《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）

> 各位同学，从这一页开始，我们进入**第 3 篇——软件测试项目实践**。
>
> 前两篇我们干了两件事：第 1 篇打地基，讲清楚什么是测试、什么是缺陷、测试分几类；第 2 篇铺方法，第 3 章讲怎么设计用例，第 4 章讲流程和规范，第 5 到第 7 章讲单元、集成、系统功能和各类专项。概括成一句：**你现在手里有一整套工具。**
>
> 可工具多不等于会干活。真实项目里没人问你"会不会等价类划分"，人家问的是"这系统两周后要上线，你打算怎么测，凭什么说它能上"。这个问题方法回答不了，得靠项目实践回答。
>
> 这一篇五章连成一条线：第 10 章测试需求分析与测试计划——先想清测什么、测到什么程度；第 11 章设计和维护测试用例——把"测什么"落成一条条能执行的用例；第 12 章部署测试基础设施——环境、数据、工具备齐；第 13 章测试执行、缺陷报告与跟踪——真的跑起来、把问题管起来；第 14 章展望。
>
> 这条线的顺序**不能颠倒**：没想清楚就写用例，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。
>
> 带着一个问题听：**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？**

> **板书：**中间写"第 3 篇 项目实践"；左侧写"第 1 篇 概念 | 第 2 篇 方法技术"；右侧竖排五章：10 计划 → 11 用例 → 12 环境 → 13 执行 → 14 展望；下方一句："从'会方法'到'能交付'"。

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 只能证明"你手气不错"，不能证明"系统可以上线"。

- 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"——也很实在，但环境搭成什么样，取决于你打算测什么，顺序上还是排在"想清楚"之后。

- 收口：这三答正好对上这一篇的章节顺序，顺序本身就是本章要教的第一条经验。

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

过渡：这一篇要交付的第一样东西，是"测试件"。我们先把这个词立住。

【10.2】软件测试工作和测试件（1 分钟）

这一页只讲一个概念：**测试件，testware**。它是"软件测试工作"的产物。

很多人以为测试的产出就是"我测过了"这四个字。不是。测试是一项**留下证据**的工作，它留下的一整套东西才叫测试件：测试计划、测试需求与测试设计规格、测试用例、测试数据、自动化脚本、环境配置说明、缺陷报告、测试报告、执行日志。这些东西有一个共同点——**别人拿着它，能把你做过的事重做一遍**。

为什么强调"能重做"？因为测试的价值在于**可复核**。你说"我测过了"，评审的人没法判断；你交一份带环境、数据、步骤、预期结果、实际结果、截图的记录，别人可以抽检、可以复现、可以在你休假时接着干。

打个生活化的比方：装修队干完活，交给你不只是"我刷好了"，还有水电走向图、材料清单、隐蔽工程照片、保修单。没有这些，你住进去心里没底；软件也一样，**测试件就是测试工作的"隐蔽工程照片"**。

行业里有个现象：出过事故的项目，事后复盘常常发现"当时其实测过"，但**记录丢了、结论没写、谁测的不**

知道。测过但没留下证据，等于没测。

所以这一章请你换一个视角：你不是在"找 bug"，你是在**生产一套可交付、可追溯的测试件**。

板书：中间写"测试件 testware"；左侧列产物：计划・需求/设计规格・用例・数据・脚本・环境配置・缺陷报告・测试报告・日志；右侧一句："别人拿着它能重做一遍＝可复核"。底部："测试工作＝生产证据链"。

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"——只说对了一件。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：测试件的完整性，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"要记住，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 常考判断："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。先看它由哪些重点工作组成。

【10.3】第 10 章 测试需求分析与测试计划（1 分钟）

我们正式进入第 10 章。这一章要解决的是测试项目**开工前的那一段**。

我先说清它的位置：它是第 3 篇的第一章，也就是说，后面四章的所有工作，都建立在这一章定下的目标、范围和计划之上。这一章定歪了，第 11 章的用例会写偏，第 12 章的环境会搭错，第 13 章的执行会失控。

这一章回答三个问题。第一，**测什么**——测试目标怎么定、测试范围怎么划、测试项怎么从需求里推出来。第二，**测到什么程度**——优先级怎么排、风险怎么识别、进入和退出准则怎么定。第三，**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。

这里要提醒一句：这一章的名字里有两个词，"需求分析"和"计划"。很多同学以为重点是后面的"计划"，因为

计划看起来像文档；其实**重心在前面的"分析"**。分析是把需求翻译成测试语言的过程，计划只是把翻译结果排成一张可执行的表。

还有一个态度问题：这一章的产物是要**签字承诺**的。你写下"系统测试 10 天完成、回归两轮"，就是在承诺资源、时间和质量。计划不是作文，是**契约的草稿**。

板书：中间写"第 10 章"；左侧"测什么：目标・范围・测试项"；中间"测到什么程度：优先级・风险・出入准则"；右侧"怎么安排：估算・进度・计划・评审"；下方一句："分析是翻译，计划是承诺"。

提问：如果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，计划写漂亮点好交差"——计划漂亮但分析空的，执行时天天改计划，改到最后没人看。

- 答"保分析"——更接近正确答案。分析对了，计划粗糙一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，但要有取舍意识：时间紧的时候，宁可把分析做扎实、把计划写成薄薄的三页。

- 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 本章的五节名称：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 常考简答："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 别把本章当成"文档写作课"，考试案例题给的是"给你一段需求，你列出测试项和优先级"，考的是分析能力。

