测试改进模型 TMM、TPI、CTP、STEP 中的评估准则，以及 CMMI 中过程域的验证和确认标准。

第 9 章：测试工具与自动化 – [90 分钟]

9.2 测试工具的概念

- （K2）从下列多种角度对不同类型的测试工具概念进行比较：“益处和风险”、“测试工具使用策略”、“工具集成”、“自动化语言”、“测试准则”、“工具部署”、“开源工具”、“工具开发”和“工具分类”。
- （K2）描述为什么和何时使用工具对于创建测试工具使用策略是非常重要的。
- （K2）理解测试工具部署的不同阶段。

9.3 测试工具分类

- （K2）从测试目标、易用性、优点、风险和用途等角度对不同测试工具进行总结和分类。
- （K2）总结在测试安全系统时对测试工具和开源测试工具的特别需求。
- （K2）描述不同测试工具的特性和运用效果，总结不同测试工具的使用方法和作用。
- （K2）描述选择使用测试工具的时机和原因，并分析它将带来的益处、风险以及结果。

第 10 章：个人技能与团队构成 – [240 分钟]

10.2 个人技能

- （K3）通过问卷方式对团队成员的能力从以下角度进行评估：软件系统的熟悉程度、专业业务知识、开发能力、软件测试能力和沟通能力。

10.3 测试团队整体实力

- （K3）通过缺口分析确定组织内空缺职位所需的技术和能力。

10.4 组织内测试的适合性

- （K2）根据不同的组织类型进行外包/内部资源和外包/内部业务的选择。

10.5 激励

- （K2）举例说明影响测试人员的积极因素和消极因素。

10.6 沟通

- （K2）举例说明测试人员如何进行专业的、客观的、有效的沟通。在分析时应考虑相关的风险和机遇。

0.5 测试分析员的学习目标

本节列举了测试分析员详细的学习目标。

本大纲的所有内容都至少在 K1 级别上进行考察。考生应可以识别、牢记和回忆这些内容。因此，下面的学习目标仅包含 K2,、K3 和 K4 的内容。

测试分析员教学大纲介绍 – [60 分钟]
（包含 ISTQB®初级大纲修订版的内容）

第 1 章：软件测试基础 – [30 分钟]

第 2 章：测试过程 – [180 分钟]
2.4 测试分析与设计
- （K2）解释在软件生命周期哪个特定阶段开展功能性测试。
- （K2）举例说明哪些因素会影响测试环境搭建。
- （K2）描述测试分析和设计如何成为发现缺陷的静态测试技术。
- （K2）举例说明什么是测试准则，以及测试准则如何应用在测试规格的说明中。

2.5 测试实现与执行
- （K2）描述测试执行的前置条件，包括测试工件；测试环境、配置管理和缺陷管理。

2.6 评估出口准则与报告
- （K3）通过给定的一组测量数据，判断测试是否达到了完成标准。

第 3 章：测试管理 – [120 分钟]
3.9.2 风险管理
- （K3）基于风险对测试用例的选择、测试覆盖率和测试数据进行优先级划分，并对测试进度和测试过程进行相应的文档化。
- （K2）列出领域测试中，基于风险的方法进行计划和执行测试所需要的全部活动。

第 4 章：测试技术 – [1080 分钟]
4.2 基于规格说明
- （K2）列出基于规格说明的测试技术发现典型缺陷的例子，并提供相应的测试覆盖率准则。
- （K3）使用以下的测试用例设计方法，对给定的软件模型编写测试用例。（需要满足模型中要求的测试覆盖率）
  - 等价类划分
  - 边界值分析
  - 判定表驱动
  - 分类树方法
  - 组对测试
  - 用例场景分析
- （K4）分析系统或需求规格说明书，选择恰当的的测试技术来测试被测对象，并按照 IEEE 829 标准，列出功能测试用例、领域测试用例和测试过程的主要内容。

4.4 基于缺陷和基于经验
- （K2）描述基于缺陷的测试用例设计技术的使用原则，并将其和基于规格说明、基于结构的测试用例设计技术区别开来。
- （K2）通过示例解释缺陷分类及其应用
- （K2）理解基于经验测试用例设计技术的原则和原因以及何时采用该技术。
- （K3）使用探索性测试的方法指定、执行和报告整个测试过程。

- （K2）根据不同的软件错误，对缺陷进行分类。
- （K4）分析系统，从基于规格说明、基于经验和基于缺陷这三种测试用例设计技术中选择适合被测对象的技术。

第 5 章：软件特征测试 – [210 分钟]

5.2 领域测试的质量属性

- （K4）举例说明第 4 章列出的哪些测试技术适用于测试软件的功能准确性、适用性、交互性、功能安全性和易用性等特性。
- （K3）通过罗列、设计、指定和执行可用性测试，覆盖指定的测试项和目标缺陷。

5.3 技术测试的质量属性

- （K2）解释测试策略中效率测试、可靠性测试、技术安全测试的目的，并举例说明这些测试可以发现的缺陷类型。
- （K2）通过一些典型的缺陷对技术测试中不同类型的非功能性测试进行描述，说明它们在软件生命周期中的典型应用，以及哪些技术适合测试设计。

第 6 章：评审 – [180 分钟]

- （K3）使用评审列表，从测试人员的角度对代码和架构进行检查。
- （K3）使用评审列表，从测试人员的角度对需求和用例进行检查。
- （K2）比较不同类型的评审方法，说明它们各自的优缺点和适用领域。

第 7 章：事件管理 – [120 分钟]

- （K4）在缺陷报告中对功能性/非功能性缺陷进行分析、分类、描述。

第 8 章：标准和测试改进过程 – [0 分钟]

此部分对测试分析员不作要求。

第 9 章：测试工具与自动化– [90 分钟]

9.2 测试工具的概念

- （K2）从以下角度比较每个测试工具的原理和特性：“风险和收益”,、“测试工具策略”、“工具集成”、“自动化语言”、“测试准则”、“工具部署”、“开源工具”、“工具开发”、“工具分类”。

9.3 测试工具分类

- （K2）从目的、用途、优点、风险等角度对测试工具进行分类并举例说明。
- （K2）将工具列表中的工具映射到不同的测试级别和测试类型。

第 10 章：个人技能与团队构成– [30 分钟]

10.6 沟通

- （K2）举例说明测试人员如何在项目中进行专业的有效的沟通。描述时应考虑风险和机遇。

0.6 测试技术分析员的学习目标

本节列出了测试技术分析员模块的详细学习目标。

本大纲的所有内容都至少在 K1 级别上进行考察。考生应可以识别、牢记和回忆这些内容。因此，下面的学习目标仅包含 K2,、K3 和 K4 的内容。

测试技术分析员大纲介绍 – [60 分钟]
（包含 ISTQB®初级大纲修订版的内容）

第 1 章：软件测试基础 – [30 分钟]

第 2 章：测试过程 – [180 分钟]
2.4 测试分析与设计
- （K2）解释在软件生命周期的哪些阶段可采用非功能性测试和基于结构的测试。解释为什么非功能性测试仅在软件生命周期的某些特定阶段进行。
- （K2）举例说明影响测试环境搭建的准则。
- （K2）描述如何使用测试分析和设计的静态测试技术发现缺陷。
- （K2）举例说明什么是测试准则，以及测试准则如何应用在测试规格说明中。

2.5 测试实现与执行
- （K2）描述测试执行的前置条件，包括测试工件；测试环境、配置管理和缺陷管理。

2.6 评估出口准则与报告
- （K3）通过给定的一组测量数据，判断是否达到了测试完成的标准。

第 3 章：测试管理 – [120 分钟]
3.9.2 风险管理
- （K2）概述基于风险管理进行测试计划编写和执行测试的方法。

第 4 章：测试技术 – [930 分钟]
4.2 基于规格说明
- （K2）列出根据特定规格说明定位典型缺陷的例子。
- （K3）使用下面列出的测试用例设计技术，对给定的软件模型编写测试用例。（测试用例需要满足给定模型中要求的测试覆盖）
  - 等价类划分
  - 边界值分析
  - 判定表驱动
  - 状态转换测试
- （K4）分析一个系统或者它的需求规格说明，根据 IEEE 829 标准，使用基于规格说明的测试技术完成测试目标，，并列出组件测试用例、非功能测试用例和测试规程的主要内容。

4.3 基于结构
- （K2）列出根据特定规格说明测试技术定位的典型缺陷的例子。
- （K3）使用下列测试设计技术在实际环境下编写测试用例（测试用例需要满足给定模型中要求的覆盖率）
  - 语句覆盖
  - 判定覆盖
  - 条件/判定覆盖
  - 多重条件覆盖
- （K4）对系统进行分析，确定为达到特定的测试目标应选择何种基于结构的测试设计技术。

- （K2）理解每种基于结构的测试设计技术和与之相关的覆盖准则，并说明何时使用这些技术。
- （K4）能够分析比较在不同的情况下选择使用何种基于结构的测试用例设计技术。

4.4 基于缺陷和经验

- （K2）描述基于缺陷测试技术的原理和采用它的原因，并说明基于缺陷测试技术和基于规格说明设计技术、基于结构测试技术之间的区别。
- （K2）举例说明缺陷分类和它的应用。
- （K2）理解基于经验测试技术的原理和采用它的原因，并说明何时使用该种技术手段。
- （K3）阐述如何使用探索性测试技术完成测试执行和报告。
- （K3）阐述为获得不同的软件缺陷，应采用何种攻击手段进行测试。
- （K4）分析目标系统，确定应该如何选择使用基于规格说明设计、基于缺陷的技术和基于经验的测试技术完成特定测试目标。

4.5 静态分析

- （K3）使用“控制流分析”，“数据流分析”方法，验证代码中是否有不正常的控制流或数据流。
- （K4）根据工具中得到的结果，判断代码中是否存在控制流或者数据流异常。
- （K2）解释调用表在评估架构质量时的用途。包括可以发现的缺陷、测试设计和测试计划，结果的局限。

4.6 动态分析

- （K2）解释如何执行代码的动态分析，总结代码动态分析可以发现的缺陷以及其局限性。

第 5 章：软件特征测试 – [240 分钟]

5.2 领域测试的质量属性

- （K2）从可以发现的缺陷的类型角度，描述属于领域测试的非功能性测试类型的特点，及其在生命周期中的典型应用和适用于测试设计的测试技术。
- （K4）详细说明属于非功能性测试的特定测试用例，这些用例可以覆盖的测试目标，以及可以发现的缺陷的类型。

5.3 技术测试的质量属性

- （K2）通过对典型缺陷的分析，描述领域测试中非功能性测试类型的特点，及其在软件生命周期中的典型应用和适用于测试设计的测试技术。
- （K2）理解、解释在软件生命周期的哪个阶段可进行安全性、可靠性和效率测试（包括 ISO9126 对应的子特性）。
- （K2）区别安全性测试、可靠性测试和效率测试可以发现的错误的类型。（包括 ISO9126 对应的子特性）
- （K2）列出安全性测试、可靠性测试和效率测试的质量属性以及这些属性对应的 ISO9126 的子特性。
- （K3）设计安全性测试、可靠性测试和效率测试的测试用例并列出对应的 ISO9126 的子特性。
- （K2）理解、解释在测试策略中包含可维护性测试、可移植性测试和辅助性测试的原因。
- （K3）设计可维护性、可移植性的非功能性测试用例。

第 6 章：评审 – [180 分钟]

- （K4）列出代码评审和架构评审可以发现的典型的缺陷。
- （K2）列出不同的评审的类型并对比各自的优势、劣势和使用的领域。

第 7 章：事件管理 – [120 分钟]

- （K4）使用良好的的缺陷报告分析、分类、描述功能/非功能性缺陷。

第 8 章：标准和测试改进过程 – [0 分钟]

对测试技术分析员，此部分不作要求。

第 9 章：测试工具与自动化 – [210 分钟]

9.2 测试工具的概念

- （K2）从如下角度比较每个测试工具原理和特性：“风险和收益”、“测试工具策略”、“工具集成”、“自动化语言”、“测试准则”、“工具部署”、“开源工具”、“工具开发”和“工具分类”。

9.3 测试工具的分类

- （K2）从目的、用途、优点、风险等角度对测试工具进行分类并举例说明。
- （K2）将工具列表中的工具映射到不同的测试级别和测试类型。

9.3.7 关键字驱动测试自动化

- （K3）用关键字筛选方法创建关键字/活动字并在测试执行工具中执行。
- （K3）使用录制-回放工具执行测试，保证回归测试的高质量，高覆盖，高频率。

9.3.8 性能测试工具

- （K3）针对目标系统特性，使用性能测试工具设计一个性能测试，包括测试计划、度量标准。

第 10 章：个人技能与团队构成– [30 分钟]

10.6 沟通

- （K2）以测试人员的角度，从专业性、目标、有效性等方面举例描述如何进行有效沟通。在描述时应考虑风险和机遇。

1. 软件测试基础

术语：

规范，测量，度量，安全关键系统，综合系统，软件生命周期。

1.1 简介

本节将介绍一些测试专业人员如测试经理、测试分析员和测试技术分析员等都应了解的测试相关主题。培训机构应在背景介绍阶段对这些主题进行讲解并提供相关的例子。例如，在“测试技术分析员”模块中，“测量和度量”（请参见 1.4 节）的主题，可以使用某个具体的技术度量（如以性能度量指标为例）进行讲解。

在章节 1.2 中，测试过程被看做是整个软件开发生命周期的一部分。该主题以初级大纲中所介绍的基本概念为基础。此外，需要特别注意的是，软件测试的过程应该与软件开发的模型以及其它 IT 过程模型相结合进行介绍。

系统可以从多个角度来影响测试的实施。章节 1.3 将介绍两种所有的测试人员都需要关注的特定系统类型：综合系统（有时也称为多系统）和安全关键系统。

高级测试人员在把大纲中所描述的不同的测试特性引入到各自的组织、团队和任务中时，将会面临一系列的挑战。

1.2 软件生命周期中的测试

测试是多种软件开发模型中的重要组成部分：

- 顺序模型（瀑布模型、 V 模型和 W 模型）
- 迭代模型 （快速应用开发模型 RAD 和螺旋模型）
- 增量模型（进化和敏捷开发方法）

作为测试策略的一部分，应该考虑和定义测试的长期生命周期特性。内容包括组织、过程定义、工具或者方法的选择等。

测试过程并不是独立存在的，它与其他的一些过程有着密切关系，比如：

- 需求工程和需求管理
- 项目管理
- 配置管理和变更管理
- 软件开发
- 软件维护
- 技术支持
- 技术文档编制

前期的测试计划和后期的测试执行都是和顺序开发模型相关的。测试任务可能是重叠的也可能是并行进行的。

变更管理和配置管理对于支持软件测试任务很重要。如果缺乏有效的变更管理，变更对系统造成的影响将无法进行评估。如果没有配置管理，同时发生的版本演进有可能丢失或失去控制。

根据项目环境的不同，除了在初级大纲中定义的各种测试级别，还可能定义了附加测试级别，比如：

- 软硬件集成测试
- 系统集成测试
- 功能交互测试
- 用户产品集成测试

每一个测试级别都应具有如下的特性：

- 测试目标
- 测试范围
- 测试依据的可追溯性
- 入口和出口准则
- 测试交付物（包括测试报告）
- 测试技术
- 测量和度量
- 测试工具
- 组织依从性和其它标准依从性

根据内容的不同，每个测试级别的目标和范围可以独立考虑，也可以在项目层面进行考虑（即避免在不同级别上进行同一类不必要的重复测试）。

测试活动应该与选定的软件开发生命周期模型相匹配，这些模型包括：顺序模型 （瀑布模型，V 模型和 W 模型）、迭代模型（快速应用开发模型 RAD 和螺旋模型）或者增量模型（进化模型和敏捷开发方法）。

例如，在 V 模型中，应用于系统测试级别的 ISTQB®基本测试过程可以描述如下：

- 系统测试计划与项目计划同时进行，测试控制直到系统测试完整的被执行后测试活动才结束。
- 系统测试分析与设计，需与需求规格说明、系统和架构设计规格说明（高层次的）和组件设计规格说明（低层次的）等同时进行。
- 系统测试环境（如测试台、测试装置）的部署应与系统测试执行工作一同在系统规划中启动，并一直持续到系统测试执行开始的前几天（尽管大部分部署工作应与编码和组件测试同期进行）。
- 在系统测试的入口准则已经全部满足（或中止）时系统测试执行才开始，这通常是指至少已经完成了组件测试并且组件之间的集成测试也已经完成。系统测试的执行会一直持续到满足系统测试的出口准则为止。
- 对系统测试的出口准则的评估和系统测试结果的报告应贯穿于测试执行的整个过程，通常在项目截止日期到来前会加大评估的频率。
- 在系统测试出口准则已经满足要求并且系统测试执行已宣告完成之后，系统测试闭包活动才开始。尽管这些活动有可能被推迟到验收测试结束之后或者整个项目活动结束之后才进行。

对于每一测试级别或者已经选定的软件生命周期和测试过程的组合，测试经理都要在编写测试计划、项目计划的时候进行调整和匹配。对那些特别复杂的项目，如系统级别的项目（通常是应用于军方或大型企业），测试过程不仅仅是进行匹配，并且需要根据项目的来龙去脉进行修改（如运用高级别的测试比运用低级别的测试能更容易地发现缺陷）。

1.3 特定系统

1.3.1 综合系统

一种综合系统是由一系列关联的组件（包括硬件、独立的软件应用程序和通讯）集成的、彼此互连以实现共同目标、没有唯一的管理结构的系统。与综合系统相关的特性和风险如下：

- 利用集成独立软件的不间断归并去避免整个系统研发从头开始。此项目标可能会实现，例如可以通过检测设备系统与有限的辅助开发相结合的手段。
- （不同项目干系人之间的）技术和组织上的复杂性会为有效管理带来风险。构建系统所涉及的不同的开发生命周期方法被一些起作用的系统所采用，而这些系统可能会导致不同群组（开发、测试、制造业、装配线、最终用户等）之间的沟通问题。总体上来说，综合系统的管理必须能够应对不同的系统集成所带来的内在技术复杂性，同时也能处理多种组织问题（如外包和离岸开发）。
- 对相关业务信息的保密和保护，不同组织和用户（如政府和私营企业）之间的接口以及法规符合性（如反垄断法案）等都意味着综合系统非常复杂。
- 由于综合系统内在特性相比独立系统来说缺乏稳定性，某一个子系统的缺陷都将影响到整个综合系统。
- 技术和功能互用性的高要求使得综合系统中对独立组件的集成测试的要求也非常严格，同时需要设计出高认可度的、良好的接口。

1.3.1.1 对综合系统进行管理与测试

综合系统通常意味着其项目管理的高复杂性和组件配置管理的复杂性。这通常意味着复杂的系统和综合系统需要做好质量保证、流程规范定义等工作。在综合系统开发中，也需要实施正式的开发生命周期控制、里程碑和评审。

1.3.1.2 综合系统的生命周期特征

相比 1.2 节描述的软件生命周期中的测试，综合系统的每一测试级别的测试都具有如下的额外特征。

- 多层次的集成和版本管理 。
- 项目的持续周期长。
- 项目成员之间的正式的信息传递。
- 各个组件的开发不同步以及大量的系统回归测试需求。
- 由于更换或更新个别过时的组件而需要进行维护测试。

对综合系统，某一层次的测试必须考虑到本层次的测试细节以及与上一层次测试的整合。例如，对某一组件的系统测试可以看成是其所属的更高层次组件的单元测试。

通常来说，综合系统中的某一独立系统要进行各个层次的测试，然后还要再根据另外的测试需求整合到综合系统中。

3.11.2 节包含与综合系统管理相关的事项信息。

1.3.2 安全关键系统

安全关键系统是指具有如下特性的系统：如果系统的操作丢失或降级（譬如由于不正确的或粗心的操作）将造成灾难性或严重的后果。安全关键系统的提供商有义务对此类损失负责和进行赔偿，而测试工作通常是为了减少这方面的损失。测试活动提供证据——系统进行了充分的系统测试可以避免造成灾难性或严重的后果。

典型的安全关键系统的例子，包括飞机飞行控制系统、自动交易系统、核电站核心控制系统和医疗系统等。

在安全关键系统的部署中应该注意下列特性：

- 需求调整和适应方法的可追溯性
- 严格的开发和测试过程
- 安全分析
- 冗余结构及其验证
- 关注质量
- 高质量的文档（文档的深度和广度）
- 更严格的审计

3.11.3 节包含与安全关键系统的管理相关的事项信息。

1.3.2.1 遵循规章

安全关键系统要依照政府、国际以及部门的规章和标准（请参见 8.2 标准符合性）。这些都要影响开发流程、组织架构或正在开发的产品本身。

需要使用审计、组织图等手段来表明这些组织架构、开发流程对规章的依从性。

为了表明开发系统或产品为适应某具体规章而做出的变更，有必要显示说明规章中的每个需求都已经充分地覆盖。在这些情况下，为描述适应变更，需要具有从需求到实现结果的、完整的可追溯性。这涉及到开发过程中的管理、开发生命周期、测试活动、确认和认证（由权威的组织提供）。

1.3.2.2 安全关键系统和复杂性

许多复杂的系统和综合系统都有安全性组件。某些时候，这些安全性要素在当前系统级别（或其子系统的级别）并不明显，而在其高一级的系统级别才显现，或在复杂系统部署时显现（如飞机的飞行控制组件、空中交通管制系统）。

例子：一个路由器本身可能不是安全关键系统，但是，如果路由信息应用的环境有安全性要求时，它就是安全关键系统了（如远程医疗服务系统）。

风险管理可以降低风险的可能性或对风险产生后的影响，是安全关键系统的开发、测试环境的必要组成部分（请参见第 3 章）。此外，失效方式和影响分析（FMEA）（请参见 3.10 节）和软件通常的失效原因分析也会应用在这种环境中。

1.4 测量和度量

一系列的度量值（数字）和度量（趋势、图等）应用在软件开发的生命周期（如计划、覆盖、工作量）中。在每种情况下都应定义一个基线，并以这个基线为基础对流程进行追踪。

下列特性应包含在内：

1. 已计划的进度、测试覆盖度及其随时间的演变
2. 需求及其变更、以及对一系列进度、资源和任务等的影响
3. 工作量和资源使用情况及其随时间的变化
4. 里程碑和范围及其随时间的变化
5. 完成任务所需的实际成本和计划成本
6. 风险及减轻风险的措施，及其随时间的演变。
7. 已发现的缺陷、已修正的缺陷及修正缺陷所需的时间

使用度量，测试人员可以从管理的角度报告测试数据，并使过程跟踪在时间上保持前后一致。

需要考虑三方面的内容：

- 度量的定义：定义一组限定的有用度量。对这些定义的度量的解释必须得到所有相关者的认可，避免将来使用这些度量时产生争议。度量可以根据过程、任务的目标，面向系统组件、个人或团队进行定义。经常存在定义了过多的度量而忽略了那些最重要的度量的倾向。
- 度量的追踪：尽可能把度量报告和集成的工作自动化，以减少生成原始度量数据所用的时间。随时间的变化，某些特定的度量数据可能会带来其它信息，这些信息没有包含在度量定义阶段约定的解释中。
- 度量的报告：目的是从管理角度迅速理解所获得的信息。报告应提供一个对某段时间度量的“快照”或对度量随时间变化的评估，并能进行趋势分析。

1.5 道德规范

引入软件测试，个人可以获取保密的、授权的信息。为保证信息规范化使用，需要遵循必要的规范。ISTQB®借鉴、引用了 ACM 和 IEEE 对于工程师道德规范的如下表述：

公共 – 认证的测试人员的行为应与公共利益保持一致。
客户和雇主 – 认证的测试人员在保证公共利益的前提下，最大限度地保证客户和雇主的利益。
产品 – 认证测试工程师应保证他们发布的（在测产品和系统中的）版本最大程度地符合专业标准。
判断 – 认证测试工程师应在其提供的专业的判断中保持公正性和独立性。
管理 – 认证软件测试管理人员和测试领导者应同意提供合乎道德要求的测试管理。
专业 – 认证测试工程师应致力于提高职业的公正性和信誉并与公共利益保持一致。
同事 – 认证测试工程师应在工作中热切的支持他们的同事并促进与软件开发人员的合作。
自身 – 认证测试工程师应终生学习并不断促进职业实践的提升。

2. 测试过程

术语：

BS 7925/2，出口准则，IEEE 829，测试用例，测试结束，测试条件，测试控制，测试设计，测试执行，测试实现，测试计划，测试规程，测试脚本，测试总结报告，测试日志。

2.1 简介

在 ISTQB® 软件测试初级认证大纲中已描述了基本的测试过程，包括以下活动：

- 计划与控制
- 分析与设计
- 实现与执行
- 评估出口准则和报告
- 测试结束活动

这些活动可以顺序进行，某些活动可以并行。例如分析与设计可以与实现与执行并行开展，而其它的活动可以顺序进行。

由于测试管理在根本上与测试过程相关，测试经理必须具备运用本章所有内容进行具体项目管理的能力。在初级认证中，测试分析员和测试技术分析员所获得的测试过程知识已经基本足够（以上所列的测试开发任务除外）。这些任务所需要的知识在本节已经基本覆盖，并在第 4 章“测试技术”和第 5 章“软件特征测试”中详细应用。

