# 一 题型

- 单选题：15分
- 判断题：10分
- 填空题：10分
- 简答题：7道题共65分

# 二 复习要点

## 1. 软件测试理论

1. 测试的目的
    i. 发现缺陷：通过执行测试用例，识别软件中存在的错误、漏洞或不符合需求的地方。
    ii. 验证功能：确保软件的功能符合需求规格说明，即“做对了该做的事”。
    iii. 验证性能与可靠性：评估软件在特定条件下的响应时间、负载能力、稳定性等是否达到预期。
    iv. 评估软件质量：通过测试结果判断软件的整体质量，为发布决策提供依据。
    v. 预防缺陷：通过早期测试（如需求评审、设计评审）减少后期修复成本。
    vi. 提供信息支持：为开发、产品、管理等团队提供关于软件状态的客观反馈。
    vii. 符合标准与规范：确保软件满足行业标准、法规或合同要求。
2. 测试流程
    i. 在软件工程中，测试流程是一套系统化、结构化的活动序列，旨在验证软件是否符合需求、发现缺陷并评估质量。通常包括以下主要阶段：
    ii. 需求分析：
        a. 理解需求文档，明确测试范围与目标。
        b. 确定哪些功能需要测试、哪些不需要。
    iii. 测试计划
        a. 制定《测试计划》文档，明确：
            a. 测试目标、范围、策略
            b. 资源分配（人员、工具、环境）
            c. 时间安排、里程碑
            d. 风险分析与应对措施
    iv. 测试设计
        a. 设计测试用例，覆盖功能、性能、安全等。
        b. 准备测试数据，搭建测试环境。

c. 评审测试用例，确保完整性。

v. 测试执行

a. 执行测试用例，记录测试结果。

b. 发现缺陷后，提交缺陷报告（含步骤、预期、实际结果等）。

c. 进行回归测试，确保修复未引入新问题。

vi. 缺陷跟踪与管理

a. 使用工具（如JIRA、禅道）跟踪缺陷状态（新建、修复、验证、关闭）。

b. 与开发团队协作，推动缺陷修复。

vii. 测试报告与总结

a. 汇总测试结果，编写《测试报告》。

b. 评估软件质量，给出发布建议。

c. 总结测试经验，改进后续流程。

viii. 常见模型中的测试流程：

a. V模型：测试活动与开发阶段并行对应（单元测试→集成测试→系统测试→验收测试）。

b. 敏捷模型：测试贯穿每个迭代，持续集成与测试，强调自动化。

c. 迭代模型：每个迭代周期都包含完整的测试流程。

ix. 测试流程的关键特点

a. 系统性：有序进行，避免遗漏。

b. 可重复性：确保测试结果可靠。

c. 适应性：可根据项目类型（如Web、移动端）调整。

3. 测试环境

i. 在软件测试中，测试环境 是指一个为执行软件测试而专门设置和配置的独立空间。它是软件、硬件、网络配置、数据以及其他支持工具的集合体，目的是模拟真实的生产（线上）环境，以便测试人员在其中验证软件的功能、性能、安全性和兼容性。

ii. 你可以把它想象成一个受控的“实验舞台”，演员（软件）在这里进行彩排，所有灯光、音效、布景（环境）都尽可能接近正式演出的现场，但不会影响真正的观众。

iii. 测试环境的核心组成要素，一个完整的测试环境通常包括以下“四大支柱”：

a. 硬件：

a. 服务器： 用于部署被测应用程序的后端服务、数据库等。可能是物理服务器、虚拟机（VM）、容器（如Docker）或云服务器实例（如AWS EC2）。

b. 客户端/终端设备： 用于测试前端界面，包括不同型号的电脑（Windows, macOS, Linux）、手机（iOS, Android）、平板、浏览器等。

b. 软件：

a. 被测软件： 待测试的应用程序本身，通常是某个特定版本（Build）。

b. 操作系统： 服务器和客户端上安装的操作系统（如 Windows Server, Ubuntu, iOS, Android）。

c. 中间件/依赖软件： 应用程序运行所必需的第三方软件，如 Web 服务器（Apache, Nginx）、应用服务器（Tomcat, Node.js）、运行时环境（Java JRE, .NET