过渡：先看这一章的目录，你心里有个时间预算。

【10.4】本章目录（2 分钟）

这是本章的路线图，我按顺序点一遍，你记结构，不用记细节。

10.1 测试目标和准则——先解决"为什么测、测到什么算开始、测到什么算结束"。这一节里有两组东西：一组是目标，包括通用目标和本项目具体目标；一组是准则，主要是进入准则。

10.2 测试需求分析——本章最重的一节，也是全章的地基。要讲清：测试分析为什么是基础、它的输入和输出是什么、分析过程分几步、范围怎么界定、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。

10.3 测试项目的估算与进度安排——把"要测的东西"换算成"要花多少人天"，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、怎么算严重性、怎么控制；策略是什么、为什么必须做取舍、有哪些常用策略。

10.5 测试计划的内容与编制——最后收口：一份测试计划到底写哪些条目、进入退出标准怎么定、计划为什么是个过程而不是一份文档。

你注意这五节的逻辑：**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"。想不明白就写，写出来的计划一定是抄模板。

时间上，10.2 占得最多，因为它是唯一一节"方法性"的内容，其余四节更偏管理口径。

> **板书：**竖排五节并标权重：10.1 目标与准则（轻）→ 10.2 需求分析（**重**）→ 10.3 估算与进度（中）→ 10.4 风险与策略（中）→ 10.5 计划编制（中）。右侧一行："前四节想，第五节写"。

提问：五节里，如果只给你两天准备一个测试项目，你会跳过哪一节？为什么？

预设回答：

- 答"跳过 10.3 估算，凭感觉排期"——现实中很多人这么干，代价是排期全靠拍脑袋，一遇延期就无法解释。

- 答"跳过 10.4 风险"——最不该跳的一节。风险不是"可能出问题"，是"一定会出问题，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"——这是最专业的答法：每节都留一个最小产出，比如风险就留一张五行的检查表。

- 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 常考对应关系题："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 记牢本章的落点是**测试计划**，前面四节都是它的输入。

过渡：进入 10.1 之前，先看一张总图：测试计划到底要重点抓哪几件事。

【10.5】软件测试计划的重点工作（3 分钟）

这一页是整章的骨架，一共有七项重点工作，我一项一项过，并且告诉你每一项对应本章哪一节。

第一，**明确测试目标**。注意"明确"两个字。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。这一项是 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：高风险的地方要多测、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"。估算方法 10.3 会讲。

第六，**测试资源、进度等安排**。人、环境、工具、数据、时间窗口，都要落到具体的人和日期上。

第七，**测试阶段出入准则**。什么条件才能开始测，什么条件才算测完——这是质量的闸门。

这七项下面还有一个括号标注的"**测试需求分析**"——它不是一个并列项，它是第二、三、四项的**共同底座**。换句话说：七项里有一半靠分析撑起来，分析塌了，计划就只剩格式。

你还应该看出另一件事：这七项跟第 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）几乎一一对应。计划不是新东西，是把规范落到这个项目上。

再看一层：这七项同时是评审的检查表。评审会上逐项问——目标可验证吗？范围写清"不测"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，计划的质量就基本托住了。

> **板书：**竖排七项编号：① 明确目标 ② 定范围（含"不测什么"）③ 测试项与优先级 ④ 风险与对策 ⑤ 工作量估算 ⑥ 资源与进度 ⑦ 出入准则。底部横线标"测试需求分析＝②③④ 的共同底座"。右侧写第 4 章八要素："人・门・料・活・果・验・出口・尺"。

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。因为目标最容易写成"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，评审时无法验证。

- 追问收口：这三项的共同病根是**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 七项重点工作的名称要能答出至少五项；② 高频判断："测试范围就是列出要测的功能"——错，不测的部分同样要写；③ 记住本章与第 4 章的对应关系，案例题里"按规范八要素评审一份计划"是常见题面。

> **过渡：** 七项里排第一的是目标。那我们就从"什么才算一个合格的测试目标"开始。

【10.6】10.1 测试的目标和准则（1 分钟）

> 进入 10.1。这一节只有两件事：**目标**和**准则**。
>
> 目标是"为什么测"，准则是"什么时候开始、什么时候算完"。为什么把这两件事放在一章的开头？因为它们决定后面所有工作的边界。目标含糊，你就不知道测多深；准则含糊，你就不知道什么时候能停。
>
> 这一节的讲法我安排成三段：先拿几条真实的"假目标"让你判断，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。
>
> 一句话概括这一节：**目标是可验证的句子，准则是不容商量的条件。**
>
> 提醒一句，这一节内容不多，但考试爱考，因为它能出一道很好的辨析题：给你四句话，问哪一句能作为测试目标。

> **板书：** 中间写"10.1 目标与准则"；左侧"目标＝为什么测（可验证）"；右侧"准则＝何时开始、何时算完（不容商量）"；下方写一句："目标是句子，准则是条件"。

> **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，对内沟通可以，写进计划不行"——很务实：口头可以用，**写进计划、写进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"做到了没有"这件事有唯一答案。

> **易错点 / 考点：** ① 本节两块内容：目标、准则（进入准则为主，退出准则在 10.5 展开）；② 常考辨析题："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 别把"测试目标"和"测试目的"当两个东西，本章是同一含义。

> **过渡：** 先做一道判断题，看看你心里的标准立住了没有。

【10.7】示例：哪个是测试目标？（2 分钟）

这一页是道判断题：下面这几句话，能作为测试的目标吗？有什么问题？

我把课上最常出现的四种句子念一遍，你跟着判。

第一句，"**保证系统上线后不出问题**"。听着很负责，问题在于它**不可验证**，而且它根本不是测试能承诺的事——上线后出不出问题，还取决于运维、数据、用户行为。测试能做的是"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？没有度量口径，测完你说达标，用户说不达标，没法裁决。

