Chapter 12 Testing Techniques: Adaptation, Specialization, and Integration

- Adaptation to Test Sub-phases
- Specialized Testing Techniques
- Integration and Web Testing Case Study

Applications of Testing Techniques

- Major testing techniques covered so far:
  - Ad hoc (non-systematic) testing.
  - Checklist-based testing.
  - Partition-based coverage testing.
  - Musa's OP for UBST.
  - Boundary testing (BT).
  - FSM-based coverage testing.
  - Markov chains and UMMs for UBST.
  - Control flow testing (CFT).
  - Data flow testing (DFT).
- Application and adaptation issues:
  - For different purposes/goals.
  - In different environments/sub-phases.
  - Existing techniques: select/adapt.
  - May need new or specialized techniques.

> 已经介绍的主要测试技术包括：
>
> - **非系统化（即兴）测试**
> - **基于清单的测试**
> - **基于分区的覆盖测试**
> - **Musa的操作剖面法用于使用基于状态的测试（UBST）**
> - **边界测试（BT）**
> - **基于有限状态机（FSM）的覆盖测试**
> - **马尔可夫链和统一建模方法（UMMs）用于UBST**
> - **控制流测试（CFT）**
> - **数据流测试（DFT）**

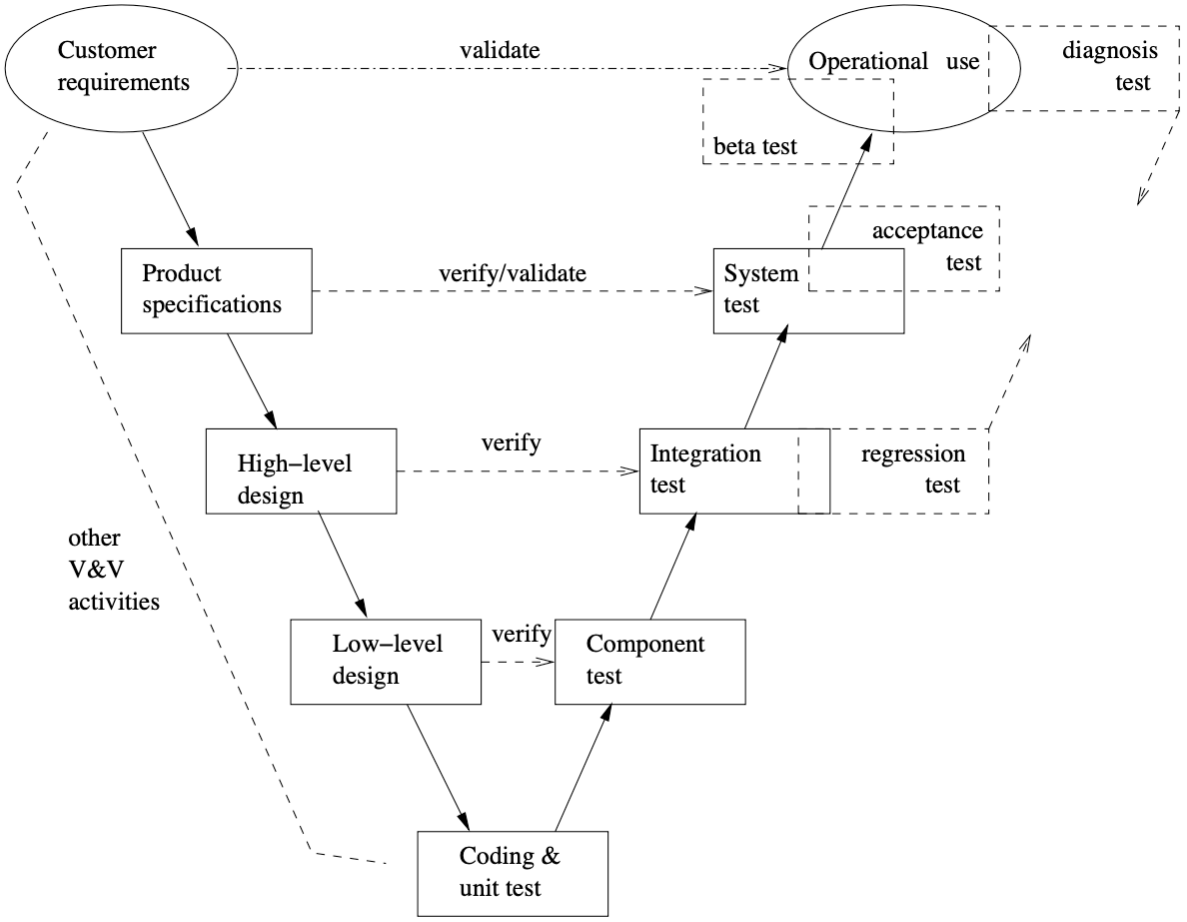
> ## 应用和调整问题
>
> 在不同目的/目标、不同环境/子阶段下的应用和调整问题，包括选择和调整现有技术，可能需要新的或专门的技术。

> ## 应用测试技术
>
> 已经介绍的主要测试技术包括：
>
> - **非系统化（即兴）测试**
> - **基于清单的测试**
> - **基于分区的覆盖测试**
> - **Musa的操作剖面法用于使用基于状态的测试（UBST）**
> - **边界测试（BT）**
> - **基于有限状态机（FSM）的覆盖测试**
> - **马尔可夫链和统一建模方法（UMMs）用于UBST**
> - **控制流测试（CFT）**
> - **数据流测试（DFT）**
>
> ## 应用和调整问题
>
> 在不同目的/目标、不同环境/子阶段下的应用和调整问题，包括选择和调整现有技术，可能需要新的或专门的技术。
>
> ### 举例解释
>
> 1. **非系统化（即兴）测试**适用于初步的软件探索阶段，当你需要快速检查软件的基本功能时。例如，开发者在编写代码后立即运行软件以检查其基本行为是否符合预期。
> 2. **基于清单的测试**适合需求详尽、操作规范明确的场景。通过对照预先定义的测试清单，可以系统地验证软件的每个功能。这在保证质量控制标准的制造业中非常有效。
> 3. **基于分区的覆盖测试**用于将输入或输出值分成不同的类别或分区，并确保从每个分区中选择测试用例。这适用于需要确保各种输入组合都被测试到的复杂软件系统。
> 4. **Musa的操作剖面法**和**基于有限状态机的覆盖测试**，以及**马尔可夫链**，都是用于理解和测试软件在不同状态下的行为的高级技术。它们适用于那些状态转换复杂，如网络协议、操作系统等软件。
> 5. **边界测试**是在输入或输出域的边界值上进行测试，用于捕获边界条件错误。这在任何需要精确输入并且错误边界可能导致严重后果的应用中都非常重要。
> 6. **控制流测试和数据流测试**专注于程序的内部结构，通过分析程序的控制流图和数据流图来识别测试用例。这些技术对于确保代码逻辑的完整性和可靠性至关重要，尤其适用于复杂的商业软件和关键系统。
>
> 每种技术都有其特定的应用场景和目标，选择和调整这些技术以适应特定的测试需求是软件测试中的一个重要方面。在实际应用中，可能需要根据项目的特定需求组合使用多种测试技术，或者开发新的测试方法来解决独特的测试挑战。