Framework）等。

d. 数据库： 特定版本的数据库系统（如 MySQL, PostgreSQL, MongoDB），并填充有测试数据。

c. 网络配置：

a. 模拟真实用户的网络条件，如局域网（LAN）、互联网访问、特定的网络拓扑、防火墙规则、带宽限制或延迟设置。

d. 测试数据：

a. 这是测试环境的“血液”。它是专门为测试用例准备的、覆盖各种场景的数据集，包括：

a. 有效数据（验证正常功能）

b. 无效数据（验证异常处理）

c. 边界值数据

d. 大量数据（用于性能测试）

b. 重要原则： 测试数据应与生产数据脱敏（去除敏感信息）后使用，或完全独立生成，绝不能直接使用真实用户数据。

4. 黑盒测试

i. 黑盒测试是一种软件测试方法，测试人员仅依据软件的需求规格说明或功能设计，将程序视为一个不透明的“黑盒子”，在不了解其内部结构、实现细节和代码逻辑的情况下，对软件的功能进行测试。

ii. 核心特点：

a. 测试依据：需求文档、功能规格说明书、用户手册等。

b. 测试对象：软件的外部行为（输入与输出）。

c. 测试人员：无需具备编程知识，但需深入理解用户需求。

d. 测试目标：验证功能是否满足需求，发现功能错误、界面问题、性能缺陷等。

iii. 主要方法：

a. 等价类划分

a. 定义：将输入数据划分为若干“等价类”，从每类中选取代表值进行测试，认为同类数据测试效果等效。

b. 要点：覆盖有效与无效输入，减少用例数量，适用于有明确范围或规则的输入。

b. 边界值分析

a. 定义：针对输入或输出范围的边界值及其附近值进行测试，因为错误常出现在边界。

b. 要点：测试最小值、略高于/低于最小值、正常值、最大值等，适用于数值型或计数型输入。

c. 因果图与决策表定义：

a. 因果图：分析多个输入条件（因）与输出结果（果）之间的逻辑关系。

b. 决策表：将因果图转化为表格，列出所有条件组合及对应动作。

c. 要点：解决多条件组合测试问题，确保逻辑覆盖完整，适用于复杂业务规则。

d. 正交实验法

a. 定义：利用正交表选取具有“均匀分散”特性的部分组合进行测试，高效覆盖多因素交互。
b. 要点：减少全面组合的用例数，适用于配置测试或多参数组合场景（如浏览器、操作系统组合）。

e. 场景法
a. 定义：基于用户实际使用场景或业务流程设计测试用例，模拟端到端操作流程。
b. 要点：关注基本流（主流程）和备选流（异常分支），贴近用户真实操作，适用于验收测试和流程测试。

f. 状态迁移法
a. 定义：将系统视为有限状态机，通过覆盖不同状态间的转换路径来设计测试用例。
b. 要点：适用于状态明确的对象（如订单状态、播放器控制），重点测试状态转换逻辑和异常回退。

5. 软件质量保证
i. 在软件测试专业课程中，软件质量保证（Software Quality Assurance，简称SQA） 是一个系统性、有计划的流程活动，其核心目标是确保软件开发过程与所采用的方法、标准和规程相符，从而预防缺陷的产生，最终交付高质量的软件产品。
ii. 核心要点：
a. 预防为主：SQA 侧重于在缺陷发生前通过改进和规范过程来预防问题，而非仅仅在后期发现缺陷。
b. 过程导向：它关注整个软件开发生命周期（SDLC）中的流程、方法和标准的符合性，确保过程被正确执行。（简而言之，SQA是通过管理和改进开发过程来保证软件质量的整套体系性工作。）
c. 管理职能：SQA 通常是一个独立于项目团队的职能，负责过程审计、评审、标准制定、培训和改进建议。
d. 与软件测试的关键区别：
a. 软件测试 是 “检测” 活动，旨在发现软件产品中存在的缺陷（关注结果/产品）。
b. 软件质量保证 是 “预防” 活动，旨在确保用来构建软件的过程是可靠的（关注过程）。
e. SQA的主要活动包括：制定质量计划与标准、过程评审与审计、产品评审、质量度量与分析、促进持续过程改进等。

6. 软件生命周期
i. 软件生命周期 是指一个软件产品从概念构思开始，到最终废弃消亡为止，所经历的一系列阶段性的、相互关联的过程的全过程。它描述了软件“从生到死”的完整旅程。
ii. V模型：强调测试与开发的对应关系。每个开发阶段（如需求分析）都有一个直接对应的测试阶段（如验收测试），将测试活动提前规划和准备。
iii. 迭代模型 & 增量模型：将整个项目分解为一系列小循环（迭代） 或小版本（增量）。每个迭代/增量都经历一个完整的迷你生命周期（分析、设计、编码、测试），逐步构建出最终产品。
iv. 敏捷模型（如Scrum, XP）：这是目前最主流的开发思想。它彻底拥抱变化，将生命周期浓缩为短周期（如2-4周）的迭代（称为Sprint）。