第三句，"**测试要充分，不能漏测**"。这句话把"充分"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，"**本轮测试覆盖新增的对公转账业务的全部 12 个业务场景，并回归个人转账的 8 个核心场景；致命与严重缺陷清零**"。这句就能当目标：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。

顺便说一句，写不出可验证的目标，通常不是文字能力问题，是**你还没想清楚要测什么**。

课上我会让你们把这几句各改一次。以第四句为模板改第一句，可以改成："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，并回归个人转账 8 个核心场景，致命与严重缺

陷清零"。改写的过程其实是在补做需求分析——句子写不下去的地方，就是还没想清楚的地方。

> **板书：**左栏写四句话（保证不出问题｜稳定可靠、界面友好｜测试要充分、不能漏测｜覆盖 12 个场景+回归 8 个核心场景，致命严重缺陷清零）；右栏逐条写病：不可验证・超出职责｜无度量口径｜绝对化｜有对象有范围有判据。底部："能不能被第三方核对？"

提问：第四句里，"致命与严重缺陷清零"这一条，属于目标还是属于准则？

预设回答：

- 答"属于目标"——它确实写在目标句里，但它的性质更像准则，下一页讲退出准则时你会看到同一条。

- 答"属于准则"——更准确的判断。它回答的是"什么时候可以停"，这是准则的职责。

- 答"都对"——这就滑向含糊了。规范的说法是：**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，但不能混着写。

- 追问收口：一句好的目标句，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/充分"；③ 案例题常给一段项目背景让你改写成目标句，改写公式是"对象＋范围＋数量＋判据"。

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）

课件给的定义很短：测试目标是**设计和执行测试的原因或目的**。请注意，它说的是"原因或目的"，不是"指标"。指标是实现目标的手段，目标是为什么要做这件事。

课件列了六类目的，我按"对内—对外—对变更"三组讲。

第一组，**对内的两类信息供给**：一是**向风险管理活动提供信息**——测试是风险管理的眼睛，你不测，风险部门只能靠猜；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，这些是管理层判断"能不能发"的依据。

第二组，**对外的两类评估**：三是**评估软件产品是否满足相关利益者的期望**——注意"相关利益者"不只用户，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，这一条本身就是测试的硬目标。

第三组，**对变更的两类评估**，这是最容易被忽略、也最实用的：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，都要验证没有副作用。

后两条合起来，就是我们常说的**回归测试的存在理由**。你会发现，真实项目里回归测试的工作量往往超过新功能测试，原因就在这儿：**改动是常态，副作用是常态**。

再强调一次这六条的性质：它们是**通用目标**，任何项目都适用。但通用目标不能直接执行，得翻译成本项目的具体目标——下一页就做这个翻译。

> **板书：**定义："测试目标＝设计与执行测试的原因或目的（不是指标）"。三组六条：对内（风险管理信息・质量信息）｜对外（满足利益者期望・合规）｜对变更（缺陷修正无副作用・变更无副作用）。右侧标："后两条＝回归测试的理由"。

提问：六条里，哪一条在你们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，但换个场景，比如给学校做学籍系统，合规就是硬指标。

- 答"缺陷修正没有副作用"——这一条忘得最狠。很多同学改完一个 bug 就提交，从不回头验证别的功能有没有被带坏。

- 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 六条目的分类记忆：对内（信息）、对外（评估）、对变更（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得带来负面效应；④ 别把"目标"和"指标"混用，指标是实现目标的手段。

过渡：通用目标有了，下一步是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）

这一页做一件事：把上一页的通用目标，翻译成本项目的具体目标。课件给了四条，我逐条讲它背后的判断。

第一条，**提供哪些质量风险信息**。这句翻译过来就是："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。比如一个银行核心系统的改造，最怕的是账务算错和批量跑批超时，那测试目标里就得写"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**。这半句是新增功能的验证，后半句是回归。请注意课件把它俩写在一起，这是在提醒你：**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"，而是"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**。功能性要求好理解；非功能性要求容易被忽略——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"，所以写目标时就要带上指标口径。

第四条，**在测试覆盖率、测试效率上的具体要求**。这一条最像"管理指标"，其实很硬：覆盖率回答"测到了几成"，效率回答"多长时间跑一轮"。没有这两个数，你就无法判断"再测一周有没有意义"。

四条合起来，构成一个项目的目标骨架：**风险指向、改动指向、要求指向、强度指向**。你写计划的时候，照着这四条各写一句，目标就不会空。

最后提醒：这四条不是抄一遍就完事，每一条都要落到**本项目的具体名词**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。

板书：四条竖排：① 提供哪些质量风险信息（怕什么测什么）② 新改动是否正确＋已有业务有无负面影响（改动↔影响成对）③ 功能性＋非功能性要求 ④ 覆盖率与效率的具体要求。右侧四个箭头写："风险指向・改动指向・要求指向・强度指向"。底部："每条都要落到本项目的具体名词和数字"。

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"翻译成"哪几类业务不容错"。

- 答"再问：月底结账那天有多少人同时用、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，就是你要写进风险的未知区域。

易错点 / 考点：① 四条内容的完整复述；② 高频考点："新改动业务与已有业务的关系"——必须成对考虑，这是回归测试目标的来源；③ 案例题常给"一句话需求"，让你落成具体目标，答题要带**业务名＋判据**；④ 别