2.2 测试过程模型

测试过程模型是一个近似和抽象的概念，它并不能完全反映现实世界中不同项目的复杂性、细节的差别和不同的测试活动。因此，并不能把模型看成是不变的真理，而应该将它作为帮助理解和组织项目的辅助手段。

本大纲使用 ISTQB®初级认证大纲（请参见上节）中描述的过程作为例子。除此以外还有其它重要的测试过程模型，下面列出了其中的三种。它们属于测试过程模型和测试过程改进模型（实用软件测试包括测试成熟度模型）并且根据所支持的成熟度进行定义。这三种测试过程模型包括 TPI®（Testing Process Improvement），将在 8.3 节测试改进过程深入讨论。

- 实用软件测试–测试成熟度模型[Burnstein03]
- 核心测试过程[Black03]
- 系统化测试和评估过程（STEP）

2.3 测试计划与控制

本节将着重讨论如何进行测试计划和测试控制的过程。

测试计划的大部分内容会在测试工作的初始阶段开展，包括测试活动和资源的识别和实现，用来满足测试策略中定义的任务和目标。

利用基于风险的测试（请参见第 3 章“测试管理”），可以在测试计划过程中采取减轻已知产品风险的措施。例如，如果知道严重缺陷通常会出现在设计规格说明中，就可在测试计划过程中规定：在编写代码之前，针对设计规格说明引入额外的静态测试（评审）。基于风险的测试还能在测试计划过程中，明确各测试活动相应的优先级。

测试依据、测试条件、测试用例和测试规程之间的复杂关系可能导致这些工作产品之间存在多对多的关联关系。只有充分理解这些关联关系，才能有效地进行测试的计划和控制。

测试控制是一个持续性行为，包括实际进度与测试计划之间的比较，报告测试的状态，以反映与计划相背离的状况。测试控制可以引导测试工作来满足任务、策略和目标等方面的要求，包括在需要的时候用以修正测试计划活动。

测试控制必须根据测试过程中产生的信息，结合项目或任务的变更条件做出回应。例如，如果动态测试在认为不会出现较多缺陷的地方发现了缺陷群，或者由于测试的开始时间被推迟因而要缩短测试执行的时间，那么必须对风险分析和测试计划进行相应的修改。这可能会要求重新设定测试的优先级，和重新安排剩下的测试执行工作。

测试计划文档的具体内容将在第 3 章“测试管理”中介绍。

测试计划和控制的度量可以包括：

- 风险与测试覆盖率
- 缺陷发现情况及其信息
- 测试件开发和测试用例执行的计划用时和实际用时之间的对比分析

2.4 测试分析与设计

测试计划阶段需要明确一系列的测试目标。而测试分析与设计的过程将使用这些测试目标达到以下目的：

- 识别测试条件
- 生成检验测试条件的测试用例

风险分析和测试计划过程中所确定的优先级准则，需要应用于从测试分析和设计到测试实现和执行的全过程。

2.4.1 识别测试条件

通过对测试依据和测试目标的分析，利用测试策略和/或测试计划中确定的测试技术，可以识别测试条件，即确定测试什么。

基于测试项的功能性和非功能性特性，测试条件的级别和结构可以通过以下几方面确定：

1. 测试依据的高数据粒度：即高层次的需求最初会生成高层次测试条件；例如验证屏幕 X 正常工作，然后可以从中衍生出一个低层次的测试条件，即验证屏幕 X 拒绝接受比正确长度少一位的帐号数字。

2. 所关注的产品风险：例如，针对高风险的功能特性，详细的低层次测试条件可能是需要定义的测试目标。
3. 管理报告和信息的可追溯性需求。
4. 决定是否仅使用测试条件，而不用开发测试用例，例如利用测试条件开展无脚本测试。

2.4.2 创建测试用例

利用测试计划或者测试策略中定义的测试技术和测试方法（请参见第 3 章），对前面识别的测试条件进行逐步的细化，来达到测试用例的设计。设计的测试用例应该是可以重用的、可以验证的以及可以追溯到需求的。

测试用例设计需要识别：

- 前置条件，例如，项目或局部测试环境的需求，及其交付计划
- 测试数据需求
- 预期结果和后置条件

定义测试的预期结果是一项特别挑战，即确定在测试中使用一个或多个测试准则。在确定预期结果时，测试人员不仅要关心屏幕上的输出，同时也要关注与数据和环境相关的后置条件。

如果清楚地定义了测试依据，理论上确定预期结果可能是简单的。但是，测试依据经常是含糊的、矛盾的、关键域缺少覆盖或者干脆没有测试依据。在这些情况下，测试人员必须掌握或者理解这些方面的专业技术。有时，甚至已经很好地描述了测试依据，但是在各种错综复杂的请求和响应之间的相互作用下，仍然很难定义预期结果，所以测试准则至关重要。测试执行中不知道如何确定测试结果的正确性，这样的执行没有多少意义，反而会在系统中生成虚假的问题报告，对系统产生错误的信心。

上述活动可以适用于所有级别的测试，尽管测试依据会有所不同。例如，用户验收测试可能主要是基于需求规格说明、用例（Use case）和定义的业务过程，而组件测试可能主要是基于底层设计规格说明。

在测试条件和测试用例的开发中，通常有不少文档输出作为测试的工作产品。在 IEEE 829 中可以找到相应的标准。这个标准讨论了适用于测试分析和设计阶段的文档类型，比如测试设计规格说明、测试用例规格说明，以及测试实现和执行过程中的主要文档类型。在实际运用中，测试工作产品的文档化的范围会有很大的差别。这些取决于如下因素：

- 项目风险（必须或没有必要记录在案）
- 文档给项目带来的附加收益
- 遵循的标准
- 使用的生命周期（例如，敏捷开发就设法通过团队内紧密和经常性的交流使文档最少化）
- 测试分析和设计过程中，对测试依据的可追溯性的要求

根据测试的范围，测试分析和设计可能需要阐述被测对象的质量属性。ISO9126 标准提供了有用的参考。具体到测试硬件/软件系统，可能会应用其有额外的属性。

结合评审和静态分析，可以改善测试分析和设计的过程。例如，基于需求规格说明实现测试分析和测试设计，是准备需求评审会议的很好的一种方法。类似地，测试工作产品，如测试用例、风险分析和测试计划也应该进行评审和静态分析。

在测试设计过程中，可能会要求定义详细的测试基础设施，尽管在实际中，这些基础设施可能需要在测试实现阶段才能最终确定。必须记住，测试基础设施不仅包括测试对象和测试件，还包括场地、设备、人员、软件、工具、外围设备、通信设备、用户权限和其它保证测试运行所需的事项。

监控测试分析和设计的度量可以包括如下各项：

- 测试条件对于需求的覆盖百分率

- 测试用例对于测试条件的覆盖百分率
- 在测试分析和设计中发现的缺陷数

2.5 测试实现与执行

2.5.1 测试实现

测试实现包括在测试规程（或测试脚本）中组织测试用例，最终确定测试数据和测试环境，编写测试执行进度表，在这些条件具备后，就可以开始测试执行。同时测试实现也包括检查测试级别的隐性和显性的入口准则。

应当设定测试规程的优先级以确保在测试策略中确定的目标能够以最有效的方法实现。例如优先运行最重要的测试规程就是一种有效的方法。

测试工作产品（测试用例和测试条件）的详细程度可能会影响测试实现中测试工作的细化程度和关联的复杂度。在一些情况下，需要应用校验标准并且测试应当提供遵循适用标准（例如，美国联邦航空管理局 DO-178B/ED 12B 标准）的证据。

如同上述 2.4 节所定义的，测试需要测试数据，并且某些情况下，这些数据量相当大。在测试实现过程中，测试人员创建输入数据和环境数据，并把这些数据存入数据库和其它类似的数据库中。同时测试人员也需要创建测试脚本和其它的数据生成器，从而在测试执行过程中可以不断地产生负载数据并发送给系统。

在测试实现过程中测试人员应当最终确定手动和自动测试的运行顺序。如果采用自动测试，测试实现包括创建测试用具和测试脚本。测试人员应当仔细检查对测试运行顺序的约束条件，识别并检查对测试环境或测试数据的依赖性。

测试实现也和测试环境有关系。在这个阶段中，测试环境必须在测试执行之前完全的建立完毕并得到验证。为实现这个目的，合适的测试环境是非常重要的。测试环境应当能够使缺陷在当前的测试条件下暴露出来。当没有失效发生时测试环境能运行正常，并且在需要的时候测试环境可以复制，例如，针对更高层次的测试所需的产品运行环境或终端用户环境。

在测试实现中，测试人员必须保证所负责测试环境的创建和维护工作是透明的和有效的，以及所有的测试件、测试支撑工具和相关的过程方法已经准备就绪，随时可用。包括配置管理、事件管理、测试日志和管理。另外，测试人员必须检验为评估出口准则和报告测试结果而收集测试数据的规程。

在测试实现中使用平衡兼顾的方法是明智的。例如，基于风险分析的测试策略常和动态测试策略混合使用。在这种情况下，一部分测试执行的工作量分配给那些没有遵循预定脚本的测试。

没有脚本的测试不是随机的或没有目标的，因为在有时间限制（请参见 SBTM）的条件下，测试结果并不是不可预知的。多年以来，测试人员开发了多种基于经验的技术，例如攻击技术（请参见 4.4 节和[Whittaker03]），错误推测技术[Myers79]和探索性测试技术。这些方法在测试分析、测试设计和测试实现中都有应用，但主要应用在测试执行中。当遵循这些动态测试策略，每个测试结果会影响后续测试的分析、设计和实现。这些轻量级的测试策略常常能有效地发现缺陷，它们需要由测试专家实现。这些动态测试策略过程是无法预测的，常常不能提供很好的测试覆盖率信息，并且在回归测试中没有特殊的工具支持很难重复这些测试。

2.5.2 测试执行

一旦提交了测试对象并且满足了测试执行的入口准则，就可以开始测试执行。测试执行的依据是测试规程，但可以给予测试人员一定的自由度，确保覆盖在测试中观察到的额外的、有趣的测试场景和系

统行为（对于在上述的额外情况中发现的任何失效，都必须描述与已经设计的测试规程之间的差异，使得失效能够被复现）。自动测试将遵循已经定义的指令而不会有偏差情况。

测试执行的核心活动是对比实际的测试结果和期望的测试结果。测试人员必须高度关注这些任务，否则，如果测试结果未能识别出测试对象中的缺陷（漏报）或者错误地将正确的行为归在错误的一类（假报），测试设计和实现的工作可能就会是浪费时间。如果期望的结果和实际的测试结果不匹配，将产生一个事件。必须仔细观察事件并确定产生的原因（产生事件的原因可能是测试对象中的一个缺陷，也可能不是），并且收集数据帮助解决事件。事件管理将在第 7 章详细讨论。

当识别一个缺陷后，应当仔细评估测试规格说明以确保其正确性。下列的诸多原因可以导致错误的测试规格说明，包括测试数据的问题、测试文档的缺陷或者是错误的执行方式。如果测试规格说明不正确，应当对它进行修改并重新运行。即使测试规格说明曾经多次成功运行，测试依据和测试对象的变动也会导致测试规格说明发生错误，所以测试人员应当对观察到的测试结果可能是由于错误的测试所导致的这种可能性保持清醒的认识。

在测试执行过程中，测试结果需要适当记录。那些执行了而未被记录测试结果的测试可能不得不重新运行以识别正确的测试结果，这将导致测试效率的低下和测试进度的延迟（说明：充分的日志能获得动态测试策略所涉及的覆盖率和重复性）。由于测试对象、测试件和测试环境可能会改进和更新，所以测试日志应当能识别明确的测试版本。

测试日志提供按顺序记录的测试执行的相关细节。

在测试过程发生的事件和单个测试都是日志的一部分，每个测试都应当唯一标识并且记录它的状态作为测试执行的成果。任何影响测试执行的事件都要记录。应当详细记录信息用于度量测试覆盖率并且文档化测试延迟、中断的原因。另外，需要记录信息以支持测试控制、测试过程报告、度量出口准则和测试过程改进。

日志的内容随测试级别和策略的不同而改变。例如，如果采用了自动化组件测试，自动化测试能自动收集大多数日志信息。如果采用了手工验收测试，测试经理可能手工编写和整理日志。在一些情况下，规范或审计需求会影响日志的内容，就像影响测试实现一样。

IEEE 829 标准包括了测试日志应当记录的信息描述。

- 测试日志标识
- 描述
- 活动和事件项

BS-7925-2 也包含了日志应当记录信息的描述。

在一些情况下，用户或客户可能参与测试执行。这可以作为增强用户或客户对系统的信心的一种手段，尽管这需要假设测试不会发现缺陷。这样的假设在早期的测试级别中常常是不适用的，但可能在验收测试时有效。

用于监测测试实现和执行的度量项可能包括：

- 测试环境配置的百分比
- 测试数据装载的百分比
- 测试条件和测试用例执行的百分比
- 测试用例自动化的百分比

2.6 评估出口准则与报告

3.6 节具体讨论了测试进度监控的文档和报告。从测试过程来看，测试进度监控需要确保收集合适的信息，以满足测试报告的要求。这包括针对最终完成目标的进度度量。

针对测试进度监控和完成情况的度量，包括与已经接受的出口准则的对应关系（测试计划时已确定）。测试进度度量可以包括以下一项或多项：

- 计划的测试条件、测试用例和测试规格说明的数目，以及按照测试是否通过分类的执行的测试数目
- 所有发现的缺陷数目，并且按严重程度和优先级对已经修复和待修复缺陷进行分类
- 提出的变更（变更需求）数目、接受（已实现）的数目和已经测试的数目
- 计划的成本与实际的成本的对比
- 计划花费与实际花费的对比
- 识别的风险，按照已经通过测试活动减轻的和待解决的风险进行分类
- 由于阻塞事件导致失去的测试时间百分比
- 重新测试项
- 计划的总共测试时间和有效的测试执行时间

针对测试报告，IEEE 829 制订了测试总结报告，具体包括下列各个部分：

- 测试总结报告标识
- 总述
- 版本变化
- 详细的评估
- 结果总结
- 评估
- 测试任务总结
- 审批

在每个测试级别结束或整个测试工作结束时，应该提交测试报告。

2.7 测试结束活动

当确定测试结束后，应当收集主要的输出成果并且交给相应的人员或归档。这些活动称为测试结束活动。测试结束活动包括 4 大类别：

1. 确保所有的测试工作已经结束。例如，所有计划的测试应该是已执行或在计划中忽略，所有已知的缺陷应该已经修复并测试通过、或推迟到下一个版本发布、或已作为系统的局限。
2. 针对不同的人员，交付相应的工作产品。例如，针对使用系统的人员或系统技术支持人员，通知已经推迟或接受的缺陷。例如，针对负责维护测试的人员，交付测试及测试环境。另外一种工作产品可以是（自动或手动的）回归测试用例套件。
3. 组织或参与（经验交流）回顾会议。从测试项目和软件开发生命周期中，总结并记录重要的经验。并且建立计划确保同样的错误将来不再重复或在项目计划中包括待解决的问题。例如：
   a. 由于较晚发现不曾预料的缺陷，团队可能意识到将来的项目中应该邀请更多的跨部门的用户代表参加质量风险分析会。
   b. 本次的估算严重有误，未来的估算任务需要考虑此问题及其原因。例如测试效率低或估算低于实际情况。
   c. 缺陷的趋势及因果分析。分析缺陷为什么发生和什么时候发生，并寻找可能的趋势，例如，后期的变更请求影响了分析和开发的质量。寻找不妥当的实践，例如，缺少了某一个测试级别，而这一测试级别又往往可以在早期发现缺陷并且更经济更节省时间。寻找由于新技术、人员变化和技术能力不足等方面导致的缺陷规律。
   d. 发现可能的过程改进机会

4. 在配置管理系统中归档所有的结果、记录、报表和其它文档及产物。例如，测试计划和项目计划应该保存到计划的归档中，并明确指明这些文档所用的系统和版本。

这些任务非常重要但经常被忽视。应该在测试计划中明确包括这些任务。

实际测试过程中通常会忽略上面说的一个或多个任务，主要因为提前解散了团队、或后续项目的资源或时间的压力，或团队异常疲劳。在合同限定的项目中，例如客户项目，合同中要明确这些需要的任务。

监控测试结束活动的度量可以包括如下各项：

- 测试用例执行的百分比
- 归入测试用例复用库的测试用例的百分比
- 已经自动化的测试用例和待自动化的测试用例的比例
- 确定为回归测试用例的百分比
- 待解决缺陷的百分比（例如，推迟解决的、没有进一步措施的、变更等）
- 识别和归档的工作产品百分比

3. 测试管理

术语：

失效模式和影响分析（FMEA），级别测试计划，主测试计划，产品风险，项目风险，基于风险的测试，风险分析，风险识别，风险级别，风险管理，风险减轻，风险类型，测试控制，基于会话的测试管理，测试估算，测试级别，测试管理，测试监控，测试计划，测试方针，测试点分析，测试进度，测试策略，宽带德尔菲法。

3.1 简介

本章涵盖了测试经理所需的相关知识点。

3.2 测试管理文档

文档编制通常是测试管理的一部分。测试管理文档的名称和范围在不同组织和项目中会有所不同，以下是在组织和项目中的通用测试管理文档的类型：

- 测试方针，描述了组织针对测试（或质量保证）的原则。
- 测试策略（测试指南），描述了组织的测试方法。包括产品和项目风险管理，将测试细分为步骤、级别或阶段，以及和测试相关的概要活动。
- 主测试计划（项目测试计划、测试方法），描述了将测试策略应用于一个特定项目，包括项目中需要开展的各种测试级别，以及这些测试级别之间的关系。
- 级别测试计划（或阶段测试计划），描述在每个测试级别中所要进行的特定活动，包括针对某个特定测试级别对主测试计划的拓展。

在一些组织和项目中，这些不同类型的文档可能会合并成一个文档，这些文档的内容可能出现在其它类型的文档中。文档中的某些内容也可能是凭直觉而非书面的形式表现出来，或者以传统的测试方法表现出来。较大、较正式的组织和项目倾向于撰写上述所有类型的文档，并将其以书面的工作产品保存，而较小、不太规范的组织和项目通常仅撰写其中的一部分。本大纲将分别描述每一个文档类型，组织和项目将根据实际要求确定具体采用哪种文档类型。

3.2.1 测试方针

测试方针描述了组织针对测试（或质量保证）的原则。它以文档形式或作为管理理念而存在，规划了组织期望达到的总体测试目标。测试方针可以由信息技术部门、研发部门或产品开发部门进行制定，不管是由什么部分制订，它都应该反映组织在测试方面的价值和目标。

测试方针有时作为更广泛的质量方针的一部分或补充。质量方针描述了管理在质量方面的整体价值和目标。

测试方针可能是一份简短的、概要的文档：

- 提供测试的定义，例如建立系统按照预期运行的信心和检测缺陷。
- 定义基本的测试过程，例如，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、以及测试结束活动。
- 描述如何评估测试的有效性和效率，例如，缺陷发现百分比（DDP）和对比测试阶段发现缺陷和发布后发现缺陷的相关成本。
- 定义期望的质量目标，例如可靠性（例如，用失效率来度量）或易用性。

- 规定测试过程改进的活动，例如，应用测试成熟度模型 TMM 或测试过程改进模型 TPI、或根据过去的项目经验改进测试过程。

测试方针既可以针对新开发项目，也可以针对维护性项目规定测试活动。测试方针也可以参考整个组织使用的测试术语标准。

3.2.2 测试策略

测试策略描述了组织采用的测试方法，包括产品和项目风险管理以及将测试细分为级别、阶段和测试的高级活动。测试策略中描述的测试过程和测试活动应与测试方针的一致。它应该包括针对组织或针对一个或多个项目所需的通用测试要求。

如初级大纲中所述，测试策略（也称测试方法）可以基于什么时候开始测试设计来进行如下分类：

- 预防型策略，尽早设计测试以预防缺陷。
- 应对型策略，在软件或系统实现后开始测试设计。

典型的测试策略（或测试方法）包括：

- 分析式的测试策略，例如基于风险的测试。
- 基于模型的测试策略，例如运行概况测试。
- 系统的方法，例如基于质量特性的测试。
- 基于与过程或标准一致的方法，例如基于 IEEE 829 文档的测试。
- 动态和启发式的方法，例如基于已知缺陷的攻击测试。
- 咨询式的方法，例如用户导向的测试。
- 回归测试的策略，例如采用广泛的自动化进行测试。

不同的测试策略可以相互组合。组织选定的测试策略应符合其需要和方法，并可根据特定的业务或项目进行裁剪。

在许多实例中，测试策略描述了项目风险和产品风险以及测试过程如何管理这些风险。在这些实例中，应清晰的对风险和测试之间的关系进行解释，并说明减轻风险和管理风险的可选方案。

测试策略可描述要执行的测试级别。在这种情况下，应给出制定每个测试级别的入口准则和出口准则的概要指导以及测试级别之间的关系（例如，不同的测试级别对应不同的测试覆盖目标）。

测试策略也可包括如下方面的内容：

- 集成规程
- 测试规格说明技术
- 测试独立性（可根据不同的测试级别而有所变化）
- 应遵守的必需的和可选的标准
- 测试环境
- 测试自动化
- 软件工作产品和测试工作产品的可重用性
- 再测试和回归测试。
- 测试控制和测试报告
- 测试度量和度量标准
- 缺陷管理
- 测试件的配置管理方法

不论是短期的还是长期的，都应该定义相应的测试策略，并将其写入一份或多份文档中。不同的测试策略适合不同的组织和项目。例如，对于安全关键的应用系统，应使用比其它应用系统更强的测试策略。

3.2.3 主测试计划

主测试计划描述如何在特定的项目中应用测试策略，包括执行哪几个测试级别以及这些级别之间的关系。主测试计划应与测试方针和测试策略相一致，若有不一致，则应解释其中的偏差和例外。主测试计划应该作为项目计划的补充，或者作为大型项目或业务的操作指南，其中描述了测试工作量。

主测试计划的内容和结构根据组织、组织的文档标准和项目形式的不同而变化，主测试计划内容通常包括以下部分：

- 测试项和不测试项
- 测试的质量属性和不测试的质量属性
- 测试时间进度表和预算（应与项目预算或组织运营预算保持一致）
- 测试执行周期及其与软件发布计划的关系
- 测试的商业价值和理由
- 测试团队与其它人或部门之间的关系和交付内容
- 为每个测试级别定义覆盖的测试项和未覆盖的测试项
- 为每个测试级别定义明确的入口准则、挂起/恢复（暂停/继续）准则和出口准则，以及测试级别之间的关系
- 测试项目的风险

对于较小的项目或业务，若只采用一个正式的测试级别，那么可将主测试计划和采用的级别测试计划合并成一个文档。例如，如果系统测试作为唯一的正式测试级别，非正式的组件测试和集成测试由开发人员执行，非正式的验收测试将作为 Beta 测试的一部分由客户执行。在这种情况下，该系统测试计划可包含本节的内容。

另外，测试通常依赖于项目中的其它活动。假如这些活动未充分文档化，尤其是它们与测试有关系并对测试产生影响，那么在主测试计划（或相应的级别测试计划）中应包含这些内容。例如，若没有定义配置管理过程，则在测试计划中应说明如何将测试对象交付给测试团队。

3.2.4 级别测试计划

级别测试计划描述在某个测试级别中所要进行的特定活动，包括级别测试计划对主测试计划的拓展。级别测试计划包含了进度、任务和里程碑等在主测试计划中不一定包含的细节。另外，级别测试计划还应包括该级别的测试规格说明应遵守的不同标准和使用的模板。

对于非正式的项目或业务，通常只撰写一份级别测试计划作为测试管理文档。在这种情况下，该级别测试计划应该包含在 3.2.1、3.2.2 和 3.2.3 节中所提到的部分内容。

3.3 测试计划文档

正如在 3.2 节所述，主测试计划的内容和结构因组织、文档标准、项目形式的不同而异。许多组织为了确保在不同项目、或不同业务之间的文档通用性和可读性，开发或改编了一些模板，包括测试计划文档的模板。

IEEE 829 “软件测试文档标准”文档包含了测试文档模板以及使用指南，包括如何准备测试计划等。该文档也描述了测试项传递的相关内容（例如，将测试项发布测试）。

3.4 测试估算

作为管理活动之一，估算是为了获得特定业务或项目中各种活动的近似成本预算和完成日期。良好的估算应该具有如下特征：

- 代表有经验的同行的集体智慧，并能得到项目干系人的支持
- 提供明确而详细的资金、资源、任务和项目干系人列表
- 针对每个估算的活动，提供最可能的成本、工作量和持续时间

尽管为估算建立了良好的项目管理实践，但软件工程和系统工程的估算依旧充满困难，包括技术和政治方面。测试估算是与项目或业务有关的测试活动中的最佳实践应用。

测试估算应包含测试过程中的所有活动，例如，测试计划和控制、测试分析和设计、测试实现和执行、测试评估和报告、以及测试结束活动。由于测试执行通常在项目的关键路径上，估算的成本、工作量，特别是测试执行持续时间是管理人员特别关心的。然而，当软件的整体质量很低或未知的时候，对测试执行的估算将比较困难，并且估算得到的结果也不可靠。通常的做法是估算需要执行的测试用例数目。在估算过程中所作的假设应该作为估算的一部分而文档化。

