《软件测试课程设计》讲稿（第 10 章）

第 12 周 · 2 学时

广东金融学院 计算机学院 | 周宇文

讲稿可照读版 | 由课件逐页撰写

第 10 章　测试需求分析与测试计划

第 12 周 · 2 学时（90 分钟）

《软件测试课程设计》第 10 章 讲稿（第 12 周 · 2 学时 90 分钟）

> 课程：软件测试课程设计（154442008） | 广东金融学院 计算机学院 | 专业选修
>
> 章节：第 10 章 测试需求分析与测试计划 —— 先想清楚"测什么、测到什么程度"，再动手
>
> 学时：**2 学时（90 分钟）**
>
> 教材：朱少民《软件测试方法和技术（第 4 版）》第 3 篇 软件测试项目实践（开篇）
>
> 配套课件：逐页还原版 **58 页**（`lecture10/`，云端 http://111.231.0.226:2345/lecture10/index.html）
>
> 教案对应：第 12 次课「测试需求分析与测试计划」
>
> 本讲重点：测试目标与准则（进入/退出准则）、测试需求分析（输入输出、从需求推导测试项、优先级与风险）、测试计划要素、测试估算、计划评审与滚动维护
>
> 本讲难点：把"要测好"翻译成可验证的测试目标；测试项与需求的追溯；在信息不全时如何估算与排期
>
> **本讲稿为可照读完整版**：每页含【口播】【板书】【提问＋预设回答】【易错点/考点】【过渡】；带 ★ 的页为可压缩页。

一、教学目标

维度	目标（课后可检验）
知识	① 能区分测试目标与测试准则，写出可验证的测试目标；② 能说出进入准则与退出准则各写什么；③ 能描述测试需求分析的输入与输出；④ 能列出测试计划的必备要素；⑤ 能说明测试估算的常用方法与其局限
能力	① 能把一段需求拆成测试项并标出优先级与风险；② 能为课程设计项目写出一份含八要素的测试计划；③ 能对一份计划做评审并指出漏洞
素质（思政）	① 目标可验证＝严谨求实；② 准出准则＝对交付负责；③ 需求"问清楚"＝沟通与责任；④ 计划评审＝对承诺负责

本讲要建立的一个总印象：测试不是"拿到代码就开始点"，而是**先从需求里把"要验证什么"捞出来，再用一份计划把人力、时间、环境、判据排布好**。第 4 章讲规范该有哪些要素，这一章就是把那些要素落成文档。

二、时间分配总表

> **容量提示**：本章课件 **58 页**，讲稿为素材全量版（口播约 2.6 万字，≈110 分钟），课时 90 分钟。建议：目标与准则、需求分析、计划要素三块慢讲，示例页快带。

时段	页码	内容	分钟	备注
第 1 节	10.1–10.11	第 3 篇开篇、测试件、测试目标与准则、进入准则	24	**目标与准则必讲**
第 1 节	10.12–10.30	测试需求分析：输入输出、分析基础、推导测试项	26	**输入输出必讲**
——	——	**课间休息 10 分钟**	——	——
第 2	10.31–	优先级与风险、测试计划要素、估算与排期	22	估算方法必讲

节	10.48			
第 2 节	10.49–10.58	计划评审、滚动维护、小结与作业	18	当堂评审一份计划

> **压缩优先级**：先压缩示例与图示页 → 再压缩模板逐条讲解 → 最后才压缩目标与准则、需求分析输入输出、计划八要素、估算方法。

三、逐页讲稿

> 每页第一段是可直接朗读的口播；其后是板书、提问与预设回答、易错点/考点、过渡语。
>
> 页码锚点与课件页一一对应（10.1–10.58）：http://111.231.0.226:2345/lecture10/index.html
>
> **口径提醒**：本章与第 4 章 4.5 节的"测试规范八要素"是同一套骨架（角色／准入／输入／活动／输出／验证／准出／度量），讲课时可明确对照。

【10.1】第 3 篇 软件测试项目实践（1 分钟）

> 课上到这儿，工具你手上已经抓了一把。等价类怎么划，流程怎么排，单元、集成、系统功能各测什么。

>

> 真实项目不问这些。它问的是另一句：这系统两周后上线，你打算怎么测，凭什么说它能上。这句话方法答不了。

>

> 所以第 3 篇换成项目实践。五章连成一条线，一章接一章。第 10 章先想清测什么、测到什么程度。第 11 章把测什么落成一条条能执行的用例。第 12 章把环境、数据、工具备齐。第 13 章真跑起来，把缺陷管住。第 14 章看往后。

>

> 顺序为什么不能颠倒？没想清就写用例，写出来的是没人认的脚本。环境没备好就开测，测出来的结论站不住。说得再直白点：前者是白干，后者是白测。

>

> 再说一层：这五章不是五个知识块，是一条流水线，上一章的产出是下一章的输入。测什么没想清，第 11 章就只能照着需求文档抄用例；用例抄得浅，第 12 章的环境和数据就配不出针对性的场景。

>

> 还有一层没人明说。前两篇你练的是挑错，这一篇要练的是交付。挑错是个人手艺，交付要别人认。别人凭什么认？凭你留下的东西说得清、查得到、接得上手。这话听着虚，落到具体就是：你休假两周，同事能不能接着把你的测试跑完。

>

> 带着一个问题进这一篇：领导把系统丢给你，说下周开始测，你的第一份交付物是什么？不是 bug 列表。是一份说清测什么、测到哪儿、怎么算测完的东西。这份东西写得糙一点没关系，写不出来才是问题。

提问：接手一个完全陌生的系统，第一周你最先干的活是什么？

预设回答：

- 答"马上打开系统点功能，先找几个 bug 出来"——这是最典型的抢跑。没有范围和优先级，你点出来的 bug 只证明你手气不错，证明不了系统能上线。

- 答"先看需求文档，弄清要测什么"——方向对了，这就是第 10 章要做的事。

- 答"先搭环境、准备数据"——也实在，但环境搭成什么样，取决于你打算测什么，顺序上还排在"想清楚"之后。

- 收口：三答正好对上这一篇的章节顺序。顺序本身就是第一条经验。

易错点 / 考点：① 第 3 篇五章的名称与顺序要能默写；② 判断陷阱"测试计划是项目经理的事，测试人员只管执行"——错，计划是测试人员的第一份专业产出；③ 项目里最常见的质量事故起点叫倒序开工：先执行、后计划。

过渡：这一篇要交付的第一样东西不是报告，是测试件。先把这个词立住。

【10.2】软件测试工作和测试件（1 分钟）

项目收尾那天，最怕听到一句"我测过了"。四个字，别人拿它没办法。

>

> 为什么没办法？因为它没法复核。测试这件事的价值，一半在找出来的问题，一半在别人能顺着你的记录再走一遍。

>

> 所以测试工作留下的不是一句话，是一整套东西，行话叫测试件，testware。测试计划、测试需求与测试设计规格、测试用例、测试数据，还有自动化脚本和环境配置说明。再往后是缺陷报告、测试报告、执行日志。这些东西的共同点只有一个：别人拿着它，能把你做过的事重做一遍。

>

> 打个生活里的比方。装修队交房，不会只说一句"我刷好了"。他还给你水电走向图、材料清单、隐蔽工程照片、保修单。缺了这些，你住进去心里没底。软件一样，测试件就是测试工作的隐蔽工程照片。

>

> 还有个反反复复出现的现象：出过事故的项目，事后复盘常发现"当时其实测过"。可记录丢了，结论没写，谁测的都不知道。测过但没留下证据，跟没测是一个结果。

>

> 再说清一件事：测试件和软件产品是两样东西。软件产品是被测的对象，测试件是这轮测试活动留下的产物。这个区分看着学究，实际有用。出问题的时候，你得能说清是产品有毛病，还是你的验证过程有缺口。两样混着叫，责任就分不清。

>

> 所以这一章请你换个身份。你不是在找 bug，你是在生产一套能交付、能追溯的测试件。这套东西攒起来，比你会用几个工具更能说明水平。

提问：测试做完了，你要交给项目组什么？如果只能回答"我测过了"，问题出在哪？

预设回答：

- 答"交测试报告"——只说对一件。报告是结论，报告背后的用例、数据、日志才是支撑结论的证据。

- 答"交缺陷列表"——缺陷列表是中间产物。它回答发现了什么，不回答还有什么没测、测到什么程度。

- 答"我口头说一下就行，反正都在我脑子里"——最危险的一种。人一走，证据链就断了。这在项目里叫单点依赖，是风险不是效率。

- 收口：测试件的完整程度，直接决定你的结论能不能被别人接受。

> **易错点 / 考点：** ① testware 的中文名"测试件"要记牢，名词解释高频；② 分清测试件与软件产品：前者是测试活动的产物，后者是被测对象；③ 判断"测试用例属于测试件"——对；"缺陷报告属于测试件"——也对，别以为测试件只指计划类文档。

> **过渡：** 这一篇要产出的第一件测试件，就是第 10 章的测试计划。

【10.3】第 10 章 测试需求分析与测试计划（1 分钟）

> 系统丢给你，只留一句：下周开始测。你第一份东西写什么？

>

> 写什么，取决于你想清了什么。第 10 章管的就是开工前那一段：目标、范围、计划。

>

> 它的位置得说清。它是第 3 篇第一章，后面四章全建在它上面。这儿定歪了，第 11 章用例写偏，第 12 章环境搭错，第 13 章执行失控。

>

> 这一章回答三个问题。测什么：目标怎么定、范围怎么划、测试项怎么从需求里推出来。测到什么程度：优先级怎么排、风险怎么识别、进出准则怎么定。怎么安排：工作量怎么估、进度怎么排、计划怎么写、怎么评审和维护。

>

> 反面例子不难找。范围没划，测试就变成能点到哪儿算哪儿，忙了一周，没人说得清还剩多少没测。目标没定，测完写报告只能写一句"未发现明显问题"，谁也不敢拿它做上线决定。

>

> 名字里两个词，很多人以为重头在后面那个"计划"，因为计划看着像文档。重心在前面那个"分析"。分析是把需求翻成测试语言，计划只是把翻译结果排成一张能执行的表。翻译错了，表排得再漂亮也是错方向。

>

> 两件事的分工，一句话能分清：分析回答凭什么这么测，计划回答谁在什么时候把它测完。

>

> 最后一个态度问题。这份产物是要签字承诺的。你写下"系统测试 10 天、回归两轮"，就是在承诺资源、时间和质量。承诺写错了是要还的，还的方式不外三种：加班、砍范围，或者带着没测过的模块上线。计划不是作文，是契约的草稿。

> **提问：**这一章只能保一件事做到位，你保"分析"还是保"计划"？

> **预设回答：**

- 答"保计划，写漂亮点好交差"——计划漂亮而分析是空的，执行时天天改计划，改到最后没人看。

- 答"保分析"——更接近正确。分析对了，计划粗糙点还能在过程中修正；分析错了，计划再精细也是错的方向。

- 答"两个都要"——态度没错，但得有取舍意识：时间紧的时候，宁可分析做扎实、计划写薄薄三页。

- 收口：分析决定方向，计划决定节奏。方向错了，节奏再好也没用。

> **易错点 / 考点：**① 本章五节名称要记全：测试目标和准则、测试需求分析、测试估算与进度安排、测试风险与测试策略、测试计划的内容与编制；② 简答常考"测试计划与测试需求分析的关系"——分析是输入，计划是输出；③ 别当成文档写作课，案例题给一段需求让你列测试项和优先级，考的是分析能力。

> **过渡：**先看这一章的目录，你心里好有个时间预算。

【10.4】本章目录（2 分钟）

> 五个小节，先把名字认全。考试问"功能点法属于哪一节"，你答不出来就是白丢分。

>

> 10.1 测试目标和准则。解决为什么测、测到什么算开始、测到什么算结束。这里有两组东西：一组是目标，通用目标加本项目的具体目标；一组是准则，以进入准则为主。

>

> 10.2 测试需求分析。全章最重的一节，也是五节里唯一讲方法的。要讲清分析为什么是基础、输入和输出各是什么、过程分几步、范围怎么界定。还要讲功能和非功能怎么分解、优先级怎么排、分析不准会带来哪

> 些风险。这一节讲透了，后面写用例、估工期都有依据；讲不透，只能照模板抄。

>

> 10.3 测试项目的估算与进度安排。把要测的东西换算成人天，再排成时间表。工具有功能点法、WBS 法、历史数据法、经验法。

>

> 10.4 测试风险和测试策略。风险怎么识别、严重性怎么算、怎么控制；策略是什么，为什么必须做取舍，常用策略有哪些。风险和策略放一节，是因为策略就是按风险分配力气的结果。

>

> 10.5 测试计划的内容与编制。收口：一份计划写哪些条目、进出标准怎么定、计划为什么是过程而不是一份文档。

>

> 五节的逻辑是：定目标、做分析、算工作量、控风险、出文档。前四节是想，第五节是写。想不明白就动手写，写出来一定是从别处抄来的模板。

>

> 时间上 10.2 占得最多。它偏方法，另外四节偏管理口径。复习按这个权重分配，别在概念上磨太久，要反复看的是 10.2。

提问：五节里只给你两天准备一个测试项目，你会跳过哪一节？

预设回答：

- 答"跳过 10.3 估算，凭感觉排期"——现实里很多人这么干。代价是排期全靠拍脑袋，一遇延期就解释不了。

- 答"跳过 10.4 风险"——最不该跳。风险不是可能出问题，是一定出问题，只是不知道哪一个先来。

- 答"哪节都不能跳，但可以压缩到一句话"——最专业。每节留一个最小产出，比如风险就留一张五行检查表。

- 收口：可以压缩篇幅，不能压缩动作。

易错点 / 考点：① 五节名称与顺序；② 对应关系题："功能点法、WBS 法属于哪一节"——10.3 估算；"风险矩阵属于哪一节"——10.4；③ 本章落点是测试计划，前面四节都是它的输入。

> **过渡：**进 10.1 之前，先看一张总图：测试计划到底要抓哪几件事。

【10.5】软件测试计划的重点工作（3 分钟）

> 一份计划送评审，二十分钟就被打回来。理由常常不是写得不好，是缺项。

>

> 七项重点工作，我一项一项过，顺带说清它在哪一节展开。

>

> 第一，明确测试目标。关键字是"明确"。目标是发现哪些质量风险、验证哪些业务、覆盖到什么程度，不是"保证系统没有 bug"。

>

> 第二，分析与确定测试范围。范围是双向的：哪些要测，哪些不测。只写要测的，是半个范围。出了事，你说不清为什么没测那块。

>

> 第三，识别测试项及其优先级。测试项是从需求里拆出来的可测单元，优先级决定先测谁、测多深。这项是 10.2 的核心产出。

>

> 第四，识别测试风险，采取相应对策。风险和测试项是一对：高风险的地方多测、早测、用更强的方法测。

>

> 第五，测试工作量估算。要回答多少人、多少天。方法在 10.3。

>

> 第六，测试资源、进度等安排。人、环境、工具、数据、时间窗口，都要落到具体的人、具体的日期上。不落到日期，就等于没安排。

>

> 第七，测试阶段出入准则。什么条件才能开测，什么条件算测完。这是质量的闸门。

>

> 七项下面挂着一个括号：测试需求分析。它不是并列的第八项，是第二、三、四这三项的共同底座。七项里有一半靠分析撑着，分析塌了，计划只剩格式。

>

> 再往深看一层。这七项跟第 4 章讲的规范八要素几乎一一对齐：角色、准入、输入、活动、输出、验证、准出、度量。计划不是新东西，是把规范落到这个项目上。

>

> 还有一层用处：这七项就是评审的检查表。目标可验证吗？范围写清不测什么了吗？测试项有优先级吗？风险有对策吗？估算有依据吗？资源落到人了吗？出入准则可判定吗？七问过一遍，计划的质量就托住了。

> **提问：**七项里哪一项最容易被写成空话？为什么？

> **预设回答：**

- 答"测试目标"——很准。目标最容易写成"保证质量、确保稳定"这类口号，一写就空。

- 答"风险"——也常见。很多计划的风险栏就一句"可能存在进度风险"，等于没写。

- 答"工作量估算"——同样高发：只写总人天，不写怎么算出来的，评审时没法验证。

- 收口：这三项的病根是同一条，不可验证。判断一份计划好不好，最快的办法是问：这句话将来怎么证明它做到了？

易错点 / 考点：① 七项名称至少能答出五项；② 判断"测试范围就是列出要测的功能"——错，不测的部分同样要写；③ 本章与第 4 章八要素的对应关系常出案例题：按八要素评审一份计划。

> **过渡：**七项里排第一的是目标。那就从什么算一个合格的测试目标讲起。

【10.6】10.1 测试的目标和准则（1 分钟）

> "保证质量"这四个字，写进计划那一行，这一项就已经没法验收了。

>

> 10.1 只有两件事：目标，还有准则。目标是为什么测，准则是什么时候开始、什么时候算完。

>

> 为什么把它俩放开头？因为它俩定后面所有工作的边界。目标含糊，你不知道要测多深；准则含糊，你不知道什么时候能停。测到哪儿停这件事，不能靠感觉，靠感觉要么早早收工，要么无限期拖。

>

> 举个对照。目标是"本轮覆盖新增对公转账的 12 个场景"，准则是"致命与严重缺陷清零才允许出测试报告"。前者说要干什么，后者说什么时候可以停。计划里这两样缺一，执行时就只能靠人临时拍板，一靠拍板就要吵。

>

> 这一节的走法我安排成三段。先拿几条真实的假目标让你判，把标准立起来。再讲测试目标的正式定义和六类目的。最后落到这个项目的具体目标怎么定，以及进入准则。

>

> 内容不多，可考试爱出辨析题：给你四句话，问哪一句能作为测试目标。这题不难，难在你心里得有一条明确的判据。

>

> 先给一句概括：目标是可验证的句子，准则是不容商量的条件。这两样分清了，后面几节都不会绕。

>

> 还有一句要紧的：准则不是写给自己看的，是给别人用的。测试经理拿它判断能不能收工，项目经理拿它排后面的活，评审拿它问一句你凭什么说测完了。写不出可判定的准则，这几问一个都答不上。

