软件测试复习大纲

第一章

//软件测试学科的发展

4个导向

- 1957年之前，调试为主 (Debugging Oriented)
- 1957～1978年，以功能验证为导向，测试是证明软件是正确的（正向思维）。
- 1978～1983年，以破坏性检测为导向，测试是为了找到软件中的错误（逆向思维）。
- 1983～1987年，以质量评估为导向，测试是提供产品的评估和质量度量。
- 1988年起，以缺陷预防为导向，测试是为了展示软件符合设计要求，发现缺陷、预防缺陷。

不同阶段

初级阶段（1957～1971）测试通常被认为是对产品进行事后检验，缺乏有效的测试方法
发展阶段（1972～1982），1972年第一次关于软件测试的正式会议，促进了软件测试的发展
成熟阶段（1983到现在），国际标准Std 829-1983，形成一门独立的学科和专业，成为软件工程学科中的一个重要组成部分

正向测试与反向测试的定义，关系

正向思维

- 测试是为了验证软件是否符合用户需求，即验证软件产品是否能正常工作

反向思维

- 测试是为了证明程序有错，而不是证明程序无错误

从从经济视角认知软件测试

- 测试的经济观点就是以最小的代价获得最高的软件产品质量。

SQA，与软件测试关系

SQA

- 软件质量保证（Software Quality Assurance，SQA）活动是通过对软件产品有计划的进行评审和审计来验证软件是否合乎标准的系统工程，通过协调、审查和跟踪以获取有用信息，形成分析结果以指导软件过程。

关系

- SQA指导、监督软件测试的计划和执行，督促测试工作的结果客观、准确和有效，并协助测试流程的改进。
- 软件测试是SQA重要手段之一，为SQA提供所需的数据，作为质量评价的客观依据。
- SQA是一项管理工作，侧重于对流程的评审和监控
- 测试是一项技术性的工作，侧重对产品进行评估和验证

第二章

缺陷定义，现象，判定准则

软件缺陷

- 从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；
- 从外部看，软件缺陷是系统所需要实现的某种功能的失效或违背。

软件缺陷的现象

- 功能、特性没有实现或部分实现
- 设计不合理，存在缺陷
- 实际结果和预期结果不一致
- 运行出错，包括运行中断、系统崩溃、界面混乱
- 数据结果不正确、精度不够
- 用户不能接受的其他问题，如存取时间过长、界面不美观

判定准则

- 将被测试系统的实际输出与所期望的输出进行比较，从而判断是否有差异，即是否为缺陷

软件缺陷产生的原因有哪?

技术问题

- 算法错误、计算和精度问题
- 接口参数传递不匹配

团队工作

- 沟通不充分，误解

软件本身

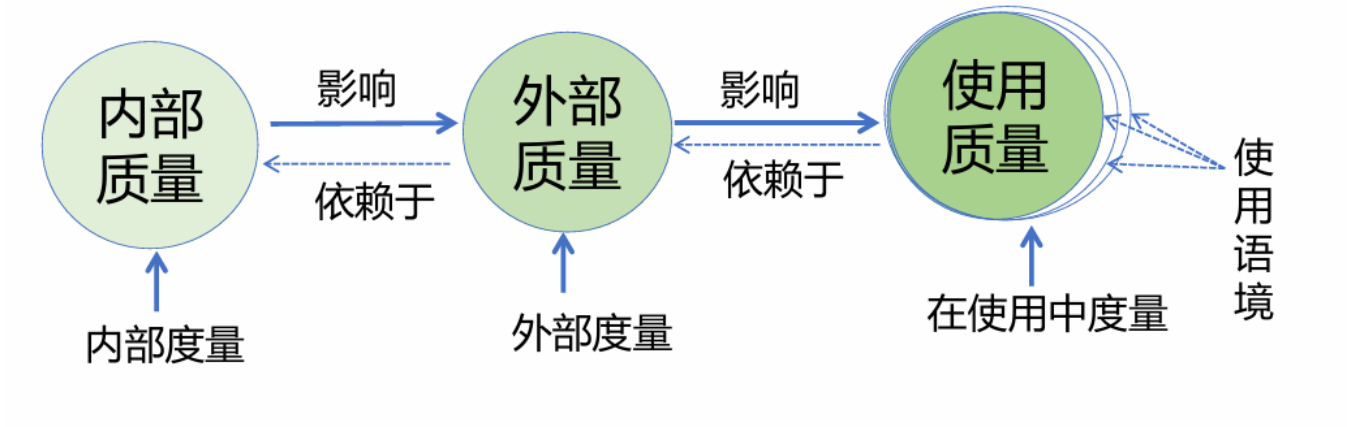
- 文档错误、用户使用场合(user scenario)，
- 时间上不协调、或不一致性所带来的问题
- 系统的自我恢复或数据的异地备份、灾难性恢复等问题

产品质量的内容，内部，外部，使用质量

产品质量

- 是人们实践产物的属性和行为，是可以认识，可以科学地描述的。并且可以通过一些方法和人类活动，来改进质量

内部质量→外部质量→使用质量

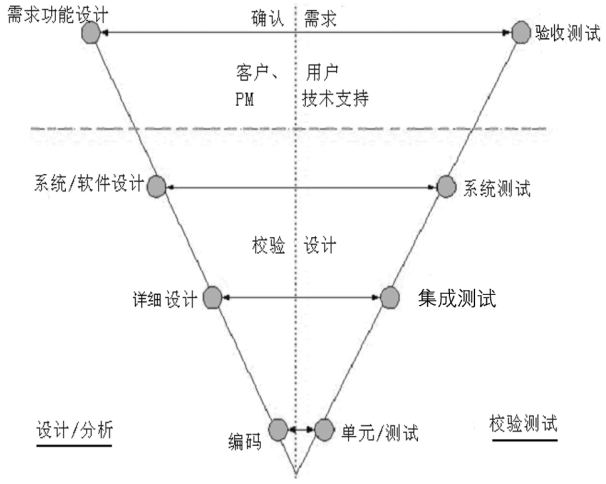


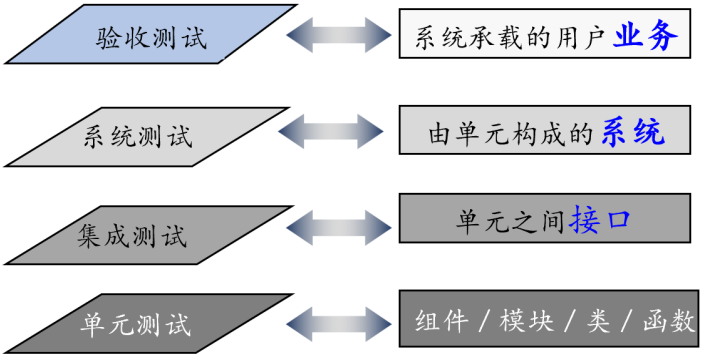
如何理解软件规格说明书缺陷

verification,validation

- 软件测试是由“验证（Verification）”和“有效性确认（Validation）”活动构成的整体
- Verification：Are we building the product right？是否正确地构造了软件？即是否正确地做事，验证开发过程是否遵守已定义好的内容。验证产品满足规格设计说明书的一致性
- Validation： Are we building the right product? 是否构造了正是用户所需要的软件？即是否正在做正确的事。验证产品所实现的功能是否满足用户的需求