测试估算应考虑影响测试活动的成本、工作量和持续时间等所有因素。这些因素包括（但不仅限于此）：

- 系统质量所要求的级别
- 被测试系统的规模
- 以前测试项目的历史数据（也可包括基准数据）
- 过程因素，包括测试策略的开发或维护生命周期和过程成熟度，以及项目估算的准确度
- 物质因素，包括测试自动化和工具、测试环境、测试数据、开发环境、项目文档（例如，需求文档、设计文档等）以及可重用的测试工作产品
- 人员因素，包括经理和技术负责人、高级执行管理人员的承诺和期望、项目团队的技能、经验和态度、项目团队的稳定性、项目团队的关系、测试和调试环境的支持、是否有合格的承包商和顾问以及相关领域的知识

其它因素也会影响测试估算，包括过程的复杂度、技术、组织、测试干系人的人数、许多测试子团队（特别是当子团队处于不同的地理位置）；项目启动、培训、定向需求；新工具的熟悉和开发、技术、定制的硬件、测试件的数目；高质量的详细测试规格说明的要求（特别是使用一个不熟悉的文档标准时）；难以确定的组件交付时间（特别是对于集成测试和测试开发而言）；以及敏感的测试数据（例如，对时间敏感的的数据）。

估算可以自底向上进行，也可以自顶向下进行。在测试估算中，通常可以使用下列技术：

- 直觉和猜测
- 过去的经验
- 工作分解结构（WBS）
- 团队评估会议（例如，宽带德尔菲法）
- 三点估算方法
- 测试点分析（TPA）[Pol02]
- 公司标准和规范
- 测试占整个项目工作量的百分比或人员编制比例（例如，测试人员与开发人员的比例）
- 组织的历史和度量标准，包括基于度量标准的模型，该模型用于估算缺陷数、测试周期数、测试用例数、每个测试的平均工作量以及回归周期的数目
- 行业平均值和预估模型，例如测试点、功能点、代码行数、估计的开发工作量或其它项目参数

在多数情况下，得出估算结果后，需要提交给管理人员，并说明测试的价值（请参见 3.7 节）。通常经过协商，需要重新对结果进行估算。理想情况下，最终的测试估算平衡了组织和项目目标在质量、时间表、预算以及特性方面的要求。

3.5 测试计划时间表的确定

通常，尽早进行测试活动的计划，可以发现和管理测试活动相关的风险，可以和其他相关的活动进行仔细和及时的协调，制订高效的计划。对于测试计划来说也是如此。除此之外，对于测试计划，根据测试估算的结果尽早进行计划还有其他的优点，包括：

- 发现和管理测试之外的项目风险和问题
- 在测试执行之前，发现和管理产品（质量）风险和问题
- 识别项目计划或其它项目工作产品中的问题
- 有机会增加分配的测试人员数目、预算、工作量和/或所需时间，以获得更高的质量
- 确认关键组件（以便能尽早发布这些组件）

由于测试很大程度上依赖于开发（发布）的进度，因此测试计划时间表应与开发紧密协调。

由于在获得完成测试计划所需全部信息时，可能已经失去了获得上述益处的机会。因此测试计划应尽早制定并尽早以草稿形式发布。随着对更多的信息的不断了解，测试计划的作者（通常是测试经理）可根据这些新得到的信息完善测试计划。使用这种迭代的方法进行测试计划的制定、发布和评审，也可以使得测试计划成为促进干系人员对测试达成共识、进行交流和讨论的载体之一。

3.6 测试进度监控

测试进度监控可以从以下 5 个主要方面进行：

- 产品风险
- 缺陷
- 测试
- 覆盖率
- 信心

在项目或业务的进行过程中，通常会以特定的方式度量和报告产品风险、事件、测试和覆盖率。这些度量值最好与测试计划中定义的出口准则进行关联。对被测产品的信心，虽然可通过调查的方式度量，但是，通常还是以主观的方式报告。

与产品风险相关的度量标准包括：

- 剩余风险的数目（包括风险的类型和级别）
- 已减轻的风险数目（包括风险的类型和级别）

与缺陷相关的度量标准包括：

- 已解决（已消除）的缺陷总数/已报告（已识别）的缺陷总数
- 失效的平均时间间隔或失效发生率
- 按不同的分类统计缺陷数：特定的测试项或组件；根本原因、缺陷来源；测试发布；引入阶段、发现或移除的阶段；在某些情况下，统计各个缺陷所有者的缺陷数
- 从报告缺陷到修复缺陷所需的时间曲线

与测试相关的度量标准包括：

- 已计划的、已设计的（开发的）、已执行的、通过的、失败的、阻塞的和跳过不执行的测试用例的总数目

- 回归测试和确认测试的状态
- 每天实际的测试小时数/每天计划的测试小时数

与测试覆盖率相关的度量标准包括：

- 需求和设计元素的覆盖率
- 风险覆盖率
- 环境与配置的覆盖率

可采用口头叙述、数字表格或绘图的方式报告度量结果，这些度量结果可用于以下方面：

- 通过测试结果的分析，揭示产品、项目或过程中发生了什么
- 提供报告，与测试项目参与者和利益干系人交流测试的发现
- 提供控制，从总体上改进测试或项目的过程并监控过程改进的结果

如何收集、分析和报告这些测试度量结果取决于使用这些结果的人，包括他们的特定需求、目的以及能力。

在根据测试结果调整或计量项目的控制工作量时，可以考虑下列方案：

- 修定质量风险分析、测试优先级和/或测试计划
- 增加资源或增加测试的工作量
- 推迟发布时间
- 弱化或加强测试出口准则

采用以上的方案，通常要求项目或业务的项目干系人达成一致，并且得到项目或业务经理的许可。

测试报告的编写方式很大程度上取决于其目标读者，例如，项目管理人员或业务管理人员。对于项目经理，可能对缺陷的详细信息感兴趣；而对于业务经理，产品风险的状态将是报告的关键。

IEEE 829“软件测试文档标准”提供了测试总结报告的一个模板。

测试控制应该根据测试计划与测试进度报告中实际状态的偏差进行。测试控制的目标是尽可能地减少实际情况与测试计划的偏差。可用的测试控制方法包括：

- 修订测试用例的优先级
- 获取额外的资源
- 推迟发布日期
- 改变项目的范围（功能性的范围）
- 修订测试完成准则（只在得到利益干系人授权的情况下进行）

3.7 测试的商业价值

虽然大多数组织认为测试在某种意义上是有价值的，但很少有管理人员（包括测试经理）可以量化、描述或明确界定测试的价值。另外，许多测试经理、测试组长和测试人员关注测试技术的细节（测试任务或测试级别的具体方面）却往往忽略了与测试相关的更重要的策略（较高级别）问题，而这些是其他项目参与者，特别是经理们所关心的。

测试从定量和定性两个方面为组织、项目和/或业务提供价值：

- 定量的价值，包括发现缺陷并在产品发布前预防或修改这些缺陷、发现缺陷并了解在产品发布前依旧存在的缺陷、通过测试减少风险并发布有关项目、过程和产品状态的信息。
- 定性的价值，包括提高产品质量的声誉、使发布更顺利和更可预测、增强和建立对产品的信心、降低产品功能失效甚至造成人员伤亡的可能性，避免承担法律责任。

测试经理和测试组长应理解哪些价值对他们的组织、项目和/或业务有意义，并能利用这些商业价值进行测试方面的沟通。

用于度量测试的定量价值和效率的成熟方法称为质量成本（或有时候称为不良质量成本）。质量成本方法将项目或业务的成本分成四个类别：

- 预防成本
- 检测成本
- 内部失效成本
- 外部失效成本

部分测试预算是用于支付缺陷检测成本，而剩余的是用于支付内部失效成本。缺陷检测成本和内部失效成本的总和通常会比外部失效成本低很多，而这正是测试的价值所在。测试经理和测试组长可通过列出这四类成本作一个有说服力的商业用例，以说明测试的价值。

3.8 分布式测试、外包测试与内包测试

在很多时候，并不是所有的测试工作都是由一个测试团队单独完成的，他们是项目团队成员的组成部分，并且和项目团队的其他成员处在同一个工作场所。如果测试工作分散在不同地点，那么可以称之为分布式测试。若测试工作由其它（有别于项目团队所在地）一个（或多个）地点的非项目成员承担，那么可以称之为外包测试。若测试工作由其他与项目团队在同一地点的非项目成员承担，那么这称为内包测试。

上述测试工作的共同点是需要明确团队之间的交流方式，并明确定义对使命、任务和交付内容的期望。项目团队必须减少对非正式交流渠道的依赖，如在走廊上的对话或同事之间的私人交流。地点、时区、文化和语言的差异使得这些工作变得更加重要。

这些测试工作的另外的一些共同点是是各个团队要使用一致的方法。如果测试小组之间或测试小组、开发小组、管理人员之间使用各自不同的方法，那么这将造成显而易见的问题，尤其是在测试执行期间。

对于分布式测试，不同工作地点之间的测试工作的划分必须要明确、合理。如果不按这个原则分配，可能导致各个测试小组分配不到其最适合的测试工作。进而致使整个测试工作受到隔阂（增加产品发布的质量风险）和重叠（降低效率）的困扰。

最后，对于各种类型的测试，整个项目团队对测试团队的信任非常重要。要相信各个测试团队虽然在组织、文化、语言和地理位置上有所不同，但他们依旧能很好地履行他们的职责。缺少信任会导致低效和延迟。缺少信任的行动包括验证彼此的活动、相互追究问题的责任以及组织内的小团体。

3.9 基于风险的测试

3.9.1 基于风险的测试简介

风险是指预料之外结果发生的可能性。只要有可能发生某些问题，使客户、用户、参与者或项目干系人减少对产品质量或项目成功的信心，即可认为存在风险。

对产品质量产生重要影响的潜在问题称为产品风险（或质量风险），例如潜在的可靠性方面的缺陷（错误）能导致在常规操作时系统崩溃。当潜在问题影响项目成功时，那么这个潜在问题就被称为项目风险（或计划风险），例如可能由于人员缺乏从而导致项目延期。

并不是所有的风险都需要同等关注。风险的级别受不同因素的影响：

- 问题发生的可能性
- 问题发生后的影响

基于风险的测试中，针对风险可以有三种不同的方法来开展测试活动：

- 必须根据已知的重要产品（质量）风险的级别分配测试工作量、选择测试技术、安排测试活动的顺序以及修改缺陷。
- 必需根据已知的重要项目（计划）风险来计划和管理测试工作，从而实现风险的减轻和缓解。
- 根据剩余的风险来报告测试结果和项目状态；例如，基于没有执行的测试、省略的测试、没有修改的缺陷或者没有进行再测试的缺陷。

针对风险的这三种应对方法应该贯穿于项目的整个生命周期中，而不仅仅是在项目初期和项目结尾。在项目进行过程中，测试人员应特别关注：

- 找出最重要的缺陷（针对产品风险），通过应用测试策略和测试计划中描述的减轻和缓解风险的方法和措施来减少风险。
- 进行风险评估。随着项目的进行，根据收集到的额外信息对风险进行识别和分析，增加或减低先前识别出的风险的可能性或影响，设定新的风险级别。

在上述两种情况中，所做的工作会影响后续的基于风险的测试。

基于风险的测试在很多方面类似于保险。一个人买保险是因为担心某些潜在风险，而忽视某些风险则是由于不担心这些风险。定量分析类似风险评估，定量分析是由保险精算师和其他保险专家进行，适用于保险业。但基于风险的测试通常依赖于定性分析。

为了正确地进行基于风险的测试，测试人员应能识别、分析并减轻一些典型的产品风险和项目风险，如有关安全、商业和经济、系统和数据安全、技术和政治因素的风险。

3.9.2 *风险管理*

风险管理主要有三个主要的活动组成：

1. 风险识别
2. 风险分析
3. 风险应对（也被称为风险控制）

这些活动在某种程度上是连续的，但在上述各节以及接下来的各节中提到的持续的风险管理，在大多数项目中是指迭代地进行这三种风险管理活动。

为了达到最高的有效性，项目的所有项目干系人都需要参与风险管理，虽然有时在实际的项目中，某些项目干系人扮演了其他项目干系人的代理人角色。例如，在针对大众市场的软件开发过程中，一部分潜在的客户可能作为大众市场的客户样本，帮助识别会严重影响他们使用软件的潜在缺陷；在这种情形中，这些被邀请的潜在客户样本就作为所有最终客户的代表。

由于测试分析员拥有特定的专业技术，他们应积极地参与到风险的识别和分析过程中。

3.9.2.1 风险识别

不论是产品风险还是项目风险，测试人员都可以通过下列的一项或多项技术来识别风险：

- 专家咨询
- 独立评估
- 使用风险模板
- 经验教训（例如，项目评估会议）
- 风险研讨会（例如，失效模式和影响分析）
- 头脑风暴
- 检查表
- 过去的经验

通过广泛邀请可能的项目干系人代表，在风险识别过程会发现尽可能多的重要风险。

使用某些方法进行风险识别时，识别过程会因为风险识别本身的风险而停止。

某些特定的技术，例如失效模式和影响分析（FMEA），要求识别每一个潜在失效模式用以分析失效模式对系统其它部分的影响（包括综合系统的上层系统））和对系统的潜在用户的影响。

其它技术，例如危害分析，要求预测风险的来源。

关于如何使用失效模式影响、失效模式和影响分析以及危急程度分析，请参见 3.10 节和[Stamatis95]、[Black02]、[Craig02]、[Gerrard02]。

3.9.2.2 风险分析

风险识别是尽量识别尽可能多的相关风险；而风险分析是研究这些被识别出的风险，特别是将这些风险明确分类并确定每个风险的可能性及影响。

风险分类是指为风险确定适当的类型。ISO 9126 标准中列出了典型的风险类型，该标准根据质量属性为风险分类。一些组织有自己的质量属性集。需要注意的是：当使用检查表作为风险识别的基础时，风险类型的分类将在风险识别过程中进行。

确定风险级别通常涉及风险的评估，即针对每个风险条目，确定其可能性和影响。风险发生的可能性一般指被测系统中存在潜在问题的可能性。换句话说，风险发生的可能性起源于技术风险。

影响技术风险的因素包括：

- 技术和团队的复杂性
- 业务分析师、设计师和程序员之间的人员和培训问题
- 团队内部冲突
- 与供应商的合同问题
- 开发组织的地理分布
- 老方法和新方法对立
- 工具和技术
- 不良的管理领导和技术领导
- 时间、资源和管理压力
- 缺乏初期的质量保证
- 变更率高
- 初期的高缺陷率
- 接口和集成问题

风险发生的影响是指风险的发生对用户、客户或其它项目干系人影响的严重性。换句话说，它起因于商业风险。影响商业风险的因素包括：

- 受影响的特性的使用频率
- 对组织形象的损害
- 商业损失
- 潜在的金融、生态或社会方面的损失或法律责任
- 民事或刑事法律制裁
- 失去经营执照
- 缺少合理的变通
- 明显的失效导致负面宣传

测试人员可以通过定量的或定性的方法设定风险的级别。如果风险的可能性和影响可定量确定，则可将两个值相乘，计算出风险发生的成本。这可以看做是一个特定风险的预期损失。

然而，通常风险的级别只能定性地确定。也就是说，可用“非常高、高、中、低或非常低”表示可能性，不能确定地说可能性是 90%、75%、50%、25%或 10%。但是不应该认为定性的方法比定量的

方法差；实际上，如果错误地使用定量方法，会误导项目干系人对于风险程度的理解和管理。非正式方法通常都是定性的和不严格的，例如在[vanVeenendaal02]、[Craig02]和[Black07b]中描述的方法。

除非风险分析是基于广泛的、有效的风险数据系统（如保险业），否则无论是定性的或定量的风险分析，它都是基于对可能性和影响的直觉判断。也就是说，个人对于这些因素的理解和看法会影响风险级别的确定。项目经理、程序员、用户、商业分析师、架构师和测试员通常对于各个风险条目的风险级别有不同的理解和看法。风险分析过程应包括一些方法，可以使得风险级别在不同干系人之间达成一致；即使在最坏的情况下，也要使用强制方法确定大家都认可的风险级别。否则，风险级别将不能用于指导风险应对活动。

3.9.2.3 风险应对

识别并分析风险之后，可以采用下面四种主要方法来处理风险：

1. 通过预防手段减轻风险的可能性和/或影响。
2. 制定应急计划，降低风险发生后的影响。
3. 将风险转交给第三方处理。
4. 忽视并接受风险。

选择哪个方法取决于该方法带来的收益和机会，以及相关的成本和潜在的额外风险。

应对策略

在大多数基于风险的测试策略中，风险识别、风险分析和制定风险应对活动是制定主测试计划和其它测试计划的基础。每个风险条目的风险级别决定了用于处理该风险的测试工作量（例如，减轻风险）。一些安全相关的标准（例如，FAA DO-178B/ED 12B、IEC 61508），规定了基于风险级别的测试技术和覆盖率。

项目风险应对

若已识别项目风险，则应将这些风险告知项目经理并制定对策。测试组织并不总是有能力降低这些风险。然而，有些项目风险能够并且也应由测试经理成功进行应对，例如下列风险：

- 测试环境和工具是否准备就绪
- 是否有足够的符合要求的测试人员
- 测试输入资源是否质量低下
- 测试输入资源是否总是在变更
- 测试工作是否缺少标准、规则和技术

风险应对的方法包括尽早准备测试件、测试设备的预测试、产品早期版本的预测试、更严格的测试入口准则、增加需求的可测性、参与早期项目产出的评审、参与对问题和变更的管理以及监控项目的进度和质量。

产品风险应对

谈到产品风险（质量风险），那么测试是减轻这种风险的形式之一。若通过测试找到了缺陷，那么就有可能在发布前解决，从而减轻风险。若经过测试未找到缺陷，那么可以确保在某些条件下（例如，已测的条件）系统能正常工作，这样也可以减轻风险。

产品（质量）风险可通过非测试活动减轻。例如，若认识到需求写得不好，有效的解决方法是通过彻底的评审来减轻风险，而不能等这些低质量的需求设计出来并转化成代码之后再去测试和决定测试的优先级。

风险的级别也用于决定测试的优先级。在某些情况下，所有高风险的测试都应该首先执行，当所有高风险的测试执行后，低风险的测试才会执行。测试执行顺序应严格地根据风险级别来制定并执行（也称作“深度优先”）；在其它情况下，测试优先级根据抽样方法来制定。抽样方法就是选择一个根据不

同风险权重的测试样本，使得该测试样本可以覆盖到所有已识别的风险。用风险覆盖率评估选出的测试样本，保证每个风险至少能被覆盖到一次（通常称作“广度优先”）。

测试人员可使用的其它风险应对方法包括：

- 选择合适的测试设计技术
- 评审和检查
- 评审测试设计
- 独立的级别
- 最有经验的人员
- 进行再测试
- 回归测试

在[Gerrard02]中引入了测试有效性的概念，测试有效性（以百分比的方式）说明了有效的测试怎样减低风险。若测试有效性低，则干系人就不会期望通过测试来减低风险。

无论基于风险的测试是深度优先还是广度优先，分配给测试的时间都可能不足。基于风险的测试使得测试人员能以剩余风险级别的方式向管理人员报告当前测试状态，让管理人员决定是否要增加测试时间或将剩余的风险转移给用户/客户、服务/技术支持人员和/或运营人员。

为进一步的测试周期调整测试

若有时间进行进一步测试，则任何新的测试周期应根据新的风险分析作调整。其主要关注点是存在全新的或有很大变化的产品风险；在测试中发现的不稳定的或易产生缺陷的区域；修复缺陷带来的风险；试图将测试专注于在测试中发现的典型错误；潜在的未测试的区域（低测试覆盖）。任何新的或额外的测试周期应根据这些风险的分析制定计划。强烈建议在每个项目里程碑时更新风险检查表。

3.9.3 软件生命周期中的风险管理

在理想情况下，风险管理应贯穿于整个软件生命周期。若组织有测试方针文档和/或测试策略，那么应在这些文档中描述在测试中管理产品风险和项目风险的基本过程。同时，如何将风险管理集成到测试的各个阶段并使其发挥作用也应在文档中列出。

无论采用哪种软件开发生命周期模型，风险识别和风险分析活动都可在项目初始阶段开始。计划的风险应对活动作为整个测试计划过程的一部分。主测试计划和/或级别测试计划中都应当描述项目风险和产品风险。风险的类型和级别会影响其所处的测试的测试级别、用于减轻风险的工作量大小、用于减低风险的测试和其它技术、判断风险减轻是否足够的标准。

在测试计划完成以后，就应开始在项目中进行风险管理（包括识别、分析和减轻）。包括识别新的风险、重新评估现有风险的级别和评估风险应对活动的有效性。例如，若在需求阶段根据需求规格说明进行风险识别和分析，那么设计规格说明完成后，应对这些风险进行再评估。另外，当正在测试的组件的缺陷数比预期多很多时，则可得出该区域存在缺陷的可能性大于预期的结论，因此应该调整风险的可能性和风险的整体级别。根据其调整的风险结果，增加该组件测试的工作量。

完成了初始的风险识别和分析活动，并开展了风险减轻活动，就可以对风险已减低到何种程度进行度量。可通过跟踪与风险相关的测试用例和追溯与已发现缺陷相关的风险进行度量。随着测试的运行和缺陷的发现，测试人员可检测风险的剩余级别，并将检测结果用于决定发布的准确时间。基于风险覆盖的测试报告的例子，请参见[Black03]。

测试报告应列出已解决和未解决的风险，以及已取得和未取得的收益。

3.10 失效模式与影响分析

失效模式和影响分析（FMEA）及其变种包括失效模式和危急程度分析（FMECA）是迭代的活动，用于分析系统范围内失效模式的影响和危急程度。这些分析应用在软件的时候也称为软件失效模式和影响分析（SFMEA）和软件失效模式和危急程度分析（SFMECA），此处的“S”指软件。在下述各节中，只会提到 FMEA，但相应信息对其它三个分析方法也适用。

测试人员必须能参与到 FMEA 文档的创建过程中去。包括理解文档的作用和如何使用文档以及运用他们的知识确定风险因子。

3.10.1 应用领域

FMEA 可以应用于：

- 分析软件或系统的危险临界值，以降低失效的风险（例如，飞机飞行控制系统等安全关键系统）
- 适合于那些具有强制性或规定（法定）的需求（请参见 1.3.2 节“安全关键系统”）
- 在项目早期消除缺陷
- 为安全关键系统定义特殊的测试因素、操作约束和设计决策

3.10.2 实现步骤

当得到概要的初步信息后，就应安排 FMEA 计划，而后随着不断得到的详细信息，逐渐将 FMEA 延伸到更详细的程度。根据可用的信息和程序的需求，可在系统或软件组件的各个层次上应用 FMEA。

对于每个关键的功能、模块或组件，迭代如下内容：

- 选择一个功能并确定其可能的失效模式，例如，这个功能会以什么样的方式失效
- 定义可能引起失效的原因、失效的影响和用于减轻或降低失效的设计原理

3.10.3 收益与成本

FMEA 有如下优势：

- 可以找出由软件失效或错误操作造成的预期系统失效
- 系统地使用 FMEA 有益于整个系统级别的分析
- 其结果可用于设计决策和/或证明设计的合理性
- 其结果可用于将测试关注在软件的特殊（关键）区域

在应用 FMEA 的时候必须考虑下列因素：

- 容易忽视失效的顺序
- 建立 FMEA 表格费时较多
- 可能难以定义独立的功能
- 可能难以识别错误传播

3.11 测试管理工作

3.11.1 探索性测试的测试管理问题

基于会话的测试管理（SBTM Session-based test management）是一种用于管理探索性测试的概念。会话是测试工作的基本单位，在会话过程中测试不会被打断，并且测试根据特定的目标关注在某个

特定的测试对象（测试章程）。在每个会话结束的时候，根据已进行的活动创建一个报告，该报告通常称为会话表。SBTM 根据已定义的过程结构开展，并生成用于辅助文档验证的记录。

一次测试会话可被分成三个阶段：

- 建立会话：建立测试环境和增进对产品的理解
- 测试设计和执行：细查测试对象，寻找问题
- 缺陷调查和报告：开始于测试人员发现某些疑似失效的时候

SBTM 会话表包含以下各项：

- 会话章程
- 测试人员姓名
- 起始的日期和时间
- 任务分解（以会话的形式）
- 数据文件
- 测试备注
- 问题
- 缺陷

在每次会话结束的时候，测试经理会在测试团队内召开一个任务报告会议。在任务报告会议中，测试经理审核会话表中的活动、改善章程、从测试人员处取得反馈，估算和计划下一步的会话活动。

任务报告的议程是简要地检验以下各项，简称之为 PROOF：

- 经过：在会话期间发生了什么？
- 结果：会话取得了什么成果？
- 展望：还有什么有待解决？
- 障碍：要进行良好的测试有些什么障碍？
- 感受：测试人员对上述项有些什么感觉？