7. 软件测试原则（比如：二八定理，百分之80的缺陷，集中在20的模块中）
    i. 书上的
        a. 测试应基于用户需求
        b. 测试要尽早进行
        c. 不能做到穷尽测试
        d. 遵守GoodEnough原则
        e. 测试缺陷要符合“二八”定理
        f. 避免缺陷免疫
    ii. ISTQB和IEEE标准的专业术语
        a. 测试显示缺陷的存在：测试只能证明软件有缺陷，不能证明其没有缺陷。（上课讲过但是书上没有）
        b. 基于需求的测试/不存在缺陷的谬论：需求可测试性 和 基于需求的测试策略
        c. 早期测试/测试左移：在软件开发生命周期中尽早测试
        d. 穷举测试的不可能性：由于输入、路径和状态的组合无限，无法覆盖所有可能场景，需通过风险评估确定测试优先级。
        e. 足够好原则：测试投入应与项目风险、时间及资源约束平衡，追求“足够好”而非“完美”的覆盖。
        f. 缺陷集群效应/帕累托原则：约80%的缺陷集中在20%的模块中，测试资源应聚焦于高风险及复杂功能区域。
        g. 避免缺陷免疫/杀虫剂悖论：重复相同的测试用例，就像重复使用同一种杀虫剂，会发现的新缺陷将越来越少
    iii. ① 诊断需对照症状（用户需求） → ② 早诊断早治疗（尽早测试） → ③ 无法检查所有细胞（无法穷尽） → ④ 根据病情决定检查项目（GoodEnough） → ⑤ 疾病常集中在某个器官（缺陷集群） → ⑥ 病毒会变异，需更新检测方法（杀虫剂悖论）
8. 测试用例设计
    i. 测试用例设计是创建一组具体、可执行测试步骤的系统化过程，目的是为了验证软件是否满足特定需求。
    ii. 一个标准的测试用例通常包含：
        a. 用例编号/名称：唯一标识
        b. 测试目标：要验证什么功能
        c. 前置条件：执行前需满足的状态（如用户已登录）
        d. 测试步骤：详细的操作描述
        e. 测试数据：具体的输入值
        f. 预期结果：软件应有的正确响应
        g. 实际结果：（执行后填写）
    iii. 核心目的：用最少量的、最具代表性的测试用例，最大程度地发现潜在缺陷，并有效地覆盖需求。
9. 测试分类（按照阶段分类、按照项目分类等）

i. 按测试阶段分类
   a. 单元测试：针对软件最小可测试单元（如函数、类）进行的测试。通常由开发人员执行，使用白盒测试方法。目的是验证代码单元是否符合设计和需求。
   b. 冒烟测试：在新版本构建完成后，对系统最基本、最核心的功能进行的初步测试。目的是确认软件的基本功能是否正常，能否进行后续更详细的测试。若失败，版本通常会被退回开发。
   c. 集成测试：在单元测试之后，将已测试的单元组合成模块、子系统或系统，测试它们之间的接口和交互是否正确。关注数据传递、接口调用和模块协同工作。
   d. 系统测试：将完整的软件系统置于实际或模拟的运行环境中，进行全面的测试。验证系统是否满足规格说明书的所有要求，包括功能、性能、安全性等。这是从用户角度进行的黑盒测试。
   e. 验收测试：在系统测试之后，由最终用户或客户进行的测试，以确定软件是否满足合同或用户需求，并决定是否接收该产品。主要包括用户验收测试（UAT）和合同验收测试。
ii. 按测试技术分类
   a. 黑盒测试：也称为功能测试或数据驱动测试。测试者将软件视为一个不透明的“黑盒”，无需了解内部结构，只关注输入与输出是否符合预期。测试依据是需求规格说明书。
   b. 白盒测试：也称为结构测试、逻辑驱动测试或透明盒测试。测试者需要了解程序内部逻辑结构，基于代码本身设计测试用例，检查程序内部路径、分支、条件等是否正确。
   c. 灰盒测试：介于黑盒和白盒之间。测试者了解系统部分内部结构（如接口定义、数据结构），并利用这些信息设计测试用例，但测试执行仍关注外部表现。常用于集成测试。
iii. 按软件质量特性分类
   a. 功能测试：验证软件的各项功能是否满足用户需求。涵盖准确性、适合性、互操作性、安全保密性、功能依从性等子特性。
   b. 性能测试：评估软件在特定条件下的性能表现。是一个广义范畴，包括：
      a. 负载测试：在正常和峰值负载下检查系统性能。
      b. 压力测试：在极限或超过容量的负载下测试系统稳定性和恢复能力。
      c. 稳定性/可靠性测试：系统长时间运行下的稳定性和无故障运行能力。
      d. 兼容性测试：检查软件在不同硬件、操作系统、网络环境、浏览器或与其他软件共存时的表现。
iv. 按自动化程度分类
   a. 人工测试：测试人员手动设计、执行并验证测试结果。优点在于灵活、智能，适合探索性测试、用户体验测试等。缺点是耗时、重复性工作易出错。
   b. 自动化测试：利用测试工具、脚本或框架自动执行测试用例，比较实际结果与预期结果。需要前期投入，但长期看能提高回归测试效率，适用于重复、机械的测试任务。性能测试、大规模回归测试常依赖自动化。
v. 按测试项目分类
   a. 界面测试(UI测试)：测试用户界面的布局、控件、美观度、易用性、一致性等是否符合设计规范和用户习惯。