软件测试不同层次测试的对象和任务





单元测试

- 单元测试针对程序系统中的最小单元---模块或组件进行测试，一般和编码同步进行。主要采用白盒测试方法，从程序的内部结构出发设计测试用例，检查程序模块或组件的已实现的功能与定义的功能是否一致、以及编码中是否存在错误。通常要编写驱动模块和桩模块

集成测试

- 集成测试，也称联调，在单元测试的基础上，将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的模块之间问题。现在提倡持续集成测试

系统测试

- 系统功能测试
  - 一般在完成集成测试后进行系统功能测试，而且基于产品功能说明书，针对产品所实现的功能，从用户角度来进行功能验证，以确认每个功能是否都能正常使用
- 系统非功能测试
  - 系统非功能性测试是将软件放在整个计算机环境下，包括软硬件平台、某些支持软件和数据等，在实际运行环境下验证系统的非功能性，

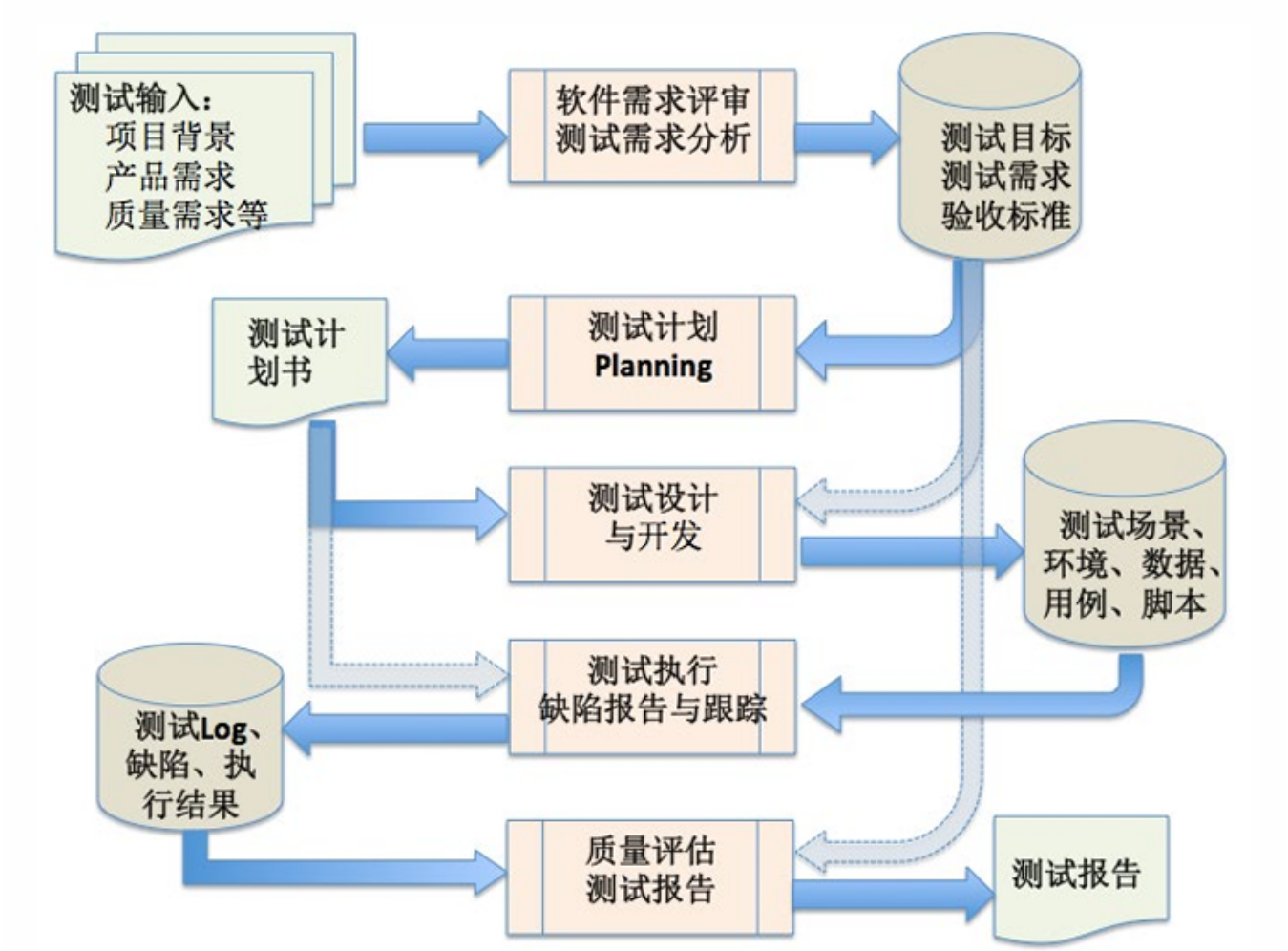
验收测试

- 验收测试的目的是向未来的用户表明系统能够像预定要求那样工作，验证软件的功能和性能如同用户所合理期待的那样

静态测试的内容包括什么？开展相关活动时采用的形式有哪些?

- 静态测试包括对软件产品的需求和设计文档、代码的评审（技术评审、文档评审等），以及对代码的静态分析等；
- 管理评审、流程评审不属于静态测试，而是属于质量保证（QA）；
- 评审的主要形式：互为评审 (Peer review)、走查 (walk-through)、会议评审 (Inspection)；
- 代码的静态分析主要采用工具进行，但人工的代码评审也不可或缺。

测试工作的流程



软件测试的工作范畴

- 测试需求分析、测试策略指定、测试计划、测试设计、测试执行、测试结果和过程评估

测试需求分析

- 明确测试范围，了解哪些功能点要测试、哪些功能点不需要测试；
- 知道哪些测试目标优先级高、哪些目标优先级低；
- 要完成哪些相应的测试任务才能确保目标的实现。