提问：一句话里出现"保证""确保""尽量"这类词，它还能当测试目标吗？

> **预设回答：**

- 答"能，意思到了就行"——意思到了，责任没到。

- 答"不能，这类词没法验证"——对，这就是这一节要立的标准。

- 答"看场合，口头沟通可以，写进计划不行"——很务实。口头可以用，写进计划和验收条件就必须换成可验证的表述。

- 收口：可验证不是吹毛求疵，它是让"做到没做到"这件事只有一个答案。

易错点 / 考点： ① 两块内容：目标、准则，以进入准则为主，退出准则放到 10.5 展开；② 辨析题"以下哪句可作为测试目标"——凡含保证、确保、尽量、彻底的多为不可验证；③ 测试目标与测试目的是同一含义，别当两个概念。

过渡： 先做一道判断题，看看你心里的标准立住没有。

【10.7】示例：哪个是测试目标？（2 分钟）

下面四句话，你挑一句能当测试目标的。我猜你会挑错，而且往负责的那个方向挑错。

>

第一句，保证系统上线后不出问题。听着很负责，毛病在两头。一是不可验证，二是它超出了测试的职责。上线后出不出事，还看运维、看数据、看用户怎么用。测试能给的只有一句：在给定范围内发现并报告风险。

>

第二句，系统要稳定可靠、界面要友好。这是形容词，不是目标。稳定是几个 9？友好是谁觉得友好？口径没有，测完你说达标，用户说不达标，没人能裁。

>

第三句，测试要做透，不能漏测。这句把"透"当成了绝对标准。测试不可能穷尽，第 2 章就讲过。能写的是用什么覆盖准则、覆盖到什么比例。

>

第四句，本轮覆盖新增对公转账业务的全部 12 个业务场景，回归个人转账的 8 个核心场景，致命与严重缺陷清零。这句能当目标：有对象、有范围、有数量、有判据。

>

四句一比，标准就出来了：能不能被第三方核对。能核对的是目标，只能表态的是口号。

>

写不出可验证的目标，通常不是你文字差，是你还没想清要测什么。课上我会让你们各改一次。以第四句为模板改第一句。前半截这样写：针对新增的对公转账业务，验证 12 个业务场景的金额、费率与限额判定符合需求文档第 4 章。后半句再补上回归个人转账 8 个核心场景，致命与严重缺陷清零。改的过程就是在补做

> 需求分析。句子写不下去的地方，就是你还没想清楚的地方。

> **提问：**第四句里"致命与严重缺陷清零"这一条，属于目标还是准则？

> **预设回答：**

- 答"属于目标"——它确实写在目标句里。可它的性质更像准则，后面讲退出准则时你会看到同一条。

- 答"属于准则"——更准确。它回答的是什么时候可以停，那是准则的职责。

- 答"都对"——这就滑向含糊了。规范说法是：目标描述要达成什么，准则描述满足什么条件可以结束。两者可以互相引用，不能混着写。

- 收口：一句好目标往往自带一个准则。分辨它属于哪一类，看它回答的是"做什么"还是"何时停"。

> **易错点 / 考点：**① 判断标准背下来：可度量、有范围、有判据、职责内；② 典型错误句式含保证、确保、彻底、尽量；③ 案例题常给一段项目背景让你改写成目标句，公式是对象加范围加数量加判据。

> **【思政落点 1】**：四句里最像负责的是第三句"不能漏测"，它把不可能的事许了出去。测试这行有条底线：不许诺做不到的事。把"保证不出问题"改成"覆盖 12 个场景、致命与严重缺陷清零"，看着像给自己上锁，其实是把边界说清、把责任接到自己身上。不夸大、不含糊、不推诿，这三条守住，别人才敢把系统交给你。

> **过渡：**标准立住了，接下来给测试目标一个正式定义。

【10.8】测试目标（2 分钟）

> 定义只有一句话，抄下来不到三十秒。麻烦在"原因或目的"这四个字，很多人顺手就读成了"指标"。

>

> 测试目标是设计和执行测试的原因或目的。指标是实现目标的手段。搞反了会出什么事？你会盯着覆盖率数字干活，忘了为什么测。

>

> 课件列了六类目的，按对内、对外、对变更三组好记。

>

> 对内两条，都是给信息。一是向风险管理活动提供信息。你不测，风险那边只能靠猜。二是提供软件系统质

> 量有关信息：缺陷分布、严重程度、模块集中度，管理层拿它判断能不能发。

>

> 对外两条，都是做评估。三是评估软件产品是否满足相关利益者的期望。相关利益者不只是用户，还有业务方、运维、监管。四是评估软件是否完全符合合规性要求。金融、医疗、汽车这些行业，合规不达标根本不能上线，这条本身就是硬目标。

>

> 对变更两条，最容易被忽略，也最实用。五是评估缺陷修正而不带来负面效应：改了 A 坏了 B，这叫修复引入缺陷。六是评估软件变更实施而不带来负面效应：版本升级、配置调整、依赖替换，都要验有没有副作用。

>

> 后两条合起来，就是回归测试存在的理由。真实项目里回归的工作量常超过新功能，原因就在这儿：改动是常态，副作用也是常态。

>

> 六条是通用目标，任何项目都适用。可通用目标不能直接拿去执行，得翻成本项目的具体目标。

>

> 顺手给个用法：听到"目的"两个字，先追一句为什么做；听到"指标"，先追一句拿什么量。追不下去的那个，通常就是还没想清的那个。

提问：六条里哪一条在你做课程设计时最容易被完全忘掉？

> **预设回答：**

- 答"合规性"——很可能。学生项目没有监管要求，可换个场景，给学校做学籍系统，合规就是硬指标。

- 答"缺陷修正没有副作用"——这条忘得最狠。很多同学改完一个 bug 就提交，从不回头验证别处有没有被带坏。

- 答"向风险管理提供信息"——也常忘，因为学生项目里风险这个概念还没建立起来。

- 收口：忘掉的那几条，恰恰是工业界最看重的。测试不是找茬，是给决策提供依据。

易错点 / 考点：① 定义原文"设计和执行测试的原因或目的"要能背；② 六条分类记忆：对内给信息、对外做评估、对变更防副作用；③ 高频考点"回归测试的必要性来自哪两条目的"——缺陷修正与变更实施不带来负面效应；④ 别把目标和指标混用。

> **过渡：**通用目标有了，下一步把它落成本项目的具体目标。

【10.9】项目的具体测试目标（2 分钟）

要求常常短得可怜：别出错，稳一点。就这几个字，你还得把它写成四句能验收的话。

>

> 第一条，提供哪些质量风险信息。它翻过来就是一句：这个项目你最怕哪里出事。怕什么就测什么，而且要把怕写出来。银行核心系统改造，最怕账务算错和跑批超时。目标就得写：验证日终批量在数据量增长三成的情况下仍在时间窗内完成。

>

> 第二条，新改动的业务是否正确实现，对已有业务是否有负面影响。前半句验新功能，后半句是回归。课件把它俩写在一起，是在提醒你：改动和影响是一对，不能只测改动。项目里最贵的事故，往往不是新功能没做对，是新功能做对了，把老功能弄坏了。

>

> 第三条，是否满足功能性要求和非功能性要求。功能好理解，非功能容易被忽略：性能、兼容、安全、可靠、易用。第 7 章专门讲过，它们的判据是指标达不达标。写目标时就得带上指标口径，不然最后又是一句"感觉还行"。

>

> 第四条，在测试覆盖率、测试效率上的具体要求。这条看着像管理指标，实际很硬。覆盖率回答测到了几成，效率回答多久跑一轮。没有这两个数，你判断不了"再测一周还有没有意义"。

>

> 四条合起来，是目标骨架的四个指向：风险、改动、要求、强度。写计划时照这四条各写一句，目标就不会空。

>

> 每条都要落到本项目的名词和数字上：具体的业务名、具体的模块名、具体的数字。写不出名词，说明分析还没做到位。

> **提问：**一个内部管理系统，业务方只提了一句"别出错，稳一点"。你怎么把它落成第二条和第三条的具体目标？

> **预设回答：**

- 答"那就写确保系统稳定运行"——又回到口号了。

- 答"追问业务方：哪几张单据最关键、出错赔钱最多"——对。这就是把"别出错"翻成"哪几类业务不容错"。

- 答"再问：月底结账那天多少人同时用、几分钟内必须出结果"——这一步把非功能要求也落下来了，性能目标就有了。

- 收口：具体目标不是你想出来的，是问出来的。问不出来的部分，就是你要写进风险栏的未知区域。

> **易错点 / 考点：**① 四条内容要能完整复述；② 高频考点"新改动业务与已有业务的关系"——必须成对考虑，这是回归目标的来源；③ 案例题给"一句话需求"让你落成具体目标，答题要带业务名加判据；④ 只写功能性要求会丢分，非功能要求是得分点。

> **过渡：**知道了目标该长什么样，接下来是最实际的一问：这些目标怎么定出来？

【10.10】如何确定测试目标？（3 分钟）

> 一个字段长度从 32 改到 64，只动一行代码。要不要回归？影响圈怎么画？先把方法摆出来。

>

> 四个追问，实际是一条推理链。

>

> 第一问，哪些业务改动会影响哪些已有业务？这是在画改动的影响圈。改一次利率计算，牵到计息、结息、报表、对账。圈画不全，回归范围就漏。漏掉的那块，正好是没人想到的那块。

>

> 第二问，系统改动会影响哪些系统功能和非功能特性？这一问把范围从业务扩到特性。有些改动不动业务规则，只动数据量、动调用链、动并发，非功能特性就受影响了。

>

> 第三问，测试覆盖率：新业务和新功能要多少，已有业务和功能呢？新的好定，老的最容易含糊过去。而恰恰是老功能，用户天天在用，出事最疼。

>

> 第四问，如何最大程度提高测试效率？看着像管理问题，其实是目标问题。效率目标会反过来约束方法选择：是上自动化每日回归，还是人工挑重点。

>

> 再看凭什么做这些判断，五个输入。产品质量要求，来自合同、标准、行业规范。业务功能关系分析，也就是业务之间的依赖图。测试范围分析，测什么不测什么。测试策略和方法选择，用什么手段。这四样共同支撑目标陈述。

>

> 四个追问的顺序有讲究：从影响出发，倒推目标。多数人定目标是"我想测什么"，专业做法是"改动影响什么，我就必须测什么"。前者凭兴趣，后者凭证据。

>

> 给个能直接套的模板：针对〔改动点〕，验证〔受影响业务或特性〕在〔条件〕下满足〔判据〕，覆盖〔范围或比例〕。填空填不满，说明还有事没问清。

>

> 回到刚才那个字段。长度改到 64，牵到表结构、导入导出模板、界面校验、上下游接口报文、历史数据迁移。要画准，看这个字段被多少地方引用。改动大小和影响大小，没有必然关系。

提问：只改一行的需求，需要回归测试吗？影响圈怎么画？

预设回答：

- 答"只改一行，不用回归"——最危险。字段长度会影响表结构、导入导出模板、界面校验、上下游接口报文、历史数据迁移。

- 答"要回归，至少测相关的增删改查和导入导出"——方向对，还能再往下推：接口报文长度约定、报表列宽、索引与性能。

- 答"看这个字段被多少地方引用，按引用关系画图"——最专业。这就是影响分析，有些团队会做静态的字段引

用扫描。

- 收口：改动大小和影响大小没有必然关系。影响圈靠分析，不靠感觉。

> **易错点 / 考点：** ① 四追问、五输入要能对上号；② 判断"改动小就不需要回归"——错；③ 案例题高频：给一段变更描述，让你列受影响的功能和非功能特性，业务与特性两个维度都要写；④ 目标句模板考试可直接套用。

> **过渡：** 目标定完了，还有一道门：什么条件下才允许开始测。

【10.11】测试进入的准则（2 分钟）

> 版本提交过来了，主流程跑不通，进度表上还剩三天。开测还是退回？

>

> 先看准入六条，按三组记。

>

> 第一组，想清楚。清楚了解项目的整体计划框架：你得知道自己在时间轴上站在哪，什么时候要交什么。完成需求规格说明书评审：需求没评审就开测，等于拿一份可能画错的地图导航。评审过的需求，才算可测的基线。

>

> 第二组，有依据。技术知识或业务知识的储备：不懂业务的测试，只能测界面。技术设计文档：接口在哪、数据怎么流，都在这儿，它是设计和集成测试的依据。

>

> 第三组，有准备。标准环境：环境不标准，测出来的问题说不清是软件的还是环境的。足够的资源：人、机器、数据、工具许可。人员组织结构定下来，谁测哪块、出事找谁，得先写清。

>

> 六条里最容易在真实项目失守的是第一条和第五条。第一条失守长这样：测试开跑了，没人知道这轮要在两周后支持哪个发布决策，于是测到一半被叫停。第五条长这样：环境今天能通，明天库挂了，团队一半时间在修环境。

>

> 为什么非要立准入？反过来说更清楚：没有准入，测试就会在残破的版本上反复空转。代码编译都不过，主流程都跑不通，你点出来的"缺陷"全是噪音。噪音一多，大家对测试结论就不信了。

>

> 准入还有个隐性作用：它是向上游要质量的抓手。开发不给能测的版本，你有权说一句"未达准入条件，测试未启动"。这句比任何催促都管用，因为它把责任边界划清了。

提问：版本提交过来了，主流程跑不通，可项目进度很紧。开测还是退回？

预设回答：

- 答"先测能测的部分，边测边等修复"——现实里很常见，但得有个前提：书面记下"未达准入，本轮结论不完整"。不写，最后延期算你头上。

- 答"直接退回，不达准入不测"——立场清楚，还要配套沟通：缺什么、什么时候能好、对进度影响多大。

- 答"随便点点，反正也是测"——最糟。残破版本上产生的记录会污染缺陷库，后面统计缺陷密度、判断收敛趋势全被带偏。

- 收口：准入的执行要点不是卡或放，是有条件放行加明确记录。

易错点 / 考点：① 六条准入条件的名称；② 判断"需求评审属于开发活动，与测试准入无关"——错，它是准入的硬条件；③ 案例题常考"版本不达准入怎么办"，标准答法是记录加有条件放行加标注结论边界；④ 结构记忆：想清楚、有依据、有准备。

过渡：准入讲完，10.1 就结束了。接下来进本章最重的一节：测试需求分析。

【10.12】10.2 测试需求分析（1 分钟）

> 多数人第一眼会把测试需求读成用户需求的另一种说法。读错的位置就在这儿：一个是软件要做什么，一个是测试要做什么。

>

> 它确实从用户需求来，可两者不是一回事。用户需求说软件要做什么，测试需求说测试要做什么，而且必须是可执行、可判定的那份清单。中间隔着一道分析工序。

>

> 四个小节，递进关系值得看清。10.2.1 测试需求分析方法，讲怎么想。10.2.2 测试需求分析技术，讲用什么工具想。10.2.3 功能测试范围分析，业务这一侧怎么划。10.2.4 非功能性的系统测试需求，质量这一侧怎么划。前两个偏思维，后两个偏对象。

>

> 方法和技术分开讲，是有道理的。方法解决顺序和取舍，技术解决手上用哪把家伙。顺序错了，工具越熟越偏；工具生疏，顺序对了还能慢慢补。所以先讲方法。

>

> 为什么这一节最重？因为后面所有活动都从这儿派生。用例从测试项派生，环境按测试项配，进度按测试项估，缺陷也按测试项归类。这儿错了，后面每一步都在错的前提上往下走。

>

> 行业里有个说法：测试项目返工最贵的三件事，需求理解错、范围划错、环境配错。三件里两件出在这一节。第三件也别急着推给运维。环境该配成什么样，同样是分析定的：要测并发，就得有能压的库；要测兼容，就得多备几套浏览器。

>

> 所以带着一个问题进来：需求文档给你了，你怎么从一份给人看的文档，得出一份给测试用的清单？这个问题答不上来，后面写用例只能靠东抄一句西抄一句。

提问：为什么说"测试需求分析"里的"需求"，不是指用户的原始需求？

预设回答：

- 答"因为它是测试的需求"——对了一半。它描述的是测试要做什么，不是软件要做什么。

- 答"它是从用户需求推导出来的"——补得到位。原始需求是输入，测试需求是经过分析、可执行、可判定的那份清单。

- 答"两者是一回事，换个说法"——这就危险了。把需求文档直接当测试清单用，会漏掉隐含需求、非功能需求、异常路径。

- 收口：测试需求是需求的可测化版本，中间隔着一道分析工序。

易错点 / 考点： ① 四小节名称与顺序；② 概念辨析"用户需求、软件需求、测试需求"三者关系——逐层派生；③ 判断"测试需求分析就是把需求文档读一遍"——错，它是重构与再表达。

过渡： 先看一张图，它说明为什么测试分析是整条链的起点。

【10.13】测试分析是基础（2 分钟）

一百条用例跑完，覆盖率报告很漂亮。可漏掉的那个模块，照样漏。

>

循环只有四步：测试分析、测试设计、执行、评估，然后回到分析。

>

分析干什么？搞清楚测什么、测到多深、优先级怎么排。产出是测试项清单、范围界定、风险和优先级。

>

设计干什么？把测试项变成具体用例：输入什么、怎么操作、预期什么。第 11 章专门讲。

>

执行就是跑用例、记结果、提缺陷。

>

评估看结果说明了什么：覆盖率够不够、缺陷收敛没有、风险还剩多少。评估的结论会回到分析。某块风险比预想的高，就得回去补分析、补测试项。

>

为什么说分析是基础？它是唯一一个错了会放大的环节。漏一个模块，设计就少一批用例，执行就少一批数据，评估就得出"质量良好"。整条链上没有任何一步会发现这个错，因为每一步都在忠实地执行上一步的错误。