3.11.2 *综合系统的测试管理问题*

综合系统测试管理相关的问题包括：

- 对构成综合系统的各个（子）系统执行测试的时候，测试管理会更加复杂，因为这种测试可能在不同的地点、由不同的组织和使用不同的生命周期模型进行。因此，综合系统的主测试计划通常使用一种正式的生命周期模型，并将重点放在管理工作上，例如里程碑和质量标准。通常有一个正式定义的质量保证过程，该过程可能是由一个单独的质量计划定义的。
- 辅助过程，例如配置管理、变更管理和发布管理，必须正式定义并取得测试管理人员的同意。这些过程必不可少，可用于确保软件发布受控、变更的引入受到管理以及被测软件基线已定义。
- 建立和管理典型的测试环境，包括测试数据，可能是一项主要的技术和组织挑战。
- 集成测试策略可能要求构建模拟器。对于综合系统，在早期的测试级别中进行集成测试模拟器的构建可能相对简单和成本相对较低，而在综合系统中为高级别的集成测试构建整个系统的模拟器可能较复杂并且成本昂贵。模拟器的计划、估算和开发经常作为一个单独的项目进行管理。
- 当测试综合系统时，不同部分间的相关性将对系统测试和验收测试造成约束。需要特别关注系统集成测试和附属的测试依据文档。例如，接口规格说明。

3.11.3 *安全关键系统的测试管理问题*

安全关键系统测试管理相关问题包括：

- 通常应用特定行业（领域）的标准（例如，运输行业、医药行业、军队）。这些标准可应用于开发过程和组织的结构或正在开发的产品。详细信息请参见第 6 章。
- 由于存在安全关键系统相关的可靠性问题，所以某些方面应正规化，例如需求跟踪、要达到的测试覆盖级别、要达到的验收准则以及必需的测试文档，以证明符合标准。
- 要证明符合组织的结构和开发过程，可以使用审查和组织结构图。
- 根据适用的标准，遵照一个预定义的开发生命周期。这种生命周期通常有特定的顺序。
- 若组织将一个系统归为“关键”系统，则在测试策略和/或测试计划中必须描述以下非功能性的特性：
  - 可靠性（Reliability）
  - 可用性（Availability）
  - 可维护性（Maintainability）
  - 安全性（Safety and security）

  由于这些特性，这种系统有时也被称作 RAMS 系统。

3.11.4 *其它测试管理问题*

缺少非功能性测试的计划可能导致被测对象存在相当大的风险。然而，许多类型的非功能性测试的成本很高，所以要考虑成本和风险之间的平衡。

非功能性测试的类型有很多，不是所有的类型都适合一个给定的应用程序。

下列因素会影响非功能性测试的计划和执行：

- 项目干系人的需求
- 需要的工具
- 需要的硬件
- 组织因素
- 通信
- 数据安全

3.11.4.1 项目干系人的需求

非功能性需求定义通常都很粗略，有时甚至根本没有提供。在计划阶段，测试人员必须有能力从受影响的项目干系人那里获得期望的非功能特性的级别并评估这些级别所代表的风险。

建议在获取需求的时候多收集不同的观点。需要必需来自不同干系人，例如客户、用户、操作人员和维护人员；否则，可能会遗漏某些需求。

在改进非功能性需求的可测性时应考虑下列要点：

- 阅读需求的次数比写需求的次数要多得多。将精力投入到编写可测的需求几乎总是划算的。使用简单的语言，简洁一致（例如，使用项目数据字典中定义的词）。在使用如“需要”（强制的），“应该”（期望的）和“必须”（最好避免使用“必须”，可以用“需要”的同义词代替）等词语时要特别小心。
- 需求的读者有不同的背景。
- 需求必须简洁明确，避免多义性。应当使用标准的格式来编写每一个需求。
- 应尽可能定量地描述需求。确定适合的度量用于表达某个特征（例如以毫秒计的性能）并说明在哪个范围内的结果可接受的或应拒绝。对于某些非功能性特性（例如易用性）而言可能比较困难。

3.11.4.2 工具需求

商业工具或模拟器特别适用于性能、效率和一些安全测试。测试计划应包括对使用测试工具的成本和时间的估算。假如需要使用专业工具，制订计划的时候应考虑新工具的学习曲线或雇用外部工具专家的成本。

复杂模拟器的开发可作为一个独立的开发项目，与其它开发项目一样，复杂模拟器的开发拥有其自身的权利并进行相应的计划。特别地，对于安全关键应用系统而言，模拟器开发的计划过程中应考虑进行验收测试或请一个独立的组织进行鉴定。

3.11.4.3 硬件需求

许多非功能性测试要求有一个仿真的测试环境，以便可进行实际的度量。被测系统大小和复杂度的不同可能对测试的计划和资金预算产生很大影响。执行非功能性测试的成本可能非常高，甚至只有有限的时间执行测试。

例如，为了验证大访问量的互联网网站是否符合可扩展性需求，可能会要模拟成千上万的虚拟用户。这对硬件和工具的成本有很大影响。通过租借（例如，“充值式（Top-up）”）性能工具的许可证方式可以将这些成本降到最低，但是这些测试的时间通常会受到限制。

执行易用性测试要建立专用实验室或使用调查表进行广泛的调查。这些测试通常在一个开发周期内只执行一次。

许多其它类型的非功能性测试（例如，安全测试、性能测试）要求在一个仿真的环境中执行。由于建立仿真环境的成本非常高，所以最实际可行的方法是使用真实的环境。此种测试执行的时间安排需要非常小心，很有可能只能在特定的时间执行（例如夜晚）。

当执行与效率相关的测试（例如性能和负载）时，应将电脑和通信的带宽列入计划。电脑和通信带宽的需求主要取决于将模拟的虚拟用户数和生成的网络流量。如果没有考虑到这个因素可能会导致度量的性能值不具有代表性。

3.11.4.4 组织因素

非功能性测试可能包括度量完整系统中的几个组件的行为（例如，服务器、数据库、网络）。若这些组件分布在许多不同的场所和组织中，则可能要花很大的精力进行计划和协调测试。例如，某些软件组件可能只在每天或每年的特定时间段才能用于系统测试，或组织只能提供有限的天数用于支持测试。如果其它组织的系统组件或人员不能确定是否能参与测试，将可能导致严重扰乱预定的测试。

3.11.4.5 通信因素

设计和执行特定类型的非功能性测试（尤其是效率测试）的能力可能取决于为了测试而修改特定通信协议的能力。在计划阶段就应注意是否存在这样的可能（例如，提供必需的兼容性的工具）。

3.11.4.6 数据安全因素

在测试计划阶段就应考虑到用于确保系统安全的手段，以确保所有的测试活动可行。例如，使用数据加密技术可能增加创建测试数据和验证结果的难度。

数据保护政策和法令可能会禁止基于生产数据生成虚拟用户。创建匿名测试数据是一件重要的任务，必须将其作为测试实现的一部分计划。

4. 测试技术

术语：

BS 7925-2，边界值分析（BVA），分支测试，因果图技术，分类树方法，条件测试，条件判定测试，控制流分析，决策路径，数据流分析，决策表测试，判定测试，基于缺陷的技术，缺陷分类，动态分析，错误推测，等价类划分，探索性测试，基于经验的技术，线性代码序列和跳转（LCSAJ），内存泄漏，条件组合测试，结对测试，路径测试，基于需求的测试，软件攻击，基于规格说明的技术，静态分析，语句测试，状态转换测试，基于结构的技术，测试章程，用例测试，野指针（Wild pointer）。

4.1 简介

本章涉及的测试设计技术包括如下类别：

- **基于规格说明（或基于行为，或黑盒）**
- **基于结构（或白盒）**
- **基于缺陷**
- **基于经验**

不管是何种级别的测试，这些技术都是互补的，并且可以针对给定的测试活动进行恰当的应用。尽管测试分析员和测试技术分析员可以采用其中的任何技术，但是针对本大纲的目的，根据最常用的使用方式，对这些技术进行下面的分类：

- **基于规格说明→** 测试分析员和测试技术分析员
- **基于结构→** 测试技术分析员
- **基于经验→** 测试分析员和测试技术分析员
- 基于缺陷→ 测试分析员和测试技术分析员

此外，本章还包括对其它技术的讨论，例如攻击、静态分析和动态分析。通常测试技术分析员采用这些技术。

请注意，基于规格说明的技术既可以是基于功能的也可以是基于非功能的。非功能的技术将在下一章讨论。

4.2 基于规格说明

基于规格说明的技术是通过分析组件或系统的测试依据文档，而不是它们的内部结构，来得到和选择测试条件或测试用例的一种方法。

基于规格说明的技术的共同特性有：

- 利用正式的或非正式的模型来描述待解决的问题、软件或其组件。
- 从这些模型中能系统地得到测试用例。

有些技术还提供覆盖准则，可以用来度量测试设计工作。完全满足覆盖准则并不意味着整个测试集的完成，而是说明该模型在当前技术下，并不能提供任何其他有用的测试。

基于规格说明的测试通常是基于需求的测试。因为需求规格说明应指定系统如何执行，特别在功能领域，而从需求中得到的测试常常是测试系统行为的一部分。在本节前面部分讨论的技术可以直接应用

于需求规格说明，从需求规格说明的文字或图示生成系统行为的模型，接下来的测试工作将按照前面描述的过程进行。

在高级大纲中，将重点探讨下列基于规格说明的测试设计技术：

名称	技术	覆盖准则
等价类划分	请参见 ISTQB®初级大纲 4.3.1 节的描述。	被覆盖的等价类数/等价类总数。
边界值分析（BVA）	请参见 ISTQB®初级大纲 4.3.2 节的描述。 注意边界值分析可能既可以用到 2 个值也可以用到 3 个值，具体使用哪个值通常根据风险来决定。	被覆盖的不同边界值数/边界值总数。
决策表测试和因果图技术	请参见 ISTQB®初级大纲 4.3.3 节有关决策表测试的描述。 使用决策表测试时，将对每个条件、关系和约束的组合进行测试。除了决策表外，还能使用称为因果图的逻辑记号图形技术。 注意有可能用压缩的决策表来测试几乎所有的条件组合。使用完整的决策表还是压缩的决策表，通常会根据风险来确定。[Copeland03]	被覆盖条件组合数/最大的条件组合数。
状态转换测试	请参见 ISTQB®初级大纲 4.3.4 节的描述。	对单个状态转换，覆盖度量是指测试过程中所覆盖的有效状态转换的百分比。这是所谓的 0 交换覆盖。对 n 状态转换，覆盖度量是指测试过程中所覆盖的有效的 n 个状态转换的组合序列的百分比，即（n-1）交换覆盖。
分类树方法，正交阵列和全结对表	识别出关注的因素或变量和能呈现的选项或值，以及识别出这些选项的单个值、成对值、三个值、或更高阶的组合。[Copeland03] 分类树方法使用图形记号来呈现测试条件（类）和测试用例覆盖的组合。[Grochtmann94]	依赖于采用的技术，例如，结对测试不同于分类树方法。
用例测试	请参见 ISTQB®初级大纲 4.3.5 节的描述。	没有正式的准则。

有时采用组合的技术生成测试。例如，在决策表中的条件可以用等价类划分方法将其划分成不同的等价类，每个等价类作为决策表的一个条件，这样更有效。测试不仅可能覆盖其条件的每一种组合，对于已分类的那些条件，还可增加测试以覆盖划分的等价类。

4.3 基于结构

基于结构的测试设计技术，有时候也称之为白盒或基于代码的测试技术，是将代码、数据、架构或系统流程作为测试设计的依据，从结构系统地导出测试。测试技术决定了所测结构的要素，提供何时结束测试导出的覆盖准则。这些准则并不意味整个测试集已完成，仅仅指所测结构已不再能基于此技术推出有用的测试。必须衡量每个准则，并且将它与每个项目或公司规定的目标相关联。

基于结构的技术的共同特性：

- 有关软件结构的信息是设计测试用例的依据，例如，代码和设计。
- 软件覆盖的范围能通过已有的测试用例来测量，并且可以系统地增加测试用例来增加覆盖率。

在高级大纲中，将讨论下列基于结构的测试设计技术：

名称	技术	覆盖准则
语句测试	测试可执行的（非注释、非空白）语句。	执行的语句数/语句总数。
判定测试	测试判定语句，例如 if/else, switch/case, for, 和 while 语句。	执行的判定结果数/总的判定结果数。
分支测试	测试分支，例如 if/else, switch/case, for, 和 while 语句。 注意：在 100%覆盖时，判定和分支测试相同，但在低覆盖水平时，两者可以不同。	执行的分支数/总的分支数。
条件测试	测试 true/false 和 case 标记语句。	执行的布尔操作数值数/ 布尔操作数值总数。
条件组合测试	测试所有可能的 true/false 条件组合。	执行的布尔操作数值组合数/布尔操作数值组合总数。
条件判定测试	测试能影响判定（分支）的 true/false 条件可能的组合。	独立影响判定的布尔操作数值数/布尔操作数值总数。
线性代码序列和跳转测试 LCSAJ（循环测试）	测试控制循环迭代的可能条件。线性代码序列和跳转测试（LCSAJ）用来测试代码特定的部分（一个线性序列的可执行代码），即从特定控制流的开始处到跳转到其它控制流或程序结尾处结束。这些代码段称为 DD 路径（decision-to-decision path）。该技术用来定义特定的测试用例和执行选择的控制流所需要的相关测试数据。设计这些测试需要使用定义控制流跳转的源代码模型。LCSAJ 能用作代码覆盖度量的基础。	执行的线性代码序列和跳转的次数 LCSAJ/总的线性代码序列和跳转次数 LCSAJ。
路径测试	识别唯一的语句序列（路径）。	执行的路径数/路径总数。

结构覆盖准则通常用于衡量采用基于规格说明和/或基于经验的技术来设计的测试集的完整性或不完整性。称为代码覆盖工具的测试工具用来捕获这些测试的结构覆盖率。如果发现重要的结构覆盖率不足（或采用的标准要求的覆盖率没有达到），则通过使用结构和其他技术增加测试用例弥补这些差距。

覆盖率分析

采用基于结构或其它技术的动态测试可以用来确定特定的代码覆盖率是否达到。对于必须达到特定覆盖率的关键系统而言，可用于提供证据（请参见 1.3.2 安全关键系统）。覆盖率分析的结果可以指出哪些代码段没有被执行，从而可以增加测试用例来执行这些代码段。

4.4 基于缺陷和基于经验

4.4.1 *基于缺陷的技术*

基于缺陷的测试设计技术是以发现的缺陷类型作为测试设计的基础，根据对缺陷已知的信息来系统地进行测试。

基于缺陷的技术也提供覆盖准则，以决定测试何时可以结束。作为比较特殊的技术，基于缺陷技术相较于基于行为和基于结构的技术，其覆盖准则的系统性要弱一些。它先给出覆盖通则，在测试设计或测试执行时关于有用覆盖限制的具体决策是自行决定的。覆盖准则并不意味整个测试集已完成，仅仅指所考虑的缺陷已不需要基于此技术作出任何有用的测试。

在高级大纲中，讨论下列基于缺陷的测试设计技术：

名称	技术	覆盖准则
分类法（类别与潜在缺陷列表）	测试人员采用列表中的分类法样本，选择一个潜在问题进行分析。分类法能列出根本原因、缺陷和失效。缺陷分类法可以罗列被测软件中最常见的缺陷，而这个列表可以用于设计测试用例。	恰当的数据缺陷和缺陷类型。

4.4.2 *基于经验的技术*

还有其它的一些测试设计技术，它们考虑缺陷历史，但并不需要系统的覆盖准则，称为基于经验的测试技术。

在基于经验的测试中，除了利用测试人员有相似应用或技术的经验外，还利用测试人员的个人技能和直觉。基于经验的测试可以有效的发现缺陷，不过在达到特定测试覆盖率水平或产生可重用的测试规程方面，不像其它技术这样适用。

当使用动态和启发式方法时，测试人员趋向于使用基于经验的测试，因为其测试比预先计划的方法对事件更灵活。另外，基于经验的测试的执行和评价是同时进行的。基于经验测试的一些结构化方法并不是完全动态的；即测试不完全是在测试执行时同时产生的。

注意：虽然下表介绍了一些覆盖准则，但是基于经验的技术没有正式的覆盖准则。

在本大纲中，将讨论下列基于经验的测试设计技术：

名称	技术	覆盖准则
错误推测	测试人员利用经验来推测潜在的可能已产生的错误，并且确定发现缺陷的方法。错误推测也用于风险分析过程中识别潜在的失效模式。[Myers97]	当采用分类时，适当的数据故障和缺陷类型。没有分类时，覆盖受限于测试人员的经验和可用的时间。
基于检查表	有经验的测试人员采用概要的列表来记录、检查或者记忆，或者通过一组规则或准则，来对产品进行验证。这些检查表是根据一组标准、经验和其它考虑建立的。用于测试基础的用户接口标准检查表来测试一个应用，是基于检查表测试的一个例子。	**检查表中每项都被覆盖。**
探测性测试	测试人员同时学习产品及其缺陷，计划要做的测试工作，设计和执行测试，以及报告结果。好的探测性测试是有计划的、交互的和创造性的。测试人员在执行过程中动态地调整测试目标，并且仅仅准备少量的文档。[Veenendaal02]	创建相应的章程来明确任务、目标和交付内容。探索性测试要予以计划，需要确定完成什么、关注何处、什么在范围内、什么在范围外以及承诺的资源。另外，需要考虑一些对缺陷和质量的启发性思考。
攻击	测试人员通过尝试迫使系统出现特定的失效，来对系统质量进行直接和集中的评价。软件攻击的原理，如[Whittaker03]中所描述的，是在其操作环境下，基于被测试软件的相互作用，这包括用户接口、操作系统内核、应用编程接口（API）和文件系统。这些相互作用基于精确的数据交换，一个或多个相互作用时的任何失调都会导致失效的发生。	测试应用程序的不同接口。主要的接口包括用户接口、操作系统、内核、API 和文件系统。

基于缺陷和基于经验的技术通过应用缺陷的知识和其它经验来提高缺陷发现率。范围从测试人员非正式的、无预先计划活动的“快速测试”到预先计划的活动再到有脚本的活动。它们通常总是有用的，而且在下列情况下有其特别价值：

- 没有可用的规格说明
- 被测试系统的文档质量很差
- 没有足够时间设计和生成测试规程
- 测试人员在领域和/或技术方面有经验
- 从脚本化测试中寻找多样性
- 分析操作失效

基于缺陷和基于经验的技术，也可与基于行为和基于结构的技术共同使用，它们可以对这些技术由于系统性弱点而产生的测试覆盖率差距进行补充。

4.5 静态分析

静态分析测试一般和软件代码或系统架构有关，并不真正运行被测试软件。

4.5.1 代码的静态分析

静态分析是不需要执行代码的测试或检查类型。有多种静态分析技术，本节将讨论这些技术。

4.5.1.1 控制流分析

控制流分析提供有关软件系统的逻辑判定点及其结构复杂度的信息。控制流分析的描述请参见ISTQB®初级大纲和[Beizer95]。

4.5.1.2 数据流分析

数据流分析是一种基于结构的测试技术，测试从设置变量到使用变量之间的路径。这些路径的术语称为定义与使用对（Du-pairs）或设置与使用对。在这个方法中，生成的测试集主要用来获得针对这些对的 100%覆盖（在可能的情况下）。

这个技术尽管称为数据流分析，但是当它追踪每个变量的设置和使用时，也可能需要贯穿软件的控制流，所以也需要考虑被测试软件的控制流。请参见[Beizer95]。

4.5.1.3 符合编码标准

静态分析阶段，可对编码标准的符合性进行评估。编码标准包括架构方面和某些编程结构的使用（或禁用）。

符合编码标准使得软件更容易维护和测试。特定的语言要求也能通过静态测试来验证。

4.5.1.4 生成代码度量

静态分析过程中能产生代码度量，有助于提高代码的可维护性或可靠性。这些度量的例子如下：

- 圈复杂度
- 规模
- 注释率
- 嵌套的层数
- 函数调用数。

4.5.2 架构的静态分析

4.5.2.1 网站的静态分析

使用静态分析工具也能评价网站的架构。目标是检查网站的树状结构是否平衡，如果不平衡将导致：

- 更多困难的测试任务
- 增加维护的工作量
- 用户导航困难

有些特定的测试工具包含网络蜘蛛引擎，通过静态分析，能提供有关网页大小和下载所需的时间、以及网页是否存在（即 http 404 错误）等信息。这可以为开发者、网站管理员和测试人员提供了有用的信息。

4.5.2.2 调用图

调用图用于以下目的：

- 设计测试来调用特定模块
- 在软件中设立一组位置，确定模块在哪里被调用
- 提供集成顺序（结对集成和邻近集成 [Jorgensen02]）的建议
- 评估全部代码结构及其架构

调用模块信息也能在动态测试时得到，此信息是指执行过程中模块调用的次数。通过合并静态分析时的调用图信息和动态分析时收集的信息，测试人员能关注经常和/或重复被调用的模块。如果这种模块较为复杂（请参见 1.3.2.2 安全关键系统和复杂性），则是首选要进行详细和广泛测试的对象。

4.6 动态分析

动态分析的原则是在软件应用运行时对它进行分析。这通常要求人工或自动地在代码中插入一些探测器。

4.6.1 概述

无法立即重现的缺陷会对测试的工作量和发布或高效使用软件的能力产生重大影响。这种缺陷可能来自于内存泄漏、不正确地使用指针和其它错误（例如系统堆栈错误）[Kaner02]。这些缺陷可能使得系统性能逐步恶化，甚至系统崩溃，针对这种特性，测试策略必须考虑这些缺陷的相关风险，并且适宜时，执行动态分析来减轻风险（通常需要使用工具）。

动态分析在软件执行时进行，可应用于：

- 通过探测野指针和系统内存丢失，阻止失效的发生
- 分析不容易再现的系统失效
- 评价网络行为
- 通过提供运行时系统行为信息来改进系统性能

动态分析可在任何测试级别进行，但在组件和集成测试中尤其有用。这需要相关技术和系统知识来描述动态分析的测试目标，尤其是结果分析。

4.6.2 探测内存泄漏

当程序可用的内存（RAM）被该程序分配后，如果后面因为程序错误而没有释放，就会发生内存泄漏。因此，程序丧失存取这块内存的能力，最后可能导致所有可用内存耗尽。

内存泄漏导致的问题随着时间的推移而发展，并且通常不能立即发现。不能很快被发现可能是因为软件最近刚安装或系统重启，这些在测试中经常发生。由于这些情况，内存泄漏的负面影响通常是在产品实际运行环境中才首次被发现。

内存泄漏的征兆是系统响应时间不断地恶化，并最终导致系统失效。尽管这种失效可以通过重新启动（重启）系统来解决，但是这常常不是一个实用的或甚至可行的解决方案。

工具能识别出代码中内存泄漏发生的区域，以便能被修正。简单的内存监测也能获得可用内存是否随着时间而减少的总体情况。如果出现了这种情况，仍然需要做进一步的分析。

还应该考虑其它类型的泄漏，如文件句柄、信号和连接池等资源。

4.6.3 探测野指针

程序中的“野指针”是在某些方面不能用的指针。例如，可能有指针“丢失”了它应该指向的对象或函数，或有些指针没有指向它应该指向的内存（例如，指针超过了数组的边界）。当程序使用野指针时，可能产生多种结果：

1. 程序可能按照预期的执行。这种情况，野指针可能存取当前没有被程序使用的内存，并且是“空闲”的。
2. 程序可能崩溃。这种情况，野指针可能导致使用其它程序运行的部分关键内存（例如操作系统）。
3. 由于程序不能存取要求的对象导致程序功能不正确。在这种情况下，尽管可能发出一个错误信息，但是程序有可能继续运行。
4. 指针可能导致数据崩溃，后续使用错误值。

注意：任何对程序内存使用的更改（例如软件更改后的新软件包）都有可能触发上面所列的后果。特别关键的是尽管使用了野指针，程序在开始的时候仍然可以按照预期的执行，但是软件在发生了某个变化后，程序出乎意料地崩溃了（甚至可能在产品运行环境中）。需要非常重视的是这种失效是潜在错误（例如野指针）的表现，而不是错误本身。（请参见[Kaner02], “Lesson 74”）

当在程序中使用工具时，不考虑其在程序执行中的后果，有可能识别出野指针。

4.6.4 性能分析

动态分析也可以用来分析程序性能。工具能帮助识别程序的性能瓶颈，并且产生大量的性能度量指标，开发人员可以根据这些指标对系统进行“调整”。这通常被称为“性能调优（Performance Profiling）”。

5. 软件特征测试

术语

辅助测试，准确性测试，效率测试，启发式评估，交互性测试，可维护性测试，运行验收测试（OAT），运行概况，可移植性测试，可恢复性测试，可靠性增长模型，可靠性测试，安全性测试，适用性测试，SUMI（软件易用性度量调查表），易用性测试。

5.1 简介

前一章描述了测试人员可用的具体的测试技术，本章则关注如何应用这些技术来评估关键属性。这些关键属性常被用来反映应用程序或软件系统的质量。

在本大纲中，测试分析员和测试技术分析员各自需要评估的质量属性将在不同的章节中考虑。ISO9126 中提供的质量属性描述作为描述这些属性的指导。

理解不同质量属性是本大纲的所有三个模块的一个基本的学习目标。针对涵盖的不同质量属性，大纲要求测试分析员和测试技术分析员进行更加深入的理解，这样才可以识别典型的风险，采取合适的测试策略和设计合适的测试用例。

5.2 领域测试的质量属性

功能性测试关注产品做什么。功能性测试的依据通常是需求文档或者规格说明文档，特定领域的专业知识或隐含的需求。功能性测试在不同的测试级别或测试阶段也不尽相同。例如，在集成测试阶段，功能性测试将测试接口模块之间的功能，其中每个模块实现某个定义的单一功能。在系统测试阶段，功能性测试将会视应用为整体进行功能测试。在综合系统测试中，功能性测试主要关注端对端的测试。这些端对端的测试跨越了已经集成的多个系统。