12.1 Testing Sub-Phases

- Annotated V-model for testing sub-phases:

  Fig 12.1 (p.204)



  - solid box: original sub-phase
  - dashed box:
    - added sub-phase or specialized testing
- Original sub-phases in V-model:
  - Operational use (not testing, strictly).
  - System test for product specification.
  - Integration test for high-level design.
  - Component test for low-level design.
  - Unit test for program code.
- Additional sub-phases/specialized testing:
  - Diagnosis test through all sub-phases.
  - Beta test for limited product release.
  - Acceptance test for product release.
  - Regression test for legacy products.

> - V 模型的原始子阶段：

    - **操作使用**（严格来说不是测试）。
    - **系统测试**用于产品规格。
    - **集成测试**用于高级设计。
    - **组件测试**用于低级设计。
    - **单元测试**用于程序代码。
- 额外的子阶段/专门的测试：
    - **诊断测试**穿越所有子阶段。
    - **Beta 测试**用于有限的产品发布。
    - **验收测试**用于产品发布。
    - **回归测试**用于遗留产品。

举例解释

1. **单元测试**通常是对单独的功能或软件部分进行的第一轮测试，目的是验证每个单元的功能是否按照设计执行。

   例如，在开发一个在线购物平台时，单元测试可能会针对用户登录、商品搜索、添加至购物车等单独功能进行。

2. **组件测试**涉及到将经过单元测试的相关单元结合起来进行测试，重点是检查不同单元之间的接口是否存在错误。

   在上述购物平台案例中，组件测试可能会检查商品搜索与结果显示是否协同工作无误。

3. **集成测试**在组件测试之后进行，此时将多个组件集成为一个完整的系统或子系统，并对整个系统进行测试以检查组件之间的接口。

   对于购物平台来说，集成测试将确保搜索、结账和用户账户管理等所有功能模块协同工作。

4. **系统测试**是在整个集成软件上进行的高级测试，以验证它是否满足需求。这是软件作为一个整体第一次进行的测试。

   在这一阶段，测试人员会模拟实际操作，确保购物平台的每项功能都能满足用户需求。

5. **操作使用**指的是软件开始运行，可能会在真实环境中使用，并可能进行 Beta 测试。

   这可能意味着邀请一小部分真实用户测试购物平台，并提供反馈以修复任何未发现的问题。

6. **诊断测试**在所有测试阶段中都可以进行，以便及早且持续地诊断和识别问题。

   例如，如果在任何测试阶段发现性能下降，就可能进行诊断测试以找到原因。

7. **Beta 测试**是在正式发布之前，由限量的终端用户在真实条件下进行的测试，以识别早期测试阶段可能未捕捉到的问题。

   对于购物平台，Beta 测试可以帮助识别用户界面或交易流程中的问题。

8. **验收测试**是由客户进行的测试，以确保系统在大规模部署前符合他们的需求。

   对于一个购物平台，客户可能会有一系列特定的验收标准，如页面加载速度和交易安全性。

9. **回归测试**用于检查先前开发和测试过的软件在更改或与新软件接口之后是否仍然表现良好。

   在购物平台中，如果添加了新的支付系统，就需要进行回归测试以确保这一更改不会影响已有的功能。

12.2.1 Unit Testing

- Key characteristics:
  - Object: unit (implemented code)
    - function/procedure/subroutine in C, FORTRAN, etc.
    - method in OO languages
  - Implementation detail ⇒ WBT. (BBT could be used, but less often.)
  - Exit: coverage (reliability undefined).
- Commonly used testing techniques:
  - Ad hoc testing.
  - Informal debugging.
  - Input domain partition testing and BT.
  - CFT and DFT.

> ## 12.2.1 单元测试
>
> - 主要特点：
>   - 对象：单元（实现的代码）
>   - 在 C、FORTRAN 等语言中的函数/过程/子程序。
>   - 面向对象语言中的方法。
>   - 实现细节 ⇒ 白盒测试（WBT）。（黑盒测试也可用，但较少见。）
>   - 出口：覆盖率（可靠性未定义）。
> - 常用的测试技术：
>   - 即兴测试。
>   - 非正式调试。
>   - 输入域划分测试和边界测试（BT）。
>   - 控制流测试（CFT）和数据流测试（DFT）。

> ### 举例解释
>
> 单元测试是针对软件或系统中最小可测试单元的测试。例如，对于一个在线购物平台，每个单元可能是一个单独的功能，如用户认证、搜索引擎或购物车。以下是这些测试技术的具体应用示例：
>
> 1. **即兴测试**可能是开发者在编写一个新功能（如用户登录功能）之后立即进行的基本测试。开发者可能会手动执行这个功能，尝试不同的输入来看它是否按预期工作。
> 2. **非正式调试**通常发生在开发者在编码过程中遇到问题时。如果登录功能无法识别有效的用户凭据，开发者可能会使用调试器或打印语句来跟踪问题。

3. **输入域划分测试**涉及将输入数据的域分成不同的区域，并从每个区域中选择代表性的测试用例。在用户登录功能的测试中，可能将输入数据（如用户名和密码）的域划分为有效输入、无效输入、边界条件等几个部分。
4. **边界测试（BT）**关注于边界值。开发者可能会测试用户名的最大长度，或者密码字段在输入特殊字符时的反应。
5. **控制流测试（CFT）**分析程序的逻辑路径。在购物车组件中，控制流测试可能会确保当用户添加商品时，商品确实被添加到购物车中，并且在显示总价时考虑了所有商品。
6. **数据流测试（DFT）**关注程序内部数据的流动。对于搜索引擎，数据流测试可能会跟踪用户的搜索词如何影响搜索结果的生成。