>

打个比方。分析是地图，设计是路线，执行是开车，评估是看看到没到。地图上少画一条路，你车开得再稳、油加得再满，也到不了那儿。

>

> 这个循环里，最该花时间的是第一步，最容易被压缩的也是第一步。这就是项目里最典型的错配。

>

> 所以给分析留看得见的时间：在计划里写成独立任务、给独立工期、指定负责人。凡是没写进计划的工作，最后都会被压成零。

>

> 还有个信号挺好用。连续几轮评估，覆盖率都卡在同一个数上，基本可以断定是范围漏了，不是用例不够。

> **提问：**这个循环里，评估发现覆盖率不足，回到哪一步？

> **预设回答：**

- 答"回到执行，多跑几遍"——跑一百遍也覆盖不到没设计出来的用例。

- 答"回到设计，补用例"——常见做法。可如果缺的是测试项这一层，补用例也只是在错的范围内补。

- 答"先回分析，看是范围划漏了还是测试项拆漏了"——最正确。先确认地图错没错，再改路线。

- 收口：判断该回哪一步很简单，问这个缺口是"没想到"还是"没写到"。前者回分析，后者回设计。

> **易错点 / 考点：**① 四步循环的名称与顺序，特别是"评估回到分析"这条回边；② 简答"为什么说测试分析是基础"——答错误会被后续环节忠实放大且难以发现；③ 判断"测试分析只在项目初期做一次"——错，评估结果会驱动它反复做。

> **过渡：**既然是基础，那它的输入是什么、输出是什么？把两头摆出来看。

【10.14】测试分析的输入与输出（2 分钟）

> 分析做不出来，多半不是不会想，是手上没料。这一节把两头摆开。

>

> 输入这一侧五类。

>

> 第一类，需求类文档：需求规格说明书、业务需求、用户故事、验收标准。这是最正统的输入。

>

> 第二类，设计类文档：架构图、接口说明、数据模型。测接口和集成的时候离不开它。

>

> 第三类，业务规则与数据：费率表、状态流转规则、真实业务数据样本。没有业务数据，很多场景根本造不出来。

>

> 第四类，历史信息：上一版的缺陷清单、生产事故记录、用户投诉。这是找问题高发区最管用的输入，比什么方法都快。

>

> 第五类，约束：标准规范、合规要求、合同条款、环境和工具的限制。

>

> 输出那一侧六项：测试目标、测试范围、测试项和测试子项、测试风险、优先级、测试策略。

>

> 这张图真正的教学点在一句话上：输入的质量决定输出的质量。很多团队分析做不好，不是方法不会，是输入残缺。需求写得含糊，设计文档过期，拿不到真实数据，历史缺陷没人整理。

>

> 所以做分析的第一件事往往不是画图，是清点输入：缺什么、向谁要、什么时候能到。这份清单本身就该写进计划的风险栏。

>

> 清点还有个笨办法挺管用：五类各写一行，写不出来的当场标出来，先去找人要，别自己硬编。编出来的输入，最后会变成编出来的结论。

>

> 还有个判断挺准：如果你发现自己只能靠猜来定测试项，那一定是输入不够，不是你水平不够。

> **提问：**需求文档写得很粗，只有一页纸，你还做不做测试分析？

> **预设回答：**

- 答"不做，等需求写细了再说"——现实里等不来，项目照跑。

- 答"做，但只按那一页纸做"——会漏得很厉害，而且漏掉的部分没人知道。

- 答"做，同时把需求不清本身列为最大风险，去找业务方、找老系统、找历史缺陷补输入"——最专业。分析可以带着不确定性往前走，但不确定性必须被记录、被暴露。

- 收口：测试分析的一项产出是输入缺口清单，它比测试项清单更能保护你。

> **易错点 / 考点：**① 输入五类、输出六项要能对上；② 判断"测试分析只需要需求文档"——错，设计文档与历史缺陷同样关键；③ 案例题常考"需求不清晰怎么办"，答补输入、记录缺口、升级风险三步。

> **过渡：**输入摆完了，把输出这一侧展开。六项成果，正好是测试计划的原料。

【10.15】软件测试分析的输出（2 分钟）

> 把"用户管理"四个字填进测试项那一栏，十有八九要被退回来。

>

> 六项输出，一项一项看。

>

> 测试目标，就是落到本项目的几句可验证的话。

>

> 测试范围，测什么、不测什么，边界写清。

>

> 测试项和测试子项，这是分析的核心产物。测试项是要测的一个能独立验证的对象，子项是它的细化。一个"转账"测试项下面，可以有本行转账、跨行转账、限额校验、余额不足、重复提交。

>

> 测试风险，哪儿不确定、哪儿复杂、哪儿历史上老出事。

>

> 测试优先级，决定先测谁、测多深、资源往哪倾斜。

>

> 测试策略，针对风险和约束，用什么手段、什么层次、什么组合。

>

> 六项的排法有讲究：前两项定边界，中间两项定内容与顺序，后两项定打法。照这个顺序写，一份测试需求文档的骨架就有了。

>

> 常见错误是把测试项写成功能菜单名。菜单名是界面怎么组织，不是测试对象。测试项要按业务能力和质量特性切："跨行转账的限额校验"是测试项，"转账页面"只是它的一部分载体。

>

> 判断切得对不对，有个简单办法：每个测试项都该能独立写出通过和失败的判据。写不出判据的，就是切得太粗或者太大。

>

> 最后补一句：清单会被下游反复引用。用例挂在它下面，缺陷归到它下面，覆盖率按它统计。所以编号和命名从一开始就规范，别等几百条了再返工。名字起得乱，最先被拖死的是统计覆盖率的那个人。

提问："用户管理模块"能不能直接作为一个测试项？

预设回答：

- 答"能，模块就是测试项"——太粗。这个项下面可能有几十条判据，没法估工作量、没法排优先级、没法判断测完没有。

- 答"不能，要拆到能写判据的粒度"——对。比如拆成新增用户字段校验、重名处理、角色分配权限生效、批量导入异常行处理。

- 答"看情况，小模块可以整块算"——可以接受，但要补一句：整块算的前提是它足够小、判据足够少。

- 收口：粒度的标准不是大小，是能不能独立判定。

易错点 / 考点： ① 六项的名称与顺序；② 高频考点：测试项的划分粒度——可独立判定、可估算、可排优先级；③ 别把测试项与测试用例混了：项是测什么，用例是怎么测；④ 案例题让你列测试项时，按业务能力切，不要按界面切。

过渡： 六项输出不是拍脑袋来的，它们来自一个有固定步骤的过程。

【10.16】测试需求分析的过程（2 分钟）

分析从哪儿开始？很多人从功能清单开始，然后就在那儿结束了。

>

五步，每步都告诉你进什么、出什么。

>

第一步，了解项目的背景、产品价值和待解决的业务问题，产出测试目标。为什么从背景起？同一个功能在不同背景下要求差得远。内部报销系统的审批，和信贷系统的审批，风险等级差着量级。

>

第二步，了解用户是谁、用户所关心的是什么，产出用户需求。用户是复数。一线操作员关心快不快、好不好操作；管理者关心数据准不准、能不能导出；审计关心留痕全不全。同一个系统，三双眼睛看的是三套需求。

>

第三步，从整体到局部、从上到下逐层分解功能特性，产出待测功能项。业务到功能，到子功能，到功能点。分解到哪儿停？停到能写判据为止。

>

第四步，基于系统的特点和质量属性分析非功能特性，产出待测非功能项。功能是能不能做，非功能是做得好不好：性能、安全、兼容、可靠、易用、可维护。这一支最容易被跳过，因为它不在需求文档的正文里。

>

第五步，基于特性的价值分析和风险分析，产出测试项优先级。价值和风险是两个维度：价值高的早测，风险高的深测，两个都高的是第一批。

>

> 顺序不能倒。它是一条从抽象到具体、从功能到质量、从内容到排序的收敛链。倒过来做，比如先排优先级再分解，你会发现手上没有可排的对象。

>

> 还有个实操上的坑：第三步和第四步是两条平行支线，不是先后关系。很多人只走功能那条线，走完就以为分析结束了。非功能那条线的入口是系统特点加质量属性，不是需求文档。

提问：第三步和第四步为什么是平行的两条线，而不是先功能后非功能？

预设回答：

- 答"因为非功能也是需求"——对，但没说到点上。

- 答"因为入口不同：功能从需求文档来，非功能从系统特点和质量属性来"——最准确。

- 答"先做功能，做完顺手想想非功能"——现实里常见。后果是非功能项永远只是顺手想想，最后一条都没测。

- 收口：平行意味着两条线都要有人负责、都要出现在清单里，谁都不是附属品。

易错点 / 考点：① 五步顺序与各步产出，简答高频；② 判断"分析过程就是功能分解"——漏了非功能那条线；③ 第二步的"用户是复数"常出案例题：不同干系人关心不同需求；④ 分解的停止条件是能写出判据。

过渡：五步走完，第一步和第五步产出的东西值得单独看。先把测试目标再摆一回。

【10.17】测试目标（2 分钟）

目标写完，回头一读，四类内容常常缺两类。这四类一条一条对。

>

> 第一条，满足业务需求，包括新改动的业务和已有业务。又是新加旧。放在分析阶段，意思是目标清单里必须同时有新功能验证和老功能回归两类条目，只列新的就不算完整。

>

> 第二条，满足质量要求，包括使用质量、功能性要求和非功能性要求。使用质量是个专门概念：软件在真实使用环境下对用户的有效性、效率、满意度、抗风险能力。它不是功能对不对，也不是性能快不快，是用起

> 来顶不顶事。

>

> 第三条，测试覆盖率要求。覆盖率是这一章反复出现的关键词，小结里还会说它既是起点也是终点。写成目标的一项，必须写清口径：按需求覆盖、按用例覆盖、按代码覆盖，还是按风险覆盖。口径不同，数字没法比。

>

> 第四条，测试效率、周期、成本等目标。这条最反直觉：测试目标里居然要写成本。理由很实在。测试是有限资源下的活动，一个全覆盖但要三个月的方案，在两周的窗口里等于零。

>

> 四条合起来，一个完整的测试目标就是四维的：业务维、质量维、覆盖维、效率维。少一维，执行时就会被某一方拉扯。只讲质量不讲效率，会被进度压垮；只讲效率不讲覆盖，会漏得莫名其妙。

>

> 写完拿四问过一遍：业务上验什么？质量上验什么？覆盖到什么程度？多长时间花多少成本？四问都有答案，目标才算齐。

提问：为什么测试目标里要写"成本"？这不是项目经理的事吗？

预设回答：

- 答"因为钱是有限的"——对，但更准的说法是：测试的所有取舍都是成本取舍。

- 答"写成本显得不专业"——恰恰相反。不写成本的计划评审时最容易被砍，因为你没告诉别人再往下测要多花多少。

- 答"成本和目标无关，那是预算"——割裂了。目标里的覆盖率 90% 和预算里的 10 人天必须对得上，对不上就是目标定错了。

- 收口：一份好目标本身就隐含了"在什么代价下达成什么"，这也是下一页讲估算的伏笔。

易错点 / 考点：① 四条内容，特别是"使用质量"这个术语；② 高频考点"测试覆盖率的口径"——按需求、用例、代码、风险，答题要说明口径；③ 判断"测试目标只包含质量目标"——错，还包含覆盖与效率成本；④ 记忆抓手：业务、质量、覆盖、效率四维。

> **过渡：**目标之后是范围。范围这一节信息密度最高，是分析的主干。

【10.18】测试范围分析（2 分钟）

范围里最该写清的那一栏，常常是空的。那一栏叫"不测什么"。

>

> 先说定义：界定影响范围的边界，说清哪些要测、哪些不需要测。两半都写，才叫范围。

>

> 输入三样。项目目标和范围文档，也就是计划，它是上位约束，测试范围不能超出项目范围。功能特性列表，这是分解的原料。业务流程、规则、业务数据，这是让测试接地气的东西，光有功能名，测不出业务逻辑。

>

> 输出四项。测试需求 Spec，正式的分析文档。测试项列表，按优先级排，这是核心产物。测试数据，数据也是范围分析的输出，因为能不能造出数据，直接决定能不能测。还有测试边界界定，以及测试项与需求映射矩阵。

>

> 映射矩阵值得单独说。一张表，行是需求条目，列是测试项，交叉处打勾。三个作用：证明每条需求都有人管；发现没有测试项覆盖的需求，也就是漏测。反过来，还能发现没有需求支撑的测试项，也就是多余或者过度设计。

>

> "哪些不测"比"测什么"难写。理由可能是本期不交付、影响极小、由第三方保证、技术上不可测、成本远高于收益。每一种都写下来。出事的时候，这一条是你唯一的解释。

>

> 判断范围划得对不对，有个很硬的标准：一个没参与分析的人，拿着这份范围，能不能判断这块不测是否合理。判断不了，说明你只写了结论，没写理由。

>

> 范围还得有人认。写完发给业务方和开发各看一遍，谁有异议当场改。等测完了再扯这块到底该不该测，怎么扯都是你输。

> **提问：**范围里写一句"第三方支付接口不在本期测试范围"，够不够？

> **预设回答：**

- 答"够了，写清楚了"——只写了不测，没写为什么、谁来保证。

- 答"不够，要写理由：由支付厂商提供测试报告，我方只做联调验证"——这才完整。

- 答"范围是测试组自己定的，不用写理由"——最危险。范围往往是和业务方、开发、项目经理共同确认的产物，单方面决定的没人认账。

- 收口："不测"那一栏的标准写法是：对象、理由、替代保障、确认人。

> **易错点 / 考点：**① 输入三样、输出四项；② 高频考点：需求映射矩阵的三个作用——正向覆盖、反向漏测、多余项识别；③ 案例题常考"哪些不测、为什么"，必须给理由；④ 别把测试数据漏掉，它也是范围分析的输出，这是拉开分差的细节。

> **过渡：**往外看是边界，往里看是内容。分析到底分析哪些东西？

【10.19】分析的内容：领域、元素（2 分钟）

> 需求文档里没有"干系人"这一节。可它是分析第一个要问的东西。

>

> 五层，从人到场景。

>

> 干系人：谁在乎这个系统。用户、业务方、运维、审计、监管，甚至竞争对手。每一类都有自己的关心点，分析就是把这些关心点收齐。

>

> 约束条件：技术栈、遗留系统、上线窗口、合规要求、预算。约束不是障碍，是设计空间。它告诉你哪些方案根本不用考虑。这一层最容易被写成抱怨清单。反过来用更划算：把约束摆明，等于提前把走不通的方案

> 划掉。省下的时间比多想的方案值钱。

>

> 愿景、目标与商业价值：这个产品为什么存在，解决谁的什么问题。这层看着最虚，用起来最实。它能帮你判断哪个功能是关键，哪个是锦上添花。

>

> 正确性、完备性、适应性：三种不同的质量追问。正确性是做得对不对；完备性是有没有缺项；适应性是换个环境、换种数据还成不成。三种追问对应三类测试项，多数人只问第一种。

>

> 目标场景决策卡：一个场景一张卡，写清触发条件、参与角色、期望结果、异常分支。卡片化的好处是可讨论、可排序、可裁剪。评审会上拿卡过一遍，缺字段的那张，就是还没想清的场景。

>

> 五层合起来，回答的是一句话：这块业务谁在用、受什么限制、为了什么、要满足哪些质量、在什么场景下被验证。

>

> 这些都不是从文档里抄来的，是问出来、读出来、推出来的。所以分析做得好不好，看你问了几个别人没想到的问题，不看你看完了几份文档。

提问：面对一份需求，你怎么保证"该问的都问了"?