只写功能性要求，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：这些目标是怎么定出来的？

【10.10】如何确定测试目标？（3 分钟）

这一页给方法，课件用四个追问加五个输入来回答"目标怎么定"。

先看四个追问，它们其实是一条推理链。

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"。任何一个需求变更，都不是孤立的：改一次利率计算，会影响计息、结息、报表、对账。影响圈画不全，回归范围就会漏。

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"：有些改动不改业务规则，但改数据量、改调用链、改并发，非功能特性就受影响了。

第三问，**测试覆盖率：新业务/功能？已有业务/功能呢？** 注意课件把新和旧并列问。新功能覆盖率通常好定，老功能覆盖率最容易被含糊过去——而恰恰是老功能，用户在用、出事最疼。

第四问，**如何最大程度提高测试效率？** 这一问看着像管理问题，其实是目标问题：效率目标会反过来约束你的方法选择——是用自动化每日回归，还是人工挑重点。

再看五个输入，它们回答了"凭什么做这些判断"：**产品质量要求**（来自合同、标准、行业规范）、**业务功能关系分析**（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）、以及由这四项共同支撑的目标陈述。

我要特别强调一点：这四个追问的顺序，就是**从影响出发倒推目标**。很多人定目标是"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。

给你一个可操作的模板：目标句＝"针对〔改动点〕，验证〔受影响业务/特性〕在〔条件〕下满足〔判据〕，覆盖〔范围/比例〕"。填空填不满，说明还有事没问清。

板书：左侧四追问（改动→影响哪些已有业务｜改动→影响哪些功能与非功能特性｜新旧业务的覆盖率各是多少｜如何提高效率）；右侧五输入（产品质量要求・业务功能关系分析・测试范围分析・测试策略与方法选择 → 目标陈述）。底部模板："针对〔改动点〕，验证〔受影响业务/特性〕在〔条件〕下满足〔判据〕，覆盖〔范围/比例〕"。

提问：一个需求只改了一行代码——把某个字段的长度从 32 改到 64。它需要回归测试吗？影响圈怎么画？

预设回答：

- 答"只改一行，不用回归"——最危险的回答。字段长度会影响数据库表结构、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，还可以再往下推：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"——这是最专业的答法，本质就是影响分析（有些团队会做静态的字段引用扫描）。

- 追问收口：所以"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 常考判断："改动小就不需要回归"——错；③ 案例题高频：给一个变更描述，让你列出受影响功能与非功能特性，答题时**业务与特性两个维度都要写**；④ 记住目标句模板，考试可直接套用。

【思政落点 1】：这一页最值得记的不是四个追问，而是它背后的态度：**目标必须可验证**。可验证意味着什么？意味着你不能用"基本满足""应该没问题"给自己留后路。工程上有一句很实在的话：**写得出的判据，才担得起的责任**。把"保证系统稳定"改成"连续 72 小时压测无错误、P95 响应 2 秒内"，看似给自己加了枷锁，实际是让协作方知道边界在哪、风险由谁承担。**严谨不是保守，是对结果负责的勇气**——这也是工程伦理里最基础的一条：不夸大、不含糊、不推诿。

过渡：目标定完了，还有一道门：什么条件下才允许开始测试。

【10.11】测试进入的准则（2 分钟）

这一页讲**测试进入的准则**，也就是"准入"。课件列了六条，我按"想清楚—有依据—有准备"三组讲。

第一组，"想清楚"：**清楚了解项目的整体计划框架**——你要知道自己在整个项目时间轴上的位置，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，等于拿一份可能错误的地图导航。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**——设计

文档告诉你系统怎么搭的，接口在哪、数据怎么流，这是设计集成和接口测试的依据。

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；**人员组织结构及其责任已确定**——谁测哪块、出了问题找谁，必须先定。

六条里，我认为最容易在真实项目里失守的是第一条和第五条。第一条失守的表现是：测试开始了，但没人知道这轮测试要在两周后支持哪个发布决策，于是测到一半被叫停。第五条失守的表现是：环境今天能通、明天数据库挂了，团队把一半时间花在修环境上。

为什么要立准入？给你一个反向的说法：**没有准入，测试就会在残破的版本上反复空转**。代码还没编译通过、主流程还跑不通，你就开始点，点出来的"缺陷"全是噪音，最后大家一起对测试结论失去信任。

准入还有一个隐性作用：它是**向上游要质量**的抓手。开发不给你一个能测的版本，你有权说"未达准入条件，测试未启动"——这句话比任何催促都有效，因为它把责任边界说清了。

板书：六条分三组：想清楚（整体计划框架・需求规格说明书已评审）｜有依据（技术与业务知识・技术设计文档）｜有准备（标准环境・足够资源・组织结构与责任）。底部写："没有准入 → 在残破版本上空转"；"准入是向上游要质量的抓手"。

提问：版本提交过来了，主流程跑不通，但项目进度很紧。你开测还是退回？

预设回答：

- 答"先测能测的部分，边测边等修复"——现实中很常见，但要有个前提：**书面记录"未达准入，本轮结论不完整"**，否则最后延期算在你头上。

- 答"直接退回，不达准入不测"——立场清楚，但要配套沟通：说清缺什么、什么时候能好、对进度的影响有多大。

- 答"随便点点，反正也是测"——最糟。你在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。

- 追问收口：所以准入的执行要点不是"卡"或"放"，而是**有条件放行＋明确记录**。

易错点 / 考点：① 六条准入条件的名称；② 高频判断："需求评审属于开发活动，与测试准入无关"——错，它是准入第一条硬条件；③ 案例题常考"版本不达准入怎么办"，标准答法是"记录＋有条件放行＋标注结论边界"；④ 结构记忆：想清楚／有依据／有准备。