每种技术都有其特定的应用场景和优势。在实践中，多种技术可能会结合使用以实现更全面的测试覆盖率。

12.1.2 Component Testing

- Key characteristics:
  - Object: component (⊃ unit), 2 types.
  - I. collection of units in C/FORTRAN/etc.
    - implementation detail ⇒ WBT.
  - II. class in OO languages
    - reusable component ⇒ BBT.
  - Exit: coverage (sometimes reliability).
- Commonly used testing techniques:
  - for traditional systems (component I) ≈ unit testing, but at larger scale
  - for OOS/COTS/CBSE (component II) ≈ system testing, but at smaller scale
    - see system testing techniques later

12.1.2 组件测试

- 主要特点：
  - 对象：组件（包含单元），有两种类型。
  - I. 在 C/FORTRAN 等语言中的单元集合。
    - 实现细节 ⇒ 白盒测试（WBT）。
  - II. 在面向对象语言中的类。
    - 可重用组件 ⇒ 黑盒测试（BBT）。
  - 出口：覆盖率（有时是可靠性）。
- 常用的测试技术：
  - 对于传统系统（组件 I），类似于单元测试，但规模更大。

    - 对于面向对象系统/商业现成软件/基于组件的软件工程（组件 II），类似于系统测试，但规模更小。
        - 后面会讨论系统测试技术。

> **举例解释**
>
> 组件测试关注于比单个单元大的构建块，这通常意味着一个由多个单元组成的组件或一个完整的类。以下是这些测试技术的具体应用示例：
>
> 1. 对于传统系统中的组件 I，例如在一个大型财务软件中，一个财务报表生成器可能会使用多个单元（如税务计算、货币转换等）。组件测试将验证这些单元作为整体的协同工作能力。
> 2. 在面向对象语言编写的系统中，组件 II 测试通常针对一个类，该类可能封装了数据和方法。例如，一个电子商务平台的购物车类可能包含添加商品、移除商品、计算总价等方法。组件测试将确保这些方法能够在一起工作，实现购物车的预期功能。
> 3. 对于面向对象系统/商业现成软件/基于组件的软件工程，组件测试可能类似于系统测试，但更关注于组件层面。如果是测试一个商业现成的数据库管理系统组件，测试可能集中在验证API的功能性、性能和安全性。
>
> 在组件测试中，通常使用的技术可能与单元测试中的技术类似，但范围更广。比如，在白盒测试中，测试者不仅需要考虑单个单元的内部逻辑，还需要考虑组件内单元间的交互。对于黑盒测试，则更加注重组件的对外行为和接口，而不深入到内部实现细节。

12.1.3 Integration Testing

- Key characteristics:
    - Object: interface and interaction among multiple components or subsystems.
    - Component as a black-box (assumed).
    - System as a white-box (focus).
    - Exit: coverage (sometimes reliability).
- Commonly used testing techniques:
    - FSM-based coverage testing.
    - Other techniques may also be used.
    - Sometimes treated as ⊂ system testing
        - see system testing techniques below.

> ## 12.1.3 集成测试
>
> - 主要特点：
>   - 对象：多个组件或子系统之间的接口和交互。
>   - 组件作为黑盒（假定）。
>   - 系统作为白盒（焦点）。
>   - 出口：覆盖率（有时是可靠性）。
> - 常用的测试技术：

    - 基于有限状态机（FSM）的覆盖测试。
    - 也可使用其他技术。
    - 有时被视为系统测试的一部分
        - 请参阅下面的系统测试技术。

> **举例解释**
>
> 集成测试是在组件测试之后的下一阶段，它专注于组件或子系统之间交互的正确性和接口的一致性。这里的关键是识别和解决在组件级别测试中可能未发现的问题。以下是集成测试的具体应用示例：
>
> 1. 在一个网络服务平台中，集成测试可能涉及检查客户端和服务器端组件的交互，如验证用户请求是如何通过网络传递并得到正确响应的。
> 2. 对于一款包含多个相互依赖微服务的软件，集成测试将确保这些微服务能够准确地交换数据，比如订单处理系统需要与库存管理和支付处理服务正确集成。
> 3. 使用**基于有限状态机的覆盖测试**，可以验证软件的状态迁移是否符合预期。比如，在一个机票预订系统中，集成测试可能会涉及检查从搜索航班到支付完成的整个流程。
>
> 集成测试通常被视为系统测试的一个子集，因为它涉及到了系统的多个组件和子系统。然而，它与系统测试的不同之处在于，集成测试侧重于组件之间的连接和交互，而系统测试则侧重于整个系统作为一个整体的行为。集成测试在软件开发生命周期中起到至关重要的桥梁作用，它帮助确认在单个组件测试后的集成环境中是否能够达到预期的工作效果。

12.1.4 System Testing

- Key characteristics:
    - Object: whole system and the overall operations, typically from a customer's perspective.
    - No implementation detail ⇒ BBT.
    - Customer perspective ⇒ UBST.
    - Exit: reliability (sometimes coverage).
- Commonly used testing techniques:
    - UBST with Musa or Markov OPs.
    - High-level functional checklists.
    - High-level FSM, possibly CFT & DFT.
    - Special case: as part of a "super"-system in embedded environment

      ⇒ test interaction with the environment.

> ## 12.1.4 系统测试
>
> - 主要特点：
>   - 对象：整个系统及其整体操作，通常从客户的角度出发。
>   - 没有实现细节 ⇒ 黑盒测试（BBT）。
>   - 客户视角 ⇒ 使用基于状态的测试（UBST）。

    - 出口：可靠性（有时是覆盖率）。
- 常用的测试技术：
    - 使用 Musa 或 Markov 操作剖面的 UBST。
    - 高级功能清单检查。
    - 可能包含控制流测试（CFT）和数据流测试（DFT）的高级有限状态机（FSM）。
    - 特殊情况：作为嵌入式环境中"超级"系统的一部分

      ⇒ 测试与环境的交互。

> **举例解释**
>
> 系统测试是软件开发生命周期中的一个高层次测试阶段，涉及整个系统的综合评估。以下是系统测试的具体应用示例：
>
> 1. 例如，在一个航空预订系统中，系统测试将验证整个系统是否满足了航班搜索、预订、支付和用户通知等客户需求。
> 2. 使用**Musa或Markov操作剖面的使用基于状态的测试（UBST）**可能涉及到模拟用户在系统中的各种可能行为路径，并确保每个路径都能正确执行。
> 3. **高级功能清单检查**可能会涵盖用户的每个预期功能，如在一个电子健康记录系统中，确保医生可以搜索患者、添加诊断信息以及生成处方。
> 4. **高级有限状态机（FSM）测试**可能涉及模拟整个系统的状态变化，例如，测试安全系统的警报、监控和响应机制是否按照设计的状态变化进行。
> 5. 在嵌入式系统环境中，例如在智能家居系统中，系统测试将特别关注该系统如何与环境（如灯光、温控和安全摄像头）进行交互，并确保操作符合用户预期和环境要求。
>
> 在系统测试阶段，通常关注系统是否能够以预期和可靠的方式满足用户的需求。这通常包括各种类型的测试，包括功能性、性能、安全性、可靠性和可用性测试。与集成测试不同，系统测试的主要目标是评估整个已集成的系统。

12.1.5 Acceptance Testing

- Key characteristics:
    - Object: whole system.
        - but defect fixing no longer allowed.
    - Customer acceptance in the market.
    - Exit: reliability.
- Commonly used testing techniques:
    - Repeated random sampling without defect fixing.
      (≈ assumption for IDRMs, Ch.22.)
    - UBST with Musa or Markov OPs.
    - External testing services/organizations may be used for system "certification".