> **预设回答：**

- 答"凭经验问"——经验有用，但不可复制，换个人就漏。

- 答"用检查单问"——对。把干系人、约束、质量、场景做成一张清单，每次逐项过一遍，这就是下一页启发式策略的雏形。

- 答"问开发就行了，他们最清楚"——开发清楚怎么实现，不一定清楚业务为什么这样要求，更不清楚用户还会怎么用。

- 收口：检查单要覆盖三类人的视角：业务方、开发、用户或运维。

易错点 / 考点：① 五组关键词；② 辨析正确性、完备性、适应性的区别；③ 判断"分析内容以需求文档为

准"——错，干系人与约束往往不在文档里；④ 场景卡的四个字段是简答得分点。

【思政落点 2】：干系人这一层，考的是你肯不肯多问一句。工程上的事故，很多不是技术不行，是有些人根本没被问到。需求方以为这么简单的规则开发肯定懂，开发以为文档没写就是不做，测试以为没提就不在范围。三方都没错，合起来就是事故。宁可多问一句被人嫌烦，也别少问一句让用户在线上替你发现。把模糊问成清楚，是这行最值钱的本事。

过渡：说到检查单，下一页就是个现成的框架，启发式测试策略模型。

【10.20】启发式测试策略（2 分钟）

四个维度摆在桌上，哪个空着，你自己一眼就看见。这就是今天这个模型。

>

HTSM，Heuristic Test Strategy Model，启发式测试策略模型。课件给的是四个要素加一个结果：项目环境、产品元素、质量标准、测试技术，交汇出可接受的质量。

>

先说"启发式"这三个字。它不是公式，是经验性的思考线索。不保证给你唯一正确答案，只保证你不漏掉重要的思考维度。老中医的问诊口诀就是这个意思：不保证诊断百分百准，但不让你忘了问某一项。

>

四个要素各管一段。

>

项目环境回答我在什么条件下干活：人、时间、工具、约束、干系人。

>

产品元素回答我在测什么东西：功能、数据、接口、结构、平台。

>

质量标准回答凭什么说它好：正确、可靠、安全、易用、可维护。

>

> 测试技术回答我拿什么手段去测：黑盒、白盒、探索式、自动化。

>

> 四者交汇，产出可接受的质量。这个说法很讲究：不是完美质量，是在这个环境下、对这个产品、按这套标准，可以接受的质量。它把测试从追求完美拉回到管理期望。

>

> 专门讲它，是为了治一种病：只从功能维度想测试。四个维度摆出来，任何一个空着，你就知道自己漏了。四列里最常空着的是质量标准那一列。功能谁都能列，凭什么叫它好，却常常写不出来。

>

> 课上可以这么用：拿一个新系统，四张纸各写一列。环境写约束，产品元素列结构，质量标准排序，技术选型。四列写完，初步策略就有了。

提问：HTSM 里的"可接受的质量"和"高质量"，差别在哪？

> **预设回答：**

- 答"没差别，都是好"——差别很大。一个是绝对标准，一个是在约束下的相对标准。

- 答"可接受是妥协，高质量是追求"——有点这个意思。但别把可接受理解成降低标准，它是把标准说清楚。

- 答"可接受的质量要和干系人对齐"——最准确。质量标准由谁定？由干系人的期望和约束共同定，测试的责任是把这条线画出来并验证。

- 收口："可接受"三个字里，藏着测试最核心的一项工作：和人对齐期望。

易错点 / 考点：① HTSM 四要素与产出；② 概念"启发式"不是算法，是思考线索，判断"启发式模型能给出唯一正确策略"——错；③ "可接受的质量"常考名词解释；④ 可与第 4 章"上下文驱动"流派对照记忆。

> **过渡：**HTSM 是思考框架，接下来把视角拉高，看分析要顾到多大一张图。

第 10 章讲稿（10.21–10.40，v3）

> 锚点与课件一一对应，不增删页。取消板书。口播按四拍：起 → 推 → 收。

【10.21】测试分析的全局观（2 分钟）

> 同一个系统，业务方、用户、产品经理、架构师各问你一遍，答案能差多远？差到能写出四份不一样的测试清单。

>

> 课件把全局观铺成五行关键词。它们不是五个要背的名词，是五个维度。第一行是视角：业务、用户、产品、系统。业务看值不值，用户看顺不顺，产品看功能全不全，系统看结构撑不撑得住。只抓一个视角，清单一定偏。

>

> 第二行是方法族：用例驱动、数据驱动、结构化、面向对象。这四样不互斥。用例驱动切场景，数据驱动切输入域，结构化切流程，面向对象切对象和关系。手上有哪样材料，就用哪把。

>

> 第三行是图形工具：用例图看角色和功能边界，数据流程图看数据怎么走，活动图看流程和并发。第四行是思维工具。头脑风暴负责发散，亲和图负责归堆。帕累托图挑关键的少数，鱼骨图追原因，直方图看分布。信息太多理不清的时候，靠这一行。

>

> 第五行落到产出：接口、功能、数据、质量、约束。前四行是过程，这一行是结果，最后都要变成清单上的条目。

>

> 名词背熟不等于有全局观。分界在这儿：换一个系统，你还知道这一场用哪把刀。练习很简单，拿你的课程设计，五行各填一行。填不出来的那行，就是你还没看到的地方。

> **提问：**帕累托图和鱼骨图，分别在分析里干什么活？

> **预设回答：**

- 答"都是画图工具，差不多"——差得远。一个管选重点，一个管找原因。

- 答"帕累托找关键的少数，鱼骨按原因分类"——准确。帕累托按贡献度排序，就是那 80/20；鱼骨图按人机料法环往下追。

- 答"帕累托排优先级，鱼骨查根因"——这就是它们在测试分析里的典型用法。

- 收口：选工具的标准是你想回答什么问题，不是哪张图好看。

> **易错点 / 考点：** ① 五个维度的名字；② 高频考点：帕累托（80/20 找关键少数）、鱼骨（因果分类）、亲和图（归类整理）三者的用途区分；③ 判断陷阱"分析方法必须统一，不能混用"——错，方法是组合着用的；④ 可以和第 3 章的方法体系对照。

> **过渡：** 往四周看完了，往下收一层，是两个最基本的手上动作。

【10.22】分析思路：整体 vs. 分解（2 分钟）

> 拆得越细，测得越全——这话你信吗？我这儿有个反例，很常见。

>

> 先看整体这一侧。课件列六样：业务流程、事件流、数据流，还有状态与状态转换。再往下是事件触发与条件约束，数据、环境、场景。六样说的是同一件事：系统作为一个整体怎么转。为什么非看整体不可？因为有些缺陷只活在部件之间。单个功能都对，串起来就错；单个状态都对，转换顺序就错。你把零件拆开逐个看，永远看不见它。

>

> 再看分解这一侧，三样：质量特性的分解，功能到子功能到孙功能，系统到模块组件再到接口。分解干的事，是把测不完的大对象变成测得了的小对象。

>

> 打个比方。整体看像看一部电影的剧情，分解看像逐个镜头查穿帮。只看剧情，漏穿帮；只看镜头，漏剧情断裂。

>

> 实操顺序我建议这么走：先整体画一张图，流程或者状态，再按图往下列测试项。这么列出来的项自带位置感，你知道它卡在流程第几步、哪一个状态转换上。

>

> 反过来，先列功能菜单再拼整体，最常见的后果是接口和状态转换成了盲区。菜单上没有"接口"这一项，也

> 没有"状态机"这一项。

>

> 那张整体图不用画得好看。能标出分支、异常、状态转换就够。它给分解提供坐标，不是拿去汇报的。所以顺序不能倒：先整体，后分解。

> **提问：**为什么接口缺陷特别容易在分解式分析里漏掉？

> **预设回答：**

- 答"因为接口不好测"——不好测是结果，不是原因。

- 答"因为分解是按功能切的，接口不属于任何一个功能"——正中要害。接口是"之间"的东西，按功能切就切丢了。

- 答"因为接口文档经常没有"——补得好。这是输入缺失，跟前面讲的"设计文档是分析输入"是一回事。

- 收口：整体视角的一个具体价值，就是把"之间"的东西捞回来：接口、状态转换、跨模块的数据一致性。

> **易错点 / 考点：**① 整体与分解各自包含什么；② 高频判断"分解得越细，分析越完整"——错，切得越细越容易丢掉"之间"的关系；③ 案例题常问"为什么接口测试容易漏"，答分解粒度与对象边界；④ 操作顺序要背准：先整体，后分解。

> **过渡：**思路有了，具体怎么下手？课件给了五条途径。

【10.23】测试需求分析途径（2 分钟）

> 拿到一个陌生系统，你第一个小时干什么？多数人是打开界面，一页页点。

>

> 课件给五条下手途径，都是想问题时的动作，不是工具。

>

> 第一条，简单化、抽象化。把细节剥掉，先看骨架。一个信贷系统剥到最后就是：收资料、评估、放款、还款、催收。抽象完你才看得清主干，不会被几百个界面晃花眼。

>

> 第二条，结构化、层次化。业务层、功能层、模块层、接口层，一层层摆好。好处很直接：能分层测、能分层追责。出了事知道是哪一层的问题。

>

> 第三条，可视化、可观察。想不清的画出来，流程图、状态图、表格。人的短期记忆装不下复杂关系，画出来才能一起讨论。这条还有第二层意思，考试爱考：不可观察的东西没法测。一个行为没有日志、没有界面反馈、没有接口返回，你凭什么说它发生过？可测试性得在需求阶段就提。等到测试才发现没法验证，只能靠猜。

>

> 第四条，类比、隐喻。陌生系统找个熟悉的对着看：这个像银行核心系统，那个像电商订单中心。类比能快速建立直觉，但要看边界。哪像哪不像，不像的地方往往就是缺陷扎堆的地方。

>

> 第五条，收集足够的信息或数据。需求、设计、数据、历史缺陷、用户反馈。材料不够就开始分析，那叫猜。

>

> 五条合起来说明一件事：分析不是天赋，是能练的动作。照这五条走一遍，至少不会漏得太离谱。我最推荐第三条，它一次解决两个问题：自己想不清，跟别人说不清。

> **提问：**第三条说"可视化、可观察"，除了画图，它在测试上还有一层什么意思？

> **预设回答：**

- 答"就是画图，方便讨论"——只答了一半。

- 答"被测系统的行为要能被观察到，否则无法验证"——对，这是测试设计的一条硬约束。

- 答"所以需求阶段就得要求加日志"——很专业。可测试性本身就是需求的一部分。很多团队到测试才发现没法验证，只能靠猜。

- 收口："可观察"是双向的：分析时把想法画出来，设计时把行为留出观测口。

> **易错点 / 考点：**① 五条途径的名称；② 高频考点：可测试性与可观察性的关系；③ 判断陷阱"类比法可以替代分析"——错，类比只给起点，边界处得实证；④ 案例题常问"系统无法验证某行为怎么办"，答补日志埋点，或者改用间接验证。

> **过渡：**下手有途径了，接着看几种分析模型，它们各有各的地盘。

【10.24】常见分析模型比较（3 分钟）

> 四种图摆一桌，考场上让你挑一个，你凭什么挑？

>

> 课件这张对比表年年考，选择题能出，选型理由也能出简答。四种模型，一种一种过，每个都说清它管什么、不管什么。

>

> 跨职能流程图。建模标准是商业建模标准——听名字就知道，它先给业务人员看，不是给程序员看。优点是复用性强、用户最容易接受。为什么容易接受？它就是泳道，一条泳道一个角色，业务人员一眼看出这步是我的活。短板是并行、异步支持差。泳道图天生顺着往下走，碰上并发分支和异步消息就画不清。

>

> 活动图，UML 家族。优点是语义最丰富：分支、合并、并发、泳道、对象流都能画。它强调行为流，也强调活动内容，适合把一段业务拆到每一步做什么。短板是容易画大，一张图几十个节点，没人看得懂。

>

> 时序图，也是 UML。强调行为流，强调协作，技术类系统更常用。技术系统为什么爱用它？因为"谁调谁"在那儿是关键。一笔交易过网关、鉴权、清算、风控、消息队列，时序图一眼看清调用顺序和消息里传了什么。业务人员用得少，因为他看不懂对象之间的消息。

>

> 数据流图，建模标准是 IDEF。强调数据流，短板很明确：没有表示出谁执行。所以它最适合计费类系统。计费的核心不是谁点按钮，是数据从哪来、到哪去、经过几次加工、口径怎么变。

>

> 四张放一起，选型就出来了。跟业务沟通选泳道图，拆细流程选活动图。要看调用协作就上时序图，盯数据加工就上数据流图。这类题的答案总落在：图的表达能力和你的分析目的一致。别背哪个最好。没有最好的模型，只有合适的。

> **提问：**一个计费系统要做测试分析，你选哪个模型？为什么不是时序图？

> **预设回答：**

- 答"数据流图，因为计费的核心是数据加工和口径"——对。计费的缺陷大多出在数据在哪里被改了、改了几次、口径对不对。

- 答"时序图更清楚"——时序图看得清调用，看不清"一笔话单过五次加工后，金额字段被谁改成什么"。这恰恰是计费的要害。

- 答"两个都用，数据流看口径，时序看调用"——最专业。模型可以组合，主模型加辅助模型。

- 收口：选型题的姿势就是主模型加辅助模型，并且说清各自回答什么问题。

> **易错点 / 考点：**① 四种模型的建模标准：商业建模标准、UML、UML、IDEF，这组对应是选择题高频；② 四个短板要记：泳道图并行异步差，数据流图不表示执行者；③ 判断陷阱"时序图适合业务人员评审"——错，业务人员看不懂对象消息；④ 记忆抓手：先想我要回答什么问题，再选图。

> **过渡：**模型是画法，画法要靠工具落地。

【10.25】分析工具（2 分钟）

> 有人把测试分析做成了画画比赛，图很漂亮，交上去没人能用。

>

> 课件列的工具很朴素。业务流程图、UML 图，用例图、活动图这些，还有建模工具、思维导图、Excel。结论先说：工具选哪个，看团队已有的习惯。可分析结果得能追溯。测试项追不到需求、缺陷追不到测试项，用什么工具都白搭。

>

> 按用途分三类。第一类，画流程和结构的：业务流程图、UML 图、专门的建模工具。解决的是复杂关系看不见的问题。

>

> 第二类，理思路的：思维导图。它在测试分析里特别好用，因为形状天然就是分解。中心是系统，一层层往外长功能项、子项。用它做功能分解，比在 Word 里敲三级标题快。哪一枝长得太粗一眼就看见，那枝就是

> 复杂、得重点测的地方。

>

> 第三类，管数据的：Excel。别嫌它土。测试项清单、需求映射矩阵、优先级表、风险清单、WBS 估算表。工业界大多数测试资产，最初都长在 Excel 里。因为它能排序、能筛选、能透视、能共享。

>

> 还有一类不显眼的输入：缺陷库和历史数据。它们不画图，可是最好用的分析材料。上一版哪里炸得多，这一版就往哪里压。

>

> 选工具有个实用标准：它产出的东西能不能直接变成下游要用的表。画得漂亮却导不出清单的图，通常只能当汇报材料。还有一条常被忘掉：工具得能共享。测试资产是团队的，锁在一个人电脑里的 Excel 等于不存在。放共享位置、有版本记录，才算数。

提问：为什么工业界大量测试资产最后还是落在 Excel 里？

> **预设回答：**

- 答"因为大家只会 Excel"——有一点道理，不是主因。

- 答"因为要排序、筛选、透视、共享，还要能贴到评审会上"——对，这是刚需。

- 答"因为专用工具太重，导入导出还麻烦"——现实理由。很多测试管理工具最后都得能导 Excel。

- 收口：会拿 Excel 建一张好的测试项表，是测试岗的基本手艺，别嫌它土。

易错点 / 考点：① 工具的三类用途划分；② 高频判断"分析必须用专业建模工具"——错，工具从属于目的；③ 追溯是分析质量的核心标准：需求、测试项、用例、缺陷四级要打通；④ 案例题常问"怎么保证可追溯"，答映射矩阵加编号体系。

> **过渡：**方法讲了三页，来看一个具体的分析示例。

【10.26】示例（2 分钟）

> 先看一段业务描述，你先别管它是什么系统。

>

> 客户提交贷款申请，系统校验资料完整性，客户经理初审，额度超过五十万转分行复审。三百字不到吧。课件这张示例没配文字说明，我就按它的路子把这段话拆一遍，你学动作。

>

> 第一个动作，读。把这段话读成角色、动作、对象、规则。上面那段读出来，四个角色，五个动作，两条规则。

>

> 第二个动作，串。把动作串成流程，标出分支和异常。资料不全怎么办，初审不通过怎么办，复审超时怎么办。异常分支往往是缺陷最密的地方。正常路径开发会自己反复点，异常路径常常没人走。

>

> 第三个动作，拆。流程里每一步变成候选测试项，每条写上判据。比如"资料完整性校验"这一项，判据是两条。必填缺失要给明确提示，不许提交。附件格式不符要拒绝，还要说明原因。

>

> 第四个动作，归。测试项分到功能线和非功能线两条线上。还是这个例子，非功能那条至少派生三条。并发提交同一份申请会不会重复建档，这是数据一致性。复审超时的时限配置能不能调，这是可配置性。客户资料展示是否脱敏，这是安全性。

>

> 四个动作走完，三百字变成十几条测试项。手艺就这四步：读、串、拆、归。

>

> 第四步是作业里最缺的。大家容易把业务描述拆成一堆功能点就收工，非功能那条线整条忘掉。每读完一段业务，追问一句：这里有没有质量属性上的要求。这一问，就是功能点和测试分析的分界。

提问：为什么说异常分支往往是缺陷最密集的地方？

预设回答：

- 答"因为异常难写"——对了一半，还有更深的原因。

- 答"因为正常路径开发会自测，异常路径常常没人走"——准确。自测的覆盖偏差直接决定缺陷分布。

- 答"因为异常处理的需求常常没写清"——也很对。需求里往往只有一句"系统应妥善处理"，妥善是什么，没人定义。

- 收口：所以分析时专门列一张异常清单：输入异常、时序异常、并发异常、外部依赖异常。

> **易错点 / 考点：** ① 四个动作的顺序；② 高频考点：异常分析的四个维度（输入、时序、并发、外部依赖）；③ 判断陷阱"测试项只从正常流程产生"——错；④ 案例题常给一段业务描述让你列测试项，功能和非功能都得有。

> **过渡：** 测试项有了，往下是最考判断力的一步：排优先级。

【10.27】明确测试项的优先级（2 分钟）

> 如果明天就要上线，你先测哪一块？

>

> 五个依据，挨个说。一，产品质量风险的识别。风险高的先测、深测。风险从哪儿来：复杂度、变更量、历史缺陷、技术新颖度、依赖外部系统。

>

> 二，功能重要性排序。按业务价值排：核心交易链路、辅助功能、边缘功能。判断方法很直接——这功能坏了，业务会不会停。

>

> 三，风险分析里的风险矩阵。把可能性、影响做成两个维度，落在"高可能乘高影响"那一格的，就是最高优先级。

>

> 四，经验与历史数据。上一版哪里问题多，生产上哪里报障多，这一版就往那儿加权。这一招最便宜，不用额外的分析成本。

>

> 五，80/20 原则。百分之二十的功能扛着百分之八十的风险或使用量。先揪出那 20%，测透它们，边际收益最高。

>

> 五个依据不是并列的。我建议先用第二和第五粗排，再用第一和第四加权，最后用第三校验。粗排定方向，加权修偏差，矩阵检查有没有明显排错。

>

> 优先级不要超过三级。P0、P1、P2 够了，分五级最后没人分得清。每一级要有定义。P0 是不通过就不能发版。P1 影响主流程但有绕行方案，P2 影响体验不影响业务。

>

> 还有一层：优先级是跟业务方对齐期望用的。时间不够砍掉 P2，得让业务方知道并且同意。所以它得写进测试项表，别留在脑子里。写下来才能被人评审、被人质疑，也才能在时间被砍时说得清砍了谁。

> **提问：**如果业务方说"每个功能都重要，都要 P0"，你怎么处理？

> **预设回答：**

- 答"那就都设 P0"——等于没有优先级，时间一紧就抓瞎。

- 答"按我的判断硬排"——容易跟业务方对立。

- 答"请他回答一个问题：上线当天只能保证三件事不出错，是哪三件"——这是最管用的办法。用资源约束逼出真实排序。

- 收口：排序的本质是取舍，取舍得有个约束条件才谈得下去。

> **易错点 / 考点：**① 五个依据；② 高频考点：风险矩阵的两个维度（可能性、影响）；③ 判断陷阱"优先级定下来就不再调整"——错，随风险滚动调整；④ 案例题常问"怎么说服业务方接受优先级"，答用资源约束把问题场景化。