测试策略指定

- 基于下列这些因素的考虑做出决定：
    - 测试方式，包括手工方式与自动化方式、静态方式与动态方式等的选择与平衡，探索式测试或基于脚本的测试、自己团队测试还是众测、外包等平衡；
    - 测试方法，包括黑盒测试还是白盒测试方法、基于数据流还是基于控制流的方法、完全组合测试方法还是组合优化测试方法等平衡；
    - 测试过程，先测什么、后测试什么，对测试阶段的不同划分等。

测试计划

测试设计

- 是解决“如何测”的问题，可以分为测试总体设计和测试详细设计：
    - 测试总体设计则主要指测试方案的设计、测试结构的设计
    - 测试详细设计主要是指测试用例的设计。

测试执行

- 手工执行：基于详细设计的测试用例来完成测试，也可以在没有测试用例的情况下进行的探索式测试。
- 自动化执行：指采用测试工具来完成，一般都需要开发自动化测试脚本，然后工具执行脚本，在后续“单元测试与集成测试、系统测试和自动化测试框架”等各章会进行详细讨论。

测试结果和过程评估

- 测试结果评估：对测试结果进行分析，如分析测试覆盖率，以了解测试是否充分；也可以基于缺陷的趋势分析和分布分析，了解缺陷是否已收敛，以及基于缺陷来评估当前被测试的版本的质量 。
- 测试过程评估，结合测试计划来进行评审，相当于把计划的测试活动和实际执行的活动进行比较，了解测试计划执行的情况和效果 。

第三章

等价类

边界值法

决策表，因果图法

各种逻辑覆盖法

路径覆盖法

功能图法，EFMS

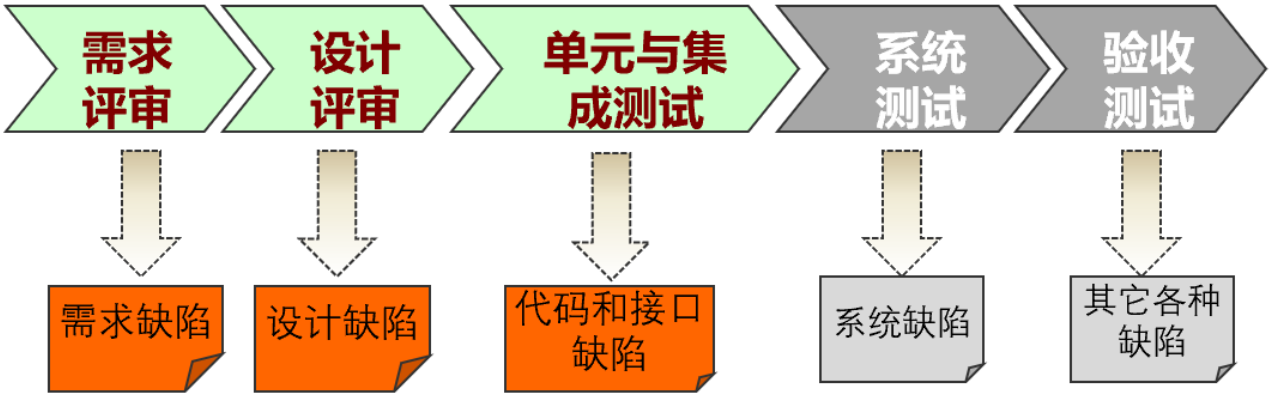
第四章

测试左移和右移，贯穿全生命周期的测试思想

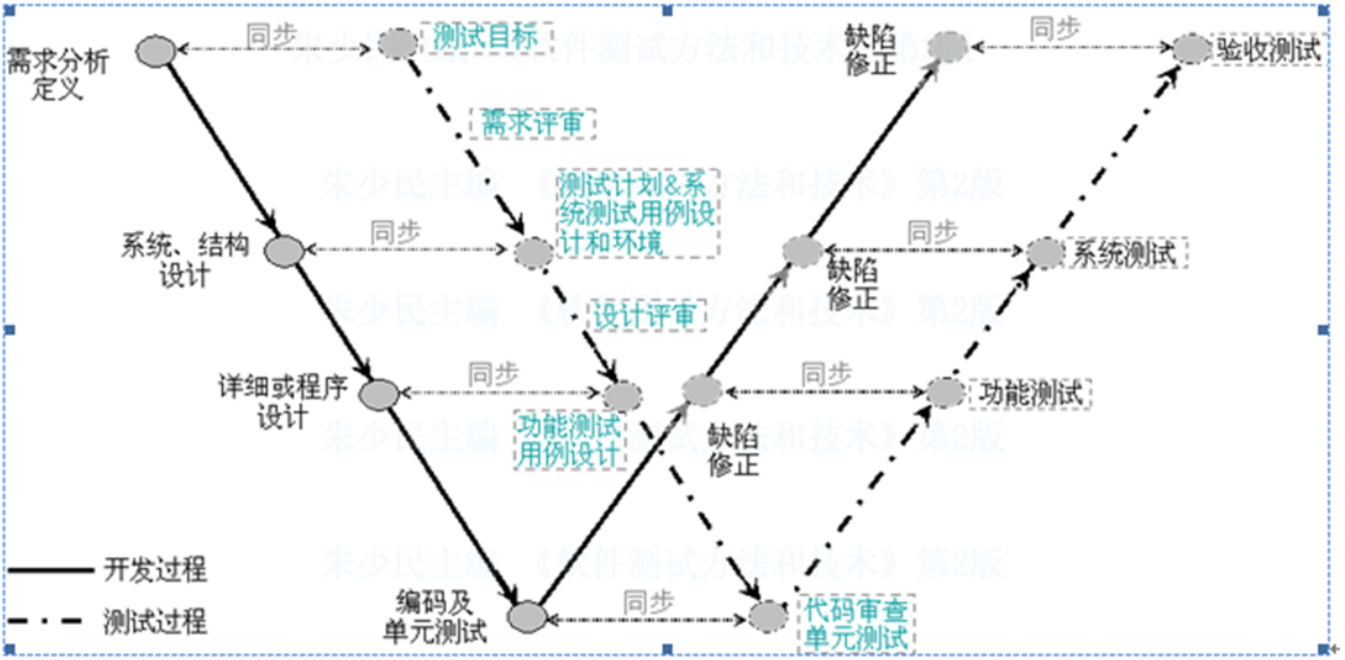
测试左移和右移

- 测试左移：不仅让开发人员做更多的测试，而且需要做需求评审、设计评审，以及第1章介绍的验收测试驱动开发（ATDD）；
- 测试右移：是在线测试（Test in Production，TiP），包括在线性能监控与分析、A/B测试和日志分析等，可以和现在流行的DevOp联系起来。