【思政落点 2】：准入和退出准则，本质上是**规则意识**。工程里最难的从来不是定规则，而是**在压力下还守规**

则：进度催得紧，主流程还没通，你要不要"先测着"？守规则的人会说：可以测，但要写清"本轮未达准入、结论仅覆盖已通部分"。这不是推责任，是把**事实和责任同时说清楚**。一个团队如果习惯"先干起来再说"，短期看很快，长期一定在返工和扯皮里耗尽。**守规则的团队才走得远**，这句话在测试岗位上尤其成立。

过渡：准入讲完，10.1 就结束了。接下来进入本章最重的一节：测试需求分析。

【10.12】10.2 测试需求分析（1 分钟）

进入 10.2，测试需求分析。这一节是全章的地基，也是四节里唯一讲"方法"的。

四个小节我念一遍：**10.2.1 测试需求分析方法、10.2.2 测试需求分析技术、10.2.3 功能测试范围分析、10.2.4 非功能性的系统测试需求**。

请注意这四小节的递进：方法是"怎么想"，技术是"用什么工具想"，功能范围是"业务这一侧怎么划"，非功能需求是"质量这一侧怎么划"。前两个偏思维，后两个偏对象。

为什么这一节最重要？因为测试的**所有后续活动都从这里派生**：用例是从测试项派生的，环境是按测试项配的，进度是按测试项估的，缺陷统计也是按测试项归类的。分析错了，后面每一步都在错误的地基上盖楼。

有个行业里的经验说法：测试项目里返工最贵的三件事——需求理解错、范围划错、环境配错。这三件里有两件直接出在这一节。

带着一个问题听：**需求文档给你了，你怎么从一份"给人看"的文档，得出"给测试用"的清单？**

板书：中间写"10.2 测试需求分析"；下方四小节：方法（怎么想）→ 技术（用什么想）→ 功能范围（业务侧）→ 非功能需求（质量侧）。右侧一句："所有后续活动都从这里派生"。

提问：为什么说"测试需求分析"这个名字里的"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，只是换了个说法"——这就危险了。直接把需求文档当测试清单用，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 高频概念辨析："用户需求 / 软件需求 / 测试需求"三者关系——逐层派生；③ 判断题陷阱："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）

这一页只有一个循环，四步：**测试分析 → 测试设计 → 执行 → 评估**，然后回到分析。信息量不大，但含义很重。

第一步，**测试分析**：搞清楚测什么、测到什么程度、优先级怎么排。产出是测试项清单、范围界定、风险与优先级。

第二步，**测试设计**：把测试项变成具体用例——输入什么、怎么操作、预期什么。这一步第 11 章会专门讲。

第三步，**执行**：跑用例，记录结果，提缺陷。

第四步，**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"？因为它是唯一一个"错了会放大"的环节。分析漏了一个模块，设计就少一批用例，执行就少一批数据，评估就会得出"质量良好"的错误结论——而且**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：测试分析是地图，设计是路线，执行是开车，评估是看看到没到。地图上少画了一条路，你车开得再稳、油加得再满，也到不了那个地方。

所以这个循环里，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。

所以我的建议是给分析留"看得见的时间"：在计划里把测试需求分析写成一条独立任务、给独立工期、指定负责人。凡是没被写进计划的工作，最后都会被压缩成零。

> **板书：**画一个环：分析 → 设计 → 执行 → 评估 →（回到）分析。环心写"测试分析"；每步下方标产出：测

试项清单｜用例｜结果与缺陷｜覆盖率与趋势。右侧一句："分析错了会放大，且无人发现"。

提问：这个循环里，如果"评估"发现覆盖率不足，应该回到哪一步？

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，但如果缺的是"测试项"这个层级，补用例也只是在错的范围里补。

- 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"的方法很简单：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 常考简答："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 判断题陷阱："测试分析只在项目初期做一次"——错，评估结果会驱动它反复进行。

过渡：既然是基础，那它的输入是什么、输出是什么？下一页把两头摆出来。

【10.14】测试分析的输入与输出（2 分钟）

这一页是一张输入输出图，课件没有铺开文字，我按课堂常用口径把它讲实。

输入这一侧，主要有五类。第一类是**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。第二类是**设计类文档**：架构图、接口说明、数据模型——测试接口和集成时离不开它。第三类是**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，很多场景根本构造不出来。第四类是**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"最有效的输入。第五类是**约束**：标准规范、合规要求、合同条款、环境与工具限制。

输出这一侧，是下一页要展开的六项：测试目标、测试范围、测试项与子项、测试风险、优先级、测试策略。

这张图真正的教学点在于：**输入的质量决定输出的质量**。很多团队的测试分析做不好，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，而是**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。

再给你一个实用判断：如果你发现自己只能靠"猜"来定测试项，那一定是输入不够，而不是你水平不够。

> **板书：**左栏输入五类：需求文档｜设计文档｜业务规则与数据｜历史缺陷与事故｜约束（标准・合规・合同・环境）。右栏输出六项（下一页展开）。中间画大箭头，箭头上写："输入残缺 → 输出失真"。底部："先清点输入，再动手分析"。

提问：如果需求文档写得很粗，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，但只能按那一页纸做"——会漏得很厉害，而且漏的部分没人知道。

- 答"做，同时把'需求不清'本身列为最大风险，并去找业务方、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**。

- 追问收口：所以测试分析的一项产出其实是"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 高频判断："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"需求不清晰时怎么办"，答"补输入＋记录缺口＋升级风险"三步。