在功能性测试中应用了大量的测试技术（请参见第 4 章）。功能性测试可以由专门的测试人员、或领域专家或开发人员（常在组件级）执行。

功能性测试考虑了如下的质量属性：

- 准确性
- 适用性
- 交互性
- 功能安全性
- 易用性
- 辅助性

5.2.1 准确性测试

功能准确性测试包括验证应用程序是否满足定义的或隐含的需求，还可包括计算的准确性验证。在准确性测试中将大量应用第 4 章中的测试技术。

5.2.2 适用性测试

适用性测试包括评估和确认功能是否适合完成预期任务。这类测试可以基于用例（Use cases）或规程（Procedures）。

5.2.3 交互性测试

交互性测试是验证应用程序在各种预定的目标环境下（硬件、软件、中间件、操作系统等）是否可以正常工作。在交互性测试时，测试团队需要识别，配置各种指定的目标环境的组合并将环境准备就绪。通过选择那些调用目标环境中出现的不同组件的功能性测试用例，进而测试这些目标环境。

交互性与多个参与交互的软件系统之间的区别相关。具有良好的交互性的软件，在不需要较大变更的情况下，可以容易地与其它系统集成。变更的数目和用来完成这些变更所需要的工作量可以作为软件交互性的一个度量指标。

例如，软件交互性测试可以关注以下设计特性：

- 软件是否使用工业界认可的通信标准，例如 XML。
- 软件是否具有自动检测与该软件交互的系统的通信请求并做相应处理的能力。

交互性测试在下列情况下尤其重要：

- 开发商业软件（COTS）或工具的组织
- 开发综合系统

这类测试主要在系统集成测试中执行。

5.2.4 功能安全性测试

功能安全性测试（渗透测试）侧重于检验软件防止未经授权对其功能和数据有意或无意地访问的能力。这个测试包括用户权限、访问和优先级的内容。此类信息应定义在系统规格说明中。安全性测试还包括许多与测试技术分析员更为相关的其它内容，这将在后面的 5.3 节论述。

5.2.5 易用性测试

了解为什么用户在使用已经符合需求的软件系统时可能会有困难，这点很重要。要做到这点，必须明白“用户”这个词的外延，它适用于不同类型的人群，从 IT 专家到儿童或者残疾人士。

一些国家机构（例如英国皇家国立盲人学院），建议网页应当方便于盲人、弱视人群、行动障碍人士、失聪人士和智障人士等残疾人使用。检查应用程序和网站是否适用于上述用户，同样也会改进针对其他人群的易用性。

易用性测试是测试软件对于用户的适用性，也就是针对下列因素进行测试，使特定用户可以在特殊环境或背景下达到特定的目标：

- 有效性：软件产品能使用户在特定的使用环境下准确完整地实现具体目标的能力
- 高效性：软件产品能使用户在特定使用环境下，耗费适量的资源去实现有效性的能力
- 满意度：软件产品能使用户在特定使用环境下感到满意的能力

可以度量的属性包括：

- 易懂性：软件的属性之一，表示用户认知软件逻辑概念及其适用性所耗费的精力
- 易学性：软件的属性之一，表示用户学习软件的使用所耗费的精力

- 可操作性：软件的属性之一，表示用户有效地和高效地实施任务所耗费的精力
- 吸引性：软件受用户喜爱的能力

易用性评估有两个目的：

- 排除易用性方面的缺陷（有时作为形成性评价）
- 对于易用性需求进行测试（有时作为总结性评价）

测试人员应具备下列专业技能或知识：

- 社会学
- 心理学
- 符合国家标准（例如美国残疾人法案）
- 人类工程学

实际实施易用性确认应尽可能地在接近系统实际运行的环境条件下运行。这个可能包括建立带有摄像头、模拟办公室、评审台、用户等等的易用性测试实验室，在此实验室，开发人员可以观察真实人员使用实际系统的效果。

许多易用性测试可能会作为其它测试的一部分来进行，例如在功能性系统测试过程中。易用性指导书对于实现在生命周期的各个阶段用统一的方法检测和报告易用性故障将大有裨益。

5.2.5.1 易用性测试规格说明

易用性测试的主要技术：

- 审查、评估或评审
- 实际实施易用性验证和确认
- 进行问卷调查

审查评估或评审

从易用性的角度对规格说明和设计进行审查和评审，可以增加用户方面的介入，以较低的成本在早期发现相关问题。

启发式评估（用户界面设计易用性的系统审查）可以作为迭代设计流程中的一个环节，用来发现设计中的易用性问题并引起关注。这涉及到一小部分的评估人员审查界面并判断其是否遵守公认的可用性原则（“启发式”）。

实际实施的确认

为了执行实施的确认，对于功能性系统测试的规范可以演化成可用性测试场景。这些测试场景测试特定的可用性属性而非功能性结果，例如学习速度或可操作性。

对于可用性，测试场景能够特别地演化为针对语法与语义的测试：

- 语法：界面的结构或语法（例如，什么可以被输入到输入区域）
- 语义：含义和目的（例如，为用户提供的合理并有意义的系统消息和输出）

开发这些测试场景的技术有：

- 例如在 BS-7925-2 标准中描述的黑盒方法
- 用纯文本或 UML（统一建模语言）描述或定义的用例

可用性测试的测试场景包括用户指导、测试前进行指导和测试后收集反馈允许的时间，以及关于如何开展的协议。该协议描述如何进行测试、时间安排、记录和活动日志等方面的内容，以及拟采用的会谈和调查方法。

调查与问卷

调查与问卷技术可应用于收集用户在可用性测试实验室使用系统时的行为观察。规范和公开的调查，例如 SUMI（软件可用性度量调查表）和 WAMMI（网站分析和度量调查表），允许采用以往的可用性测试数据做基准。另外，由于 SUMI 提供了具体的可用性测试数据，为其作为完成/验收标准提供了很好的机会。

5.2.6 *辅助测试*

对于在使用中有特殊需要或限制的用户，软件的辅助性是非常重要的。这包括残疾人士。辅助测试必须要考虑参照相关标准，如网站内容可访问性指南和法规、残疾歧视法案（英国，澳大利亚）和 Section 508 法案（美国）。

5.3 技术测试的质量属性

测试技术分析员所关注的质量属性重点在于产品“如何”运行，而不是它在功能性方面上做“什么”。这些测试可以发生在任何一个测试级别，但是与下面的测试有特殊的关联性：

组件测试（特别是实时嵌入式系统）

- 性能基准
- 资源使用

系统测试和运行验收测试（OAT）

- 根据风险和可用资源，包括下面提到的任何一个质量属性和子属性。
- 在这个级别上，面向技术的测试目的在于测试特定的系统，例如，硬件和软件的组合，包括服务器、客户端、数据库、网络和其它的资源。

在软件成为产品后，往往会由另一个单独的团队或组织继续进行测试。从正式投入运营前的测试中获得的质量属性度量，可以作为供应商和运营商间的服务级别协议（SLA）的基准。

可以考虑以下的质量属性：

- 技术安全性
- 可靠性
- 有效性
- 可维护性
- 可移植性

5.3.1 *技术安全性测试*

安全性测试在两个重要方面不同于其它形式的领域或技术测试：

1. 选择测试输入数据的标准技术可能遗漏重要的安全性问题
2. 安全性故障的现象完全有别于其它类型测试中发现的现象

当软件不仅实现设计中的功能，还执行许多非预期的额外动作时，软件将存在许多安全性漏洞。这些副作用代表了诸多软件安全性最大威胁中的一个。例如，一个媒体播放器可以正确播放音频，但同时会在未加密的临时存储中写入文件，这类副作用给软件盗版商留下可乘之机。

防止信息被未授权者非法使用是安全性最为关注的问题。安全性测试就是试图通过对系统的薄弱环节进行攻击的方法去破坏系统的安全性策略，例如：

- 对应用程序或数据进行未授权的复制（如盗版）
- 未授权的访问控制（例如，用户能够执行其没有权限执行的任务）

- 缓存区溢出（缓存区超限），可能由于在用户界面输入框中输入超长字符串造成。发生缓存区溢出，则存在运行恶意代码指令的可能性
- 服务拒绝，阻止用户与应用程序的交互。（例如，通过发送“骚扰”请求使网络服务器超载）
- 在网络上窃听数据传输来获得敏感信息（例如，在信用卡交易中）
- 破解保护敏感信息的加密代码
- 逻辑炸弹（在美国有时被称为“复活节彩蛋”），可能被蓄意的埋进代码中，在特定条件被激活（如在某个特定日期）。当逻辑炸弹激活时，它们可能执行恶意行为，如文件删除或格式化磁盘。

具体的安全性方面可以分为如下几类：

- 用户接口相关
  - 未授权访问
  - 恶意输入
- 文件系统相关
  - 访问文件或者存储库中的敏感数据
- 操作系统相关
  - 敏感信息的存储，例如在内存中存储未经加密的（用户）口令。通过恶意输入使系统崩溃就可能得到这个信息。
- 外部软件相关
  - 与系统中使用的外部组件交互。这些可能存在于网络级别（如不正确的包或消息传递），或在软件组件级别（如软件依赖的一个组件失效）。

值得注意的是，这些对于系统安全性的改进可能会影响其性能。建议在完成安全性改进后，考虑再次进行性能测试。

5.3.1.1 安全性测试规格说明

以下方法可用于完成安全性测试：

- 进行信息检索
  可以使用各种工具来创立系统的概况或网络图。这些信息可能包括员工姓名、物理地址、内网的详细信息、IP 地址、使用的软件或硬件的标识和操作系统版本。
- 进行漏洞扫描
  可以使用各种工具对系统进行扫描。这类工具不会给系统造成危害，但可以识别已经违反或将导致违反安全性策略的漏洞。也可以使用检验表来识别特定的安全漏洞，www.testingstandards.co.uk网站提供类似的检验表。
- 通过使用获得的信息，研制“攻击方案”（也就是企图破坏某一系统安全性策略的测试行为的方案）。需要在攻击方案中制定针对各种接口（用户界面，文件系统）的输入，以检测出最严重的安全性缺陷。
- 在[Whittaker04]中描述的各种“攻击”，是针对安全性测试技术的宝贵资料。更多的关于“攻击”的信息，请参见 4.4 节。

5.3.2 可靠性测试

可靠性测试的目的之一是对软件成熟度一段时间的统计度量进行监控，并将其与既定目标相比较。该度量指标可以选用平均故障间隔时间（MTBF）、平均修复时间（MTTR）或其它故障强度度量（例如，每周特定严重程度的故障的数量）。所监控的值在时间上的分布可以表示为可靠性增长模型。

可靠性测试可以有两种形式，一是重复进行预定的测试，二是随机测试（可以从用例库中选取或由统计模型产生测试用例）。进行这些测试颇为耗时（数日量级）。

软件成熟度在时间上的度量指标分析可以作为出口准则（例如，进行产品发布）。特定的度量指标，如 MTBF 和 MTTR，可以设立为服务水平协议并在实际系统运行中进行监控。

软件可靠性工程和测试（SRET）是可靠性测试的标准方法。

5.3.2.1 健壮性测试

通过功能测试可对软件处理非期望输入（所谓的逆向测试）的容错性进行评估，而通过技术测试则可对系统中由外部故障引发的容错性进行评估。所谓外部故障通常是指由操作系统报告的故障（例如，磁盘已满、过程或服务无效、未发现文件、内存无效）。系统级的容错性测试可以通过专门工具来完成。

5.3.2.2 可恢复性测试

可靠性测试的进一步表现是对软件系统按照规定的方式从软硬件故障中恢复至正常状态的能力进行评估。可恢复性测试分为故障转移测试和备份/恢复测试。

某些软件故障可引发严重后果，因而会采取专门的软硬件措施以保障系统即使在出现故障时仍能运行。针对这种情况要进行故障转移测试。例如，故障转移测试适用于金融损失风险极高或存在关键安全隐患的情况。这种对于灾变性事件引发故障的可恢复性测试也被称为“灾难恢复”测试。

典型的硬件方面的措施包括平衡各个处理器负载，集群化服务器、处理器或硬盘，以便在一个硬件失效的时候，另一个马上接管。（例如，磁盘冗余阵列 RAID）。典型的软件方面的措施则可在一个所谓的非相似冗余系统（Redundant dissimilar system）中实现多个独立软件系统（例如，航空飞行控制系统）。冗余系统一般结合了上述的软件和硬件措施，根据独立实例的个数（2、3 或 4 个）可以分别称之为二重、三重或四重系统。软件非相似性的实现，即令两个（或多个）互不相关的开发团队按相同的软件需求，开发不同的软件来提供相同的服务。因而使非相似冗余系统在相似的缺陷输入情况下不至于导致相同的后果。这些增强系统可恢复性的措施会直接影响其可靠性，可在可靠性测试中予以考虑。

故障转移测试专门针对这类系统，它模拟故障模式或将其在受控环境中执行。针对故障转移机制进行测试，可以保证数据不至丢失或缺损且系统保持原有服务水平（例如，功能可用性和响应时间）。关于故障转移测试的更多信息，请浏览 www.testingstandards.co.uk。

备份/恢复测试关注使故障影响最小化的过程性指标。这类测试对进行不同形式的备份并将其在数据丢失或缺损时进行恢复的（通常写入手册的）过程进行评估。测试用例的设计要保证能够覆盖该过程的关键路经。可通过技术评审预想这些场景并针对实际的安装过程验证各类手册。运行验收测试（OAT）在产品环境或类似环境中实施这些场景，确认它们的实际效用。

备份/恢复测试的指标可以包括：

- 完成各类备份（例如全备份或增量备份）所需时间
- 恢复数据所需时间
- 承诺的数据恢复水平（例如，24 小时内数据的恢复、1 小时内特定传输数据的恢复）

5.3.2.3 可靠性测试规格说明

可靠性测试基本上是基于不同的使用模式（有时称为“运行概况”）常规地进行，或根据风险状况来进行的。可以使用随机或伪随机方法产生测试数据。

需要合理选取可靠性成长曲线，可使用工具对一组故障数据进行分析，确定最切合现有有效数据的可靠性成长曲线。

可靠性测试可以专门针对内存泄漏进行。此类测试的规格说明要求重复地执行特定的内存存取行为，并保证占用的内存能够正确地释放。

5.3.3 效率测试

效率质量属性是通过针对时间行为和资源行为的测试来进行评估。下面针对时间行为的效率测试在性能测试、负载测试、压力测试和可扩展性测试四个方面进行介绍。

5.3.3.1 性能测试

根据所关注的非功能性需求，性能测试可以分为几个不同的测试类型，包括性能、负载、压力和可扩展性方面的测试。

特定性能测试的关注点在于组件或系统在规定的时间内和特定的条件下（见下文中的负载测试和压力测试节的有关内容）响应用户或系统输入的能力。不同的性能的度量方法取决于不同的被测对象。对于一个单独软件组件，其性能可以根据 CPU 主频来判定。而带客户端的系统，其性能则要根据系统处理特定用户请求的响应时间来判定。对于那些由多种组件（如客户端、服务器、数据库）构成的系统，则要进行各组件之间的性能测试，从而发现系统的性能瓶颈。

5.3.3.2 负载测试

负载测试的关注点在于系统在处理预计实际负载级别增长方面的能力，实际负载级别增长是由多用户并发使用所产生的事务请求导致的。可以测量和分析各种典型的使用情况（运行概况）下系统的平均响应时间。请参见[Splaine01]。

负载测试有两种类型，多用户测试（与实际环境下用户数量相当）和海量测试（大量用户）。并检查这两类测试的响应时间和网络流量。

5.3.3.3 压力测试

压力测试的关注点在于系统在峰值负载或超出最大载荷情况下的处理能力。在压力级别逐渐增加时，系统性能应该按照预期缓慢下降，而不致失效。压力条件下，尤其需要测试系统功能的完整性和一致性，发现功能缺失或数据的不一致。

另一目标就是压力测试还可以发现系统崩溃的临界点，从而发现系统中的最薄弱环节。在进行压力测试时，允许向系统中适时地添加额外的组件（如内存，CPU 处理能力，数据库存储量）。

在极端负载测试中，将模拟产生各种条件组合，这种组合会引起系统突然的过量负载。脉动测试则是在低使用率的情况下，间歇地施加数次极端负载测试。通过这些测试，可以考察系统处理负载变化的能力并确定系统是否能够正确地请求和释放资源。请参见[Splaine01]。

5.3.3.4 可扩展性测试

可扩展性测试，关注于系统满足未来更高效率需求的能力。测试的目标是评判系统在不超出议定的限度或不失效情况下的增长能力（例如容纳更多的用户，存储更多的数据）。了解这些限度后，可以设定一些阈值并在运行过程中进行监控和预警出现的问题。

5.3.3.5 资源使用测试

资源使用效率测试针对系统资源的使用情况（如内存空间、磁盘容量和网络带宽）进行评估，并对其在正常负载和压力情况下的状况进行比较，如高级别的交换量和数据量。

例如，内存使用量（有时称为“内存占用”）在实时嵌入式系统的性能测试中具有重要作用。

5.3.3.6 效率测试规格说明

各种类型的效率测试（例如性能、负载、压力测试）的规格说明基于对运行概况的定义。它们代表用户与应用程序交互时不同的行为方式。一个应用程序可以具有数个运行概况。

使用监控工具，或者根据所预计的使用量，可以得到每个运行配置的用户数量（在已有真实系统或可参照系统的情况时）。可以遵循一些算法预计，或者由业务部门提供，这个预计值对于确定可扩展性测试的运行概况非常重要。

运行概况是测试用例的基础，通常由测试工具产生。在这种情况下，所谓“虚拟用户”特指运行调优中的一个模拟用户。

5.3.4 *可维护性测试*

一般而言，可维护性测试涉及对软件进行分析、变更和测试的难易程度。进行可维护性测试的方法包括静态分析和检查表方法。

5.3.4.1 动态可维护性测试

动态可维护性测试的关注点在于文档化规程，开发这些规程是为了维护一个特定应用程序（如进行软件升级）。测试时，可以选取维护场景作为测试用例，确保使用文档化的规程就可以达到所要求的服务水平。

这种测试方法尤其适用于下列情况：基础设施相对复杂，并需要多个部门协作支持。它同时可以作为运行验收测试（OAT）的一个部分。[www.testingstandards.co.uk]

5.3.4.2 可分析性（纠错性维护）

这种测试方法的关注点在于收集诊断并解决系统问题所需要的时间。一个简单的度量标准就是诊断并解决系统问题所需要的平均时间 。

5.3.4.3 可变性、稳定性和可测试性 （适应性维护）

系统的可维护性还可以根据系统变更（例如修改代码）所需的工作量进行衡量。该工作量由各种因素决定，例如软件设计方法（例如面向对象设计）和编码标准等，这种形式的可维护性测试可以通过分析和评审来完成。 可测试性涉及到对所作变更进行测试所需的工作量。稳定性则涉及到系统对于变更的反应。稳定性较低的系统，只要发生变更便会引发大量的负面冲击（俗称“连锁反应”）。[ISO9126] [www.testingstandards.co.uk]

5.3.5 *可移植性测试*

可移植性测试通常和软件移植到某个特定的运行环境中的难易程度相关，包括第一次建立或从现有环境上移植。可移植性包括可安装性、共存性/兼容性、适用性和可替换性。

5.3.5.1 可安装性测试

可安装性测试是针对那些用于在目标环境安装软件的安装程序所进行的测试。它可以包括安装操作系统的软件或在客户个人电脑上安装软件产品的安装向导软件。典型的可安装性测试应完成下列目标：

- 使用安装向导或遵照安装手册的步骤（包括执行必需的安装脚本），确认是否可以成功地进行软件安装。其中包括选择相应的选项针对不同的软硬件配置进行安装，以及进行不同程度地安装（如完全安装或部分安装）
- 测试安装软件是否能够正确处理安装过程中所出现的失败（例如无法安装某些 DLL）现象，而不致于使系统处于某个不确定的状态（如软件只安装了一部分或造成错误的系统配置）
- 测试部分（不完全的）安装/卸载能否完成
- 测试安装向导是否可以成功地识别无效的硬件平台或操作系统配置
- 衡量是否能够在一定时间内或在一定步骤内完成整个安装过程
- 确认是否可以成功地进行软件降级或卸载

通常，在执行了安装测试，检查是否存在安装方面的问题（如配置错误、功能缺失）之后，才进行功能测试。可用性测试一般和安装测试同时进行（例如，确认在安装过程中向用户了提供明确的指导、反馈或出错信息）。

5.3.5.2 共存

如果不存在相互依赖关系的计算机系统可以在同一环境（例如 同一个硬件平台）中运行，而不影响彼此的行为（如资源冲突），则称之为是兼容的。例如，当新的或升级的软件被大量装入已经安装了应用程序的环境（例如服务器）时，需要执行兼容性测试。

在一个没有安装其它应用程序的系统上，可能检测不出软件的兼容性问题，但如果将其部署到另一个安装了其它应用程序的环境（如产品环境），则可能会出现兼容性的问题。

典型的兼容性测试的目标包括：

- 评估在运行环境中加载其它应用程序所导致的功能上的负面影响（例如，当服务器上运行多个应用程序时的资源分配冲突）
- 评估因修复或升级操作系统给应用程序带来的影响

通常是在系统测试和用户验收测试完成后，再进行兼容性测试。

5.3.5.3 适应性测试

适应性测试就是测试一个应用程序是否能够在所有特定的目标环境（硬件、软件、中间件、操作系统等）中正确地运行。在进行针对适用性的测试时，需要明确各种指定的目标环境并完成配置，供测试部门使用。这些运行环境的测试通过选择一组调用环境中存在的各种组件的功能测试用例来完成。

适应性还涉及到通过完成一个预定过程将软件移植到各种特定运行环境的能力。测试可以对该过程进行评估。

适应性测试还可以与可安装性测试共同进行，然后辅以功能测试，以检验软件在其它运行环境中是否会出现问题。

5.3.5.4 可替换性测试

可替换性所关注的是系统中软件组件能够被替换的能力，尤其对于那些以商业软件（COTS）为特定组件的软件系统。

在集成过程中会有一些可替换的组件集成构成一个完整的系统，因而可替换性测试可以与功能集成测试并行进行。可以通过技术评审和审查进行系统的可替换性评估，其关键点在于可被替换组件的接口是否定义得非常清楚。

6. 评审

术语

审计，IEEE 1028，非正式评审，审查，审查组长，管理评审，主持人，评审，评审员，技术评审，走查。

6.1 简介

成功的评审过程需要合理计划、积极参与和有效跟踪。培训机构必须确保测试经理了解其在计划和跟踪活动方面所具有的职责。测试人员必须积极参与评审过程，提出其独特的看法。测试人员应该接受过正规的评审培训，以便更好的理解在任何技术评审过程中各自的角色。所有的评审参与者必须对管理良好的技术评审做出承诺。正确的开展评审，对于整体交付质量而言，是最简单、最有成效的方式。关于评审的国际标准请参见 IEEE 1028。

6.2 评审的原则

评审是静态测试的一种类型。评审的主要目标通常就是发现缺陷。评审员通过直接检查文档发现缺陷。

最基本的评审类型在《初级认证大纲》（2005 版）的 3.2 节有详细描述，在下面的 6.3 节也将列出。

相关的源文档（描述项目需求的文档）和标准（项目所必须遵循的标准）一旦就绪，最好尽早开展各种评审。如果缺少某一文件或者标准，则不能发现所有文档之间存在的缺陷和不一致，只能发现单个文档中的缺陷和不一致。待评审的文档必须事先提供给评审员，以使评审员能有充足的时间来熟悉文档内容。

所有类型的文档都可以进行评审，例如，源代码、需求规格说明、概念、测试计划及测试文档等等。动态测试通常会在源代码评审之后开展，它的目的是发现静态检查无法发现的缺陷。

评审可以导致三种可能的结果：

- 文档可以不需修改或者只需要较少修改即可使用
- 文档必须修改，但是不需要进一步的评审
- 文档必须进行大范围的修改，并且需要进一步评审

至于典型的正式评审涉及的所有人员的角色和职责，在《初级认证大纲》中都已覆盖，即经理、主持人或组长、作者、评审员和记录员。其他有可能参与评审的人员包括决策者或者利益干系人，以及客户或者用户代表。另外一个可选角色有时会在审查中用到，那就是读者，读者角色在会议中对工作产品的某些部分进行解释。除了评审职责，每个评审员可各分配一项基于缺陷的职责，以寻找特定类型的缺陷。单个产品可以采用不止一种评审类型。例如，一个小组可以进行一项技术评审，以决定在接下来的迭代中实施哪些功能。对于确定要包含的功能，也许需要对规格说明进行审查。

6.3 评审类型

《初级认证大纲》中介绍了以下评审类型：

- 非正式评审
- 走查
- 技术评审
- 审查

实际过程中的评审可能是这些评审类型的混合，例如运用规则集合的技术评审。

6.3.1 *管理评审和审计*

除了《初级认证大纲》中介绍的评审类型，IEEE1028 还叙述了以下评审类型：

- 管理评审
- 审计

管理评审的主要特征如下：

- 主要目的：监控进展、评估状态和为下一步行动做出决策
- 由或者为对项目或系统负有直接责任的经理执行
- 由或者为项目干系人或决策者执行，例如，一个更高级别的经理或总监
- 检查与计划的一致性、与计划之间的偏离性，或者管理规程的充分性
- 包含评估项目风险
- 结果包括行动项和待解决的问题
- 期望参与者进行各项准备工作，必须记录决策