12.1.5 验收测试

- 主要特点：
  - 对象：整个系统。
    - 但不再允许修复缺陷。
  - 市场中客户的接受程度。
  - 出口：可靠性。
- 常用的测试技术：
  - 不修复缺陷的重复随机抽样。 （≈ 对于 IDRMs 的假设，见第 22 章。）
  - 使用 Musa 或 Markov 操作剖面的使用基于状态的测试（UBST）。
  - 可以使用外部测试服务/组织进行系统“认证”。

举例解释

验收测试是软件发布前的最后测试，以确保系统满足客户的需求并准备进入市场。以下是验收测试的具体应用示例：

1. 在一个企业资源规划（ERP）系统的实施项目中，验收测试可能由客户执行，以确保所有的业务流程在系统中都能正确无误地运行，并且满足合同中规定的性能标准。
2. **不修复缺陷的重复随机抽样**可能在一个电子商务平台上进行，测试团队会执行大量的订单处理操作来模拟现实环境中的系统负载，并注意系统在高负载下的表现。
3. 使用**Musa或Markov操作剖面的使用基于状态的测试（UBST）**，在一个在线教育平台的验收测试中，可能会模拟不同类型的用户（学生、教师、管理员）的行为，确保每个用户角色的体验都符合预期。
4. 对于重要系统，如医疗信息系统或汽车安全系统，通常需要**外部测试服务或组织**进行认证，以确保系统达到了特定的安全和质量标准，准备进行市场发布。

在验收测试阶段，不再关注缺陷的修复，而是评估产品是否达到了可发布的状态。这意味着此阶段发现的问题可能会导致推迟发布，或者需要在产品发布后通过补丁或更新来解决。这个阶段的测试通常由或与客户密切合作的测试人员进行，因为它的目标是评估产品是否满足了用户的实际业务需求和预期。

12.1.6 Beta Testing

- Key characteristics:
  - Object: whole system
  - Normal usage by customers.
  - Exit: reliability.
- Commonly used testing techniques:
  - Normal usage.
  - Ad hoc testing by customers. (trying out different functions/features)

- Diagnosis testing by testers/developers to fix problems observed by customers.

12.1.6 Beta 测试

- 主要特点：
  - 对象：整个系统
  - 客户的正常使用。
  - 出口：可靠性。
- 常用的测试技术：
  - 正常使用。
  - 客户进行的即兴测试（尝试不同的功能/特性）。
  - 测试人员/开发人员进行的诊断测试，以修复客户观察到的问题。

举例解释

Beta 测试是在产品发布前的最终测试阶段，通常由实际用户在真实环境中进行，以确保产品在实际使用中的表现符合预期。以下是 Beta 测试的具体应用示例：

1. 如果有一款新的社交媒体应用即将上市，公司可能会选择一群目标用户进行 Beta 测试，这些用户将使用该应用的所有功能，比如发布更新、添加好友和私信，以确保功能在日常使用中无误。
2. 在 Beta 测试阶段，**客户进行的即兴测试**可能包括随机使用不同的应用功能，如在一个新的文件分享服务中上传和下载文件，以测试其性能和稳定性。
3. 在测试期间，如果用户报告了问题，**诊断测试**会由开发团队执行，以确定问题的原因并解决它。例如，在游戏软件的 Beta 测试中，如果玩家报告了一个关于图形显示的错误，开发者将进行诊断测试以查找并修复这个问题。

Beta 测试的主要目的是收集最终用户的反馈，识别可能在之前的测试阶段中未被捕捉到的问题，确保产品的质量，并最终确定产品是否准备好进行全面发布。这个阶段的反馈对于确定用户体验的质量、揭示操作中的问题以及了解产品在市场中的实际表现至关重要。

12.1.7 Testing Sub-Phases: Comparison

- Key characteristics for comparison:
  - Object and perspectives.
  - Exit criteria.
  - Who is performing the test.
  - Major types of specific techniques.
- “Who” question not covered earlier:
  - Dual role of programmers as testers in unit testing and component testing I.
  - Customers as testers in beta testing.
  - Professional testers in other sub-phases.

- Possible 3rd party (IV&V) to test reusable components & system acceptance.

12.1.7 测试子阶段：对比

- 对比的关键特点：
  - 对象和视角。
  - 出口标准。
  - 谁在执行测试。
  - 主要的特定技术类型。
- 之前未提及的“谁”问题：
  - 程序员在单元测试和组件测试 I 中充当测试员的双重角色。
  - 在 Beta 测试中，客户作为测试者。
  - 在其他子阶段中，专业测试人员。
  - 可能由第三方（独立验证和验证 IV&V）来测试可重用组件和系统验收。

对比说明

各测试子阶段的关键对比点可以总结如下：

1. **对象和视角**：
   - **单元测试**和**组件测试 I**：由开发者进行，关注单个功能或类。
   - **集成测试**：测试组合后组件间的接口。
   - **系统测试**：从用户角度审视整个系统。
   - **验收测试**：确认系统是否满足用户需求，通常由用户或客户进行。
   - **Beta 测试**：由实际用户在真实环境中测试。
2. **出口标准**：
   - **单元测试**和**组件测试 I**：通常是代码覆盖率。
   - **集成测试**：可能是接口覆盖率。
   - **系统测试**：关注功能性、性能、安全性等。
   - **验收测试**：用户满意度和合同要求。
   - **Beta 测试**：产品稳定性和用户反馈。
3. **谁在执行测试**：
   - **单元测试**和**组件测试 I**：程序员。
   - **集成测试**和**系统测试**：可能是内部专业测试团队。
   - **验收测试**：客户或用户。
   - **Beta 测试**：最终用户。
   - **组件测试 II**和**系统验收**：有时由第三方（如IV&V）进行。
4. **主要的特定技术类型**：
   - **单元测试**：白盒测试技术，如控制流和数据流测试。