过渡：输入摆完了，我们把输出这一侧展开——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）

这一页列出测试分析的六项输出，它们是本章承上启下的关键，因为它几乎就是测试计划的原料清单。

第一，**测试目标**——上一节已经讲过，落到本项目的那几句可验证的话。

第二，**测试范围**——测什么、不测什么，边界写清。

第三，**测试项和测试子项**——这是分析的核心产物。测试项是"要测的一个可独立验证的对象"，子项是它的细化。举例：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"若干子项。

第四，**测试风险**——哪些地方不确定、哪些地方复杂、哪些地方历史事故多。

第五，**测试优先级**——决定先测谁、测多深、资源往哪倾斜。

第六，**测试策略**——针对风险和约束，用什么手段、什么层次、什么组合去测。

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

这里要提醒一个常见错误：很多同学把"测试项"直接写成"功能菜单名"。菜单名是**界面组织方式**，不是测试对象。测试项应该按业务能力和质量特性来切，比如"跨行转账的限额校验"是一个测试项，而"转账页面"只是它的一部分载体。

判断你切得对不对，有个简单办法：**每个测试项都应该能独立写出通过/失败判据**。写不出判据的，说明切得太粗或者太大。

再补一句：测试项清单是要被下游反复引用的——用例挂在它下面，缺陷归到它下面，覆盖率按它统计。所以它的编号和命名要从一开始就规范，别等到几百条时再返工。

> **板书：**六项竖排：目标 → 范围 → 测试项与子项 → 风险 → 优先级 → 策略。右侧注："前二定边界・中二定内容与顺序・后二定打法"。底部写："每个测试项必须能独立写出通过/失败判据"。

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。

- 追问收口：粒度标准不是"大小"，是"**能不能独立判定**"。

易错点 / 考点：① 六项输出的名称与顺序；② 高频考点：测试项的划分粒度标准——可独立判定、可估算、可排优先级；③ 别把"测试项"与"测试用例"混为一谈：项是"测什么"，用例是"怎么测"；④ 案例题常给一段业务描述，让你列测试项，注意按业务能力切，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分析过程。

【10.16】测试需求分析的过程（2 分钟）

这一页给的是分析过程的五个步骤，我按顺序走一遍，每一步都告诉你"输入什么、产什么"。

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**。为什么要从背景开始？因为同一个功能在不同背景下要求完全不同：一个内部报销系统的"审批"，和一个信贷系统的"审批"，风险等级差着量级。先懂背景，才知道深浅。

第二步，**了解用户是谁、用户所关心的是什么**，产出**用户需求**。注意"用户是谁"是复数：一线操作员关心快不快、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**。这就是最经典的分解：业务 → 功能 → 子功能 → 功能点。分解到什么程度停？停到能写判据为止。

第四步，**基于系统的特点与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与风险分析**，产出**测试项优先级**。价值和风险是两个不同维度：价值高的要早测，风险高的要深测，两者都高的就是第一批。

五步的顺序不能颠倒，因为它是一条**从抽象到具体、从功能到质量、从内容到排序**的收敛链。你把它倒过来做，比如先排优先级再分解，你会发现没有可排的对象。

还有一个实操提醒：第三步和第四步是**两条平行支线**，不是先后关系。很多人做分析只走功能那条线，走完就以为分析结束了——非功能那条线的入口，是"系统特点＋质量属性"，不是需求文档。

> **板书：**五步竖排带箭头，每步右侧写产出：① 背景・价值・业务问题 → 测试目标；② 用户是谁・关心什么 → 用户需求；③ 逐层分解功能特性 → 待测功能项；④ 系统特点・质量属性 → 待测非功能项；⑤ 价值分析・风险分析 → 优先级。右侧标："③④ 是两条平行支线"。

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，但没说到点子上。

- 答"因为它们的入口不同：功能从需求文档来，非功能从系统特点和质量属性来"——这是最准确的回答。

- 答"先做功能，做完了顺手想想非功能"——现实中常见，后果是非功能项永远是"顺手想想"，最后一条都没测。

- 追问收口：平行意味着**两条线都要有人负责、都要出现在清单里**，谁都不能是附属品。

> **易错点 / 考点：** ① 五步顺序与各步产出，简答高频；② 判断陷阱："分析过程就是功能分解"——漏了非功能那条线；③ 记牢第二步的"用户是复数"，案例题常考"不同干系人关心不同需求"；④ 分解的停止条件：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"在分析视角下再看一遍。

【10.17】测试目标（2 分钟）

这一页又把测试目标拿出来讲，但视角变了——上一节讲"目标是什么"，这一页讲"目标里必须装哪四类内容"。课件给了四条。

第一，**满足业务需求，包括新改动的业务、已有业务**。注意又是"新＋旧"的组合。这一条在分析阶段的含义是：你的目标清单里，必须同时有"新功能验证"和"老功能回归"两类条目，不能只列新的。

第二，**满足质量要求，包括使用质量、功能性要求和非功能性要求**。这里"使用质量"是个专门概念，指的是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，而是"用起来顶不顶事"。

第三，**测试覆盖率要求**。覆盖率是本章反复出现的关键词，小结里还会说它是"起点也是终点"。作为目标的一项，它要写清口径：是按需求覆盖、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，**测试效率、周期、成本等目标**。这一条最反直觉：测试目标里居然要写成本。其实很合理——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，你会看到一个完整的测试目标应该是**四维的**：业务维、质量维、覆盖维、效率维。少一维，目标就会在执行中被某一方拉扯——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。

一个实用的检查法：写完目标，用四个问题各问一次——"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。

> **板书：** 四维写成一个十字：上"业务（新＋旧）"｜下"质量（使用质量・功能性・非功能性）"｜左"覆盖率（口径要写清）"｜右"效率・周期・成本"。底部："四问检查法：验什么・验多深・覆盖多少・花多大代