软件测试生命周期



w模型



- 测试过程和开发是同时开始同时结束的，两者是同步关系

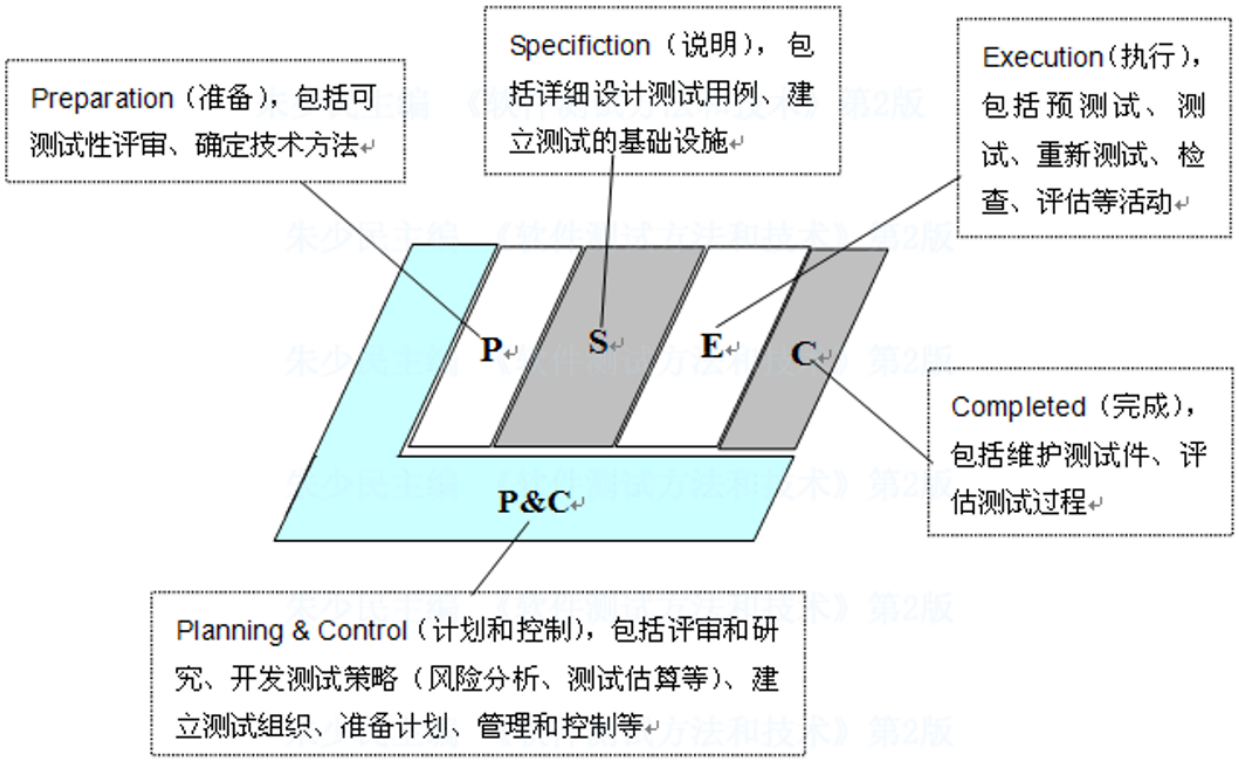
- 测试过程是对开发过程中的阶段性成果和最终的产品进行验证的过程，两者相互依赖
    - 前期测试更多依赖开发，后期反过来

TMAP定义，几个阶段，模型基石及关系

定义

- TMap (Test Management Approach，测试管理方法)是一种结构化的、基于风险策略的测试方法体系, 目的能更早地发现缺陷，以最小的成本、有效地、彻底地完成测试任务，以减少软件发布后的支持成本。
- TMap所定义的测试生命周期由计划和控制、准备、说明、执行和完成等阶段组成

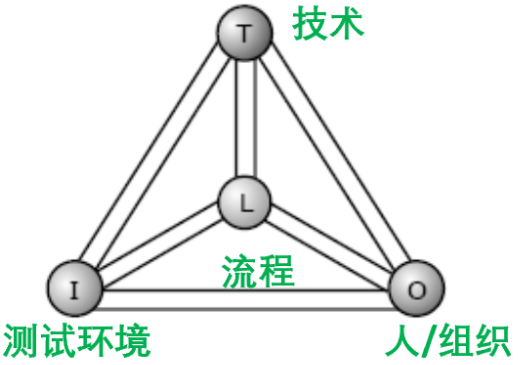
阶段



基石

TMap三大基石

- 与软件开发生命周期一致的测试活动生命周期（L）
- 坚实的组织融合（O）
- 正确的基础设施和工具（I）
- 可用的技术（T）



SBTM基本要素，结果，原理

要素

- Session(会话)是一段不受打扰的测试时间（通常是90分钟），是测试管理的最小单元。
  - 每个session关联一个特定的、目标明确的测试任务（mission）
- Charter (章程，即测试指导)：对每个session如何执行进行简要的描述，相当于每个session需要一个简要的计划（提纲）
  - 一系列Session相互支持，有机地组合在一起，周密地测试了整个产品。
- A session sheet (测试报告)： 相当于测试报告，供第三方（如测试经理、ScrumMaster等）进行检查的材料。它最好能被工具扫描、解析和合成。
- Debriefing（听取口头报告）：口头汇报，更准确地说，是测试人和其lead/Manager之间的对话。

结果：Session Sheet

- Session charter (包含任务陈述、测试范围等)
- Tester name(s)/测试执行者
- Task breakdown :TBS（Test/Bug/Setup）度量（耗时），这些数据配合简报有助于估算测试速度、评估测试效率
- 测试数据、数据文件：为测试数据复用提供了基础
- Note/ 测试笔记：测试过程中随时记录的有价值的信息，叙述了测试故事：为什么测试，如何测，为什么这样的测试已足够好
- Issues/ 问题/风险：测试过程中的问题和疑惑（未来测试的参考资料）
- 缺陷：测试的直接产出

原理

- 通过结构化的测试会话来组织和管理探索性测试活动，以提高测试效率和质量。

测试几个学派的特点

- 分析流派：认为测试是严格的和技术性的，在学术界有许多支持者
- 标准流派：将测试视为衡量进度的一种方法，强调成本和可重复的标准
- 质量流派：强调过程规范性，监督开发人员并充当质量的看门人
- 上下文驱动流派：强调人的价值，寻找涉众关心的bug
- 敏捷流派：强调自动化测试，使用测试来快速验证开发是完整的

TMM，TPI，CTP, STEP定义，特点

TMMi(Testing Maturity Model integration)

- 测试成熟度模型集成
- 过程能力描述了遵循一个软件测试过程可能达到的预期结果的范围。TMMi的建立，得益于以下3点：
  - 充分吸收、CMM/CMMi的精华；
  - 基于历史演化的测试过程；
  - 业界的最佳实践。
- 5个别级的一系列测试能力成熟度的定义，每个级别的组成包括到期目标、到期子目标活动、任务和职责等。
- 一套评价模型，包括一个成熟度问卷、评估程序和团队选拔培训指南。
- 2-5级别内容