> **过渡：**优先级排完是什么样子？接着示例往下看。

【10.28】示例 – 续（2 分钟）

> 排顺序这事，看着像行政活，其实最考判断。

>

> 上一页手里是一堆测试项。排队列，就是给它们定先后。

>

> 第一条原则，先跑通主干，再钻细节。第一批一定是主业务流程的端到端验证，从入口走到出口，确认这条路通。主干不通，后面的细节测试全在流沙上跑，测出来的缺陷还会被主干缺陷盖住。

>

> 第二条，高风险模块早测。早测值钱在早发现早改。同一个缺陷，设计阶段改和上线前一晚改，代价差着数量级。缺陷成本随阶段往上走，这是基本规律。

>

> 第三条，依赖关系决定顺序。A 功能依赖 B 接口，B 就得先测。只看重要性不够，还得看拓扑：底层的、被依赖的先测。

>

> 第四条，为回归留位置。回归用例的优先级通常高于新增的低优先级用例，因为它保护的是已经在用的东西。

>

> 排完得到一张表：测试项、优先级、依赖、预计工作量、责任人。这张表就是下一页要汇总的东西。

>

> 有个细节常被忽略：同一个测试项，不同阶段优先级可能不一样。批量导入在功能测试阶段是 P1，到性能阶段可能升 P0，因为导入是最容易把系统压垮的操作。优先级带着场景走，不是绝对数。

>

> 再给个做法：依赖单列一栏。排期先按依赖画拓扑，再按优先级微调。不这么干，你的顺序执行时天天被阻塞，优先级排得再准也跑不动。

提问：为什么"批量导入"这类功能在性能阶段优先级会升高？

预设回答：

- 答"因为它慢"——是现象，不是原因。

- 答"因为它是资源消耗最大的操作，一次性处理大量数据，内存、数据库、事务都吃紧"——准确。

- 答"因为用户往往集中在某个时点导入，比如月初，形成峰值"——这就把业务特征说清了。

- 收口：所以优先级要分开看功能重要性和风险重要性，两者不一定一致。

> **易错点 / 考点：** ① 排优先级的四条原则；② 高频考点：缺陷成本随阶段上升，越晚发现越贵；③ 判断陷阱"优先级只由业务重要性决定"——错，风险、依赖、阶段都算；④ 案例题常给一组测试项让你排序并说理由，理由要落到四条原则上。

> **过渡：** 前面这些结果汇总起来，就是一份测试需求分析结果。

【10.29】示例：测试需求分析结果（2 分钟）

> 分析做完了交什么？交一份能让别人接着干的东西。

>

> 课件这页没铺文字，可产出物的样子很固定。通常四张表。

>

> 第一张，测试项清单：编号、测试项名称、所属功能或特性、来源需求编号、优先级、备注。编号别随手编，用"业务域-功能-序号"，像 TRF-001、ACC-003，一眼看出归属。

>

> 第二张，需求与测试项映射矩阵。行是需求，列是测试项，交叉打勾。三个用途：证明覆盖、发现漏测、发现多余。

>

> 第三张，风险清单：风险项、可能性、影响、严重性、应对措施、责任人、跟踪状态。

>

> 第四张，非功能测试项表。性能、安全、兼容、可靠、易用，每一类写清指标口径、测试方法、依赖条件。

>

> 四张表靠编号打通。这是作业里最弱的一环：表跟表之间没有编号关联，看着齐全，追溯不了。

>

> 再加一张容易被忽略的：不测项清单。哪些需求本期不测、为什么、谁确认的。评审的时候这张最能体现专业度，因为它证明你想过边界。

>

> 还有每条测试项的判据，哪怕只写一句通过条件。判据逼你想清"测到什么算过"。没判据的测试项，到执行阶段一定变成"看着差不多就行"。

>

> 合格的标准很硬：把这份结果交给一个没参与分析的人，他能不能据此排出测试计划。能，合格；不能，说明还缺东西。

> **提问：**为什么"不测项清单"在评审时特别有价值？

> **预设回答：**

- 答"显得我们考虑周全"——表面理由。

- 答"因为它把为什么没测提前说清了，事后不会被追责"——实用，也是真实动机之一。

- 答"因为评审最容易出分歧的就是边界：有人以为测了，有人以为没测"——说得最透。边界不清是项目扯皮的第一来源。

- 收口：边界是写给所有可能误解你的人看的。

> **易错点 / 考点：**① 分析结果的四张表；② 高频考点：需求映射矩阵的双向用途（正向查覆盖，反向查漏测）；③ 判断陷阱"分析结果就是一份测试项清单"——错，还有映射、风险、非功能和"不测项"；④ 编号体系是案例题的加分点。

> **过渡：**分析做得再好也可能不准。下面专讲造成不准的风险。

【10.30】造成测试分析不准确的风险（3 分钟）

> 有一类漏测，事后没人能怪你。

>

> 那是"不知道自己没测"。课件这页列了一串让分析不准的原因，我按七组归一下。

>

> 范围类：项目范围不清楚，项目之间耦合性强。耦合最麻烦，你以为动的是自己家，其实牵着别人家一大片。

>

> 需求类：产品业务没说清，具体需求不明确，需求变更频繁。变更频繁的后果不是多干点活，是你分析出来的测试项永远追不上需求，分析变成一次次重做。

>

> 复杂度类：系统复杂，分解分解不动。分解不动的时候，人容易退回"按界面点"，这是能力不够时的本能。

>

> 可测性类：系统特性不可测试。没日志、没接口、状态看不见，测试项只能写成人工观察，判据就变成主观的。

>

> 数据类：业务数据种类多、处理环节多。组合爆炸，是漏项的重灾区。

>

> 代码类：代码质量差、结构不好。同一组用例，在烂代码上跑出来的现象你解释不清，成本和判断力一起往上走。

>

> 环境类：环境配置复杂，用户配置项很多。配置项一多就是组合，组合恰恰最难穷尽。

>

> 右边那条线索比左边七条值钱。它把系统切成三段：新开发、继承特性、平台/维护特性。每段都标着问题高发区，箭头指向测试重点。意思是不同来源的代码，缺陷分布不一样。新开发集中在没做全、没做对。继承特性集中在改坏了、兼容破了。平台与维护集中在配置、部署、环境适配。用法就一句：按代码来源调整测试重点，别平均用力。

提问：为什么继承特性、也就是老代码，反而是问题高发区？老代码不是最稳吗？

> **预设回答：**

- 答"老代码稳定，不会出问题"——这是最大的误解。它的稳定有个前提：没人动它。

- 答"因为改的人不熟、文档缺失、没人敢动"——准确。典型风险是改一处、坏一片，而坏的那片往往没用例护着。

- 答"因为它跟新功能耦合，边界模糊"——也到位，新老系统并存的时候更明显。

- 收口：写回归用例，重点恰恰是那些看起来不会坏的老功能。

> **易错点 / 考点：** ① 七组原因至少答得出五组；② 高频考点：按代码来源区分测试重点（新开发、继承、平台维护）；③ 判断陷阱"老代码稳定，可以少测"——错，改动后的老代码是高发区；④ 案例题常问需求频繁变更怎么应对，答短周期、可追溯、测试项滚动更新。

> **【思政落点 3】**：测试分析做不准，最常见的原因不是技术不够，是**不肯承认自己没看懂**。需求里有歧义、有缺口，有人硬着头皮往下写用例，写完一大堆，最后测的不是人家要的。承认"这块我还没弄清"，然后去问、去确认、把假设写下来——这才是专业。工程上最贵的不是问问题，是把错误的前提一路执行到底。

> **过渡：** 分析走到这儿可以收口了。下面用六个问题做小结。

【10.31】小结：从基本问题出发完成测试分析（2 分钟）

> 这一节要收了，我想留一张能带走的东西给你。

>

> 课件把分析的全部成果收成五条。一，分解了产品质量目标，把"质量要好"拆成一条条能验的目标。二，基于风险给特性分了类，哪些高风险，哪些常规。三，定了测试的深度、广度和优先级，深度是测多细，广度是测多宽。四，定了测试的总体框架，知道各种测试活动怎么排。五，预估了测试结果，也就是缺陷趋势：大概出多少、集中在哪。

>

> 另半边是六个问题，这是整节的方法论内核。测试的对象和范围是什么，测试目标是什么。重点和难点是什么，深度和广度多少。先测什么再测什么，怎么评价测试的效果。

>

> 六问的价值在通用。拿到任何一个陌生系统，把这六条逐条答一遍，测试分析就完成了一半。

>

> 六问和五条成果是对得上的。前两问对目标和范围，三四问对深度广度和重难点。第五问对先后顺序和优先级，第六问对评价。答的时候别跳格，跳掉的那格最后都会变成返工。问题的顺序，就是分析的顺序。

>

> 建议把这六问抄在笔记本第一页。以后做课程设计、做实习项目，先答六问，再动手写文档。

>

> 还有一个用法：六问也是审别人计划的模板。拿一份计划逐条问过去，很快能看出哪一格是空的。

提问：六个问题里，哪一个最容易被答成空话？

> **预设回答：**

- 答"测试目标"——很可能，它最容易写成口号。

- 答"评价测试的效果"——也常见。很多人到项目结束都没想清"怎么证明这轮测试做得好"。

- 答"重点和难点"——同样高发。大家会写"重点是对公业务"，却不写难点在哪、为什么难。

- 收口：凡是要判断力的问题都容易写成空话。对策是每答一问都给出一个能核对的对象或数字。

> **易错点 / 考点：**① 六个基本问题是本章高频简答，得能默写；② 五条成果的名称；③ 判断陷阱"测试分析结束的标志是文档写完"——错，标志是它能指导测试计划和设计；④ 记忆抓手：对象、目标、重难、深广、先后、评价。

> **过渡：**分析完了，接下来把"测什么"换算成"多少工作量"。

【10.32】10.3 测试估算与进度安排（1 分钟）

谁能把"这系统测完要几天"答得不心虚？

>

> 进入 10.3，测试估算与进度安排。这节解决一个很俗但躲不开的问题：要多少人、多少天。

>

> 四小节。10.3.1 测试工作量估算，10.3.2 工作分解结构表方法。10.3.3 资源的安排，10.3.4 测试里程碑和进度表。

>

> 顺序本身有逻辑。先算总量，再算细项，然后配人配物，最后排成时间表。一条从数字走到日历的路，每一段的产出都得能被下一个人接手。

>

> 为什么单开一节讲估算？因为在测试项目里，估算是所有人都盯着、又基本靠拍脑袋的那个数字。拍高了被砍，拍低了天天加班，事后还说不清到底估错没估错。更麻烦的是，估错的代价不在测试组内部。数是给别人的承诺，改一次就得跟所有人重谈一次。所以这一节的分量，比它的篇幅重得多。

>

> 这节给五种估算方法，精度不同，成本也不同。功能点靠规模换算，WBS 靠任务分解，历史数据靠相似项目，经验法靠人。学它们不是为了算出精确数字，是为了让你的数字有出处、能解释、能修正。知道哪个假设一变，这个数会跟着怎么变，比那个数本身有用。

>

> 带着一个问题听：领导问"这个系统测完要几天"，你答的那个数是怎么来的。

提问：为什么测试估算比开发估算更容易被质疑？

> **预设回答：**

- 答"因为测试看起来就是点点鼠标"——外行的直觉，可它确实是质疑的来源。

- 答"因为测试的工作量依赖别人，版本什么时候来、质量多差，都不由你控制"——这是根子上的原因。

- 答"因为测试没有看得见的产出物"——部分对。测试的产出是证据和判断，不像代码那样摆在那儿。

- 收口：所以估算时把假设写出来。"按上一版缺陷密度估""按版本按期交付估"，假设一变数字跟着变，这才站得住。

易错点 / 考点：① 四小节名称与顺序；② 高频判断"测试估算是纯技术问题"——错，它强依赖上游交付的质量和进度；③ 估算三条底线：有依据、有假设、有区间。

> **过渡：**先看一个具体场面：拿到一张表，你判断得出它是干什么用的吗？

【10.33】这是干什么的？（1 分钟）

先别急着往下翻，看这张表。

>

课件这儿只有一张图，配一句问话：这是干什么的？答案我先不给，教你怎么读它。

>

一，看列头。有任务、工期、开始、结束、责任人、前置任务的，是进度表。有任务、工作量也就是人天、单价的，是估算表。有任务、人力、时间占比的，是资源分配表。列头定表的身份。

>

二，看粒度。任务是粗到"测试阶段"这一级，还是细到"编写某个模块的用例"这一级？粒度决定它能不能管进度。粗到只看得到阶段，它只能回答大概还要多久。细到每个动作一格，维护成本会压过收益。

>

三，看有没有依赖。有前置任务、有里程碑的，说明这份计划排过序，不是一堆任务的堆砌。三样看完，这张表是什么、能不能用，心里基本有数了。

>

为什么花一分钟讲读表？因为项目里你接手的往往不是从零做计划，是接手别人做了一半的计划。看懂别人计划的形状，比会画一张新表更常用。形状看不懂，你连找谁问都不知道。

>

落点在这儿：估算、进度、资源，最后都长成表。表的结构决定它能不能被执行、被跟踪。这三张表也不是三件东西，是同一份计划的三层，列头不同，回答的问题就不同。下面几页就把它们一张张做出来。

提问：一张表里只有任务名称和工期，没有责任人也没有前置任务，它能用来管项目吗？

预设回答：

- 答"能，有工期就能排"——排不了。没有前置关系，你不知道谁等谁；没有责任人，延了没人认。

- 答"不能，缺责任人和依赖"——对，这正是它的问题。

- 答"当汇报材料可以，当管理工具不行"——分得很准。汇报看结果，管理看责任和依赖。

- 收口：做计划宁可少写几个任务，也要把责任人和依赖写全。

> **易错点 / 考点：** ① 三种表的识别特征，用列头法；② 高频考点：进度表必备要素，任务、工期、起止、责任人、前置依赖、里程碑；③ 判断陷阱"有工期就是进度计划"——错。

> **过渡：** 先解决最上游的问题：工作量到底怎么算出来。

【10.34】测试工作量估算方法（2 分钟）

> 领导问一句"几天测完"，你脱口而出的那个数，敢不敢签字？

>

> 五种方法，从粗到细说。

>

> 功能点方法：用软件规模反推测试工作量。好处是有相对客观的规模基准，难处是得有历史生产率数据才换得动。

>

> WBS 方法：把测试工作拆成任务树，逐项估再汇总。最常用，也最容易被接受，因为每一项都能说清为什么是这么多天。

>

> 历史数据推算：找相似规模、同类型的项目外推。精度不错，前提是你有历史数据。这也是成熟团队要建项目度量库的原因。

>

> 经验法：靠团队或专家小组判断。别小看它。有经验的人估出来的数常常比公式准。他把"这次需求特别糙""这个组新人多"这类因素算进去了，公式算不进去。

>

> 综合方法：几种各算一遍，取交集或加权。最稳，也是工业界的常态，没人只信一种。

>

> 取舍原则很简单。项目越新、越没历史数据，越靠经验法和 WBS。越成熟、数据越全，越能上功能点和历史推算。

>

> 这里有个坑，每年都有人踩：估算结果给区间，别给点值。"18 到 22 人天"比"20 人天"专业，因为它承认了不确定性。

>

> 还有个做法值得学：估两遍。第一遍自下而上按 WBS 加总，第二遍自上而下凭经验给总数。两边差太多，说明有一处判断偏了，回去查那一处，那里就是风险点。

提问：为什么有经验的团队反而更爱用专家判断，而不是公式？

预设回答：

- 答"因为公式不准"——有一点，但公式在某些场景是准的。

- 答"因为专家能把定性因素算进去：需求糙、人员新、依赖多"——这就是原因。公式处理不了"这次甲方特别难沟通"。

- 答"因为专家判断快"——也是实情，评审会上要的是当场能定的数。

- 收口：专业做法是专家判断加 WBS 校验。先凭经验给数，再用分解逐项核对，差异大的地方就是风险点。

易错点 / 考点：① 五种方法名称；② 高频考点：WBS 与功能点的区别，前者拆任务，后者按规模换算；③ 判断陷阱"经验法不科学"——错，有数据支撑的经验判断精度很高；④ 估算给区间不给点值。

过渡：五种方法里功能点最有公式感，先看它。

【10.35】软件规模估算：功能点方法（2 分钟）

有没有一把量软件规模的尺子，换语言、换框架都不变？

>

> 功能点方法就是照这个思路做的。核心是一个公式加五类元素。公式：AFP，调整后功能点，等于 UFP，未

> 调整功能点数目，乘以 AF，影响因子。两步走：先数出未调整的功能点，再用影响因子调整。

>

> 五类元素是被数的对象。外部输入数 EI：系统从外面收进来的输入，录入表单、导入文件、接口请求。外部输出数 EO：系统往外给的输出，报表、导出、通知。外部查询数 EQ：只读的查询，列表页、详情页。内部逻辑文件 ILF：系统自己维护的主要数据实体，客户表、订单表。外部接口文件 EIF：本系统要用、但由别的系统维护的数据，比如从核心系统读到的客户主数据。

>

> 为什么分五类？因为实现和测试的复杂度不一样。同样是"一个功能"，只读查询和带事务的输入，测起来差得远。功能点用这五类把规模量化了。

>

> 对做测试的人，它的用处在这儿：给了你一把和实现技术无关的规模标度。不管 Java 还是 Python，功能点数是稳定的，所以能横向比项目、推工作量。

>

> 局限也得说清。功能点量的是功能规模，不量质量难度。一个高安全要求的模块和一个普通模块，功能点数可能一样，测试工作量差几倍。所以它只是起点，还得用风险和质量要求去修正。