> - **组件测试**：根据组件类型，使用白盒或黑盒测试技术。
> - **集成测试**：关注于有限状态机等交互测试技术。
> - **系统测试**：综合使用多种测试技术，包括使用基于状态的测试。
> - **验收测试**：通常是黑盒测试，专注于系统的外部行为。
> - **Beta 测试**：主要依赖用户的正常使用和反馈。
>
> 每个子阶段都有其特定的目的和方法，有效地应用这些测试技术需要考虑测试对象、测试执行者以及测试的目的和预期成果。第三方测试（IV&V）可能在任何一个阶段被引入，特别是在对高安全性、高可靠性或有严格合规要求的系统进行测试时，以保证客观性和彻底性。

12.1.8 Testing Sub-Phases: Summary

- Summary: Table 12.1 (p.209)

Sub-phase	Persp.	Stopping	Who	Tech.
unit	WBT	coverage	programmer	db, s-list, BT, CFT, DFT
component				
type-I	WBT	coverage	programmer	s-list, BT, CFT, DFT
type-II	BBT	both	tester/3p	BT, CFT, DFT
integration	WBT	coverage	tester	FSM, CFT, DFT
system	BBT	both	tester	f-list, FSM, Musa, Markov
acceptance	BBT	usage	tester/3p	Musa, Markov
beta	BBT	usage	customer	normal usage

> 这个总结为软件开发中的各种测试阶段提供了概览，为如何进行每个阶段的测试、何时视为完成以及谁应负责测试活动提供了指导。对于团队来说，这是一个有助于分配责任和选择适合其软件产品的测试方法的实用资源。

12.2 Specialized Testing

- Specialized testing tasks:
  - Some do not fit into specific sub-phases.
  - Different goals (other than reliability).
  - Non-standard application environment.
- Our coverage:
  - Defect diagnosis testing.
  - Defect-based testing.
  - Regression testing.
  - Testing beyond programs.
  - Testing for other goals/objectives.

12.2 专项测试

- 专项测试任务：
  - 有些并不适合特定的子阶段。
  - 目标不同（不仅仅是可靠性）。
  - 非标准的应用环境。
- 我们的覆盖范围：
  - 缺陷诊断测试。
  - 基于缺陷的测试。
  - 回归测试。
  - 超越程序的测试。
  - 针对其他目标/目的的测试。

举例解释

专项测试是软件测试的一个领域，它聚焦于特定目的的测试，有时候并不属于传统的软件测试子阶段。以下是一些专项测试的具体示例：

1. **缺陷诊断测试**可能在软件发现缺陷后立即进行，目的是确定缺陷的具体原因。例如，如果一个视频游戏崩溃，开发团队会进行诊断测试以确定是内存管理问题、图形渲染错误，还是其他问题。
2. **基于缺陷的测试**是根据先前发现的缺陷特点来设计测试用例。例如，如果发现电子表格软件在处理特定格式的文件时存在问题，测试将专注于这类文件的进一步测试。
3. **回归测试**通常在软件更新后进行，以确保新的代码没有破坏旧的功能。在移动应用的更新中，开发者可能会运行一个测试套件，确保新功能的添加没有引入任何新的错误。

> 4. **超越程序的测试**可能包括软件与硬件的交互、用户界面的可用性测试，甚至是整个系统的安全性评估。例如，在智能家居系统中，测试可能包括设备通讯的可靠性和用户控制界面的直观性。
> 5. **针对其他目标/目的的测试**可能包括性能测试、安全性测试、兼容性测试等。举例来说，对于在线购物平台，性能测试可能关注网站在高流量下的响应时间，而安全性测试则确保交易数据的加密和保护。
>
> 专项测试是为了满足特定的测试需求而设计的，通常需要定制的方法和工具，有时候还需要特殊的技能和经验。这些测试不仅仅是为了确认软件的可靠性，而是为了确保软件能够在各种条件下满足用户的预期和行业标准。

12.2.1 Defect Diagnosis Testing

- Context of defect diagnosis testing:
  - In followup to discovered problems by customers or during testing.
  - Pre-test: understand/recreate problems.
  - Test result: faults located.
  - Followup with fault removal and re-run/re-test to confirm defect fixing.
- Defect diagnosis testing:
  - Typically involve multiple related runs.
  - Problem recreation as the starting point.
  - Perturbation and observation.
  - Domain knowledge important.
  - More recorded defect information ⇒ less reliance on defect diagnosis.
  - Defect-based techniques (below) useful.

> ## 12.2.1 缺陷诊断测试
>
> - 缺陷诊断测试的上下文：
>   - 跟进客户发现的问题或在测试过程中发现的问题。
>   - 测试前：理解/重现问题。
>   - 测试结果：定位到故障。
>   - 后续操作包括移除故障并重新运行/测试以确认缺陷修复。
> - 缺陷诊断测试：
>   - 通常涉及多次相关运行。
>   - 问题重现作为起点。
>   - 干扰和观察。
>   - 领域知识很重要。
>   - 记录的缺陷信息越多 ⇒ 对缺陷诊断的依赖越小。
>   - 基于缺陷的技术（见下文）很有用。

> **举例解释**
>
> 缺陷诊断测试是在软件开发过程中非常重要的一环，通常在用户反馈或测试过程中发现问题后执行。例如：
>
> 1. 假如一个客户报告他们的电子邮件应用在尝试打开某个特定附件时崩溃，缺陷诊断测试的第一步是重现这个问题。测试人员将尝试使用相同的附件和相同的操作来看看是否能够触发崩溃。
> 2. 测试人员可能需要运行多个测试，使用不同的数据集和操作序列来定位问题。例如，如果问题似乎与内存使用有关，测试人员可能会监控内存使用情况，当应用崩溃时检查内存状态。
> 3. 如果一个在线支付系统在处理交易时出现间歇性失败，测试人员可能需要对系统进行干扰测试，如模拟网络延迟或数据库连接失败，以观察系统的反应并定位问题。
> 4. 缺陷诊断测试的有效性在很大程度上取决于领域知识。熟悉业务逻辑和技术架构的测试人员可能更容易理解和识别问题的根源。
> 5. 当测试人员有足够的缺陷记录信息时，他们可以使用这些信息来快速定位新报告的问题是否与之前的缺陷相关联。记录详细的缺陷信息可以减少在后续问题诊断时的工作量。
> 6. 在使用基于缺陷的测试技术时，测试人员会专注于那些与已知缺陷相关的特定条件或代码路径，以确保缺陷被正确修复

12.2.2 Defect-Based Testing

- General idea and generic techniques:
  - Focus: discovered or potential defects (and related areas).
  - Ad hoc testing based on defect guesses.
  - Risk identification ⇒ risk-based testing. (Part IV, esp. Ch.21)
  - Defect injection and mutation testing.
- Defect injection and testing:
  - Inject known defect (seed known fault).
  - Test for both seeded and ingenuous faults.
  - Missed faults ⇒ testing technique↑.
  - Also used in reliability modeling.
- Mutation testing ≈ defect injection testing, but systematic mutants used.