说明：测试经理应该参与和发起测试过程的管理评审。

审计是非常正式的，通常用来证明符合某些期望，符合适用的标准或者合同条款。因此，审计在揭示缺陷方面是效率最低的。

审计的主要特征为：

- 主要目的：对过程、法律法规、标准等的符合性提供独立的评估
- 审计组长对审计活动负责，并且担任审计主持人的角色
- 审计员通过访谈、见证和检查文档方式，收集符合性的证据
- 结果包含观察项、建议、纠正措施和通过/不通过的评估。

6.3.2 *特殊工作产品的评审*

评审可以根据被评审的工作产品名称或者活动来进行描述，例如：

- 合同评审
- 需求评审
- 设计评审
  - 概要设计评审
  - 关键设计评审
- 验收评审/资质评审
- 运行预备评审

合同评审可能与合同里程碑关联，通常作为安全关键或安全相关系统的管理评审。它涉及的角色包括经理、客户和技术人员。

需求评审可以是走查、技术评审或者审查，可以考虑安全相关和依赖性需求，也可以是功能和非功能需求。需求评审可以包括验收准则和测试条件。

设计评审是典型的技术评审或审查，它涉及的角色包括技术人员和客户或项目干系人。概要设计评审提出对某些技术设计和测试的初步方法；关键设计评审覆盖了所有被提议的设计解决方案，包括测试用例和测试规程。

验收评审是用来获得管理层对系统的批准。这也可以归为资质评审，它是一种常见的管理评审或审计。

6.3.3 *正式评审的开展*

《初级认证大纲》描述了正式评审的 6 个阶段：计划、启动会议、个人准备、评审会议、修改和跟踪。被评审的工作产品应当与资格或评审员相适应。例如，测试计划对应测试经理，商业需求或测试设计对应测试分析员，功能规格说明、测试用例或测试脚本对应测试技术分析员。

6.4 评审的引入

为了能够将评审成功引入一个组织，需要采用以下的步骤（顺序不一定如此）：

- 获得管理层支持
- 让各位经理了解成本、利益和执行方面的问题
- 评审步骤、形式和基础结构（例如，评审度量数据库）的选择并文档化
- 对评审技术和规程进行培训
- 获得参与评审工作的人员和评审对象作者的支持
- 执行试点评审
- 通过节约成本证明评审的好处
- 将评审用于最重要的文档。例如，需求、合同、计划等

对降低或避免修复缺陷的成本或者这些缺陷将导致的后果进行度量，可用来评估评审引进是否成功。也可以用因早期发现和修改缺陷少花时间而节约的成本来度量。

评审过程应随着时间持续的监控和改进。经理必须意识到，学习新的评审技术是一项投资，其收益不是立即可见的，但随着时间的推移会越来越明显地显示出来。

6.5 评审的成功因素

有许多因素可以帮助开展成功的评审。实施评审并不难，但是如果未充分考虑以下这些因素，评审可能会以各种方式走入歧途。

技术因素

- 确保正确执行针对评审类型所定义的过程，特别是对于比较正式的评审类型，如审查
- 记录评审所耗费的成本（包括所耗费的时间）和所得到的收益
- 评审早期的草稿或者部分文档，以提前识别出各种缺陷类型，防止后来它们被引入整个文档
- 在启动一个评审过程之前，确保文档或部分文档已为评审准备就绪（即应用入口准则）
- 运用组织特有的常见缺陷检查表
- 根据不同的目标，例如：文档清理、技术改进、信息转移或进度管理，运用不止一种评审类型
- 需要根据文档内容作出重大决策的文档应该进行评审或审查。例如：在管理评审授权项目重大开支之前，需要认真审查方案、合同或概要需求
- 为了评估而不是清理，以抽样调查某一限定的文件子集
- 鼓励发现最重要的缺陷，注重内容而非形式
- 持续改进评审过程

组织因素

- 即使在最后期限的压力下，经理也应确保花费足够的时间用于评审活动
- 切记评审中耗费的时间和预算并非与发现的缺陷成比例
- 对于在评审中发现的缺陷，要给予足够的时间进行修改
- 永远不要用评审中的度量数据进行个人绩效评估
- 对于不同类型的评审，要确保能有合适的评审员参与
- 为评审提供培训，特别是较为正式的评审类型
- 成立评审负责人论坛来相互分享经验和想法
- 确保人人参与评审，并且每个人自己的文档内容都得到了评审
- 将最强大的评审技术用于最为重要的文档
- 确保建立的评审团队，具有良好的平衡性，由不同技术和背景的人员组成
- 支持过程改进的活动必须能支持解决系统问题
- 对通过评审过程所取得的改进表示认可

人员问题

- 对项目干系人进行教育，使他们明白评审会发现缺陷，并且对于修改和再评审要给予时间
- 要确保评审对于作者来说是一次正面的、积极的经历
- 在一种“无责备”的氛围中，乐于接受缺陷的识别
- 要确保评审意见具有建设性、有益性和客观性，而非主观性
- 作者不同意或者不愿意的情况下，不进行评审
- 鼓励大家对待评审文档的最重要的方面，进行深层次思考

有关评审和审查方面的更多资料，请参见[Gilb93]和[Weigers02]。

7. 事件管理

术语

IEEE 829，IEEE 1044，IEEE 1044.1，异常，配置控制委员会，缺陷，错误，失效，事件，事件日志，优先级，根本原因分析，严重性

7.1 简介

测试经理、测试人员必须熟悉缺陷管理过程。测试经理关注过程，包括识别方法、缺陷追踪和缺陷移除。测试人员主要着眼于准确记录测试领域发现的事件。在缺陷生命周期的每一步，测试分析员和测试技术分析员都有不同的目标。测试分析员根据商业和用户需求评估系统行为，例如用户是否知道在面对某种特定信息或行为时应如何操作。测试技术分析员将着重评估软件自身的行为和对问题的技术研究，如在不同的平台和内存配置下测试同一个失效。

7.2 何时可以发现一个缺陷？

事件是意外发生并需要进一步调查的问题。事件是对由缺陷导致的失效的识别。事件可能会记录为缺陷报告也可能不记录。缺陷是已经确认的需要通过修改工作项来解决的实际问题。

缺陷可以通过静态测试发现。而失效只能通过动态测试发现。软件生命周期中的每一阶段都应该提供用于发现和排除潜在缺陷的方法。例如，在开发阶段，代码和设计的评审应该用于发现缺陷；在动态测试阶段，执行测试用例以发现失效。从总体上来说，越早发现并修正缺陷，缺陷造成的系统质量成本越小。应该记住的一点是，缺陷既可能存在于测试件中也可能存在于测试对象中。

7.3 缺陷生命周期

所有的缺陷都有生命周期，尽管某些缺陷会跳过某个阶段。缺陷的生命周期（按照 IEEE 1044-1993 的描述）由以下 4 个步骤组成：

- 第 1 步：识别
- 第 2 步：调查
- 第 3 步：改正
- 第 4 步：总结

在每步骤中都包含三个用于获取信息的活动：

- 记录
- 分类
- 确定影响

7.3.1 第 1 步：识别

识别步骤发生在当潜在的缺陷（事件）被发现时。可以在软件生命周期的任何一个阶段发生。在发现缺陷的时候，应记录识别缺陷的数据项。包括如下信息：缺陷发生的环境、发现人、描述信息、时间和负责人（如果可以提供）。识别可以按照识别潜在缺陷的某些属性进行分类，包括项目活动、项目阶段、潜在的原因、可重现、征兆和异常造成时的产品状态。这些信息可以对可能造成的严重性划分等级，确定对项目时间表和项目成本产生的影响。

7.3.2 第2步：调查

在进行了识别后，需要对每个潜在的缺陷进行调查。这步主要用于发现与缺陷有关的事项和解决方案建议，也可能是没有任何行动（例如，潜在的缺陷已经不是事实上的缺陷）。在本步骤，将记录附加的信息，并对前一步骤中提供的分类、影响信息进行再评估。

7.3.3 第3步：改正

根据前面的调查结果，就可以开始改正步骤。改正包括那些解决缺陷的需要的行动，还有流程修订/改进的行为以及为避免类似缺陷引入将来项目中所采取的策略。对每个变更都必须执行回归测试、再测试，以及进展测试。在本步骤，将记录附加的信息，并对前一步骤中提供的分类、影响信息进行再评估。

7.3.4 第4步：总结

在总结阶段，所有附加的数据项都需要记录，总结的分类包括关闭、延期、合并或归于另一项目。

7.4 缺陷要素

IEEE 1044-1993 描述了一组适用于缺陷生命周期的强制要素。另外也定义了大量的非强制性要素。根据 IEEE 1044.1 在执行 IEEE 1044-1993 时，可以将 IEEE 标准中定义的要素和公司中使用的要素之间进行映射。允许公司有自己的命名规格，无需严格保持与 IEEE 1044-1993 命名规则一致。IEEE 允许多个公司和组织之间的缺陷信息的差异性。

不论是否将与 IEEE 保持一致作为目标，针对缺陷提供的要素应可以提供足够的信息，使得缺陷是可处理的。可处理的缺陷报告具有以下特性：

- 完整性
- 简明性
- 准确性
- 客观性

除了解决特定的缺陷之外，缺陷报告还应该为精确分类、风险分析和过程改进提供信息。

7.5 度量和事件管理

缺陷信息中需要包含足够的信息用来帮助测试过程监控、缺陷密度分析、发现和修正度量和缺陷的收敛（打开和关闭）。另外，缺陷信息应该用于过程改进，将缺陷阶段信息、根本原因分析、缺陷趋势分析作为风险减轻调整策略的输入。

7.6 事件沟通

有效的沟通可以在事件管理中避免相互指责，支持收集和解释目标信息。事件报告的准确性、合适的分类和客观的表述有助于保持缺陷报告者和缺陷修复人员之间的专业关系。测试人员应考虑到缺陷的相对重要性并提供可用的客观信息。

缺陷会诊会议应致力于对缺陷进行适当的优先级判定。缺陷的跟踪工具不能替代有效的沟通，缺陷会诊会议也不能作为缺陷跟踪工具的替代。有效的沟通和良好的工具支持有助于实现高效的缺陷跟踪过程。

8. 标准和测试改进过程

术语

能力成熟度模型（CMM），集成能力成熟度模型（CMMI），测试成熟度模型（TMM），集成测试成熟度模型（TMMI），测试过程改进 （TPI）

8.1 简介

测试过程建立和改进的来源很多。本章首先把标准作为测试相关主题的一个有用（有时是必须）信息来源。测试经理和测试人员的学习目标是，知道有哪些可用的标准及其各自的适用范围。培训机构需要特别强调与学员正在学习的模块相关的特定标准。

测试过程建立之后，需要不断改进。在 8.3 节中，首先描述改进的一般性问题，然后介绍了一系列的可用于测试过程改进的特定模型。测试经理需要理解此节的所有材料，同时，作为改进活动当中的重要参与者，测试分析员和测试技术分析员也需要理解用于改进活动的相关模型。

8.2 标准的考量

这里以及初级大纲中涉及了一些标准。许多标准涉及与软件相关的多个主题，例如：

- 软件开发生命周期
- 软件测试与测试方法论
- 软件配置管理
- 软件维护
- 质量保证
- 项目管理
- 需求
- 软件语言
- 软件接口
- 缺陷管理

培训大纲无意列出或者推荐特定的标准。测试人员应该了解标准是如何创建的以及在工作环境当中如何应用。

标准有不同的来源：

- 国际性的或者具有国际性目标的标准
- 国家性的标准，如国际标准在某个国家中的实际运用
- 特定领域的标准，例如适应特定领域的国际或者国家标准，或为特定领域而开发的国际或者国家标准。

本章接下来将讨论使用标准时的注意事项。

8.2.1 标准概论

8.2.1.1 标准的来源

标准是由专业人士建立的，并反映了集体的智慧。国际标准有两个主要来源：IEEE 和 ISO。国家以及特定领域的标准同样重要和有用。

对于测试人员来说，他们需要了解在工作内容和环境中所应用的标准，是正式标准（国际，国家或者特定领域的标准）还是内部标准和推荐实践方法。随着标准的演进和修改，需要指出标准的版本号（或者发布日期）以保证其一致性。在标准的参考文献使用中，没有指明使用文献的版本号以及发布日期的，即认为是参考最新的版本。

8.2.1.2 标准的可用性

标准可以作为一种工具来促进建设性的质量保证。所谓建设性的质量保证的目标是减少引入的缺陷而非只是在测试中发现缺陷（分析性的质量保证）。并不是所有的标准都适用于所有的项目，标准中的信息可能对项目有益，也可能对项目造成干扰。教条式地遵循标准不能帮助测试人员在工作产品中找出更多的缺陷[Kaner02]。然而，标准可以提供一些参考性的框架，以及测试解决方案的依据。

8.2.1.3 一致性和冲突

有些标准和其它标准之间缺少一致性，甚至在定义方面存在冲突。这时标准的使用者可根据其使用环境和背景决定不同标准的可用性。

8.2.2 国际标准

8.2.2.1 ISO

ISO [www.iso.org]是“国际标准组织”（最近改为“国际标准化组织”）的缩写。国际标准化组织是由各国标准化团体（ISO 成员团体）组成的世界性的联合会。该国际组织为软件测试人员提供了许多有用的标准，例如：

ISO 9126:1998，现已分成以下的标准和技术报告（TR）

- ISO/IEC 9126-1:2001 软件工程—产品质量—第 1 部分：质量模型
- ISO/IEC TR 9126-2:2003 软件工程—产品质量—第 2 部分：外部度量
- ISO/IEC TR 9126-2:2003 软件工程—产品质量—第 3 部分：内部度量
- ISO/IEC TR 9126-2:2004 软件工程—产品质量—第 4 部分：质量度量

- ISO 12207:1995/Amd 2:2004 系统和软件工程—软件生命周期过程
- ISO/IEC 15504[1]-2:2003 信息技术—过程评估—第 2 部分：执行评估

以上所列并没有涵盖所有可供使用的标准，根据工作内容和环境，其它 ISO 标准也有可能适用。

8.2.2.2 IEEE

IEEE [www.ieee.org]是“电气与电子工程师协会”的简称，她是被一百多个国家承认的美国专业组织。IEEE 为测试人员建议了许多有用的标准，例如：

- IEEE 610:1991 IEEE 标准计算机词典，一本汇集了 IEEE 标准计算机词汇的专著
- IEEE 829:1998 IEEE 软件测试文档标准
- IEEE 1028:1997 IEEE 软件评审标准
- IEEE 1044:1995 IEEE 关于对软件异常进行分类的指引

以上所列并没有涵盖所有可供使用的标准，根据工作内容和环境，其它 IEEE 标准也有可能适用 。

8.2.3 国家标准

许多国家本身设有自己的特定标准，其中的某些标准可直接应用于软件测试，有些对软件测试有用。例如英国的 BS-7925-2:1998“Software testing, Software component testing（软件测试，软件组件测试）”提供了本大纲所涉及的许多关于测试技术的信息，体现在以下方面：

[1] ISO 15504 也称为 SPICE，来源于 SPICE 项目

- 等价类划分
- 边界值分析
- 状态转换测试
- 因果图
- 语句测试
- 分支/判定测试
- 条件测试
- 随机测试

BS-7925-2 同时为组件测试提供了过程描述。

8.2.4 特定领域的标准

各种技术领域当中同样存在着标准。一些行业根据其特定的技术领域背景，对其它标准进行了裁剪。这里提及的内容，仍然集中在软件测试、软件质量和软件开发。

8.2.4.1 航空电子系统

RTCA DO-178B/ED 12B “Software Considerations in Airborne Systems and Equipment Certification（航空系统和设备认证的软件注意事项）”适用于民用航空器中所使用的软件。这个标准同样适用于开发（或者验证）民用航空器软件时所使用的软件。对于电子航空软件，美国联邦航空管理局（FAA）和国际联合航空局根据软件的关键程度指定其接受测试的结构化的覆盖要求。

关键程度等级	潜在影响	所需的结构覆盖
A 灾难性的	阻碍安全飞行和着陆	条件判断，分支，语句
B 危险的/严重的	安全边界或性能急剧下降 机组人员不能正确而全面地完成任务 小数乘客受到严重或致命伤害	判定和语句
C 主要的	安全边界的显著下降 机组人员的工作量显著增加 乘客不舒适可以包括伤害	语句
D 轻微的	飞机安全边界 能力的轻微降低 机组人员的工作量轻微增加 乘客的某些不方便	无
E 无影响	对飞机的性能没有影响 没有增加机组人员的工作量	无

对于待认证的民用航空器上的软件，根据其应用领域的关键程度不同， 必须达到各自相应的结构覆盖等级。

8.2.4.2 航天工业

有一些工业领域会对其它的标准进行裁剪以适合他们特定的领域。其中的一个例子是航天工业中的 ECSS（欧洲空间标准化合作组织）[www.ecss.nl]。根据软件的关键等级，ECSS 推荐了一系列的方法和技术，这些内容与 ISTQB®初级和高级大纲一致，包括：

- SFMECA - 软件故障模式、影响以及严重度分析
- SFTA - 系统故障树分析
- HSIA - 软硬件相互分析

- SCCFA - 软件常见失效分析

8.2.4.3 食品与药物管理局

对于面向联邦法规法典第 820 章 21 条规定的医药系统，美国食品及药物管理局（FDA）推荐了特定的结构与功能测试技术。
FDA 同样也推荐了与 ISTQB®初级和高级大纲内容一致的测试方针和原则。

8.2.5 *其它标准*

有很多适用于不同产业领域的标准。其中有些是针对特定的产业或者领域的修改，也有一些适用于特定任务或者为如何使用标准提供解释。

测试人员需要知道应用于自己的专业领域，产业和工作环境的不同标准（包括内部标准、最佳实践等等）。有时适用的标准依据特定合同的不同层次的适用性确定。这时测试经理必须知道应该遵循的标准，并且保证维持适当的一致性。

8.3 测试改进过程

正如测试是用于改进软件，软件质量过程是用于改进软件开发过程（以及软件交付）的。过程改进同样适用于测试过程。有很多方式和方法可以用于改进软件测试和含有软件的系统的测试。这些方法通过提供指导方针和改进的方面实现针对过程的持续改进（并延伸到交付）。

测试成本经常在项目总成本中占据主要部分。然而，在众多的软件过程改进模型当中只有极少部分关注于测试过程，例如 CMMI（请参见下文）。

测试改进模型，如测试成熟度模型（TMM）、系统化测试和评估过程 （STEP）、关键测试过程（CTP）和测试过程改进（TPI）。这些模型的开发，弥补了这方面的空缺。TPI 和 TMM 提供了跨组织级别的度量并作为“基准”进行参照比较。至今，产业上有许多改进模型可供选择。除了本节提及的模型之外还可以考虑测试组织成熟度（TOM）、测试改进模型（TIM）以及软件质量等级（SQR）。现在，也有许多局部的模型正在使用。测试的专业人员需要对所有能用的模型进行研究以决定最适合他们情况的选择。

本节内容所提及到的模型只是说明模型是如何工作的以及模型包含哪些内容，而不是推荐它们。

8.3.1 过程改进简介

过程改进与软件开发过程和测试过程有着密切的联系。从个人犯过的错误中吸取的教训，有助于改进整个组织开发和测试软件的过程。戴明改进循环（Deming improvement cycle）：计划、实施、检查、行动（Plan、Do、Check、Act，PDCA），已经使用了多年，直到现在，测试人员在过程改进当中仍在使用。

过程改进的一个重要前提就是要相信系统质量在很大程度上受开发该软件的过程质量所影响。软件工业上质量的提升，减少了软件维护对资源的要求，由此提供了更多的时间在未来创造更多更好的解决方案。

通过使用过程模型对组织的成熟过程进行度量，过程模型提供了改进的起点。同时根据评价的结果，模型同样为组织过程的改进提供了框架。

过程评估明确了过程能力，并推动过程改进。同样，过程改进的效果需要通过过程评估进行度量。

8.3.2 过程改进类型

有两种模型：过程参照模型和内容参照模型

1. 过程参考模型是在进行评估时所使用的指导框架，通过与模型比较评估组织的能力，并在此框架内评估组织。
2. 内容参考模型用于在评估完成之后进行过程改进。

有些模型同时涵盖了上述的两个方面。

8.4 改进测试过程

为了达到更高的成熟度和更专业的水平，IT 业已经开始参照测试过程改进模型开展工作。行业标准模型正在帮助开发跨组织的度量与分析（可用于比较）。阶段式模型，如 TMMI 和 CMMI 提供的标准对不同的公司和组织进行比较。持续性模型允许组织拥有更多的自由度，根据组织确定的优先级实施过程改进。测试业除了过程的改进需要，还有几个标准可以应用，包括 STEP、、TPI 和 CTP。这些将会在以下各节进行讨论。

上述的四项测试过程评价模型允许组织根据其当前的测试过程，确定自身的能力程度。进行评估时，TMMI 和 TPI 为测试过程的改进提供了明确的路线。STEP 和 CTP 则是向组织提供方法以判定哪些过程改进能带来最大的回报，并让组织选择合适的路径。.

一旦决定过程需要评审和改进，所要遵循的步骤是：

- 启动
- 度量
- 设定优先级和计划
- 定义和重新定义
- 运作
- 确认
- 进化

启动

在这项活动中，在过程改进的项目干系人、对象、目标、范围，流程改进的覆盖范围等方面要得到确认并取得一致。在此活动中，还要选择改进所要用到的过程模型。可以在已发布的模型中选择合适的模型或选择自定义的模型。

在过程改进活动开始之前，需要定义出过程改进成功的标准和在整个过程改进中进行衡量的方法。

度量

进行一致的度量过程，最终得到一个列表，该列表包括所有可能的过程改进点。

优先级与计划

将度量过程中得到的列表进行优先级排列。该优先级排列可能参照的依据有投资回报率、风险、与组织战略的一致程度、可测量的定量或者定性收益等。

设定好优先顺序之后，需要编写并且实施改进计划。

定义与重新定义

基于识别出的过程改进需求，如对所需的新过程进行定义，需要改进的现有过程进行重新定义并为实施做好准备。

运作

过程改进制定成熟后便可开始实施。过程改进包括需要的任何培训、指导、先导性实验和最终的全面执行。

确认

在完全执行了过程改进之后，对改进前协商确定的收益进行确认是非常关键的，比如，实现收益目标。同样重要的是，需要确认过程改进活动是否满足了所有的过程改进成功标准。

进化

依赖于使用的过程模型，这个阶段是下个成熟度的起始点，同时也是在这个阶段决定是否重新开始过程改进还是就此停止。

使用评估模型是一种通用方法，该方法确保通过标准化的途径，利用行之有效的实践方法进行测试流程改进。然而，过程改进也可以不使用模型，例如，可以使用分析法和回顾会议。

8.5 使用 TMM 改进测试过程

测试成熟度模型（TMM）包含 5 个等级，并延伸补充 CMM。每一个等级都包括已定义的过程域，组织在升级到更高一个等级之前，需要完全满足前一个等级的过程域。TMM 既提供过程参考模型也提供内容参考模型。

TMM 的等级分别是：

等级 1：初始级

初始级表示测试过程没有正式的文档化和结构化。测试在编码后以特别的方式执行，与调试没有区别。测试的目的被理解为证明软件正常工作。

等级 2：定义级

要达到等级 2，组织已设定测试方针和目标，并引入了测试计划过程，同时具有基本的测试技术和方法。

等级 3：集成

等级 3 要求组织的测试过程和软件开发周期集成在一起，并且在正式标准，过程和方法中进行了文档化。组织有清晰的软件测试功能，并有相应的监督和控制措施。

等级 4：管理&度量

达成等级 4 的条件是测试过程可有效地度量、管理并能适用于特定项目。

等级 5：优化

最后的等级表示组织成熟度达到如下状态：测试过程的数据可以用于防止错误的发生，而且注意力集中在优化已建立的过程。

要达到特定的等级需要实现一系列的预先定义好的成熟度目标和附属目标。这些目标根据活动、任务和责任等进行定义，并依据管理者、开发/测试人员和客户/用户的特殊“考虑”来评估。关于 TMM 的更多信息，请参见[Burnstein03].