b. 安全性测试：评估软件系统抵御恶意攻击、保护数据和功能安全的能力。包括漏洞扫描、渗透测试、权限验证、数据加密测试等。
c. 文档测试：验证与软件相关的文档（如用户手册、安装指南、在线帮助、API文档）的准确性、完整性、可读性和一致性。
d. 兼容性测试：如前所述，属于性能测试的一个子集，但常作为独立测试项目。
e. 安装/卸载测试：验证软件在不同环境下安装、升级、卸载过程的正确性和完整性。

vi. 其他分类（没法具体归类，但是测试行业中经常使用这些测试）
a. α测试（Alpha测试）：在软件开发接近完成时，在受控环境下由开发方组织内部用户或测试团队模拟用户行为进行的测试。目的是发现重大缺陷。
b. β测试（Beta测试）：在α测试之后，软件正式发布前，将产品交给真实的外部用户在实际使用环境中进行的测试。目的是获得用户真实反馈，发现潜在问题。测试环境不可控。
c. 回归测试：当软件被修改（如修复缺陷、增加新功能）后，重新执行之前的部分或全部测试用例，以确保修改没有引入新的缺陷（回归缺陷），且原有功能依然正常。这是软件维护阶段的核心测试活动。
d. 随机测试（探索性测试的一种）：不基于详细的测试用例，而是凭借测试人员的经验、直觉和对系统的理解，随机或自由地进行测试。旨在发现那些结构化测试可能遗漏的缺陷，是脚本测试的有效补充。

10. 测试用例设计方法
i. 核心思想：设计测试用例的目标是：用最少的用例，发现最多的缺陷。关键在于如何系统地、高效地“组合”或“选择”测试输入、操作和环境。
ii. 黑盒测试设计方法，基于软件外部规格说明，不关心内部实现。主要针对功能需求。
iii. 白盒测试设计方法：基于程序内部逻辑结构，主要针对代码覆盖率。
iv. 基于经验和直觉的测试设计方法

11. 回归测试的理解
i. 回归测试是指在软件修改（例如修复缺陷、新增功能、优化代码）后，重新执行之前已通过的测试用例，以确认这些修改没有引入新的缺陷（回归缺陷），并且没有破坏软件原有的、正常的功能。

12. 缺陷
i. 产生原因
a. 需求不明确
b. 软件结构复杂
c. 编码问题
d. 项目期限短
e. 使用新技术
ii. 软件缺陷的分类
a. 按照测试种类划分：按照测试种类可以将软件缺陷划分为界面缺陷、功能缺陷、性能缺陷、安全性缺陷、兼容性缺陷等。

b. 按照缺陷的严重程度划分：按照缺陷的严重程度可以将软件缺陷划分为严重缺陷、一般缺陷、次要缺陷、建议缺陷。
c. 按照缺陷的优先级划分：按照缺陷的优先级不同可以将软件缺陷划分为立即解决缺陷、高优先级缺陷、正常排队缺陷、低优先级缺陷。
d. 按照缺陷的发生阶段划分：按照缺陷的发生阶段不同可以将软件缺陷划分为需求阶段缺陷、架构阶段缺陷、设计阶段缺陷、编码阶段缺陷、测试阶段缺陷。

iii. 软件缺陷管理工具
a. Bugzila
b. 禅道
c. Jira

# 2. 自动化测试

1. 自动化测试工具
2. 自动化测试的目的
3. 自动化测试应用（iwebshop）

# 3. 接口测试

1. 接口测试工具
2. 接口测试的目的
3. 响应状态码
4. PostMan接口测试的应用

# 4. 性能测试

1. 性能测试类型
2. 性能测试指标
3. 性能测试应用