>

> 顺手还有个用法：这五类元素本身就是一张检查表。每一类问一句"这类东西我们测了吗"，能捞出不少漏项。

> **提问：**一个系统的功能点数和去年那个项目一样，测试工作量就一定一样吗？

> **预设回答：**

- 答"一样，规模相同嘛"——最常见的错。规模相同只说明东西一样多，不说明一样难测。

- 答"不一定，还要看质量要求、技术栈、团队熟练度、代码质量"——准确。

- 答"还得看是不是老系统改造，改造比新建难测"——很好的补充。

- 收口：功能点给的是基数，工作量等于基数乘一串修正系数。

> **易错点 / 考点：**① AFP＝UFP×AF 这个公式要能默写；② 五类元素的英文缩写和中文名称对应，选择题高频；③ 高频辨析：ILF 是内部维护，EIF 是外部维护；④ 判断陷阱"功能点可以直接当工作量"——错，中间还要换

算和修正。

> **过渡：**基数有了，接下来看真正拉动测试工作量的是哪些因素。

【10.36】测试工作量估算（3 分钟）

同一套系统，两个人估出来的数能差一倍。

>

> 差在没想清工作量由什么决定。课件列六条因素，我把结论留到最后，那句最重。

>

> 一，测试任务由质量需求、测试目标决定。目标定得高，活就多。要"验一遍主流程"和要"证明扛得住峰值"，工作量差一个量级。

>

> 二，测试范围由产品新功能特性、或者测试任务决定。范围是第一个乘数，范围翻倍，工作量基本翻倍。

>

> 三，代码质量越低，测试工作量越大，回归测试的次数和频率都会加大。烂代码缺陷密度高，修一轮带出新的，回归就得反复做。质量差的代码，会让测试掉进"永远回归不完"的循环。

>

> 四，处在不同的开发阶段，工作量不同。同一个功能，早期介入和后期补测成本不一样，后期还要叠加返工和赶工。

>

> 五，自动化程度高，工作量越低。是越低，不是为零。自动化把重复执行的成本降下来，脚本开发和维护的成本得算进去。所以它对高频回归划算，对一次性验证不划算。

>

> 六，应用领域、技术、编程语言不同，估算方法也不同。金融、电信这类高可靠领域测试深度要求高；嵌入式、移动端还多出设备和兼容矩阵的成本。

>

> 课件那句结论照背：测试工作量是根据测试范围、策划任务和开发阶段来确定的。后半句是，测试范围和测试任务是估算的主要依据。范围加任务，就是这个意思。

>

> 最后补一条实践提醒。估算最容易犯的错是只算执行、不算准备。环境搭建、数据准备、用例编写、缺陷复测、报告撰写，这些常常占一半以上。只按跑用例估出来的数，一定不够。

提问：如果只按"执行用例的条数"估工作量，会低估哪些部分？

预设回答：

- 答"用例编写本身"——对，写用例常常比跑用例更花时间。

- 答"环境和数据准备"——这是最大的一块隐性成本，数据难构造时更明显。

- 答"缺陷复测和回归"——很重要。一个缺陷从提出到关闭，要走提出、修复、复测、回归、关闭几轮，每轮都要人。

- 收口：按任务类型分块估：计划、设计、环境、执行、缺陷、报告，六块都要有数。

易错点 / 考点：① 六条影响因素；② 课件那句结论是简答标准答案，得背准；③ 高频判断"自动化程度越高工作量越低，所以自动化一定划算"——错，要看回归频次；④ 准备成本常被低估，这是实践要点。

【思政落点 4】：估算这件事，最容易出的不是算错，是**先有结论再凑数字**。领导要两周，你就把工作量往两周凑。这种估算比不估算更糟，因为它给了团队一个假的安全感。按功能点、按 WBS 一步步算出来，哪怕结果不好看，那也是真实的信息——让人据此做决策，而不是拿数字哄人。

过渡：估算最常用的工具是 WBS，下面专门讲它。

【10.37】工作分解结构表方法 WBS（2 分钟）

为什么"测整个系统要多久"永远估不准，"给这张表加五个字段校验"却能估准？

>

> WBS 就是从这个问题里长出来的。三步。

>

> 第一步，列出本项目需要完成的各项任务。是任务，不是阶段。"测试阶段"不是任务，"编写登录模块测试用例"才是任务。

>

> 第二步，每个任务继续往下细分，可以分很多层，细到不能再分。标准是什么？能直接估算、能派给一个人、能判断做完没做完。三条都满足，就别再往下拆。

>

> 第三步，按任务的层次编号，形成完整的工作分解结构表。编号不是形式主义。2.3.1 这样的编号让父子关系一眼看清，后面的估算汇总、进度跟踪、责任分配都能挂在编号上。

>

> 课件给了它准的原因：测试工作量的估算依赖测试任务的细化，对每项任务分解，再按子任务估。通常分解粒度越小，估算精度越高。道理很朴素：估"测整个系统"没法估，估"这张表加五个字段校验要半天"就能估。

>

> 但有个反向的边界：粒度不是越细越好。拆到"打开浏览器"那一级，管理成本超过估算收益。工程上的经验是拆到半天到两天的量级，短到估得准，长到值得管。

>

> 还有个用法常被忘掉：WBS 不只是估算工具。它同时是责任分配工具和进度跟踪工具。同一棵树，挂上人就是责任表，挂上日期就是进度表。

提问："测试阶段"能不能作为一个 WBS 任务？

预设回答：

- 答"能，它是个大任务"——当层级名可以，当可估算的任务不行。它没法派给一个人，也没法判断做完没做完。

- 答"不能，得拆到具体活动"——对，比如拆成编写用例、搭环境、执行、复测、出报告。

- 答"看层级，第一层可以叫测试阶段，下面再拆"——这是标准做法。上层是分类，下层才是任务。

- 收口：判断一个节点是不是任务，问三句：谁做，做多久，做完什么样。

易错点 / 考点：① WBS 三步法；② 课件那句"粒度越小，估算精度越高"是简答得分点，但要知道它有下限；

③ 高频考点：WBS 的三个用途，估算、责任、跟踪；④ 判断陷阱"WBS 只用于估算"——错。

过渡：看一棵真实的 WBS 树长什么样。

【10.38】示例：WBS 方法（2 分钟）

一棵树画出来，最容易被漏掉的是哪一枝？

>

先看骨架。顶层是"某项目测试工作"。第一层按阶段分四到六块。测试计划与需求分析，测试设计与用例开发。测试环境与数据准备，测试执行，缺陷管理与回归，测试总结与报告。

>

第二层按对象分。测试执行下面再分单元测试、集成测试、系统功能测试、专项测试、验收支持。

>

第三层按模块或特性分。系统功能测试下面分存款、交易、清算、报表。

>

第四层才是具体活动：编写用例、评审用例、执行、记录结果。到这一层基本就能填人天了。

>

编号跟着长出来。4.3.2.1，就是第四块执行下第三类系统功能、第二组交易的第一项活动。看编号就知道它在树上的位置。

>

三个用法。估算：自下而上汇总人天得总量，再跟自上而下的经验值对照。找漏：把 WBS 和测试项清单对一遍，有没有测试项落不到任何任务上。有，就是计划漏了活。排期：同一层的兄弟任务大多能并行，父子任务得串行。

>

最容易漏的那一枝是沟通与评审类任务。用例评审、计划评审、缺陷评审、日报周报，不写进 WBS 就不算工作量，可它们吃掉的工时不小。

>

> 最后一句：WBS 要有人维护。拆完不更新，两周后它就跟现实脱节，变成一张只能挂在墙上的图。

> **提问：**WBS 和测试项清单怎么配合使用？

> **预设回答：**

- 答"两个是一回事"——不是。测试项清单回答测什么，WBS 回答干什么活，两者是多对多的关系。

- 答"拿测试项去核对 WBS 有没有漏活"——正确用法之一。每一项测试都得有对应的设计任务和执行任务。

- 答"用测试项的数量来分派 WBS 的工作量"——也很实用。某个模块六十个测试项，大致能推出它占多少工作量。

- 收口：两边对齐了，计划才是有内容的计划，不是一张有格式的表格。

> **易错点 / 考点：**① 测试 WBS 常见的四层结构；② 高频考点：WBS 编号的层级含义；③ 判断陷阱"WBS 做完等于进度排完"——错，还要加依赖、资源、日历；④ 沟通评审类任务要写进去，这是实践加分点。

> **过渡：**估算和分解都有了，接下来给测试加上风险这个维度。

【10.39】10.4 测试风险与测试策略（1 分钟）

> 时间被砍掉一半的那天，你手里得先有个说法。同样是砍，砍得有没有依据，差别很大。

>

> 进入 10.4，测试风险与测试策略。这节看着最像管理，其实最考判断。

>

> 三小节。10.4.1 测试风险管理计划，10.4.2 基于风险的测试。10.4.3 测试策略的确定。

>

> 它解决两个问题。哪些地方可能出事、出多大的事、怎么办，这是风险。资源不够、时间不够的时候这场仗怎么打，这是策略。往后三块，先回答风险从哪来、怎么定级、拿什么措施压。再回答一句：资源不够时先保哪块。这些都得落在纸面上，装在你脑子里不算。很多测试项目不是死在测不出来。是死在不知道在哪停手，什么都想测，最后什么都没测透。

>

> 为什么放同一节？因为它们是一体两面。风险告诉你哪里要用力，策略告诉你怎么用力。只识别风险不定策略，清单就只是一张吓人的表。只谈策略不看风险，那就是平均用力。

>

> 这节还有个实在的用处：它是向上沟通的武器。你能说清"这三块风险最高，只给我两周，我只能覆盖前两块"。你就不再是来要人、要时间的那个部门，而是能给取舍方案的角色。

>

> 带着一个问题听：时间被砍一半，你砍什么、保什么，依据是什么。

> **提问：**为什么说风险清单是测试人员向上沟通的武器？

> **预设回答：**

- 答"因为能证明我们很忙"——忙证明不了价值。

- 答"因为它把来不及变成了有依据的取舍"——准确。说不出风险，你只能说时间不够；说得出风险，你能说砍掉这块会带来什么后果。

- 答"因为管理层只听得懂风险"——有点道理，管理层更关心后果，不太关心过程。

- 收口：所以风险清单要落到后果上，别写"这块很复杂"。

> **易错点 / 考点：**① 三小节名称；② 高频考点：风险与策略的关系，风险定方向，策略定手段；③ 判断陷阱"风险管理是项目经理的事"——错，测试风险主要由测试组识别。

> **过渡：**先立概念：测试风险到底指什么。

【10.40】测试风险（2 分钟）

> 发版会上，"测完了"这三个字，你敢不敢往下签？

>

> 先把测试风险这个词的范围划清。课件讲两件事：风险从哪来，风险管理怎么干。

>

> 来源五条，得背。软件系统的复杂性，需求频繁变更。测试范围分析不到位、存在未知区域。测试的充分性也就是覆盖率不足，还有一些不确定性。

>

> 两条展开。范围分析不到位、存在未知区域，这条最要警惕。它不是已知没测，是不知道自己没测。已知的缺口能沟通、能补；未知的缺口只能等它上线自己暴露。

>

> 这一条讲的是覆盖率。测试不可能穷尽，所以任何测试都带着没覆盖到的风险。这是先天的，不是工作失误。

>

> 风险管理三个环节：风险识别、风险分析、风险防范。识别是列出可能出什么事，分析是算它多严重、多可能，防范是拿什么措施。顺序不能颠倒：没识别出来就无从分析，没分析就不知道要防到什么程度。

>

> 课件还有一句判断：软件测试总是存在较高的风险，风险管理就是设法降低或缓解测试过程中的风险。降低或缓解这个措辞要留意。风险消灭不掉，只能管。谁的报告写"风险已消除"，你得多看一眼。

>

> 最后一个视角转换：风险不是坏消息，风险是决策依据。管理层定能不能发版，靠的就是你这份清单。

提问："不知道自己没测"和"知道没测但来不及"，哪种更危险？

> **预设回答：**

- 答"知道没测更危险，因为明知故犯"——从责任看确实更重，可从可控性看反而更高。

- 答"不知道更危险，因为无法应对"——按风险管理算是更准的判断。未知的风险定不了价，也沟通不了。

- 答"都危险，处理方式不同：已知的做取舍决策，未知的靠探索式测试去挖"——最完整。

- 收口：测试里有一类活动专门对付"未知的未知"：探索式测试、交叉测试、缺陷模式的类比迁移。

易错点 / 考点：① 五条风险来源；② 三环节名称与顺序；③ 高频判断"风险管理就是消除风险"——错，是降低或缓解；④ "未知区域"这个提法要记住，案例题常考。

> **过渡：**三环节的第一步是识别，它有具体做法。

第 10 章讲稿（10.41–10.58，v3）

> 锚点与课件一一对应，不增删页。每页五项：口播、提问、预设回答、易错点 / 考点、过渡。口播按四拍：起 → 推 → 收。

【10.41】风险识别（2 分钟）

> 开项目会的时候，大家提的风险是不是总那几条？需求会变、人会累、环境不稳。提完散会，下次再提还是这几条。凭脑子想风险，结局就是这样。

>

> 课件给的答案很朴素：建一张风险项目检查表。为什么非得是表？人在会上只会想起最近出的事、最容易想起的事。上个月刚炸的模块，所有人都记得。三年没炸过的老模块，没人提。表能把要想的一条条摆上桌面，逼你逐个看过去。

>

> 表从哪来？两个来源：历史资料和头脑风暴。历史资料是上一版的生产事故、缺陷库里反复出问题的模块、同行业的经典事故。头脑风暴是把开发、测试、运维、业务拉到一张桌子上，各说各的担心。不同角色担心的事完全不一样。开发怕方案没验过，运维怕部署和监控，业务怕规则边界。测试最容易犯的错，是只从测试的视角看风险。

>

> 有个动作最常被跳过。课件的要求是：识别并确定程度，给出预防或处理措施。三件事都做完才算完。只列清单不定级，评审第一句就问：这条有多大？你打算怎么办？答不上来，清单等于没写。

>

> 骨架课后自己搭：需求类、技术类、资源类、进度类、质量类、组织类。六类过一遍，漏项就少了。

>

> 风险识别不是一次会。每个阶段开头重扫一遍，风险自己在变。

提问：头脑风暴为什么一定要跨角色？

预设回答：

- 答"人多想法多"——只是表面。

- 答"不同角色担心的东西不一样，合起来才是全貌"——对。开发怕技术难点，运维怕部署和监控，业务怕规则边界。

- 答"还能顺便把风险说成共识"——这是额外收获。清单没人认，执行时没人配合。

- 收口：风险识别的产出，一半是清单，一半是共识。

易错点 / 考点：① 检查表是课件点名的办法，别答成"开会讨论"；② 三个动作缺一不算完成：识别、定程度、给措施；③ 高频：头脑风暴的跨角色价值；④ 案例题给项目背景让列风险，按需求、技术、资源、进度、质量、组织六类答，基本不漏。

过渡：风险列出来了，下一步得给它算个大小。

【10.42】风险分析常用的模板（2 分钟）

两个公式，一张九宫格。风险分析里最像算数的地方，就这一处。

>

第一个：风险严重性 = 可能性 × 影响。是乘，不是加。为什么用乘？它说的是"两个都小才算真小"。可能性再低，影响大到业务停摆，这条不能放过。影响再小，天天发生，也够烦。加法会把一头的极端值摊平，乘法不会。

>

第二个：风险危害 = 可能性 × 后果严重程度 × 范围。比前一个多一个范围。多出来的这个因子最要紧：同一个缺陷，影响一个用户和影响全部用户，危害差着量级。所以评一条风险，你要多问一句：这会碰多少人、多少笔业务？

>

再看那张矩阵。横轴影响程度，分低中高。纵轴发生的可能性，也分低中高。九格，从很低、低、中等、高到很高。三乘三，最经典的那种。

>

> 矩阵两个用法。一是定级：高可能乘高影响那一格，先处理；低可能乘低影响那一格，接受或者挂着看。二是沟通：把风险画进格子里，比写一段话管用。"这个在最高那档"，非技术的人也听得懂。

>

> 应试有个点：对角线看着对称，其实不对称。测试里通常影响优先。影响极大的，可能性再低也要留应对，资金类、合规类都是这样。案例题里这一条很吃分。

>

> 最后一句：分析结果要定期重算。同一个风险，版本往前走，两头都在变。三个月前的红，现在可能已经绿了。

> **提问：**一个"千次操作错一次、一错就差一分钱"的问题，算高风险还是低风险？

> **预设回答：**

- 答"可能性低，算低"——只看了一头。

- 答"一分钱而已，不影响什么"——不懂业务。一分钱会污染整批对账，可能全量重跑，甚至把资金卡住。

- 答"按危害算：可能性低，但后果重、范围是整批，综合偏高"——对。

- 收口：定级前必须问清后果和范围，这两项不能测试组自己拍。

> **易错点 / 考点：**① 两个公式的因子，特别第二个多出来的"范围"；② 3×3 矩阵怎么读、怎么定级，选择题高发；③ 陷阱："可能性低就不用管"——错，影响优先；④ 案例题要定级，答案里得写可能性、影响、范围三条依据。