> ## 2.2.2 基于缺陷的测试
>
> - 一般理念和通用技术：
>   - 焦点：发现的或潜在的缺陷（及相关区域）。
>   - 基于缺陷猜测的即兴测试。
>   - 风险识别 ⇒ 基于风险的测试。（尤其是第四部分，第21章）
>   - 缺陷注入和变异测试。
> - 缺陷注入和测试：
>   - 注入已知缺陷（植入已知故障）。

    - 同时测试植入的和自然发生的故障。
    - 错过的故障 ⇒ 提升测试技术。
    - 也用于可靠性建模。
- 变异测试 ≈ 缺陷注入测试，但使用系统化的变异体。

> **举例解释**
>
> 基于缺陷的测试专注于利用已知的缺陷信息来指导测试活动，以发现和修复类似的或相关的问题。例如：
>
> 1. 如果在一个电商网站中发现了一个涉及价格计算的缺陷，基于缺陷的测试可能会专门测试所有涉及价格计算的代码路径，包括打折、税费计算和货币转换等功能。
> 2. **缺陷注入**涉及故意在软件中引入已知的缺陷，然后运行测试以确保测试可以发现这些故障。例如，开发者可能会在代码中故意引入一个数组越界的错误，然后运行测试套件看是否能够捕捉到这个问题。
> 3. **变异测试**是通过对软件的现有代码进行小的修改（称为“变异”）来模拟可能的错误，并确保测试能够发现这些“变异”导致的故障。这种测试检查软件测试套件的质量，确认它能够检测出微小的、故意引入的变化。
> 4. 如果在进行缺陷注入测试时测试套件没有发现植入的缺陷，这表明测试套件需要提高测试技术的效果，可能包括增加测试用例的数量、覆盖更多的路径或改进测试用例的设计。
> 5. 基于缺陷的测试也可以用于**可靠性建模**，通过观察测试中缺陷的发现率和类型来预测软件在实际使用中的可靠性。
>
> 基于缺陷的测试是一种高级的测试方法，它帮助开发团队更深入地理解软件中的潜在问题，并改进测试策略以便更好地捕捉和预防未来可能出现的错误。

12.2.3 Regression Testing

- Context of regression testing:
    - In software maintenance and support: – ensure change/⇒ negative impact.
    - In legacy software systems:
        - ensure quality of remaining functions,
        - during develop ment/product update,
        - new part ≈ new development,
        - focus: integration sub-phase & after.
    - Re-test to verify defect fixing as well as no unintended consequences.
- Regression testing techniques:
    - Specialized analysis of change: Δ-analysis.
    - Focused testing on (new) Δ-part.
    - Integration of old and new.

> 12.2.3 回归测试
>
> - 回归测试的上下文：
> - 在软件维护和支持中：确保改变不会导致负面影响。

- 在遗留软件系统中：
  - 确保剩余功能的质量，
  - 在开发/产品更新期间，
  - 新部分 ≈ 新的开发，
  - 焦点：集成子阶段及之后。
- 重新测试以验证缺陷修复以及没有不期望的后果。
- 回归测试技术：
- 对变更的专门分析：差异分析（Δ-analysis）。
- 针对（新的）差异部分的集中测试。
- 旧的和新的整合。

> **举例解释**
>
> 回归测试是确保在软件更新、修补或其他变更后，原有功能不受影响的重要测试过程。以下是回归测试的具体应用示例：
>
> 1. 假设一个CRM系统新增了一项功能，允许销售人员追踪潜在客户的社交媒体活动。在这个新功能发布之后，回归测试将确保已有的客户数据管理、销售报告等功能仍然正常工作。
> 2. 使用**差异分析（Δ-analysis）**，开发团队会检查新代码对旧版本的具体影响。例如，在一个在线银行系统中，如果修改了资金转账的代码，回归测试将集中在该功能和可能受影响的其他功能上。
> 3. 在添加了一些新的API端点到一个现有的Web服务之后，回归测试会包括对这些新端点的测试，同时也会测试现有端点以确保它们没有因为新添加的代码而出现问题。
> 4. 当旧的软件系统经历更新时，例如引入新的安全措施或用户界面的改进，回归测试的焦点通常是在集成阶段，确保新旧组件可以无缝集成。
>
> 回归测试是软件质量保证的关键组成部分，尤其是在多次迭代的软件开发周期中。它帮助捕捉到由于修改导致的任何新问题，同时确保原有的功能仍然如预期那样工作。

12.2.4 Other Specialized Testing

- Testing beyond programs:
  - Embedded and heterogeneous systems:
    - test interactions with surroundings.
  - Web testing, in case study later.
- Testing to achieve other goals:
  - Performance testing;
  - Stress testing;
  - Usability testing, etc.
- Dynamic analysis and related techniques:
  - Simulation to reduce overall cost.
  - Prototyping, particularly in early phases.

- Timing and sequencing analysis.
- Event-tree analysis (ETA), Chapter 16.

12.2.4 其他专项测试

- 超越程序的测试：
  - 嵌入式和异构系统：
    - 测试与环境的交互。
  - 网页测试，稍后案例研究。
- 实现其他目标的测试：
  - 性能测试；
  - 压力测试；
  - 可用性测试等。
- 动态分析及相关技术：
  - 模拟以降低总成本。
  - 尤其是在早期阶段的原型制作。
  - 时间和顺序分析。
  - 事件树分析（ETA），第16章。