TMMI Foundation（请参见www.tmmifoundation.org）定义了 TMM 的后续标准：TMMI。TMMI 是基于由伊利诺斯州技术研究院开发的 TMM 框架延伸的针对测试过程改进更细节化的模型。TMMI 是使用 TMM 的实践经验总结，同时 TMMI 定位在作为 CMMI 的一个补充。

TMMI 的结构主要基于 CMMI 的结构（过程领域、通用目标、通用实践、特定目标、特定实践。）

8.6 使用 TPI 改进测试过程

TPI 模型（测试过程改进）用连续的方式表述，而 TMM 模型用不连续的阶段来表述。

测试过程优化计划包括了一系列的关键域。这些关键域建立在生命周期的 4 大支柱（组织、基础设施、工具及技术）之内。对于关键域的评估，级别是由 A 到 D， A 是最低级。如果关键领域非常不成熟，也有可能达不到初始 A 级的要求。有些关键领域可能只能评为 A 或者 B（如估算和策划），而其它（如度量与分析）的评级范围可以是 A、B、C 或者 D。

对于给定的关键领域所处的级别，可通过对 TPI 模型所定义的检查点进行测量获得。例如，当关键领域所要求的检查点都满足了 A 级和 B 级的要求，那么这个关键域就达到了 B 级。

TPI 模型定义了众多过程和等级之间的依赖关系。这些依赖关系保证了测试过程的均衡发展。例如，如果关键域“报告”没有相应地达到 A 级，那么关键领域“度量与分析”也就不可能达到 A 级（即如果没有测试报告，度量与分析便没有意义）。对于依赖关系的使用，在 TPI 模型当中是可选的。

TPI 主要是一个过程参考模型。

测试成熟度矩阵提供了每个关键域的各个等级（A/B/C/D）和总体测试过程成熟度等级的映射关系。这些总体等级包括：

- 受控
- 高效
- 优化

在 TPI 的评估过程当中，将会使用定量的度量和定性的访谈以建立测试过程成熟度等级。

8.7 使用 CTP 改进测试过程（CTP）

如 BLACK02 所述，使用关键测试过程（CTP）评估模型的基本前提是存在某个特定的关键测试过程。这些关键过程，如果实施良好，将会造就成功的测试小组。相反，如果这些活动实施的程度不够，即使是极有天赋的测试人员和测试经理也不会获得成功。这个模型识别了 12 个关键测试过程。

CTP 主要是一个内容参考模型。

关键测试过程模型是一个允许对模型根据应用场景进行裁剪的改进途径，包括：

- 特定挑战的识别
- 良好过程特点的识别
- 过程改进实施顺序和重要性的选择

核心测试过程模型可以适用于所有软件开发生命周期模型。

使用 CTP 的过程改进，始于对现有测试过程的评估。评估识别了哪些过程强，哪些过程弱，并根据组织的需要提供具有优先度的建议。不同特定背景下的评估不同，但一般而言，CTP 评估将对下列定量的度量数据进行检查：

- 缺陷发现率
- 投资回报率
- 需求覆盖率和风险覆盖率
- 测试发布管理费用
- 缺陷报告拒绝率

除了上述的定量数据以外，CTP 评定过程中通常还会对下面的质量因素进行定性的评估：

- 测试小组角色和效率
- 测试计划的效用

- 测试小组测试水平，领域知识和技术
- 缺陷报告的效用
- 测试结果报告的效用
- 变更管理的效用和平衡

评估过程中识别出薄弱过程域后，需要开始制定改进计划。模型为每个核心测试过程提供了通用改进计划，但评估小组需要对它们进行合适的裁剪。

8.8 使用 STEP 改进测试过程

STEP（系统化测试和评估过程），与 CTP 比较类似，而不像 TMMI 和 TPI，并不要求改进需要遵循特定的顺序。

STEP 方法的基本前提包括：

- 基于需求的测试策略
- 从生命周期初期开始测试
- 测试用作需求和用例模型
- 由测试件设计导出软件设计（测试驱动开发）
- 及早发现缺陷或完全的缺陷预防
- 对缺陷进行系统分析
- 测试人员和开发人员一起工作

STEP 根本上是一个内容参考模型。

STEP 方法是基于如下的思想：测试是一个生命周期活动，在需求正规化时开始直到系统退役。STEP 方法强调“先测试后编码”，而这种方针的实现途径是使用基于需求的测试策略以保证在设计和编码之前，已经设计了测试用例以验证需求规格说明。该方法识别并关注改进测试中的三个主要阶段：

- 计划
- 产物
- 度量

STEP 的评估过程中，使用定量的度量和定性的访谈。定量的度量和分析包括：

- 不同时期的测试状态
- 测试需求和风险覆盖
- 缺陷趋势，包括发现、等级和分类分项数据
- 缺陷密度
- 缺陷移除效率
- 缺陷发现率
- 缺陷引进、发现和移除等阶段
- 测试成本，包括时间、工作量和资金

量化的因子包括：

- 已定义的测试过程的效用
- 客户满意度

某些情况下 STEP 评估模型可以与 TPI 成熟度模型结合起来使用。

8.9 能力成熟度模型集成（CMMI）

CMMI 可以通过两种方法或方式实施：阶段性方式或者连续性方式。在阶段性方式中，有 5 个“成熟度等级”，每一个等级都是建立在前一个等级的过程域基础之上。在连续性方式中，允许组织专注于其优先改进的过程域而无需关注原有的等级。

阶段性方式主要包含在 CMMI 中，以确保与 CMM 保持共性，而一般认为，相比之下，连续性方式更加灵活。

在 CMMI 模型中，确认和验证的过程域参考静态和动态的测试过程。

9. 测试工具与自动化

术语

调试工具，动态分析工具，仿真器，故障传播工具，超链接测试工具，关键词驱动测试，性能测试工具，模拟器，静态分析工具，测试执行工具，测试管理工具，测试准则

9.1 简介

本节将对初级认证大纲进行详细说明，首先介绍一般性概念，接下来会详细描述一些特定的工具。

即使有些内容会更贴近测试经理、测试分析员或测试技术分析员，但对于专业测试人员来说，必须对于概念具有基本理解。有关基本理解方面的内容将在稍后适当的地方加以讨论。

工具可以用不同的方法进行归类，其中包括根据它们的实际使用者，如测试经理、测试分析员或测试技术分析员来分类。这种分类反映了本大纲的模块结构，并将在本章接下来的内容中使用。一般而言，这部分内容讨论的工具主要和某些特定的模型相关，也不排除某些工具（如测试管理工具）与多个模型相关。在这种情况下，培训机构将给出在特定环境下应用工具的例子。

9.2 测试工具的概念

测试工具可以极大程度地提升测试工作的效率和准确度，前提是要用正确的方法使用正确的工具。
测试工具要作为一个独立的方面，在一个良好运作的测试组织当中进行管理。通常认为测试自动化是测试执行的同义词，但大多数手动过程是由不同的测试自动化活动构成的，这意味着在使用正确工具的前提下，大多数的测试领域都能一定程度的实现自动化。

任何测试工具的版本、测试脚本或测试场景，在作为测试依据时，都应该处于配置管理之下并与被测试软件的特定版本绑定。任何测试工具都是测试件中重要组成部分并得到相应的管理，例如：

- 在创建测试工具之前建立架构
- 保证脚本与工具版本、补丁、版本信息等等处于配置管理之下
- 创建和维护库（测试用例中相同概念或内容的复用）、把测试工具执行情况文档化（如组织中如何使用工具的过程）
- 为后续开发实现结构化的测试用例，例如，使它们具有可扩展性和维护性。

9.2.1 *测试工具和自动化当中的成本收益和风险*

应该经常进行成本与收益分析，以确保投资能得到良好的回报。成本与收益分析报告的主要特征是围绕着以下成本项，对不使用工具的工作量所带来的成本和使用工具时的成本（时间转化为成本、直接成本、经常性成本和非经常性成本）进行对比：

- 启动成本
  - 知识收集（工具学习曲线）
  - 适时的评估工作（工具比较）
  - 与其它工具的集成工作
  - 对于初始成本当中，工具的购买、改编或者开发的补偿
- 经常性成本
  - 工具持有的成本（包括维护、许可费用、支持费用、持续的知识水平提升）

- 可移植性
- 可用性和依赖性（如果缺乏这方面特性）
- 持续成本评估
- 质量改进，以保证对所选工具的最优使用

仅仅基于试验性自动化项目的商业实例往往会遗漏一些重要的成本，例如系统变更时维护、升级和完善测试脚本的成本。测试用例的持续性是指它在需要重新编写前的有效使用时间。执行自动化测试脚本的首个版本所需的时间，通常远远高于手工执行测试的时间，但前者可能更快、更方便地开发出更多的相似的测试脚本。而且随着时间的推移，能极大地增加优秀测试用例的数量。另外，在实施之后，未来的自动化可以显著地改进测试覆盖率和测试效率。工具的应用必须基于长期的商业实例。
在特定的层次上，每一个测试用例都必须仔细考虑是否适合自动化。许多自动化项目的实施都是基于现有的手工测试用例，而没有评审每一个特殊用例在自动化过程中所带来的实际收益。而任何一个测试用例的组合（测试套件）往往包含着手动的、半自动的和全自动的测试。

除了在初级认证大纲中涵盖的主题之外，也需要考虑以下方面：

其它收益：

- 自动测试执行时间变得更易于预测
- 由于测试用例是自动的，所以在项目后期的回归测试和缺陷确认测试将更加快捷和安全
- 对于自动工具的使用，可以提升测试人员和测试小组的技能水平
- 在并行、迭代和增量开发中，自动化可以为每个软件包提供更好的回归测试
- 能覆盖一些手工无法进行的测试类型（例如性能和可靠性）

额外风险：

- 不完备的或不正确的手工测试被自动化了（自动化测试只是机械地转化而隐含着风险），
- 当被测试软件改变时，测试件（如脚本）难以维护，需要多次修改
- 由于只有自动的、程序化的测试在运行，测试活动中缺少测试人员的直接参与，这将会降低缺陷发现率。

9.2.2 测试工具策略

测试工具通常不只运用在一个项目上。依赖于项目的投资和周期，在单个项目或者版本中它也许不能对投资给予足够的回报，但在软件后继版本开发中会得到足够的回报。例如，在维护阶段有大量的测试工作（对于每一个缺陷修正都需要执行一系列可观的回归测试），只要处在维护阶段的系统生命周期足够长，测试工具就能在其自动化的过程中带来超过成本的收益。另外的一个例子是，在手动测试中，非常容易产生人为失误（如输入错误），因此大量数据的自动输入、输出数据与期望数据的自动比较（即测试结果和预期结果的比较）都体现了自动化的优势。

对于使用和依赖许多不同测试工具（针对不同阶段和目的）的公司来说，设立长期的测试工具策略以帮助对不同工具的版本和工具支持的引入和淘汰等做出正确的决策是最为明智的。在大型公司的集中使用工具的领域，建议为工具使用者、策略、工具范例或者脚本语言提供规范。

9.2.3 工具间的集成与信息交换

通常而言，在测试（开发）过程中，都会使用多个测试工具。举一个例子，公司会同时使用静态分析工具、测试管理和报告工具、配置管理工具、事件管理工具和测试执行工具。这种情况之下，判断工具彼此之间是否能以有利的方式实现集成和信息交换，则显得非常重要。例如，测试管理系统直接获得所有测试执行状态，从而可提供测试进展的即时更新，并能直接跟踪特定测试用例的需求。如果在测试管理数据库和配置管理系统中都存贮测试脚本，将会带来更多的工作量且更容易产生错误。如果测试人员希望在测试用例执行过程中发送事件报告，就需要集成缺陷跟踪系统和测试管理系统。如果作为独立的静态分析工具能直接向测试管理系统报告事件、警告和反馈，一定会带来许多便利。

向同一个商家购买成套的测试工具不意味着它们就能集成运行，不过这是个合理的要求。如果公司有足够的时间，需要全面评估这些方面的问题，包括相对于纯手工处理时信息遭篡改或者丢失的风险而言，自动信息交换的成本有多大。

某些新概念，如 Eclipse 那样的集成开发环境设法通过提供开发和测试工具的公共接口，使处于同一环境下的工具的集成和使用变得更为容易。工具供应商可以通过创建基于 Eclipse 框架的插件成为 Eclipse 的一部分，让它看起来和使用时感觉与其它工具一样。这同时也为用户创造了有利的条件。说明：这不意味着用户使用相似的接口，工具之间就自动提供了集成和信息交换。

9.2.4 *自动化语言：脚本和脚本语言*

有些情况下，使用脚本和脚本语言能更好地实现和扩展测试条件和测试用例。例如，当对一个 Web 应用进行测试时，借助脚本可以绕过用户界面而对 API（应用程序接口）进行充分的测试。另外一个例子就是，针对用户界面的自动化测试可以完成所有可能的输入组合，而这是手动测试不可能做到的。.

脚本语言的能力差异非常大。需要注意的是，脚本语言可以从一般的程序设计语言变到特定的标准符号。例如信令协议中的 TTCN-3.

9.2.5 *测试准则的概念*

测试准则（Test oracles）一般用于确定期望的结果。它们可以执行被测试软件相同的功能，所以这样的测试准则并不常见。在旧的系统被相同功能的新系统取代时，旧的系统就可以作为准则使用。在交付系统的功能出现问题时，同样可以使用准则。对需要交付的高性能软件进行功能测试时候，可以建立或者使用低性能基准以得出期望的结果。

9.2.6 *测试工具部署*

每一个自动化工具都是有版权的软件，并可能依赖于某些硬件或软件。无论工具是购买的、经过修改的，还是自己开发的，都需要文档化并经过测试。有些工具需要和环境集成在一起，而有的工具则是单独使用比较好。

被测试系统无论在个人硬件、操作系统、嵌入式软件还是在非标准的配置中运行，都有必要开发或修改一种工具用于测试以适应特定的环境。此时推荐进行成本收益分析，涵盖初始实施和长期维护。

在自动化测试工具的部署中，直接把手动测试用例自动化并不是明智之举，应该重构测试用例以更好地为自动化服务。包括格式化测试用例、考虑复用模式、使用变量代替硬编码值以扩展输入和充分利用测试工具的优势。而测试工具的优势在于能够跳过（Traverse）、重复或者改变指令并提供更好的分析和报告机制。对许多自动化测试工具来说，必须具有可编程能力，以用于开发有效的测试程序（脚本）和测试套件。通常，如果大型的测试套件没有很好地设计，将难以升级和管理。针对测试工具的编程和设计技术的培训，有助于充分发挥测试工具的作用。

即使在手动测试用例被自动化之后，定期地手工执行测试用例依然是重要的，这样可以保持关于测试怎么运作和如何验证正确的操作等方面的知识。

当一个工具投入使用和测试脚本数量不断增加时，可能需要增加其它工具所具有的功能特性，这并不是总能实现的，因为工具并不都具有开放式的接口，而且有时会使用非标准的脚本语言。明智的做法是使用带有开放式架构的有插件程序或者 API（应用程序接口）的工具。这就保证了测试脚本作为测试件的适用性。

对于每一种工具，不管在哪个阶段使用，需要考虑下面的特性列表。这些特性适用于评估和构建工具。在每一个过程域当中，工具的作用有强有弱。如果有一份同类工具的对比列表则相当有用。

- 分析（概念的理解，自动或者手工提供的输入或信息）
- 设计（自动或手工生成的）
- 选择（根据一系列指标，如覆盖率，进行手工或者自动的选择）
- 执行（人工，自动，驱动，重启等）
- 评估（例如，测试基准）和演示。这类通常被认为是日志或者报告功能（包括手工和自动的方式，例如比较、方式、规范、标准的产生）

9.2.7 开源测试工具的使用

用于测试安全关键系统的工具需要遵循相关的标准。并不推荐在安全关键系统当中使用开源工具，除非它们达到了一定的认证等级。

开源软件的质量是基于其曝光率、历史和受质疑的使用范围，并不能假定比市面上其它任何软件具有更多（或者更少）的正确性。需要组织质量评审以鉴定测试工具的正确性。

对于某些工具类型来说，更容易将正面的评估结果和错误的工具执行混淆在一起（例如，跳过了某些执行过程却没有报告所跳过的内容）。还要仔细考虑许可费用。同样希望通过对代码的修改以增强所共享的工具的功能特性。

9.2.8 开发自己的测试工具

测试人员和设计者为了加快他们的工作开发了许多测试工具。开发自己测试工具的另外一个原因就是市场上缺乏合适的软件、特定的硬件或测试环境。这些工具通常能有效地完成事先定义的任务，但同时也非常依赖创建工具的人。为了能让其它人对它们进行维护，需要对这些工具文档化。在组织中推广工具时，对其开发目的、目标、优点和可能存在的缺点等进行评审是非常重要的。一般来说，这类工具的最初版本不一定能带来效益，但将在原有基础上不断满足新的需求而对工具进行了大量的扩展。

9.2.9 测试工具分类

除了根据工具所支持的活动进行分类（沿用初级大纲的概念），还有其它一些分类，包括:

- 根据测试执行的级别分类（组件、集成、系统、验收）
- 根据处理和支持的故障分类
- 根据测试方法和测试技术分类（详见下文）
- 根据测试的目的分类，如度量、驱动、记录、比较
- 根据特定的领域分类，如交通模拟与信号、网络、协议、交换、电视屏、专家系统等
- 根据所支持的不同测试领域分类，如数据输入、环境、配置或者其它概念范围
- 根据工具应用方法分类：现货供应、框架（对适应性而言）、插件（也就是Eclipse），标准或认证测试套件、内部工具开发

最后，工具可以根据它们的实际使用者进行分类，如测试经理、测试分析员或测试技术分析员。这种分类反映了本大纲的模块，并会在本章接下来的内容当中使用。初级大纲有专门讨论工具的章节。
下节是关于工具的附加信息。

9.3 测试工具类别

本节包括如下目标：

- 在ISTQB®初级大纲第6部分的基础上提供关于工具类别的附加信息，例如测试管理工具、测试执行工具和性能测试工具。
- 介绍新工具类别

请参照 ISTQB®初级大纲第 6 部分的内容以了解这部分内容没有提及到的其它工具类别的一般信息。

9.3.1 测试管理工具

对于测试管理工具的一般信息，请参照 ISTQB®初级大纲 6.1.2.

测试管理工具需要具有跟踪以下信息的能力：

- 测试工件（Artifact）的跟踪
- 复杂环境中测试环境数据的获取
- 不同测试场所（跨测试组织）的相同测试任务（Test sessions）在不同测试环境中的并发测试套件的有关数据
- 度量与分析，例如：
  - 测试条件
  - 测试用例
  - 执行时间（测试用例、套件、回归测试套件）和其它重要的计时，包括有助于管理决策的平均值
  - 测试用例的数量，测试工件和测试环境
  - 通过/失败百分比
  - 未执行的测试用例数量（及未执行的原因）
  - 趋势
  - 需求
  - 测试工件之间的关系和可追溯性
- 测试管理工具支持的概念，例如：
  - 测试工件的组织者，测试用例的责任人和执行者
  - 测试条件和测试环境
  - 回归测试套件、测试任务
  - 日志和失效处理信息
  - 环境的重新启动（和重新初始化）
  - 使用测试度量与分析报告记录测试工件达到文档化测试过程的目的

测试管理工具提供给测试管理人员、测试分析员和测试技术分析员应用。

9.3.2 测试执行工具

对于测试执行工具的一般信息，请参照 ISTQB®初级大纲 6.1.5

测试执行工具主要为测试分析员和测试技术分析员在各种测试级别中使用，用于运行测试和验证测试结果。使用测试执行工具的主要目的包括下列一个或多个方面：

- 减少成本（工作量或者时间）
- 运行更多测试
- 让测试更具可重复性

测试执行工具最常用于自动化回归测试。

测试执行工具执行用程序语言编写的一系列命令，通常被称为脚本语言。工具中所用命令的详细程度能描述单个按钮操作、键盘输入和鼠标移动。具体的测试脚本对于被测试软件的变化非常敏感，特别对于图形用户界面（GUI）的改变。

编写脚本可以从录制脚本（由捕获/回放工具完成）、使用现有脚本、模板或者关键词创建或编写脚本等开始。脚本是一种象其它软件一样工作的程序。捕获（或者录制）可以用于记录非系统性测试的审计跟踪。大多数的测试执行工具包括了比较器，其功能是比较实际结果和所存贮的期望结果。测试的趋势（如同编程一样）是从详细的低级命令转向更“高级”的语言，此处同样会涉及库、宏和子程序的使用。系列的命令会标上一个名字——测试的关键词（Key-word）驱动或者行为词（Action-word）驱动。这样做的好处是区别命令与数据。这与在编写脚本中为减少用户的工作量而使用模板是相同的概念。

有些测试工具应用失败的主要原因是测试人员编程和认知水平的低下，他们不清楚在自动测试执行过程中，测试工具只能解决部分的问题。非常重要的一点是任何测试执行自动化都需要管理、工作量、技术和注意力，如测试架构和配置管理。这也同样意味着测试脚本会有缺陷。测试件架构的使用可能有着工具提供商赋予的独立性。购买工具之后，一般的思路是需要遵循工具的标准，例如脚本的结构和命名。然而，测试自动化会在组织测试的最佳方法和工具如何发挥作用之间建立起衔接。

9.3.3 *调试与故障解决工具*

故障解决需要使用工具来缩小产生错误的范围。在没有证据表明什么原因导致系统失效情况下，同样需要解决故障。故障解决工具包括跟踪和模拟环境，与软件进行交互或者从系统中提取系统信息，从而缩小产生问题的范围。

编程人员使用调试和跟踪工具重现问题并进一步研究程序的状态。调试器和跟踪器可以使编程人员做以下的事情：

- 逐行运行程序
- 在程序的运行过程中暂停程序
- 设置和检查程序变量

需要清楚了解调试（和调试工具）与测试工作相关，但不是测试（调试工具不是测试工具）。调试和跟踪工具可以被测试人员用于故障解决以更好地定位失效来源和决定缺陷报告将发送到何处。调试、跟踪和故障处理工具主要被测试技术分析员使用。

9.3.4 *错误传播和错误注入工具*

错误传播和错误注入工具是应用于测试的两种不同的工具。错误传播会像编译器那样规则地创造单一或者有限种类的代码错误。这类工具也可以与变异测试（Mutation test）技术（也叫变异测试工具）一起使用。

错误注入工具集中于改变测试组件的实际接口（当源代码不可用时），也可以特意（重新）植入一个特定的故障以检查 1）软件是否可以处理故障（容错性）。或者 2）评价测试套件当中的某个测试是否能找出特意插入的故障。错误传播和错误注入工具主要被测试技术分析员在用于编码级别的工作，同样也可以让测试分析员为了测试分析而操纵数据库中的数据或者在数据流中植入错误以测试系统的行为。

9.3.5 *模拟与仿真器*

模拟工具是在编码或者其它系统难以获得、昂贵或者不可行（处理核灾难的测试软件）时支持测试。一些模拟器和测试控制工具也可以支持或者模拟故障表现，因此，可以检查错误状态和错误场景。使用这类工具的最主要风险是不能检测到对于某些系统相当重要的资源相关的错误（如计时问题等）。

仿真器是一种特殊的模拟器，由软件编写构成用于模拟硬件。使用仿真器的好处在于实现更精细的测试。仿真器其中的一个优势就是跟踪、调试和重新创建依赖于时间的起因，而这是真实的系统无法实现

的。构建仿真器的成本不菲，但其可以让系统“慢动作”运行，这对一些并行、依赖于时间的和复杂的系统的分析来说，是非常值得的。.