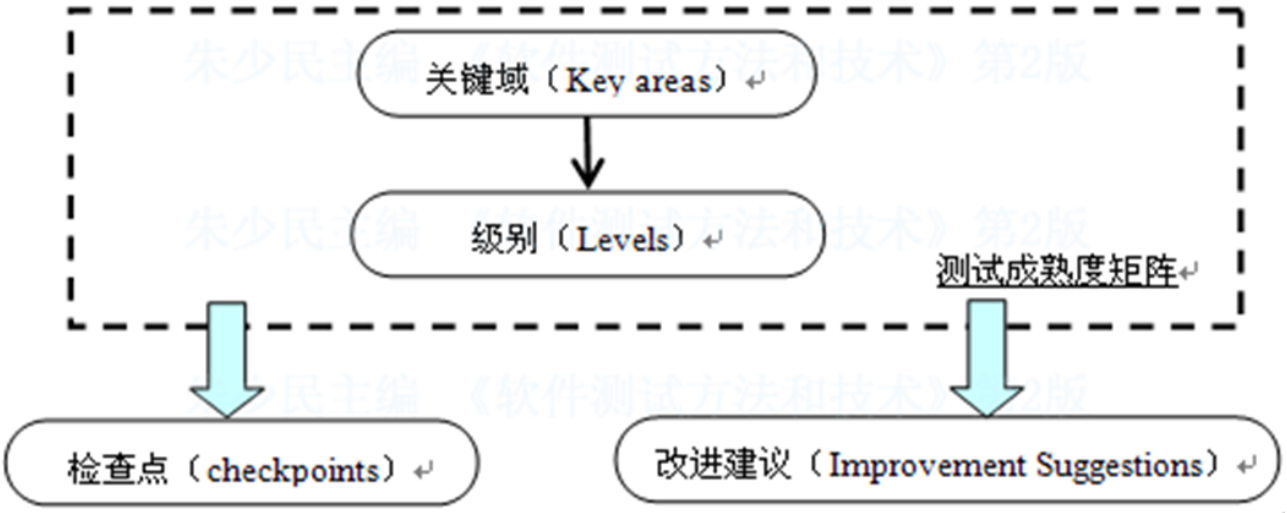
TMMi 2～5级别内容

	描述	特征	目标
2	Phase Definition（阶段定义级）。 测试目标是验证软件符合需求，会采用基本的测试技术和方法。	测试被看做是有计划的活动； 测试同调试分开； 但编码完成后才进行测试工作。	启动测试计划过程； 为基本的测试技术和方法制度化。
3	Integration（集成级）。 测试不再是编码后的一个阶段，而是把测试贯穿在整个软件生命周期中。测试是建立在满足用户或客户的需求上。	具有独立的测试部门； 根据用户需求设计测试用例； 有测试工具辅助进行测试工作； 没有建立起有效的评审制度； 没有建立起质量控制和质量度量标准。	建立软件测试组织； 制订技术培训计划； 测试在整个生命周期内进行； 控制和监视测试过程。
4	Management and Measurement（管理和度量级）。 测试是一个度量和质量控制过程。在软件生命周期中评审作为测试和软件质量控制的一部分	进行可靠性、可用性和可维护性等方面的测试； 采用数据库来管理测试用例； 具有缺陷管理系统并划分缺陷的级别； 还没有建立起缺陷预防机制，且缺乏自动地对测试中产生的数据进行收集和分析的手段。	实施软件生命周期中各阶段评审 建立测试数据库并记录、收集有关测试数据； 建立组织范围内的评审程序； 建立测试过程的度量方法和程序； 软件质量评价。
5	Optimization（优化级） 具有缺陷预防和质量控制的能力；已经建立起测试规范和流程，并不断地进行测试过程改进。	运用缺陷预防和质量控制措施； 选择和评估测试工具存在一个既定的流程； 测试自动化程度高； 自动收集缺陷信息 有常规的缺陷分析机制	应用过程数据预防缺陷。统计质量控制 建立软件产品的质量目标 持续改进测试过程； 优化测试过程。

TPI(Test Process Improvement)

内容

- 是基于连续性表示法的测试过程改进的参考模型，是在软件控制、测试知识以及过往经验的基础上开发出来的



- 20个关键域

1. 测试策略
2. 生命周期模型
3. 介入时间
4. 估计和计划
5. 测试规格技术
6. 静态测试技术
7. 度量
8. 测试自动化
9. 测试环境
10. 办公环境
11. 承诺与动力
12. 测试功能与培训
13. 方法的范围
14. 沟通
15. 报告
16. 缺陷管理
17. 测试件管理
18. 测试过程管理
19. 评估
20. 底层测试

TPI级别

- 为了了解过程在每个关键域所处的状态，即对关键域的评估结果，通过级别是来体现。模型提供了4个级别，由A到D，A是最低级。根据测试过程的可视性改善、测试效率的提高、或成本的降低以及质量的提高，级别会有所上升。

TPI检查点和建议

- 为了能客观地决定各个关键域的级别，TPI模型提供了一种度量工具——检查点。每个级别都有若干个检查点，测试过程只有在满足了这些检查点的要求之后，才意味着它达到了特定的级别
- 检查点帮助我们发现测试过程中的问题，而建议会帮助我们解决问题，最终改进测试过程。建议不仅包含对如何达到下个级别的指导，而且还包括一些具体的操作技巧、注意事项等。

TPI NEXT

- 商业驱动作为测试过程提升的基础
- 为改进目标和度量设定优先级
- 确保商业可以引导和控制改进的过程

CTP(Critical Test Process)关键测试过程

定义和内容

- 内容参考模型、上下文相关的方法，并能对模型进行裁剪
- 使用CTP的过程改进，始于对现有测试过程的评估，通过评估以识别过程的强弱，并结合组织的需要提供改进的意见
- 计划（Plan）、准备（Prepare）、执行（Perform）和完善 (Perfect)；计划和完善主要是管理工作，准备和执行是实践工作

CTP的12个关键过程

- 测试
- 建立上下文关系和测试环境
- 质量风险评估
- 测试估算
- 测试计划
- 测试团队开发
- 测试（管理）系统开发
- 测试发布管理
- 测试执行
- 缺陷报告
- 测试结果报告
- 变更管理

STEP(Systematic Test and Evaluation Process，系统化测试和评估过程)

内容

- 是一个内容参考模型
- 基于需求的测试策略
- 在生命周期初始开始进行测试
- 测试用作需求和使用模型
- 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