举例解释

超出传统软件程序测试范畴的其他专项测试包括：

1. 在测试**嵌入式和异构系统**时，比如智能手表或自动驾驶汽车，测试不仅仅局限于软件本身，还需要确保软件与硬件、网络和物理环境正确交互。例如，自动驾驶汽车的软件需要在实际的驾驶环境中进行测试，以确保其感应器能够正确响应真实世界的条件。
2. **网页测试**可能包括检查网站在不同浏览器和设备上的兼容性、性能、以及如何处理大量用户请求。例如，电子商务网站在大型促销活动中的测试，以确保它能够处理高流量和交易压力。
3. **性能测试**关注软件在特定工作负载下的表现，例如测试数据库的响应时间和事务处理速率。
4. **压力测试**则是将系统推至极限，以观察其在极端条件下的表现，例如模拟超过正常操作范围的用户数量来测试一个在线服务。
5. **可用性测试**评估软件的用户友好程度，例如通过用户测试会话来看用户完成特定任务的容易程度。
6. **动态分析**和相关技术，如**模拟**，可以用于在不产生实际成本的情况下评估系统的行为。这对于大型系统特别有用，因为它可以在构建整个系统之前识别潜在问题。
7. **原型制作**是早期阶段常用的技术，允许开发团队快速构建并测试软件的关键方面，以便收集用户反馈并改进产品。
8. **时间和顺序分析**以及**事件树分析（ETA）**是评估系统如何在不同条件下响应特定事件序列的技术。例如，安全关键系统的测试会使用这些技术来评估在发生硬件故障或安全漏洞时的系统行为。

这些专项测试技术为软件测试提供了一种全面的方法，不仅考虑代码正确性，还包括整个系统的行为和性能以及用户体验。这些方法通常与传统的测试方法相结合，以确保在所有方面都满足用户和市场的需求。

12.3 Test Integration

- General idea:
  - Many activities and tasks.
  - Different techniques.
  - Individual advantages and limitations.
  - Much commonality exists.
  - Possibility of integration?
- Test integration: Advantages
  - combined strength ⇒ benefit↑.
  - common elements ⇒ cost↓.
  - flexibility↑.

> ## 12.3 测试整合
>
> - 基本概念：
>   - 许多活动和任务。
>   - 不同的技术。
>   - 各有优势和限制。
>   - 存在许多共性。
>   - 是否有整合的可能?
> - 测试整合的优势：
>   - 结合优势 ⇒ 效益提升。
>   - 共同元素 ⇒ 成本降低。
>   - 灵活性提高。

> **举例解释**
>
> 测试整合的目的是将不同的测试活动和技术结合起来，以充分利用每种技术的优点，弥补单一技术的不足，并提高测试过程的效率。例如：
>
> 1. 在一个软件开发项目中，开发团队可能会整合单元测试和集成测试的方法。在单元测试中，他们可能使用白盒测试来测试代码的具体实现。而在集成测试中，则可能使用黑盒测试来验证不同组件的交互。
> 2. 在进行系统测试时，团队可能会整合性能测试和安全性测试。例如，可以在一次运行中测试软件的响应时间（性能测试）和其对恶意攻击的抵抗力（安全性测试）。
> 3. 回归测试可以整合变更分析和自动化测试。例如，每当软件代码发生变更时，变更分析可以确定哪些部分可能受到影响，而自动化测试则可以迅速运行那些部分的测试用例。
> 4. 在软件维护阶段，可以整合基于风险的测试和缺陷诊断测试。当软件需要更新时，基于风险的测试可以帮助确定哪些区域最可能出现问题，而缺陷诊断测试则可以用于解决这些问题。

> 通过整合不同的测试技术和方法，测试团队可以更高效地工作，减少冗余工作，加速测试过程，同时提高软件产品的整体质量。整合测试还意味着测试活动可以更灵活地适应项目需求的变化，以及软件开发周期的不同阶段。

12.3.1 Hierarchical Web Testing

- Case study from Chapter 10 continued:
  - Web navigation modeled by FSMs.
  - UBST using UMMs to overcome state explosion problem of FSMs.
  - Guiding existing web testing. (they typically focus on a small unit/facet)
  - Lack of structure for overall hits ⇒ use of simplified OPs (Musa OPs)
- Overall approach:
  - Top-tier: flat (Musa) OP.
  - Middle-tier: UMMs.
  - Bottom-tier: existing web testing.

12.3.1 分层网页测试

- 第10章案例研究继续：
  - 网页导航通过有限状态机（FSMs）建模。
  - 使用统一建模方法（UMMs）进行使用基于状态的测试（UBST），以克服FSMs的状态爆炸问题。
  - 指导现有的网页测试。（它们通常聚焦于小单元/方面）
  - 整体点击缺乏结构 ⇒ 使用简化的操作剖面（Musa OPs）
- 总体方法：
  - 顶层：平面式（Musa）操作剖面。
  - 中层：UMMs。
  - 底层：现有的网页测试。

> **举例解释**
>
> 分层网页测试是一种结构化的测试方法，它将不同级别的测试整合在一起，以提供一个全面的网页测试策略。例如：
>
> 1. **顶层测试**关注于整个网站的用户行为模式。使用Musa操作剖面来模拟用户的访问模式和行为，比如网站上最常被访问的页面和用户行为的流量分布。
> 2. **中层测试**利用UMMs来处理复杂的网站状态。UMMs可以帮助测试人员理解在网站的不同部分之间导航时的可能状态和转换，这有助于识别在复杂交互中可能出现的问题。
> 3. **底层测试**包括对网站的具体单元或功能进行测试。这可能涉及到测试表单的提交，验证链接是否正确，或者确保在不同浏览器和设备上的兼容性。

> 在分层网页测试中，顶层的测试帮助定义测试的重点区域和优先级，中层提供了一种方式来有效地组织和管理测试活动，而底层则保证了具体功能的质量。通过这种分层方法，测试团队能够系统地覆盖从整个网站的用户体验到具体技术细节的各个方面。这种方法既可以节省时间和资源，也可以确保问题被全面地识别和解决。

12.3.2 Existing Web Testing

- Web functionality testing:
  - Focus on the web components identified in Ch.10.
  - HTML syntax checking via various tools.
  - Link checking.
  - Form testing.
  - Verification of end-to-end transactions.
  - Java and other program testing.
- Beyond web functionality testing:
  - Load testing.
  - Usability testing.
  - Browse rendering.