价"。

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，但更准确的说法是：**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，不写成本的计划在评审时最容易被砍，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"必须能对上，对不上就是目标定错了。

- 追问收口：所以一份好的目标，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 判断陷阱："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。范围这一页信息密度很高，是分析的主干。

【10.18】测试范围分析（2 分钟）

这一页讲范围分析，一句话概括课件的意思：**界定影响范围的边界，说清哪些要测、哪些不需要测**。

先看输入，课件给了三样：**项目目标和范围文档（计划）**——这是上位约束，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。

再看输出，四样：**测试需求 Spec**——正式的分析文档；**测试项列表（按优先级）**——核心产物；**测试数据**——注意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"；**测试边界界定**与**测试项与需求映射矩阵**。

映射矩阵这个东西值得单独说。它是一张表：行是需求条目，列是测试项，交叉处打勾。它的作用有三个：一是证明**每条需求都有人管**；二是发现**没有测试项覆盖的需求**（漏测）；三是发现**没有需求支撑的测试项**（多余或过度设计）。

关于"哪些不需要测"，我要多说两句。范围分析里最难写的不是"要测什么"，而是"**为什么不测**"。原因可能是：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。每一种都要写下来，因为出事的时候，这一条就是你唯一的解释。

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

板书：左侧输入三项（项目目标与范围文档｜功能特性列表｜业务流程・规则・业务数据）；中间"分析"；右侧输出四项（测试需求 Spec｜测试项列表按优先级｜测试数据｜边界界定＋需求映射矩阵）。底部两行："不测什么＋为什么不测"；"映射矩阵的三大作用：证明覆盖・发现漏测・发现多余"。

提问：范围里写"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"和"谁来保证"。

- 答"不够，要写清理由：由支付厂商提供测试报告，我方仅做联调验证"——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，单方面决定的范围没人认账。

- 追问收口：所以"不测"这一栏的标准写法是：**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"，答题必须给理由；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：范围往外看是边界，往里看是内容。分析到底分析哪些内容？

【10.19】分析的内容：领域、元素（2 分钟）

这一页问的是"分析的对象到底是什么"。课件给了几组关键词：**干系人、约束条件、愿景与目标商业价值、正确性/完备性/适应性、目标场景决策卡**。我按"人—约束—价值—质量—场景"五层讲。

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。每一类干系人都有自己的"关心点"，分析就是把这些关心点收集齐。

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。

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？这一层看着最虚，其实最实用：它能帮你判断"哪个功能是命门，哪个是锦上添花"。

正确性、完备性、适应性：这是三种不同的质量追问。正确性是"做得对不对"；完备性是"该有的有没有"；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，很多人只问第一种。

目标场景决策卡：这是把"场景"落成卡片的做法——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。

五层合起来，其实就是回答一句话：**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。**

提醒一句：这一页的所有内容都不是"文档里抄来的"，是**问出来的、读出来的、推出来的**。分析能力的一半是提问能力。

> **板书：**五层竖排：干系人（谁在乎）→ 约束条件（设计空间）→ 愿景・目标・商业价值（为什么存在）→ 正确性・完备性・适应性（三种质量追问）→ 目标场景决策卡（触发・角色・期望・异常）。底部："分析能力的一半是提问能力"。

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，这就是下一页要讲的启发式策略的雏形。

- 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"。

- 追问收口：所以检查单要覆盖**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 高频辨析：正确性 / 完备性 / 适应性三者的区别；③ 判断陷阱："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 3】：这一页的关键词是"干系人"。工程里很多事故不是技术不行，是**该问的人没问**：需求方以为"这么简单的规则开发肯定懂"，开发以为"需求没写就是不用做"，测试以为"文档没提就不在范围"。三方都没错，合起来就是一场事故。所以"问清楚"不是能力问题，是**责任意识**：宁可多问一句被人嫌烦，也不要少问一句让用户在线上替你发现。**把模糊问成清楚，是工程人最值钱的本事**，也是对他人、对结果负责的最直接方式。

过渡：说到"检查单"，下一页就是一个现成的框架——启发式测试策略模型。

【10.20】启发式测试策略（2 分钟）

这一页介绍 **HTSM，Heuristic Test Strategy Model，启发式测试策略模型**。它出自测试圈很有影响的一套方法论，课件的核心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 → 可接受的质量**。

先解释"启发式"这三个字。启发式不是公式，它是**经验性的思考线索**：不保证给你唯一正确答案，但能保证你不漏掉重要的思考维度。就像老中医的问诊口诀，不保证诊断百分百准，但不会让你忘了问某一项。

四个要素怎么用？**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、可维护等等。**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**——注意这个说法很讲究：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"。这句话把测试从"追求完美"拉回到"管理期望"，非常务实。

我为什么要专门讲这个模型？因为它能治一种常见的病：**只从功能维度想测试**。你把这四个维度摆在桌上，任何一个维度空着，你就知道自己漏了。

课堂上你可以这样用它：拿到一个新系统，四张纸各写一列，项目环境写约束，产品元素列结构，质量标准排序，测试技术选型——写完你就有一份初步策略了。

板书：中间写"HTSM 启发式测试策略模型"；四角写：项目环境（人・时间・工具・约束）｜产品元素（功能・数据・接口・结构・平台）｜质量标准（正确・可靠・安全・易用・可维护）｜测试技术（黑盒・白盒・探索・自动化）；中心写"可接受的质量（不是完美质量）"。

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"可接受的质量是要和干系人对齐的"——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。