比较

- STEP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。
- 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

第五章

代码评审的形式及各自特点

互查

- 自由，不正式，容易开展，开发人员互相审查彼此编码

走查

- 相对正式，采用讲解、讨论和模拟运行方式进行查找错误的活动

会议评审

- 以会议的形式，制定目标，流程和规则

单元测试定义，作用，目标

定义

- 单元测试是对软件基本组成单元（如函数、类的方法等）进行的测试。

作用

- 单元质量是系统质量的基石，单元测试可以比系统测试测得更彻底

单元测试的目标

- 单元模块被正确编码
- 信息能否正确地流入和流出单元；
- 在单元工作过程中，其内部数据能否保持其完整性，包括内部数据的形式、内容及相互关系不发生错误，也包括全局变量在单元中的处理和影响。
- 在为限制数据加工而设置的边界处，能否正确工作。
- 单元的运行能否做到满足特定的逻辑覆盖。
- 单元中发生了错误，其中的出错处理措施是否有效。

> 单元模块被正确实现（主要指编码），包括功能、性能、安全性等，但一般主要介绍单元功能测试。
> （参数）输入是否正确传递和得到保护（容错），输出是否正常
> 内部数据能否保持其完整性，包括变量的正确定义与引用、内存及时释放、全局变量的正确处理和影响最低
> 代码行、分支覆盖或MC/DC达到要求，如高于80%或95%

桩程序，驱动程序

- 驱动程序（Driver）也称作驱动模块，用以模拟被测模块的上级模块，能够调用被测模块。在测试过程中，驱动模块接收测试数据，调用被测模块并把相关的数据传送给被测模块。

> 简单说就是你负责测试的模块没有main()方法入口，所以需要写一个带main的方法来调用你的模块或方法。这个就是驱动测试

- 桩程序（Stub），也称桩模块，用以模拟被测模块工作过程中所调用的下层模块。桩模块由被测模块调用，它们一般只进行很少的数据处理。

> 这里解释一下桩，桩是指用来代替关联代码或者未实现的代码。如果函数B用B1来代替，那么，B称为原函数，B1称为桩函数。打桩就是编写或生成桩代码。

集成测试的概念

- 集成测试是将软件集成起来，对模块之间的接口进行测试。
- 顾名思义，集成测试是将软件集成起来后进行测试。集成测试又叫子系统测试、组装测试、部件测试等。
  - 模块内的集成，主要是测试模块内各个接口间的交互集成关系；
  - 子系统内的集成，测试子系统内各个模块间的交互关系；
  - 系统内的集成，测试系统内各个子系统和模块间的集成关系。
- 集成测试的测试依据；概要设计书，详细设计说明书，主要是概要设计说明书

单一系统的集成测试的集成模式及优缺点

自顶向下

- 定义：从系统的顶层模块开始，逐步向下集成和测试。
- 优点：
  - 测试早期即可验证系统的主要控制逻辑。
  - 测试驱动程序较少，减少开发工作量。
- 缺点：
  - 需要大量桩程序来模拟底层模块。
  - 底层模块的测试可能被延迟。

自底向上

- 定义：从系统的底层模块开始，逐步向上集成和测试。
- 优点：
  - 底层模块的测试可以尽早完成。
  - 桩程序需求较少。
- 缺点：
  - 需要开发大量驱动程序来调用底层模块。
  - 系统的整体功能验证可能被延迟。

混合策略

- 定义：同时从顶层和底层开始集成，最后在中间层完成集成。
- 优点：
  - 顶层和底层模块可以同时测试，节省时间。
  - 减少桩程序和驱动程序的开发工作量。
- 缺点：
  - 需要协调顶层和底层模块的集成进度。
  - 中间层模块的测试可能较复杂。

//微服务特点，测试目标？测试基本步骤？

特点

- 模块化
- 独立部署
- 分布式架构

目标

- 验证服务功能
- 接口测试
- 性能测试
- 容错测试
- 集成测试
- 安全性测试
- 兼容性测试

集成测试的目标

- 集成测试，也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。
- 目标在于检验与软件设计相关的程序结构问题。如数据穿过接口时可能丢失；一个模块与另一个模块可能有由于疏忽的问题而造成有害影响； 把子功能组合起来可能不产生预期的主功能；个别看起来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有错误等。

第六章

功能测试的基本思路

回归测试需要解决的问题。回归测试的策略和方法

解决的问题

- 回归缺陷：原来正常工作的功能，没有发生需求变化，而由于受其它改动影响而产生的问题。
- 回归测试就是为了发现回归缺陷而进行的测试。如果没有回归测试，产品就带着回归缺陷被发布出去了，造成严重后果。

回归测试策略

- 再测试全部用例
- 基于风险选择测试
- 基于操作剖面选择测试

- 再测试修改的部分

第七章

性能测试目标

- 获取系统性能某些指标数据
- 为了验证系统是否达到用户提出的性能指标
- 发现系统中存在的性能瓶颈，优化系统的性能

什么是性能测试?

- 性能测试（performance test）就是为了发现系统性能问题或获取系统性能相关指标而进行的测试。一般在真实环境、特定负载条件下，通过工具模拟实际软件系统的运行及其操作，同时监控性能各项指标，最后对测试结果进行分析以确定系统的性能状况。