> ## 12.3.2 现有的网页测试
>
> - 网页功能性测试：
>   - 关注于第10章中识别的网页组件。
>   - 通过各种工具进行HTML语法检查。
>   - 链接检查。
>   - 表单测试。
>   - 端到端交易的验证。
>   - Java及其他程序的测试。
> - 超出网页功能性测试范畴：
>   - 负载测试。
>   - 可用性测试。
>   - 浏览器渲染。

> **举例解释**
>
> 现有的网页测试涉及到多种测试类型，专注于确保网页的各个方面都按预期工作。例如：
>
> 1. **HTML语法检查**是基础的网页测试，确保网页代码遵循正确的标准，以便能在各种浏览器上正确显示。这可以通过W3C验证服务或类似的工具来完成。

> 2. **链接检查**是确保网页上所有的链接都是活动的，没有导向错误页面或者死链的过程。这通常通过自动化工具来进行，可以快速识别和修复断链。
> 3. **表单测试**确保用户输入的数据正确处理，并且所有的表单元素如输入框、下拉菜单、复选框等都能正确工作。
> 4. **端到端交易验证**涉及到模拟用户完成一个典型的业务流程，如在电子商务网站上从选购商品到完成支付的过程，确保整个流程无缝、安全地执行。
> 5. **Java及其他程序的测试**涉及到对网页内嵌程序的功能性和安全性测试，确保它们不会导致网页崩溃或其他意外行为。
>
> 在功能性测试之外，也需要进行其他类型的测试：
>
> 1. **负载测试**模拟大量用户同时使用网站的情况，测试网站的性能和稳定性，确保在高流量下依然能保持响应速度。
> 2. **可用性测试**关注用户界面的设计和用户体验，如导航的简便性、内容的易读性等。
> 3. **浏览器渲染**测试确保网页在不同的浏览器和设备上都能正确呈现，包括移动设备和桌面设备。
>
> 通过结合这些测试技术，开发团队可以确保网站在技术上是健全的，同时也提供了良好的用户体验。

12.3.3 Web Testing (from Ch.10)

- Testing web navigations:
  - FSM-based testing in Chapter 10.
  - Web crawling via robots.
- UMMs for web testing (Chapter 10).
  - Availability/usage of web logs.
  - Some observations:
    - skewed top hit pages and x-references
    - the impact of structural hierarchy

> ## 12.3.3 网页测试（来自第10章）
>
> - 测试网页导航：
>   - 第10章中基于有限状态机（FSM）的测试。
>   - 通过机器人进行网页爬取。
> - 用于网页测试的统一建模方法（UMMs）（第10章）：
>   - 网页日志的可用性/使用。
>   - 一些观察结果：
>   - 偏斜的顶部点击页面和交叉引用
>   - 结构层次的影响

> **举例解释**
>
> 网页测试尤其关注于网站的导航和结构，确保用户可以有效地找到他们需要的信息，并且网页之间的链接正确无误。以下是针对网页导航和结构测试的一些方法和观察：
>
> 1. **基于有限状态机（FSM）的测试**是一种检验网页导航逻辑的方法。通过建立网站导航的状态机模型，测试人员可以确保从一个页面到另一个页面的所有可能路径都被覆盖，并且每个转换都按照预期工作。例如，确保从主页到产品页面再到购物车的路径是可行的，并且每个步骤都符合设计。
> 2. **网页爬取**技术通过使用自动化脚本（如爬虫机器人）来遍历网站的所有链接，检查死链或无效链接，以及页面加载速度。这有助于快速发现链接问题或内容缺失的问题。
> 3. **统一建模方法（UMMs）**在网页测试中利用网站的实际使用数据（如网页日志）来指导测试活动。通过分析网页日志，测试人员可以发现哪些页面被频繁访问，哪些交叉引用最常见，以及用户的行为模式。这些信息可以用来确定测试的重点区域，特别是那些对用户来说最重要的页面和功能。
> 4. 对网页日志的**观察**可以揭示某些页面（如首页或热门产品页面）有极高的访问量，而其他页面则访问量较少。这种偏斜的访问模式提示测试人员需要重点关注那些高访问量的页面。同时，网站的结构层次如何影响用户的导航和页面间的链接也是一个重要考量。
>
> 通过结合这些方法和观察，网页测试可以更有针对性地进行，从而提高测试的有效性和效率。这有助于确保网站不仅在技术上无误，而且能够满足用户的需求和期望。

12.3.4 Hierarchical Web Testing

- Overall approach:
  - Top-tier: flat (Musa) OP
    - for simplicity and skewed distribution.
  - Middle-tier: UMMs
    - importance of highly used navigations.
  - Bottom-tier: existing web testing – no need to re-invent wheels
- Implementation support:
  - TAR (top access report) ⇒ top-tier
  - CPR (call-pair report) to form clusters

    ⇒ middle tier UMMs
- UMM refinement ⇒ bottom-tier
  - low-level Markov chains and
  - traditional (WBT-)testing models
- Implementation of the hierarchical web test- ing strategy: Fig 12.2 (p.218)

Top Level	Top Access Report (TAR) a flat list of frequently accessed services in ranking order (may be grouped by interconnection in customer usage scenarios)
Middle Level	Unified Markov Models (UMMs) for groups of TAR entries linked by CPR (call-pair report) (may be expanded into lower-level UMMs or other models)
Bottom Level	Detailed UMMs/other Models associated with frequently visited or critical nodes of UMMs (may correspond to testing models other than UMMs)

12.3.4 分层网页测试

- 总体方法：
  - 顶层：平面式（Musa）操作剖面
    - 出于简化和偏斜分布的考虑。
  - 中层：统一建模方法（UMMs）
    - 高频使用导航的重要性。
  - 底层：现有的网页测试 - 无需重新发明轮子
- 实施支持：
  - TAR（顶级访问报告）⇒ 顶层
  - CPR（调用对报告）形成聚类

    ⇒ 中层UMMs
- UMM细化 ⇒ 底层
  - 低级马尔可夫链和
  - 传统的（基于白盒测试的）测试模型
- 分层网页测试策略的实施：图12.2（第218页）

举例解释

分层网页测试策略通过结合不同层次的测试方法，旨在高效地覆盖网站的测试需求。以下是该策略的具体实施示例：

1. **顶层**使用Musa操作剖面来确定哪些网页或功能由于用户访问频率高而成为测试的重点。例如，电子商务网站的首页、热门产品页面和搜索功能可能是顶层测试的焦点，因为它们对用户体验至关重要。

2. **中层**使用UMMs来分析和测试网站的主要导航路径。这可以基于用户行为数据，如网站日志，来确定哪些导航路径被频繁使用。在这一层级，可以对电子商务网站的购物流程（从选购到结账）进行聚焦测试。
3. **底层**依托现有的网页测试技术和框架来进行具体的功能性和界面测试，比如链接检查、表单验证和跨浏览器测试等。这一层不需要开发新的测试方法，而是利用现有的工具和技术来确保网页的基本功能按预期工作。

通过TAR和CPR报告，测试团队可以有效地识别需要重点测试的网页和导航路径。TAR报告揭示了访问量最高的页面，而CPR报告帮助测试人员理解用户如何从一个页面导航到另一个页面，这些信息对于制定测试策略至关重要。

综合使用这些分层方法，可以使测试活动更有目标性，提高测试的效率和覆盖率，同时确保网站的主要功能和导航路径满足用户的期望和需求。