> **过渡：**公式是工具，来看一份真的测试风险。

【10.43】示例：测试风险（2 分钟）

> 风险清单写十行不难，写对十行很难。课件底部留了一句问话：还有什么高风险的项？就顺着这句往下补。

>

> 先看常见那几类。需求变更频繁，用例跟着反复重写。关键模块代码质量差，缺陷密度高，回归永远跑不完。

> 第三方接口不可控，联调卡在那儿。环境不稳，结论不可靠。数据难造，有些场景根本测不了。人员经验不足，高风险模块测不深。

>

> 我最盯的是另外几条，它们常被漏掉。

>

> 上游交付延期。测试的窗口是被上游压出来的，这是最普遍的进度风险。

>

> 性能和容量没底。功能都测过了，可从没在接近生产的数据量下跑过。上线首日的事故，多从这里来。

>

> 数据迁移和老版本兼容。凡是碰老数据、老配置、老版本的，几乎必出问题。

>

> 权限和安全边界。越权、信息泄露这类，功能测试天生覆盖不到。

>

> 回滚方案能不能用。出了问题退得回去吗？回滚本身也要测，这一整条经常被忘干净。

>

> 什么算高风险，有个笨办法好使：新、复杂、关键、历史多。没做过的，依赖多的，坏了业务就停的，以前老出问题的。四条占两条以上，就进高风险清单。

>

> 写法也有讲究。条目写成原因到后果。比如：第三方支付接口联调窗口只有两天，若延期，支付相关三十个测试项无法执行。评审的人一眼能判断要不要给资源。

提问：回滚方案为什么也算测试风险？

> **预设回答：**

- 答"回滚是运维的事，跟测试没关系"——最常见的误解。回滚脚本、数据回退、版本切换，都验过吗？

- 答"回滚没测过，真出事退不回去"——对，后果是把故障时间放大好几倍。

- 答"回滚本身也是一次变更"——很专业。反向变更一样带副作用，同样要验。

- 收口：高风险清单要多问一句，正常路径之外，还有哪些异常处置路径没验过。

> **易错点 / 考点：** ① 常见测试风险的类别；② 高频：高风险项的四个判断词（新、复杂、关键、历史多），占两条即可；③ 陷阱："回滚归运维，不在测试范围"——错；④ 条目要写成能评估的样子：原因加后果，带上数量。

> **过渡：** 风险认清了，接下来是怎么办。

【10.44】风险常用控制方法（2 分钟）

> 排期表上一天空格都没有的时候，这份计划多半要出事。上游任何一天延期，都直接压到测试窗口上。

>

> 风险控制有五条，我按事前、事中、事后重排，好记。

>

> 事前两条。第一，做好风险管理计划，制定应急处置方案。是应急方案，不是注意事项。真出事那晚，第一步打给谁、第二步切哪个开关，得写在纸上。第二，估算资源、时间、预算时留余地。留余地不是偷懒，是承认不确定。计划排得刚好，等于假设一路顺利。

>

> 事中两条。第三，采取措施避免可以避免的风险。环境不稳就早点搭两套，数据难造就提前动手，能绕开的先绕开。第四，把高风险转成低风险。转移不是甩锅，是改结构：一个大版本拆成两次小发布，一次全量迁移改成分批迁移。风险还在，可每次的爆炸范围小了。

>

> 事后一条。第五，降低躲不掉的风险带来的损失。灰度发布、开关控制、限流降级、快速回滚，这些手段的共同点是把全量失败变成局部失败。

>

> 五条里最被低估的是第二条。现实中的计划常常刚好排满，一个意外就全线延期。延期挤测试，测试被挤就少测，少测后期事故多，事故多又没时间补。这个圈一旦转起来，越转越紧。

>

> 有句话可以直接搬到评审会上。这个排期没有缓冲，上游任何一天延期都会压掉测试窗口。请确认能不能接受这个风险。

提问："把高风险转移为低风险"，测试里怎么转？举个场景。

预设回答：

- 答"买保险"——那是财务上的转移，测试里更常见的是改结构。

- 答"大版本拆成两次发布，第一次只上低风险功能"——对，单次影响面变小。

- 答"全量迁移改成分批，每批验完再往下"——典型。

- 收口：转移的实质是改交付结构，让风险落在能控的粒度上。

易错点 / 考点：① 五条控制方法；② 高频：缓冲的意义，为什么留余地不算偷懒；③ 陷阱："风险转移就是推给别人"——错，是改结构；④ 案例题问"排期无缓冲怎么办"，答书面提示风险加给取舍方案。

过渡：风险这套讲完，换个词：策略。

【10.45】为什么要制定测试策略？（2 分钟）

资源永远不够用。这话听着像牢骚，可它就是测试这行的起点。

>

> 课件引了两段英文定义。第一段：Strategy，the art of devising or employing plans or stratagems toward a goal。策略是为达成目标去设计、运用计划与谋略的艺术。抓两个词。art，艺术，说明它没有唯一解。toward a goal，朝向目标，说明它得服务目标。脱离目标的策略，只是花架子。

>

> 第二段：an adaptation that serves an important function in achieving the goals。策略是一种适应，在达成目标里起重要作用。adaptation 这个词是关键。适应，意味着策略是对约束的回应。什么都给足，照流程做就行，哪里用得上策略。

>

> 两段合起来，回答了标题那句问话：为什么要有测试策略？因为资源不够、约束常在、目标又必须达成。策略就架在"必须做到"和"做不到全做"中间。

>

> 换个直白的说法：打不赢全面战争的时候，怎么把关键那几仗打赢。这就是策略。

>

> 这也解释了一件事。学生作业里最缺策略，因为课程设计没有真实的时间压力，什么都测一遍就行。没有约束，长不出策略。所以后面的案例，你要带着资源不够的假设去读。

>

> 再补个反例。没有约束，测试只回答一个问题：还有什么没测。有了约束，才要多答一个：这么多没测的里头，先测哪个、放掉哪个。后面这个，才是策略。

> **提问：**如果时间、人力、环境全给足，还要测试策略吗？

> **预设回答：**

- 答"不要，全测一遍就行"——理论上如此，现实里没有这种项目。

- 答"还要，因为'全测'本身也测不完"——准。穷尽测试不成立，还是得选。

- 答"还要，策略也管方法选择和层次划分"——补得好。策略不只是砍什么，还有怎么测。

- 收口：策略两层，选测什么，和决定怎么测。

> **易错点 / 考点：**① 定义里两个关键词，art 和 adaptation；② 陷阱："策略就是砍需求"——错，还包括方法与资源安排；③ 案例题问资源被压缩怎么调策略，要给取舍依据；④ 一句常考："没有约束就没有策略"。

> **过渡：**定义完了，看策略里到底装了什么。

【10.46】测试策略及其内容（3 分钟）

> 你自己写过的那份测试策略，里头有没有一句"不做什么"？多数人写不出来，因为写的其实是任务清单。

>

> 先看课件给的定义：在受到条件限制、不利因素的影响下，为了实现测试项目的目标而不得不采取的对策；

> 针对不同测试阶段、不同测试对象，范围、方法、类型要做必要调整。

>

> 这句话三个要害。一是受条件限制、有不利因素，前提是约束。二是为了实现目标不得不，说明策略是被逼出来的，不是想出来的。三是分阶段、分对象调整，一份策略管不了整个项目。

>

> 四项内容。第一，针对风险采取对策，工作量、时间有压力时怎么办，标准怎么取舍，任务优先级怎么排。第二，保证测试用例能执行、后续回归能跑起来。策略再漂亮，落不了地也没用。第三，选定用哪些技术和工具。第四，考虑影响资源分配的特殊情况，比如某个模块要专用设备，某类测试要外部专家。

>

> 最后那个词是重点：trade-offs，取舍，有得有失。策略的核心是我明确放弃什么，换来什么。不肯放弃的，那不是策略，是愿望。

>

> 给你一个句式，作业里能直接用：鉴于某个约束，我们选择这么做，放弃什么，代价是什么风险，由谁确认。这句写得出来，策略就站得住。写不出来，说明还在许愿。

>

> 考试也在这儿扣分。策略题最容易只写"要做什么"，不写"放掉什么"。阅卷看的就是有没有取舍意识。

提问：一份策略里没出现"不做什么"，算合格吗？

> **预设回答：**

- 答"算，写了要做什么就够"——那是清单，不是策略。

- 答"不算，没有取舍就没有策略"——对。不写放弃什么，等于宣称资源无限。

- 答"资源真够的话可以不写"——可以，但要在文档里补一句：本次资源满足全范围，无取舍。把"没有取舍"也做成结论。

- 收口：连"没有取舍"都要写出来，因为它也需要被人确认。

易错点 / 考点：① 定义的三个要害；② 四项内容的名称别混；③ 高频简答："为什么说策略的实质是取舍"；④ 陷阱："策略定了就不再变"——错，随阶段和风险滚动调。

> **过渡：**道理讲完，来道硬的：时间砍一半，你怎么测。

【10.47】问题：减少 50%时间，如何测试？（2 分钟）

> 把测试时间砍掉一半，你怎么办？这个问题每年都能吵起来。

>

> 先看现场常见的错答。第一种，加班把时间补回来。短期能顶，长期撑不住，人一累判断力下降，漏得更多。第二种，按原计划测一半用例。听着公平，其实最糟。随便砍一半，关键路径也可能被砍掉。第三种，直接跟老板说做不到。态度老实，可没给方案。

>

> 正路是分层做取舍。五步。

>

> 第一步，重排优先级。只保 P0 和一部分 P1，P2 明确写进"本轮不测"。要写在纸上，别只在嘴上说。

>

> 第二步，缩深度，不缩广度。宁可每个关键功能都走一遍，也别少数功能测到极致。广度不够，风险是未知的；深度不够，风险是已知的。

>

> 第三步，提效率。高频回归用例自动化，环境一键部署，数据准备脚本化。这些一次投入、长期回报，压周期时最划算。

>

> 第四步，改结构。分批发布、灰度上线。把一次全量验证，拆成多次小范围验证。让生产环境的小流量，接住一部分测试环境的活。

>

> 第五步，把风险写清楚。出一份未覆盖清单，配上对应风险，请决策者确认。这条不是甩责任，是把取舍变成组织的决定，而不是测试组自己悄悄让步。

>

> 五步里最要紧的是第二步的判断：宁可测浅，不可测漏。

> **提问：**为什么"缩深度"比"缩广度"安全？

> **预设回答：**

- 答"深度不好判断，广度好判断"——有道理。广度有清单能核对，深度很难说够深了。

- 答"整块没测是未知，测得粗是已知"——准。未知的风险没法估，已知的能兜。

- 答"用户先撞主流程，深度问题多在边界才露"——补得好，符合缺陷暴露的顺序。

- 收口：时间紧，先保都碰到，再谈挖多深。

> **易错点 / 考点：**① 五步取舍法；② 高频："缩深度不缩广度"的理由；③ 陷阱："所有用例减半执行"——错，要按优先级砍；④ 案例答案要素：重排优先级、自动化提效、分批灰度、风险清单加确认。

> **过渡：**取舍原则有了，看几个已经在用的策略。

【10.48】示例：测试策略（2 分钟）

> 课件连问三句：之前的回归策略还记得吗？Pairwise 和正交试验法隐含了什么策略？还有什么常用的测试策略？三句都得接住。

>

> 第一问，回归策略。常见三类。完全回归，全部用例再跑一遍，最稳也最贵。选择性回归，按影响分析挑出受影响的用例跑，性价比高，吃的是影响分析的质量。随机回归，按比例抽样，适合用例量巨大、变更又很局部的场景。选哪一类，本身就是一次取舍：拿时间换风险。

>

> 第二问，Pairwise 隐含什么策略？四个字：采样代表全体。组合爆炸，你测不完所有组合。于是先假设多数缺陷来自两个参数的交互，再用少量组合覆盖全部两两配对。这个假设本身就是策略：用已知的缺陷规律，换一个跑得动的用例规模。它不保证不漏，但它让漏变得可预期。

>

> 第三问，还有什么常用策略。课件给了两个。交叉测试，不同的人测别人写的模块，用不熟悉去对抗思维定势。还有 ST 加 ET，也就是测试执行的两段论。先拿脚本化测试打底，再用探索式测试去挖脚本想不到的地方。

>

> 三问放一起看，会看出一件事：策略无处不在。选回归范围是策略，选用例生成方法是策略，选人员组合是策略，选执行顺序也是策略。

>

> 所以这一节不是背名词。你要长出一个反射：每次做测试决定，都问自己一句，我这用的是哪种策略，放掉了什么。反射长出来，你就从执行的人变成设计的人。

提问：选择性回归最依赖什么？最大的风险在哪？

> **预设回答：**

- 答"依赖用例库大小"——不是关键。

- 答"依赖影响分析准不准，分析漏了，该回归的没回归"——对，用分析质量换执行成本。

- 答"还依赖变更记录够不够细"——补得好。改了什么说不清，影响分析无从下手。

- 收口：选择性回归得配两样东西，变更清单和影响矩阵。缺一样，就退化成凭感觉回归。

易错点 / 考点：① 三类回归策略的名称与适用场景；② 高频：Pairwise 的隐含假设，多数缺陷源于两参数交互；③ 陷阱："完全回归永远最优"——错，成本可能扛不住；④ 记 ST 加 ET 的组合思路：脚本打底，探索补漏。

> **过渡：**第三问里的两段论值得单独拆，它是本章最实用的一个模型。

【10.49】两段论（2 分钟）

项目里最容易出事的那段时间，是里程碑前还是里程碑后？多数人的直觉是前头，实际经验是后头。

>

> 两段论把测试执行切成两段。图上一半是发现缺陷，强调效率。里头有单元和集成测试、新功能验证、自动

> 化每日验证 SVT 和 BVT、非功能性的系统测试。这一段越快越好，目标是把缺陷尽快挖出来，把质量底数摸清。为什么讲效率？缺陷发现得越早越便宜，这一段问题最多，也最需要快速迭代。

>

> 另一半是验证功能，降低风险。功能验证、剩余缺陷验证、确认测试、回归测试、探索式测试、针对高风险区域的功能测试。目标变了，不再是挖，而是确认：确认修好了，确认没带坏，确认风险能控。这一段讲的不是效率，是可信度。

>

> 中间那条线标着关键、里程碑。分界就在这儿。线之前，你在探质量。线之后，你在证质量。判据、节奏、心态，全不一样。

>

> 两段混了会出什么事？最常见的是里程碑之后还在大改代码。测试已经进到证明模式，改动一来，之前验过的结论全被打断，回归永远追不上，最后带风险上线。所以两段论不只是测试策略，它也是对项目节奏的约束：过了里程碑，改动要有门槛。

>

> 还有个细节。探索式测试被放在第二段。很多人以为探索式是早期乱点，实际它更常用来在后期定向挖高风险区域。为什么？因为到那时候，你才知道风险在哪。

提问：探索式测试为什么多放在第二段，而不是一开工就做？

预设回答：

- 答"早期没东西可测"——不准确，早期也有可测对象。

- 答"后期才知道风险在哪，探索才有方向"——对。探索的价值取决于假设质量，假设来自前面积累的信息。

- 答"它能补第二段的盲区"——到位。证明模式容易只看已知，探索负责找未知。

- 收口：先广撒网，再定点挖。探索两段都能用，目的不同。

易错点 / 考点：① 两段各含哪些活动，SVT 和 BVT 属第一段；② 高频：两段目标差异，发现缺陷对验证功能；③ 陷阱："过了里程碑还能随便改代码"——错，这正是要防的；④ 分界线是"关键／里程碑"。

过渡：策略不是凭空来的，它被八个东西卡着。

【10.50】测试策略影响因素（2 分钟）

> 同一套测试方法，换个团队就落不了地。差别常常不在方法上。

>

> 影响策略的因素，课件列了八个。我一条条过，顺带说它把你推向哪种策略。

>

> 测试方式，静态还是动态，用不用探索式。偏静态省钱，覆盖行为少；偏探索灵活，难度量。

>

> 测试方法，黑盒还是白盒。有源码、有覆盖率工具，白盒才用得上；系统是外购的，只能黑盒。

>

> 测试层次，单元、集成、系统。层次决定你在哪个粒度上发现问题。单元最好定位，系统最接近真实。

>

> 测试人员，责任、能力、独立性。这条最常被忽略。团队里没人懂性能调优，你写再细的性能策略也执行不了。独立性问的是，测试的人是不是写代码的人。

>

> 用例的选择和优化，有没有优先级。有优先级，时间不够才能砍尾部；没有，只能全跑或者乱挑。

>

> 测试环境，搭起来简单不简单，能不能自动部署。环境能不能一键起，直接决定你能跑几轮回归。

>

> 测试工具，能不能用、好不好用。可获得性和上手难度，会改掉整个性价比。

>

> 质量标准，国内标准还是美国 DO-178C 那类严格标准。标准决定什么叫达标，它是策略的天花板。航空那套标准对追溯和覆盖的要求，会重塑整条测试流程。

>

> 八条里，第四和第六最硬：人和环境。很多策略不是想错了，是人和环境撑不住。

> **提问：**为什么说环境能力就是测试频率的上限？

> **预设回答：**

- 答"环境快就能多跑几轮"——只说到现象。

- 答"每轮回归都要部署、造数据、清理，环境不行，这些时间把轮次吃掉了"——对。

- 答"环境不稳还会让结论不可信，跑三轮不如跑一轮靠得住"——更进一步。不可靠的环境把测试变成赌博。

- 收口：所以部署和环境这块不是杂活，它决定策略的上限。

> **易错点 / 考点：**① 八个因素的名称；② 高频：人员和环境的约束作用；③ 陷阱："策略只由测试目标决定"——错，八因素一起卡；④ DO-178C 是"标准影响策略"的典型例子。

> **过渡：**分析、估算、风险、策略都齐了，最后一步是把它们装进一份文档。

【10.51】10.5 测试计划的内容与编制（1 分钟）

> 10.5 是本章最后一节，也是收口：测试计划的内容与编制。

>

> 三个小节。10.5.1 测试计划的内容，10.5.2 测试项目的计划过程，10.5.3 制定有效的测试计划。

>

> 这个顺序有点意思。先讲计划里写什么，再讲计划怎么做出来，最后讲怎么才能做好。一般文档课不是这么排的，多半先甩一个模板，照着填。这里把过程和有效性放进了同一节。

>

> 为什么这么排？它想强调一件事：测试计划的价值不在文档本身，在编制它的过程。从模板复制、把空格填满、交上去，格式再标准也没用。经过讨论、有假设、有取舍、有风险应对的计划，三页纸也是好计划。

>

> 三节之间是递进的。内容决定评审时看不看得出问题；过程决定这份计划会不会写出来就过期；有效性决定它落到地上有没有人照着做。三个都缺，计划就只是一份漂亮附件。评审会上没人挑得出错，交上去也没人照着做。