系统性能表现

外部表现

- 启动系统、打开页面越来越慢
- 查询数据，很长时间才显示列表
- 网络下载速度很低，如5k/s

内因

- 资源耗尽,如CPU使用率达到100%
- 资源泄漏,如内存泄漏 ，最终会导致资源耗尽
- 资源瓶颈,如线程、GDI、DB连接等资源变得稀缺

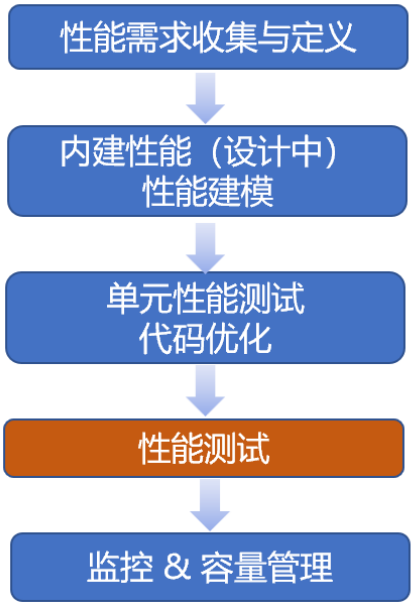
按照测试目的分，性能测试类型

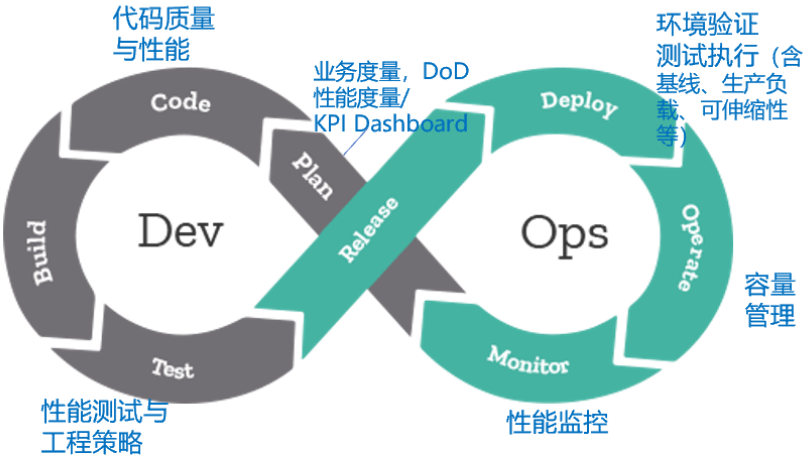
- 性能基准测试：在系统标准配置下获得有关的性能指标数据，作为将来性能改进的基线(Baseline)
- 性能验证测试：验证系统是否达到事先已定义的系统性能指标、能否满足系统的性能需求
- 性能规划测试：在多种特定的环境下，获得不同配置的系统性能指标，从而决定系统部署的软硬件配置选型
- 容量测试可以看作性能的测试一种，因为系统的容量可以看作是系统性能指标之一
- 压力测试 (Stress test)：模拟实际应用的软硬件环境及用户使用过程的高负载、异常负载、超长时间运行，从而加速系统崩溃，以检查程序对异常情况的抵抗能力，找出性能瓶颈、不稳定等问题。
- 渗入测试（soak test），通过长时间运行，使问题逐渐渗透出来，从而发现内存泄漏、垃圾回收（GC）或系统的其他问题，以检验系统的健壮性
- 峰谷测试（peak-rest test），采用高低突变加载方式进行，先加载到高水平的负载，然后急剧降低负载，稍微平息一段时间，再加载到高水平的负载，重复这样过程，容易发现问题的蛛丝马迹，最终找到问题的根源

什么是压力测试

- 模拟实际应用的软硬件环境及用户使用过程的高负载、异常负载、超长时间运行，从而加速系统崩溃，以检查程序对异常情况的抵抗能力，找出性能瓶颈、不稳定等问题。

性能测试的基本过程





什么是安全性测试

- 旨在评估系统的安全性，确保系统能够保护数据和资源免受未经授权的访问、使用、泄露、破坏或篡改。安全性测试的目标是发现系统中的安全漏洞，并验证安全机制的有效性。

渗透测试实施策略

- 全程监控：采用类似wireshark的嗅探软件进行全程抓包嗅探
- 择要监控：对扫描过程不进行录制，仅仅在数据分析后，准备发起渗透前才开启软件进行嗅探
- 主机监控：仅监控受测主机的存活状态
- 指定攻击源：多方监控指定源（某主机）进程、网络连接、数据传输等
- 对关键系统，可以采用对目标的副本进行渗透测试

软件安全性测试有哪两种？有什么关系和区别？

- 安全功能测试 (Security Functional Testing)：数据机密性、完整性、可用性、不可否认性、身份认证、授权、访问控制、审计跟踪、委托、隐私保护、安全管理等
- 安全漏洞测试 (Security Vulnerability Testing)：从攻击者的角度, 以发现软件的安全漏洞为目的。安全漏洞是指系统在设计、实现、操作、管理上存在的可被利用的缺陷或弱点

安全性测试的任务

- 全面检验软件在需求规格说明中规定的防止危险状态措施的有效性和在每一个危险状态下的反应
- 对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案，进行针对性测试
- 在异常条件下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态
- 对安全性关键的软件单元、组件，单独进行加强的测试，以确认其满足安全性需求

安全性测试方法按内外部分为哪两种?

基于威胁的方法

- 从软件外部考察其安全性，识别软件面临的安全威胁并测试其是否能够发生

基于漏洞的方法

- 从软件内部考虑其安全性，识别软件的安全漏洞，如借助特定的漏洞扫描工具

什么是XSS攻击和sql注入攻击，如何进行测试和防范

XSS 跨站脚本攻击

- 允许恶意用户将代码植入到“供其它用户使用的web页面” 中
- 输入验证：对用户输入进行严格的验证，拒绝恶意脚本。
- 输出编码：对动态生成的HTML内容进行编码，防止脚本执行。
    - 使用HTML实体编码，例如将<编码为`<`。

sql注入

- 根据SQL语句的编写规则，附加一个永远为“真”的条件，使系统中某个认证条件总是成立，从而欺骗系统、躲过认证，进而侵入系统
- 使用参数化查询
- 限制数据库权限

web安全性测试可从哪些方法开展

- 检查应用系统架构,防止用户绕过系统直接修改数据库
- 检查身份认证模块，用以防止非法用户绕过身份认证
- 检查数据库接口模块，用以防止用户获取系统权限
- 检查文件接口模块，防止用户获取系统文件
- 检查其他安全威胁

什么是软件可靠性？可从哪几个指标度量？各自的定义

定义

- 在规定的一段时间和条件下，软件能维持其性能水平的能力有关的一组属性，可用成熟性、容错性、易恢复性三个基本子特性来度量。
- 成熟性度量：通过错误发现率DDP（Defect Detection Percentage）来表现。DDP越小，软件越成熟。
    - DDP=测试发现的错误数量/已知的全部错误数量
- 容错性度量，容错测试是一种对抗性的测试过程。在这种测试中，通过各种手段让软件强制性地发生故障，或将把应用程序或系统置于（模拟的）异常条件下，以产生故障，例如设备输入/输出（I/O）故障或无效的数据库指针和关键字等
- 恢复性度量，恢复性的测试先设法（模拟）使系统崩溃、失效等，然后计算其系统和数据恢复的时间来做出易恢复性评估。

容错测试的要点?