测试分析员或测试技术分析员会根据所需要的模拟种类，使用这些工具。

9.3.6 静态和动态分析工具

对于静态和动态分析工具的一般信息，请参见 ISTQB®初级大纲 6.1.6“性能和监控工具”。

9.3.6.1 静态分析工具

静态分析工具可以在软件生命周期的任何时候使用，也可以在软件开发的任何级别或阶段中使用，这最终取决于工具所能提供的度量方法。静态分析工具以警告的方式报告它们的发现。未经证实的警告被称为假报（False positive）。真报（被证实的警告）才是真正的问题，在执行当中可能会导致系统失效。由于需要合适的故障排除技术，辨别出真假性则是困难和耗时的。最近更多的静态分析工具可使用在编译过程中动态绑定的信息，从而能更有力地发现真正的问题，产生更少的假报。测试技术分析员使用这些工具。

9.3.6.2 动态分析工具

动态分析工具提供软件运行状态的实时信息。这些工具一般用于识别未分配的指针、检查指针算法、监督内存的分配、使用和释放，从而可标记内存泄漏和显示那些难以用静态方法发现的错误。因为内存问题是动态产生的，所以应该在不同层次上而不是针对大规模复杂的系统使用内存检测工具。需要注意的是，不同的商业测试工具的实施方法不同，因此跟踪和报告不同类型的内存或资源（如堆栈）问题。结论是，两种不同的内存管理工具识别不同的问题。当某些编程语言（C，C++）的内存管理由程序员控制的时候，内存检测工具特别有用。测试技术分析员使用这些工具。

9.3.7 关键词驱动测试自动化

关键词（有时称为“动作词”）大多（但并非专有地）用来代表系统中的高级商业活动（如“取消订单”）。每个关键词特地用来表示被测试系统里一系列的具体交互活动。在某些特殊的测试用例中会使用关键词序列（包括相应的测试数据）。[Buwalda01]

在自动化测试当中，一个关键词作为一个或者多个可执行的测试脚本所实施。工具会读入使用关键词编写的测试用例并调用执行关键词的相应的测试脚本。脚本以高度模块化方式实施而易于映射到特定的关键词。开发模块化的脚本需要良好的编程技术。

关键词驱动的自动化测试的主要优势包括：

- 可以由特定应用或业务领域的专家来定义关键词。这可以使测试用例规格说明的任务变得更有效。
- 对于业务领域的专业人士可以在自动测试用例执行过程中受益（一旦关键词以脚本方式被实施）
- 用关键词描写测试用例便于维护，因为它们不会因被测试软件的细节变化需要做大量的修改
- 测试用例规格说明独立于实施。关键词可以通过各种脚本语言和工具实现。

基于关键词的自动测试大多数的使用者是领域专家和测试分析员。

9.3.8 性能测试工具

对于性能测试工具方面的通用知识，请参考 ISTQB®初级大纲 6.1.6“性能和监控工具”

性能测试工具有两种主要功能：

- 负载生成
- 给定负载下系统响应的度量与分析

负载生成是通过脚本实现预先定义的操作特征（请参见 5.3.3 节）。可通过单个用户的操作获取最初的脚本（可能用到捕获/回放工具），然后通过性能测试工具转化为特定的操作特征。其实施必须考虑每个事务处理数据的变化（或者一系列的事务之中）。

性能工具通过模拟大量并发用户（即虚拟用户）、在特定容量的输入数据条件下产生负载。与捕获/回放工具相比，许多性能测试脚本通过通讯协议层来再现系统和用户之间的交互作用，而非通过图形用户界面来模拟用户与系统之间的交互作用。只有很少的负载生成工具是通过用户界面的方式产生负载。

性能测试工具使用广泛的度量手段以完成测试实施过程及其之后的分析。典型的度量与分析需要关注以下方面：

- 虚拟用户的数量
- 虚拟用户执行的事务的数量和种类
- 由用户产生的特定事务请求的响应时间
- 基于测试日志的报告，负载与响应时间的图形

性能测试工具实施时主要考虑因素：

- 保证负载产生的硬件和网络带宽
- 被测试系统所使用的通讯协议和测试工具的兼容性
- 工具的灵活性以保证不同的运行特征易于执行
- 必要的监视、分析和报告机制

由于性能测试工具开发的工作量比较大，需要获取（购买）性能工具。但是，针对某些有技术保护的产品或者负载特征和机制相对简单时，宜于自己开发有针对性的性能测试工具。性能测试工具为测试技术分析员所用。

说明：性能相关的缺陷往往对于被测系统有深远的影响。当有紧急的性能要求时，对关键组件（通过驱动程序和桩）进行性能测试优于等待整个系统的性能测试。

9.3.9 Web 工具

超链接测试工具是用于扫描和检查网站中未中断或丢失的超链接。有些工具也提供额外的信息，如结构图（站点树状图）、下载速度和文件大小（通过 URL）、点击率和容量。这些工具也可以用于控制 SLA（服务等级协议）符合性。测试分析员和测试技术分析员使用这些工具。

10. 个人技能与团队构成

术语

测试的独立性

10.1 简介

所有的专业测试人员都应该意识到要能很好地完成具体的任务需要一定的个人技能。本节首先集中讨论个人技能，然后讨论有关测试经理的一些具体问题，例如团队整体实力、组织、激励以及沟通。

10.2 个人技能

软件测试的个人能力可以通过经验或者在不同工作领域的培训中获得。以下的每一条对于测试人员的基础知识都有帮助：

- 软件系统的使用
- 领域或业务的知识
- 参与软件开发过程不同阶段的各种活动，包括分析、开发和技术支持等
- 参与过软件设计活动

软件系统的用户熟悉系统用户方的有关问题，并且非常了解系统是如何操作的、对系统产生最大影响的失效在哪里以及系统的预期反应应该是什么。具有领域专业知识的用户知道哪些领域知识对业务来讲是最重要的以及这些领域知识如何影响满足业务需求的能力。这些知识常用来帮助排列测试活动的优先级、创建实际的测试数据和测试用例，以及验证或提供用例。

软件开发过程的知识（需求分析、设计和编码）有助于深入了解错误如何引入、错误在哪里能检测以及如何预防它们的引入。有技术支持的经历提供用户体验、期望和可用性需求的知识。软件开发经历很重要，因为高端测试自动化工具的使用要求具有编程和设计方面的专业知识。

具体的软件测试技能包括了分析规格说明书、参与风险分析、设计测试用例的能力以及运行测试和记录结果的努力。

尤其是测试经理，他的项目管理知识、技能和经历是很重要的，因为测试管理就像运行一个项目，例如制定计划、跟踪进度和向项目干系人报告。

人际交往技能，例如批评与被批评、影响力和谈判力在测试中的作用都很重要。一个技术上能够胜任的测试人员有可能不胜任这个岗位，除非他能拥有和使用必要的人际沟通技巧。除了与他人一起有效工作外，成功的专业测试人员还必须非常有条理，关注细节，拥有良好的书面表达能力和口头沟通技巧。

10.3 测试团队整体实力

在组织中员工的选择是最重要的管理职能之一。除了工作要求的一些特定个人技能外，还有很多因素要考虑。当选择某个人加入团队时，必须考虑团队的整体实力。这个人是否与测试团队中已经存在的技能和性格互补？重要的是考虑测试团队成员性格多样性和技术技能组合的优势。一个强大的测试团队能够处理不同复杂性的多个项目，同时也能成功地处理与其它项目组成员的人际关系。

新的团队成员必须快速融入小组并且得到充分的指导。在团队中每个人应该被赋予一个规定的角色。这可以建立在个人评估过程的基础之上，目的是为了在团队获得整体成功的同时，个人也能获得个人的成功。主要是将个人性格和团队角色进行匹配，培养个人天赋技能，同时提高他们的技能组合。

需要记住的一个重点是，没有人是完美的。但可以在个人长处和短处的平衡之上建立一个强大的团队。要维持和建立团队知识以及增加团队的灵活性需要团队内部进行交叉培训。

10.4 组织内测试的适应

在测试如何适应组织结构方面，组织之间差异很大。尽管质量在整个软件开发生命周期内是每个人的责任，但一个独立的测试团队对于软件产品的质量有很大帮助。测试功能的独立性在实践中差异很大，下面列出了从最低到最高程度的独立性：

- 非独立的测试人员
  - 在这种情况下不存在独立性，开发人员测试他们自己的代码
  - 如果开发人员有时间做测试，那么他就会确定代码是否按照他设想的方式运行，这种方式可能符合实际需求也可能不符合
  - 开发人员能够快速地修复发现的缺陷
- 测试由编写该段代码以外的不同开发人员执行
  - 开发人员和测试人员之间独立性很小
  - 一个开发人员测试另一个开发人员的代码可能不情愿报告缺陷
  - 开发人员的注意力通常集中在正面的测试用例上
- 执行测试的测试人员（或测试组）是开发团队的一部分
  - 测试人员（或测试组）向项目经理报告
  - 测试人员的注意力更多地集中在验证对需求的符合上
  - 因为测试人员是开发团队中的一员，他除了测试以外可能还有开发职责
- 测试人员来自于业务组织、用户团体或者其它非开发的技术组织
  - 单独报告给项目干系人
  - 质量是团队的主要关注点
  - 技能开发和培训的重点是测试
- 外部测试专家针对特定的测试目标执行测试
  - 测试目标可以是可用性、安全性或性能
  - 质量应该是这些专家关注的焦点，但是可能取决于报告结构
- 测试由公司之外的组织来执行
  - 实现最大化的独立
  - 知识传递可能不充分
  - 需要明确的需求和定义一个很好的沟通框架
  - 外部组织的质量必须被定期审核

在开发和测试组织之间存在不同程度的独立。要清楚的认识到需要在独立程度和知识传递之间进行平衡。较低的独立性有利于增长知识，但是也会引入有冲突的目标。独立的程度也可能由采用的软件开发模型所决定，例如：在敏捷开发中测试人员常常是开发团队中的一部分。

一个组织可以采用上述选择进行不同的组合。测试可以由开发组织及独立的测试组织分别进行，也许最后由外部组织进行最终认证。重要的是清楚每个测试阶段的职责和期望，并且在进度和预算约束下设定那些要求以便保持最终产品的最佳质量。

外包是使用外部组织的形式之一。外包组织可以是在你的公司现场，也可以是在你的公司之外的你所在国或者其他国家（这种情况有时称为离岸外包），为你提供测试服务的公司。外包带来挑战，特别是跨国外包，应该考虑下面的很多因素：

- 文化的差异
- 外包资源监督
- 信息传递和沟通
- 知识产权的保护
- 技能水平，以及技能开发和培训
- 员工流动
- 准确的成本估算
- 质量

10.5 激励

有很多方法可以激励担任测试岗位的个人，它们包括：

- 肯定所完成的工作
- 管理层的认可
- 项目组和同事之间的相互尊重
- 所做工作的充分回报（包括薪水、绩效考核和奖金）

一些项目的影响使得这些激励机制难以实施。例如，测试人员很可能面对一个不可能在最后期限完成的项目。测试人员尽个人最大努力推动团队对质量的关注，并且投入了额外的时间和精力，但是由于一些外部因素影响产品可能还在不应该发布的时候就被发布了。结果可能是尽管测试人员已经尽了最大的努力，但仍是一个质量很差的产品。如果测试人员的贡献没有被理解和衡量，不管最终产品是否成功，对测试人员都很可能是个打击。

测试团队必须确保跟踪一些适宜的度量数据，一来证明他们很好地完成了测试，二来减轻风险并准确地记录测试结果。如果没有收集并发布测试度量数据，测试团队很容易因为良好的工作没有得到应得的认可而受挫。

认可不仅是指无形的尊重和认可，它也体现在升职机会、薪酬等级和职业晋升通道中。如果不尊重测试组，也就不会有这些机遇。

当测试人员对于项目价值做出显著贡献时，就应该得到认可和尊重。在一个单独的项目中，测试人员从项目的概念阶段开始参与，并在项目的整个生命周期中持续工作，其贡献能够得到迅速体现。在测试过程中，当测试人员对项目顺利开发做出了突出的贡献，他们就会赢得认可和尊重，当然这个贡献也应从质量成本的减少和风险缓解方面等进行量化。

10.6 沟通

测试团队的交流主要出现在三个层面：

- 测试产品的文档：测试策略、测试计划、测试用例、测试总结报告、缺陷报告等
- 文档审查的反馈：需求、功能规格说明书、用例、组件测试文档等
- 信息收集与传播：与开发人员、其他测试组成员、管理者等的相关交流

所有的交流必须是专业的、客观的并且有效的，目的是建立和维护测试团队的尊严。对于其他人的工作产品提供反馈意见，特别是建设性的反馈意见时，反馈必须保证客观性和一定的策略。

所有的沟通应该侧重于实现测试目标以及改进产品及其开发过程的质量上。测试人员与广大听众包括用户、项目组成员、经理、外部测试组和顾客的沟通必须是对于目标听众而言有效的。例如为开发团队设计的缺陷趋势报告，对于向管理层汇报来说可能过于详细。

11. 参考文献

11.1 标准

本节列出了在本大纲中提到的标准。

11.1.1 按章节顺序

下列每一章参考如下标准：

- 第 2 章
  BS-7925-2, IEEE 829, DO-178B/ED-12B.
- 第 3 章
  IEEE 829 D0-178B/ED-12B.
- 第 4 章
  BS 7925-2.
- 第 5 章
  ISO 9126.
- 第 6 章
  IEEE 1028.
- 第 7 章
  IEEE 829, IEEE 1044, IEEE 1044.1.

11.1.2 按字母表顺序

下列标准在如下各章涉及到：

- [BS-7925-2] BS 7925-2 （1998） Software Component Testing
  第 2 章和第 4 章
- [IEEE 829] IEEE Std 829™ （1998/2005） IEEE Standard for Software Test Documentation （currently under revision）
  第 2 章和第 3 章
- [IEEE 1028] IEEE Std 1028™ （1997） IEEE Standard for Software Reviews
  第 6 章
- [IEEE 1044] IEEE Std 1044™ （1993） IEEE Standard Classification for Software Anomalies
  第 7 章
- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
  第 5 章
- [ISTQB] ISTQB Glossary of terms used in Software Testing, Version 2.0, 2007
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne systems and Equipment certification, RTCA/EUROCAE ED12B.1992.
  第 2 章和第 3 章

11.2 文献

[Beizer95] Beizer Boris, “Black-box testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Black02]: Rex Black, “Managing the Testing Process （2nd edition）”, John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

[Black03]: Rex Black, “Critical Testing Processes”, Addison-Wesley, 2003, ISBN 0-201-74868-1

[Black07]: Rex Black, “Pragmatic Software Testing”, John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Burnstein03]: Ilene Burnstein, “Practical Software Testing”, Springer, 2003, ISBN 0-387-95131-8

[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation” Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Craig, Rick David; Jaskiel, Stefan P., “Systematic Software Testing”, Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, “Risk-based e-business testing”, Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Gilb Tom, Graham Dorothy, “Software inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black “Foundations of Software Testing”, Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann （1994）, Test case design using Classification Trees, in: conference proceeedings STAR 1994

[Jorgensen02]: Paul C.Jorgensen, “Software Testing, a Craftsman’s Approach second edition”, CRC press, 2002, ISBN 0-8493-0809-7

[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4

[Koomen99]: Tim Koomen, Martin Pol, “Test Process Improvement”, Addison-Wesley, 1999, ISBN 0-201-59624-5.

[Myers79]: Glenford J.Myers, “The Art of Software Testing”, John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Pol02]: Martin Pol, Ruud Teunissen, Erik van Veenendaal, “Software Testing: A Guide to the Tmap Approach”, Addison-Wesley, 2002, ISBN 0-201-74571-2

[Splaine01]: Steven Splaine, Stefan P.,Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0

[Stamatis95]: D.H. Stamatis, “Failure Mode and Effect Analysis”, ASQC Quality Press, 1995, ISBN 0-873-89300

[vanVeenendaal02]: van Veenendaal Erik, “The Testing Practitioner”, UTN Publsihing, 2002, ISBN 90-72194-65-9

[Whittaker03]: James Whittaker, “How to Break Software”, Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker04]: James Whittaker and Herbert Thompson, “How to Break Software Security”, Peason / Addison-Wesley, 2004, ISBN 0-321-19433-0

11.3 其它参考文献

下列参考信息来源于因特网。

虽然在本大纲发布时这些链接是有效的，但 ISTQB 不保证这些链接今后依旧有效。

- 第 5 章
  - www.testingstandards.co.uk
- 第 6 章
  - Bug Taxonomy: www.testingeducation.org/a/bsct2.pdf
  - Sample Bug Taxonomy based on Boris Beizer's work: inet.uni2.dk/~vinter/bugtaxst.doc
  - Good overview of various taxonomies: testingeducation.org/a/bugtax.pdf
  - Heuristic Risk-Based Testing By James BachJames Bach, Interview on What IsTesting.com. www.whatistesting.com/interviews/jbach.htm
  - www.satisfice.com/articles/et-article.pdf
  - From "Exploratory & Risk-Based Testing （2004） www.testingeducation.org"
  - Exploring Exploratory Testing , Cem Kaner and Andy Tikam , 2003
  - Pettichord, Bret, "An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord
- 第 9 章
  - www.sei.cmu.edu/cmmi/adoption/pdf/cmmi-overview06.pdf
  - TMMi www.tmmifoundation.org/

12. 附录 A – 大纲背景

高级认证资质的目标

- 使测试在软件工程领域获得基本的、专业的认可
- 提供测试职业发展的标准框架
- 使获得专业认证的测试人员被雇主、客户和同行认可，提高测试人员的背景状况
- 促进全部软件工程学科内的一致且良好的测试实践。
- 确定行业有关的、有价值的测试主题。
- 使软件供应商雇用认证的测试人员，并且通过宣传他们的测试人员招聘政策获得超越竞争对手的商业优势。
- 为测试人员和对测试感兴趣的人员提供获得国际认可的专项资质机会。

本认证的报考要求

参加 ISTQB 高级认证软件测试考试的条件是：

- 拥有 ISTQB 认可的考试委员会或成员委员会颁发的 ISTQB 初级资质。
- 具有若干年的软件测试或开发经验，工作年限由授予高级认证的考试委员会或成员委员会确定。
- 遵守本大纲的道德规范。

推荐考生参加 ISTQB 成员委员会授权课程。但是，参加任何 ISTQB 考试不强制要求参加培训。

在本国际认证之前，ISTQB 认可的成员委员会或考试委员会已经认定的专业人员或者授权发放的软件测试高级认证将与本国际认证同等有效。高级认证不会过期，不需要重新更新。证书上注明授予日期。

ISTQB 认可的成员委员会控制每个参与国的本地事务。成员委员会的职责由 ISTQB 确定，在各个国家实施。成员委员会的职责包括授权培训机构和安排考试，考试由一个或多个签约的考试委员会直接或间接实施。

13. 附录 B – 读者须知

13.1 考试委员会

本高级大纲需要《ISTQB 初级认证大纲》（2005 版）的知识内容和知识级别。

ISTQB 考试委员会可以根据大纲内的任何主题进行命题。

建议每个考题所分配的不同分值应与各自主题的学习目标相对应。例如：K1 知识级别的考题比 K3 知识级别的分值低，K4 知识级别的考题分值更高。

13.2 考生与培训机构

要获得高级证书，考生必须通过初级认证，并且通过考试委员会对考生是否具有参加高级资格认证所需要的实践经验的考察。具体考察标准可以参考相关考试委员会以明确考生达到特定的实践经验条件。ISTQB 建议获得高级认证的基本经验要求是考生具有至少 5 年软件工程的实践经验，或者如果考生拥有工科学士或同等学历，需要具有至少 3 年实践经验。

获得专业软件测试的高级能力需要掌握本大纲内容之外的更多内容。鼓励考生和培训机构除了掌握本大纲要求的内容外，还要多花时间阅读和研究其它内容。

本大纲提供了测试书籍和标准的参考清单供考生和培训机构阅读，从而使考生能够更详细地理解指定的主题内容。

14. 附录 C – 培训机构须知

14.1 模块化

本大纲包括三个模块的内容，分别是：

- 高级测试经理
- 高级测试分析员
- 高级测试技术分析员

通过全部模块者获得“全面高级测试专业”证书。

14.2 培训时间

14.2.1 每个模块的培训

建议 3 个不同模块课程分别需要的教学时间如下：

- 高级测试经理 5 天
- 高级测试分析员 5 天
- 高级测试技术分析员 5 天

这是根据每个模块的章数和各章的学习目标指定的，每章都列出了每个角色的最小教学时间。

培训机构可以付出比规定时间更多的时间，考生可以付出更多的时间阅读和研究。课程表可以与大纲的顺序不同。

课程不必连续讲述。培训机构可以选择不同的时间组织课程教学，例如 3+2 天的测试管理，或者 2 个正常工作日外加 3 天的测试分析或测试技术分析。

14.2.2 通用性

培训机构可以决定只一次性讲述通用主题，以减少时间和避免重复。根据模块的情况，相同的主题可以从不同的角度讲述。

14.2.3 资料

在准备培训资料时，应该使用大纲包含的成熟的标准作为参考。使用的各个标准的版本必须是大纲引用的当前版本。大纲中没有引用的其它出版资料、模板或者标准可以在教学时使用或参考，但不作为考试内容。

14.3 实践练习

实践作业（小练习）应该包含期望学员能够应用学到的知识的所有方面（K3 或更高的学习目标）。根据大纲内容的学习目标和主题描述，确定讲课和练习内容。

15. 附录 D – 建议

适用于本教学大纲各章的建议已汇编合并在一个附录中，如下所列项目可供检查。

此列表基于在初级大纲中涉及到的 7 项测试基本原则，并针对测试中出现的疑问提供了一些有用的建议。此附录中所提供的列表没有把所有的情况都列出来，提供了一个在教学中可供参考的示例。培训机构可从此列表中选取与教学模块最相关的内容进行教学。

15.1 业界建议

当以一种工业化的方式执行测试时，测试会面对很多困难。高级测试人员应该将此大纲中描述的建议与他们的组织、团队、任务以及软件组件的背景相结合。以下所列的建议包含一些会对测试结果的表现产生**消极**影响的方面。需要注意的是以下建议并没有包含所有的情况。

- 编写测试计划时只注重功能测试
  缺陷并不仅局限于功能方面或是单一用户。多用户的相互作用可能会对处于测试阶段的软件产生影响。
- 测试环境的配置不充分
  如果有多个类型的处理器、操作系统、虚拟机、浏览器和各种外设可以组合成许多可能的配置，而测试只局限在其中的部分配置组合时，可能会留下大量未被发现的潜在缺陷。
- 直到最后时刻才执行压力和负载测试
  压力和负载测试的结果可能需要软件（包括但不限于基础构架）方面做出重大调整。鉴于这些调整可能会需要相当多的资源来实施，如果这些测试直到计划引入生产之前才进行的，其结果可能会对项目产生极为不利的影响。
- 不对文档进行测试
  用户会得到开发完成的软件以及相关文档，但如果文档与软件不匹配，用户可能不会使用软件的所有潜在功能甚至完全放弃此软件。
- 不对安装程序进行测试
  安装程序，以及备份和恢复程序，都只运行非常有限的次数。然而，这些程序比软件本身更为重要，因为如果不能够安装软件，就不能使用软件。
- 在转向下一项测试任务之前，坚持完全的做完当前的任务
  虽然一些软件开发全生命周期模型建议按次序执行开发任务，但是在实践当中许多任务往往需要（至少部分的）并行。
- 未能正确地识别风险所在的区域
  某些区域会被识别出存在风险并且被彻底地测试。但是，原来没有识别出风险的区域可能由于未被测试或测试不充分而变得风险很高。
- 测试的输入和规程过于细致
  就测试人员主动定义测试的输入和规程而言，如果不给他们足够的发挥余地的话，他们就不会主动的去测试那些看上去没有错误但实际上却可能包含一些潜在缺陷的区域。
- 对“不相关”的异常视而不见
  “不相关”的现象或结果可能指向了潜伏在表象下的缺陷。
- 检查软件期望实现的功能却不检查不期望实现的功能
  如果限制测试人员仅对产品中期望的功能进行测试的话就很有可能错过产品中不期望被实现的部分（例如，额外的和非期望的功能）。
- 测试套件只有它们的所有者才理解
  测试人员可能会由于职责的调整而转换岗位。其他的测试人员会去阅读并理解之前编写的测

试。如果该测试人员没有提供可读性和可理解性高的测试规格说明书，那将对整个测试产生消极的影响以至于整个测试对象不能够被理解或者整个测试被作废。

- 测试只针对用户看得见的接口
  软件的结构不只局限于用户接口。进程间通信、批处理和其它中断同样也会影响软件的正常工作甚至导致缺陷。
- 缺乏缺陷报告和配置管理
  事件报告和管理以及配置管理对于确保整个项目（包括开发和测试）取得成功是非常重要的。一个优秀的测试人员会去考虑修正发现的缺陷而不是发现很多缺陷后却没有详细的记录它们以至于不能修正。
- 只增加回归测试
  随着时间的推移测试套件的更新不能局限于检查以前发现的缺陷是否再次出现。随着代码的升级，测试套件需要覆盖到那些新的功能并对软件的其它部分做回归。
- 没有为下一次测试做好经验总结
  当软件交付给用户或在市场上发售时，并不意味着测试任务的结束。因为很有可能发行一个新的软件版本或发布，所以应保存好相关的测试记录并传递给测试人员负责的下一次测试。
- 尝试对所有的测试进行自动化
  自动化测试看上去是一个好主意，但是自动化测试本身就是一个开发项目。并不是所有的测试都可以进行自动化测试，有些测试手工执行比自动执行还要快。
- 期望重复运行所有的手工测试
  当再次执行先前的手工测试时，期望执行之前所有的测试是不切实际的。测试人员们的情绪是有起伏的，并且他们往往有意或无意地会倾向于聚焦在软件的特定区域。
- 使用图形化自动测试工具降低测试成本
  一个图形化捕获/回放工具是一笔重要的初期投资，必须根据理解和评估的相关成本，将工具用于支持已经定义的策略。
- 希望回归测试能够发现很多新的缺陷
  回归测试通常不会发现大量的新缺陷，主要因为它们是之前已经做过的测试（例如，为同一个软件之前的版本所做的测试），并且应该在那些之前的测试执行中已经发现了缺陷。但这并不意味着把回归测试全盘否定，仅仅是因为回归测试的效率（指发现新缺陷的能力）要比其它的测试更低。
- 热衷于代码覆盖得到的那些简单的数字
  由于能够提供很多数据和图片，从管理的角度出发代码覆盖和度量是一件很令人感兴趣的事情。但是，数字不能反映一个测试的效率或针对性。例如，对于代码覆盖而言 100%是一个不错的目标，但是，它的可行性如何？充分性又如何呢（也就是它的语句、条件或修正的条件判定覆盖）？
- 仅仅因为测试没有增加覆盖率就将它们从回归测试套件中删除
  在回归测试件中，一些测试可以或者应该删除，而另外一些则需要添加。不应该仅仅根据测试是否增加覆盖率来决定是否删除测试，因为很多针对性的测试用例（指其能够检查出的缺陷类型）并不影响测试覆盖率。例如，代码覆盖不是唯一的覆盖类型；测试可能基于多种原因（如：特定的值或一系列事件）而不是简单的覆盖。
- 将测试覆盖率作为测试人员的绩效指标
  覆盖率是度量测试完整性的一种方法，而不是工作表现或人力效率的指标。不同的组织和项目可用不同的度量方法来评估他们的效率。在使用这些方法时必须经过深思熟虑，以避免不良影响。
- 彻底放弃覆盖度量
  覆盖的形式多种多样（例如，代码语句覆盖、条件覆盖、模块覆盖、功能覆盖等）。使用一定的工作量去获得充分的度量数据是很有意义的。因为这些数据对测试任务很有用，所以没有理由把所有的覆盖度量都放弃。

以上大部分要点已经在本大纲的相关章节中有所提及。

当介绍测试原则并在组织中实际运用时，不仅仅要考虑以上所提及的要点，还要考虑其它额外的信息资源，例如：

- ISTQB 考试大纲（初级和高级）
- 本大纲的参考文献
- 如在 8.3 节所涵盖的参考模型。

16. 索引