>

> 这一节有两个落点要盯住。一是进入和退出标准，它是质量的闸门。二是计划的评审和滚动维护，它决定这份计划是活的还是死的。

>

> 留个问题给你下课自己想：你写过的那些计划，后来有人看过第二遍吗？

>

> 答案要是没有，那份计划大概只活到了评审那天。

> **提问：**为什么"填模板"式的测试计划没有价值？

> **预设回答：**

- 答"因为没人看"——那是现象。

- 答"因为空填满了，判断没做：范围为什么这么划、风险为什么这么排"——对。计划的核心是判断，模板只给位置。

- 答"模板让计划看起来完整，把没想清楚的地方盖住了"——很尖。形式的完整，是思考缺失最好的掩护。

- 收口：评审一份计划，别数页数，看每个数字后面有没有理由。

> **易错点 / 考点：**① 三个小节的名称；② 陷阱："格式比内容重要"——错；③ 本节两个落点：出入标准、评审与滚动维护。

> **过渡：**先看这份计划里到底有哪些条目。

【10.52】测试计划内容（2 分钟）

> 计划写什么，课件给了一份清单。清单不长，每条都有理由。

>

> 先记定义，简答常考：软件测试计划是指导测试过程的纲领性文件，描述测试目标和测试范围、测试项及其优先级、测试方法、测试风险与策略、资源、任务安排和进度等。

>

> 条目按逻辑分组记，比死背顺序有用。

>

> 身份和背景：测试计划标识符，也就是文档编号；项目总体情况简介。编号不是形式。计划改到第七版，你得知道评审的是哪一版。

>

> 对象和判据：测试项，test item，含需要测试的功能；测试项通过和失败的标准；测试中断和恢复的规定。中断恢复这条很少有人写，可它很实用。环境挂了、版本回退、关键人请假，测试怎么停、怎么续，得事先说。

>

> 方法和策略一条。交付物一条，测试完成要交哪些材料。执行安排一组：任务和进度表、环境要求、人员安排与职责、培训需求。风险一组：潜在的问题和风险。最后还有一个兜底项。

>

> 细节点看教材附录 B，考试以教材为准。

>

> 这里跟第 4 章接一下。第 4 章讲的规范八要素，角色、准入、输入、活动、输出、验证、准出、度量，正好是这份清单的骨架。角色对人员职责，准入对进入准则，输入对测试项，活动对方法与任务。后四个也一对一：输出对提交材料，验证对评审，准出对通过标准，度量对进度和覆盖率。拿八要素扫一遍，漏项一眼看见。

>

> 一个实操建议：第一页写一页纸的摘要，目标、范围、关键风险、里程碑、需要的支持。决策的人只翻第一页，后面几十页是给执行的人看的。

提问："测试中断和恢复的规定"为什么值得写进计划？

> **预设回答：**

- 答"因为可能出事"——太笼统。

- 答"因为被打断后最难的是从哪接着测：哪些结论还有效，哪些要重跑"——对，核心是判定结论还有没有效。

- 答"还牵扯版本回退，退回去以后旧的测试记录怎么算"——很实际，项目里天天发生。

- 收口：这条要写清三件事，什么算中断、中断期间结论怎么标、恢复后从哪一步继续。

> **易错点 / 考点：** ① 定义句和主要条目，简答高频；② 高频：与第 4 章八要素的对应；③ 陷阱："计划只写测试组内部的事"——错，还有环境、培训、交付物、跨组依赖；④ 细节以教材附录 B 为准。

> **过渡：** 清单里有两条被单独拎出来讲，就是进入和退出标准。

【10.53】进入/退出标准（2 分钟）

> 整个计划里最硬的两条，是进入准则和退出标准。

>

> 先看定义。进入准则，是对测试切入点的确立，满足什么条件才启动测试。退出标准，是某个阶段结束或者里程碑达到事先定义的要求。

>

> 翻成大白话。进入准则是门票，退出准则是放行条。没门票就开测，是在一个残破的版本上空转。没放行条就宣布测完，是在赌运气。

>

> 课件还有一句：每个阶段都要控制进入和退出标准，为的是保障测试质量。要盯住"每个阶段"这四个字。不是只有系统测试才有。单元、集成、系统、验收，每个阶段都该有自己的门票和放行条。只写一半的标准，执行时一定被含糊过去。

>

> 这两条为什么必须事先定义？事后定的标准，一定是被结果倒推出来的。项目延期了，就把退出标准从缺陷清零改成主要缺陷清零。技术上叫改口径，管理上叫移动球门。球门一动，标准就没有公信力。

>

> 还有个实操要求：标准要能判定。代码质量良好不算标准。编译通过、冒烟用例全过、没有未解决的阻塞级缺陷，这才算。判断办法跟前面讲目标时一样，换个人能不能核对。

>

> 最后说它的管理价值。这两条是测试组最硬的筹码。它把"我觉得还不能上线"变成"未满足退出标准第 3 条"。前者是意见，后者是事实。

>

> 还有一句：进入准则和退出标准最好成对写。有门票没放行条，执行时一定被含糊过去。

> **提问：**项目已经延期，老板要求降标准先上线，你怎么接？

> **预设回答：**

- 答"照做"——可以做，但必须留书面记录：哪几条被豁免、对应什么风险、谁批准。

- 答"坚决不同意"——立场清楚，可没方案，容易被绕过。

- 答"给三个选项：延期三天补测关键项，或带风险上线并准备回滚，或缩范围先上"——最专业。把降标准变成选方案。

- 收口：测试的专业性，在于把质量决策翻译成可选项。

> **易错点 / 考点：**① 两个定义，名词解释高频；② 高频：标准必须事先定义、必须能判定；③ 陷阱："退出标准可以随进度调"——错，调要走变更并留记录；④ 案例题问被要求降标准怎么办，答书面记录加给选项加写明风险。

> **【思政落点 5】**：事先定好标准、按标准放行，练的是规则意识。定规则不难，难的是进度压到头还守得住。把"我觉得可以上"换成"第 3 条没过"，是把个人判断交给公开的标准。守这一条，比多补几个用例更能护住用户。

> **过渡：**退出标准长什么样？课件给了一个系统测试的例子。

【10.54】示例：系统测试结束标准（2 分钟）

系统测试什么时候算结束？这个问题不能靠感觉回答。

>

课件分两档。第一档，非严格系统，用基于测试用例的准则。功能性测试用例通过率达到 100%，非功能性测试用例通过率达到 95%。

>

两个数字的差别有讲究。功能必须 100%，功能不对，业务直接用不了，没有妥协空间。非功能给到 95%。这一块常受环境、数据、时段影响，客观上难稳定复现，留一点余量是现实的。

>

但有个坑要说清：通过率 95%，不等于可以有两成非功能项没测。通过率的分母是已执行的用例。分母本身不全，通过率再高也是假的。所以用这条标准，必须同时规定用例执行率。

>

第二档，严格系统，金融、航空这类，要补基于缺陷密度的准则。一条是连续几个小时或几天，测试发现的缺陷密度低于某个阈值，这叫缺陷收敛，说明系统在趋稳。另一条是代码的缺陷密度要高于某个阈值，比如上一个版本的平均值。

>

第二条乍看很怪。为什么要求缺陷密度高？

>

它的逻辑是：测出来的缺陷太少，可能是测得不深，不是质量好。这一版缺陷密度明显低于历史均值，先别高兴，先怀疑覆盖不够。

>

两条合起来看：用例通过率回答功能对不对，缺陷密度回答测得够不够、系统稳不稳。前者看结果，后者看过程。严格系统两条都要有。

提问：缺陷密度低于历史平均，为什么反而要警惕？

> **预设回答：**

- 答"因为质量变好了"——可能，但这是最不该先下的结论。

- 答"可能测得不够深：用例没设计到、执行不彻底、环境跑不通"——对。

- 答"也可能是缺陷漏报了，或者被合并统计了"——补得好，数据口径也会骗人。

- 收口：看到异常漂亮的数据，先验证数据从哪来，别先庆祝。

> **易错点 / 考点：** ① 两档准则的具体数字，100% 和 95%，选择题高频；② 高频："代码缺陷密度要高于阈值"的原因；③ 陷阱："通过率 100% 就说明测完了"——错，还要看执行率和覆盖；④ 分清结果判据和过程判据。

> **过渡：** 最后两页，把"计划"这件事重新认识一遍。

【10.55】软件测试计划更是一个过程（2 分钟）

> 计划这个词，课件特意用了英文的动名词：Planning。

>

> 名词是那份文件，动名词是持续做计划这件事。差一个后缀，差两种工作方式。

>

> 为什么计划必须是个过程？三个理由。

>

> 假设会变。你计划时假设需求两周内冻结，第三周它又变了。假设环境下周就绪，结果拖了一个月。假设一垮，靠假设搭起来的计划就失效。计划要是一动不动，多半已经和现实脱节。

>

> 认知会变。测了一周，你对系统的理解远超第一周。风险排序变了，优先级变了，策略该跟着变。

>

> 计划本身还是沟通工具。每次更新，都是一次重新对齐：谁负责什么、什么时候交、风险变没变。这个过程的价值，常常大于文档本身。

>

> 所以做法是滚动维护。固定节奏，比如每周更新一次，每次更新三件事：进度的实际值、风险的变化、下一步的安排。别一版到底，那只是把计划变成历史文件。

>

> 谁来更新？项目经理牵头，测试负责人动笔，开发给交付时间。更新完发给谁？发给要拿它做决定的人，不是塞进共享盘就算完。这三件事不落实，滚动维护也撑不住两个月。

>

> 配套还有个动作，本章小结里点了名：计划评审。评审不是走过场，它让计划被承诺。开发确认交付时间，业务确认范围，项目经理确认资源。没经过评审的计划，只是测试组自己的一厢情愿。

> **提问：**一份三个月没更新过的测试计划，说明什么？

> **预设回答：**

- 答"说明计划做得好，一直没变"——几乎不可能。三个月里需求、环境、人员不可能一点不变。

- 答"说明没人用了，实际靠口头协调"——最可能是这样。

- 答"说明它成了归档文件，只在对审计时有用"——也真实，不少计划只活到评审那天。

- 收口：判断一个团队的测试管理水平，最快的问法是，你们计划上次更新是什么时候。

> **易错点 / 考点：**① "Planning 是过程"这个观念，简答高频；② 高频：滚动维护要更新哪三项；③ 陷阱："计划批准了就不能改"——错，变更要走流程，但必须改；④ 计划评审的作用要答到：把计划变成共同承诺。

> **【思政落点 6】**：承诺这件事，写"十天完成"很容易，难的是到期那天敢把实际进度摆出来，说清差在哪、准备怎么补。工程里最伤人的不是延期，是瞒着延期，到最后一刻才说做不完，一屋子人的排期全崩。报实情、改计划、把话说在前面，这就是诚信落在具体工作里的样子。

> **过渡：**最后用一页把整章收起来。

【10.56】本章小结（2 分钟）

> 一章讲完，回头看它给了你什么。

>

> 小结五条，一条条对回去。

>

> 第一条，从产品需求到测试需求，逐层分析：业务、功能、用例、场景。这是 10.2 的主线。分析的实质是逐层翻译，不是抄需求。

>

> 第二条，清楚测试上下文，目标、条件、环境。这是 10.1。目标要能验证，准入要明确。同一个测试项，上下文变了，测法就变。

>

> 第三条，测试覆盖率是起点也是终点。这句最值得琢磨。起点是说，分析阶段就得定覆盖口径，按需求覆盖还是按风险覆盖。终点是说，评估时用它判断够不够。起点的口径没定，终点就没法判断。这是很多项目的通病。

>

> 第四条，风险识别、分析与防范。10.4 的三环节。风险消灭不掉，只能降低、只能缓解。

>

> 第五条，有效制定测试计划，计划评审也不能省。这是 10.5。计划是过程，评审让它变成承诺。

>

> 五条里第三和第五最能看出专业度。覆盖率是技术上的严谨，评审是管理上的严谨。缺一头，测试就只剩"做了很多事"，而不是"能对结果负责"。

>

> 串一句：分析决定测什么，估算决定花多少，风险决定先测谁，策略决定怎么测，计划把这些写成承诺。这句背下来，这一章就没白学。

>

> 再给个复习法。把五条还原成五张表：测试项表、需求映射矩阵、WBS 估算表、风险清单、计划正文。表画得出来，说明真会了。只会背文字，题目一变就懵。

> **提问：**"覆盖率是起点也是终点"，起点没定口径会怎样？

> **预设回答：**

- 答"测完不知道算不算测完"——对，没口径就没判据。

- 答"各方理解不一致，评审时吵起来"——很现实。开发按代码覆盖看，业务按需求覆盖看。

- 答"最后靠感觉，或者靠时间到了收工"——最常见的结局，也最不专业。

- 收口：口径要在计划评审时写死，比事后争论便宜得多。

> **易错点 / 考点：**① 五条小结，简答高频；② 高频："覆盖率是起点也是终点"的含义；③ 陷阱："计划评审是形式"——错；④ 能把五节串成一句话，案例题好拿分。

> **过渡：**理论讲完，下面的实验把这一章走一遍。

【10.57】实验 7：制定测试计划（2 分钟）

> 实验 7 是把前面所有内容走一遍的地方。

>

> 七步，一步一说，每步说清错在哪。

>

> 第一步，小组讨论，分析 SUT，重新明确测试目标，含功能和非功能要求。"重新明确"四个字要紧，不是抄项目简介，是把目标写成能验证的句子。易错：只写功能，忘了非功能。

>

> 第二步，定范围，哪些要测、哪些不测。易错：只写要测的，不写不测的和理由。这一步的产出要是双向清单。

>

> 第三步，把范围分解成测试项，含功能测试项和专项测试项。易错：按界面菜单切，不按业务能力切。专项那部分，性能、安全、兼容，经常整块漏掉。

>

> 第四步，用 WBS 或者敏捷里的扑克牌估算法，估每个测试项的工作量，再汇总。易错：只估执行不估准备，汇总完也不跟总工期对。扑克牌估算好在快，还能把分歧逼出来。两个人估同一件事差五倍，这件事一定有没想清楚的地方。

>

> 第五步，识别测试风险，分析和列出主要的几条。易错：写成"可能存在进度风险"这种空话。要写成原因到后果。

>

> 第六步，一起讨论对策。这一步的产出是对策，不是注意事项。

>

> 第七步，按 10.5.1 的主要内容，把前面汇总成一份简要计划。易错：前六步直接粘过来，不做取舍和摘要。第一页要是摘要。

>

> 交付要求说清楚：正文不超过五页，附一张测试项表、一张 WBS 估算表、一张风险清单。页数限制是故意的，它逼你取舍。取舍正是这一章要教的东西。

提问：扑克牌估算法为什么能提高估算质量？

> **预设回答：**

- 答"因为它快"——是优点，不是质量提升的原因。

- 答"因为它把分歧暴露出来，差五倍说明有人没想清楚"——对，分歧本身就是信息。

- 答"因为大家先独立出牌，不会被第一个说话的人带走"——很专业，这就是它比开会拍数字强的地方。

- 收口：估算的价值，一半在数字，一半在靠分歧发现理解偏差。

易错点 / 考点：① 七个步骤的顺序和要点；② 高频：估算要含准备成本；③ 陷阱："实验只交一份文档"——错，还要有能追溯的表；④ "不测项加理由"是必写内容。

> **过渡：**这一章到这儿。下一章把测试项变成一条条能执行的用例。

【10.58】感谢聆听（1 分钟）

第 10 章到这儿就结束了。

>

回头看一眼它给了你什么。一句话说，它给的是一套把模糊需求变成可执行承诺的办法。目标要能验证，范围要写清边界，测试项要能独立判定。优先级要有依据，风险要能定级、要有对策，估算要有出处，计划要能评审、能滚动维护。

>

这一章没教你怎么点界面、怎么跑工具。它教的是动手之前把事想清楚。这部分活最不出彩。没有人会因为你写了一份好计划给你鼓掌。可它决定后面所有事是不是白干。

>

下一章第 11 章，接着用这一章的产物：测试项变成测试用例，用例变成能维护的资产。你今天写的每一个测试项，后面都要有对应的用例、对应的执行记录。

>

作业提醒。实验 7 那份五页计划，按七步走。别落下两样：不测项清单，还有风险的原因到后果写法。这两处最容易被扣分，也最能看出有没有真做取舍。

>

课上到这儿，考试在前头。复习按重要性排，顺序是这样。先是 10.2 的分析。再是 10.5 的计划和出入标准。然后是 10.4 的风险与策略。最后是 10.3 的估算。简答必背四样：六个基本问题、计划的定义句、两个风险公式、两档结束标准。

>

计划不是文档，是过程。这句话带出教室，比背五个名词管用。

>

谢谢大家。下课。

提问：如果把这一章浓缩成一句能贴在工位上的话，你会写哪句？

预设回答：

- 答"计划不是文档，是过程"——好，抓住了 10.5。

- 答"风险消灭不掉，只能管"——也好，抓住了 10.4。

- 答"没有判据的目标都是口号"——最锋利，抓住了 10.1 到 10.2 的主线。

- 收口：三句都对，说的是同一件事的三个面：说清楚、算明白、敢承诺。

易错点 / 考点： ① 本章主线一句话概括；② 复习优先级：10.2 > 10.5 > 10.4 > 10.3；③ 简答必背：六个基本问题、计划定义句、两个风险公式、两档结束标准。

过渡： 下一章，设计并维护测试用例。

四、思政落点清单（共 6 处）

序号	所在页	落点主题
1	10.7	四句里最像负责的是第三句"不能漏测"，它把不可能的事许了出去。测试这行有条底线：
2	10.19	干系人这一层，考的是你肯不肯多问一句。工程上的事故，很多不是技术不行，是有些人根
3	10.30	测试分析做不准，最常见的原因不是技术不够，是不肯承认自己没看懂。需求里有歧义、有
4	10.36	估算这件事，最容易出的不是算错，是先有结论再凑数字。领导要两周，你就把工作量往两
5	10.53	事先定好标准、按标准放行，练的是规则意识。定规则不难，难的是进度压到头还守得住。
6	10.55	承诺这件事，写"十天完成"很容易，难的是到期那天敢把实际进度摆出来，说清差在哪、