- 容错测试是一种对抗性的测试过程。在这种测试中，通过各种手段让软件强制性地发生故障，或将把应用程序或系统置于（模拟的）异常条件下，以产生故障，例如设备输入/输出（I/O）故障或无效的数据库指针和关键字等

什么是A/B测试？有什么特点

- A/B测试（ABTest） 是将用户分成不同的组，同时在线试验产品的不同版本，通过用户反馈的真实数据来确定哪一个方案更好
- 先验性：采用流量分割与小流量测试的方式，先让线上部分小流量用户使用以验证设计，再根据数据反馈来推广到全流量，减少产品损失
- 并行性：同时运行两个或两个以上版本的试验完成对比分析，而且保证每个版本所处的环境一致的，避免流程周期长的问题，节省验证时间
- 科学性：基于统计的数据来做出决策，避免主观或经验的错误决策

第八章

软件本地化，国际化，全球化，相互关系

I18N国际化

- 支持Unicode字符集、双字节的字符；
- 分离程序代码和显示内容
- 消除Hard code
- 使用Header files 去定义经常被调用的代码段；
- 改善翻译文本尺寸，具有调整的灵活性
- 支持各个国家的键盘设置；
- 支持文字排序和大小写转换；
- 支持各个国家的度量衡，时区，货币单位格式等的设置；
- 国际化用户界面设计（自我定义）。

L10N本地化

- 翻译
- 地区文化、宗教
- 度量衡和时区等
- 软件用户界面（UI）
- 联机文档 (帮助文档和功能性的PDF文档)
- 热键设置

unicode与utf-x关系，特点

Unicode

- 定义：Unicode是一种字符编码标准，旨在为全球所有语言的字符分配唯一的编码。它解决了不同语言和字符集之间的兼容性问题。
- 作用：Unicode定义了一个统一的字符集（字符的编号），但不规定具体的存储方式。

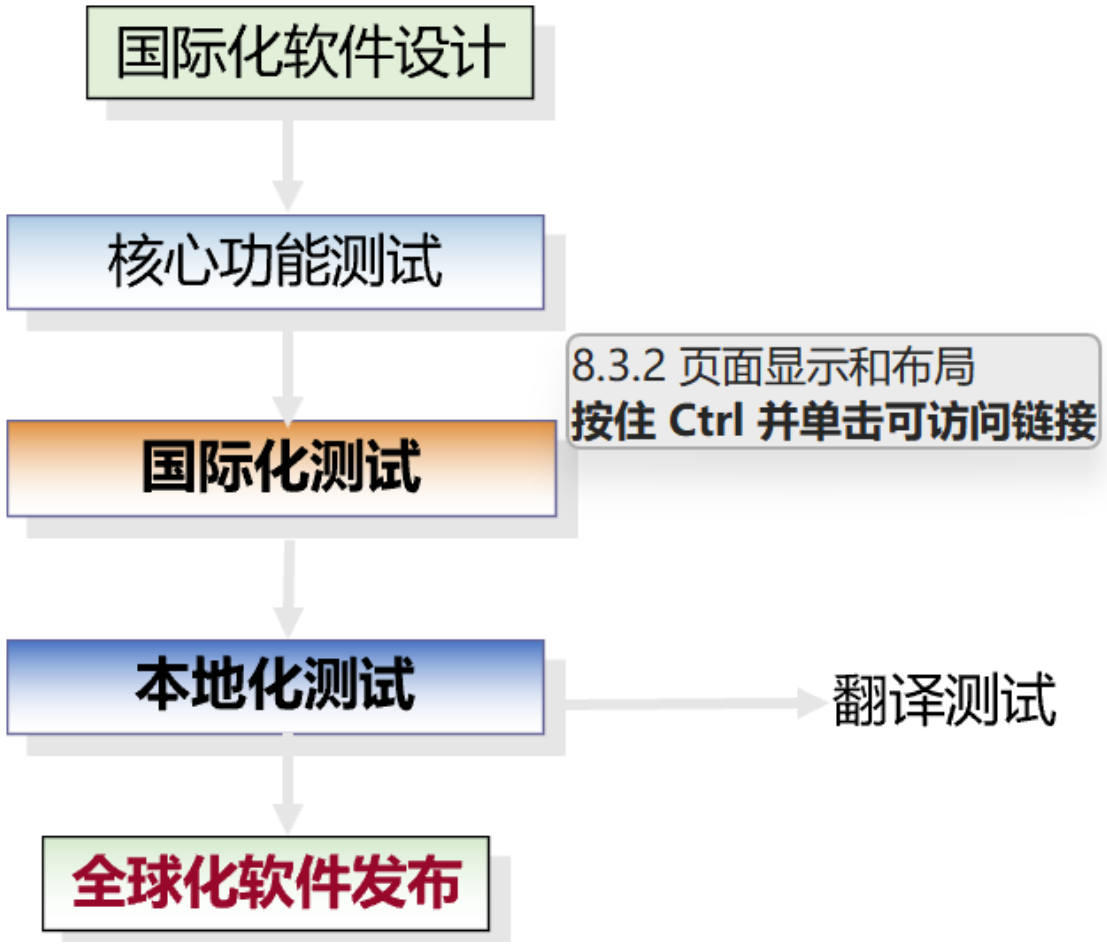
- 范围：Unicode字符集的范围是从U+0000到U+10FFFF，支持超过百万个字符。

UTF-x：

- 定义：UTF-x（Unicode Transformation Format）是Unicode的具体编码实现，用于将Unicode字符转换为二进制数据以便存储或传输。
- 关系：UTF-x是Unicode的实现方式，负责将Unicode字符集映射到实际的字节序列。
- 常见类型：
  - UTF-8：一种变长编码，每个字符使用1到4个字节。
  - UTF-16：一种变长编码，每个字符使用2或4个字节。
  - UTF-32：一种固定长度编码，每个字符使用4个字节。

软件本地化基本步骤

- 建立配置管理体系，跟踪目标语言各个版本的源代码
- 创造和维护术语表
- 源语言代码和资源文件分离、或提取需要本地化的文本
- 把分离或提取的文本、图片等翻译成目标语言
- 把翻译好的文本、图片重新检入目标语言的源代码版本
- 如果需要，编译目标语言的源代码
- 测试翻译后的软件，调整UI 以适应翻译后的文本
- 测试本地化后的软件，确保格式和内容都正确



本地化测试主要有哪些工作

- 功能性测试，所有基本功能、安装、升级等测试；
- 翻译测试，包括语言完整性、术语准确性等的检查；
- 可用性测试，包括用户界面、度量衡和时区等；
- 兼容性调试，包括硬件兼容性、版本兼容性等测试；
- 文化、宗教、喜好等适用性测试
- 手册验证，包括联机文件、在线帮助、PDF文件等测试

软件本地化测试完整路线

第九章

自动化测试与测试自动化

- 通过平台、系统或工具自动地完成测试的某类工作都可以归为测试自动化
- 自动化测试更侧重的测试用例或测试数据生成、测试执行和测试结果呈现等自动化

如何理解测试自动化

测试自动化实现的原理，几种技术

自动化测试的流程

几种脚本技术

自动化功能测试基本构成

TA框架的构成及各部分特点

- Harness/IDE
- 脚本语言(Script Language)
- 代理 (Agents)
- 工具 (Tools)
- 任务安排 (Scheduler)
- 报告 (Report)