- 追问收口：所以"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 高频概念："启发式"不是算法，是思考线索，判断陷阱"启发式模型能给出唯一正确策略"——错；③ "可接受的质量"这个提法常考名词解释；④ 与第 4 章"上下文驱动"流派可以对照记忆。

过渡：HTSM 是思考框架，接下来把视角拉高，看分析要顾到多大的一张图。

【10.21】测试分析的全局观（2 分钟）

这一页讲"全局观"，课件用五行关键词铺开：业务/用户/产品/系统，用例驱动/数据驱动/结构化/OO 方法，用例图/数据流程图/活动图，头脑风暴/帕累托图/鱼骨图/亲和图/直方图，接口/功能/数据/质量/约束。

这五行其实是五个维度，我给你归类。

第一行，**视角**：业务、用户、产品、系统。同一个东西，四个视角看到四套需求——业务看价值，用户看体验，产品看功能，系统看结构。分析时要能切换视角，只用一个视角必然片面。

第二行，**方法族**：用例驱动、数据驱动、结构化、面向对象。它们不是互斥的，是四把不同的刀：用例驱动切场景，数据驱动切输入域，结构化切流程，OO 切对象与关系。

第三行，**图形工具**：用例图看角色与功能边界，数据流程图看数据怎么流，活动图看流程与并发。

第四行，**思维工具**：头脑风暴发散，帕累托图找那 20% 的关键项，鱼骨图找原因，亲和图归类，直方图看分布。这些是质量管理的经典工具，用来处理"信息太多理不清"的场面。

第五行，**分析对象**：接口、功能、数据、质量、约束——这五个就是最终要落到清单上的东西。

一句话总结这一页：**全局观不是知道更多名词，而是知道在什么场合用哪把刀**。名词背得再熟，遇到一个具

体系统还是只会按菜单点，那等于没有全局观。

补一个练习方法：拿你正在做的课程设计，把这两个字的五行各填一行内容。填不满的那一行，就是你的分析还没覆盖到的维度。

板书：五行横排：视角（业务・用户・产品・系统）｜方法族（用例驱动・数据驱动・结构化・OO）｜图形工具（用例图・数据流图・活动图）｜思维工具（头脑风暴・帕累托・鱼骨・亲和・直方）｜分析对象（接口・功能・数据・质量・约束）。右侧："全局观＝知道什么场合用哪把刀"。

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，鱼骨找原因的分类"——准确。帕累托按贡献度排序（80/20），鱼骨按人机料法环分类追因。

- 答"帕累托用来排优先级，鱼骨用来分析缺陷根因"——这就是它们在测试分析里的典型用法。

- 追问收口：所以工具的选择标准是"**你要回答什么问题**"，不是"哪个图好看"。

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，方法是组合使用的；④ 与第 3 章的方法体系可以对照。

过渡：全局观之后是两个最基本的手上功夫：整体看还是拆开看。

【10.22】分析思路：整体 vs. 分解（2 分钟）

这一页讲两种最基本的分析思路：**整体**和**分解**。这两条思路必须成对使用，缺一条就会出问题。

先看**整体**这一侧，课件列了六样：业务流程、事件流、数据流、状态与状态转换、事件触发与条件约束、数据/环境/场景。这六样的共同点是——它们描述的是"系统作为一个整体怎么运转"。为什么必须整体看？因为很多缺陷只存在于**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。这类缺陷，你把它拆成零件是永远看不见的。

再看**分解**这一侧，课件列了三样：质量特性的分解、功能→子功能→孙功能、系统→模块/组件→接口。分解

的价值是把"测不完的大对象"变成"测得了的小对象"。

两边的差别，可以打个比方：**整体看是看一部电影的剧情，分解看是逐个镜头检查**。只看剧情，你会漏掉穿帮镜头；只看镜头，你会漏掉剧情断裂。

实操上，我建议的顺序是：**先整体画一张图（流程或状态），再按图分解出测试项**。这样分解出来的项天然带着"位置感"——你知道每一项在流程的哪一步、哪个状态转换上。

反过来做，先按菜单列功能再拼整体，最常见的后果是：**接口和状态转换成了盲区**。因为菜单上没有"接口"这一项，也没有"状态机"这一项。

还有一个实操技巧：整体那张图不必画得漂亮，能标出分支、异常、状态转换就够了。它的作用是给分解提供坐标，不是给领导汇报。

> **板书：**左右两栏。左（整体）：业务流程・事件流・数据流｜状态与状态转换｜事件触发与条件约束｜数据・环境・场景。右（分解）：质量特性分解｜功能→子功能→孙功能｜系统→模块/组件→接口。底部："先整体画图，再按图分解"；"只分解不整体 → 接口与状态成盲区"。

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，按功能切就会切丢。

- 答"因为接口文档经常没有"——补得好，这是输入缺失的问题，也和前面讲的"设计文档是输入"呼应。

- 追问收口：所以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的内容；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 案例题常考"为什么接口测试容易漏"，答"分解粒度与对象边界"；④ 记住操作顺序：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）

这一页给五条"下手"的途径，都是很实操的思维动作。

第一条，**简单化、抽象化**。把复杂系统剥掉细节，先看骨架：这个系统本质上在做什么？一个信贷系统剥到最后就是"收资料—评估—放款—还款—催收"。抽象之后你才看得清主干，不会被几百个界面晃花眼。

第二条，**结构化、层次化**。把对象按层次摆好：业务层、功能层、模块层、接口层。层次化最大的好处是**可以分层测试、分层追责**——出了问题知道是哪一层的事。

第三条，**可视化、可观察**。把想不清楚的东西画出来：流程图、状态图、表格。人的短期记忆装不下复杂关系，画出来才能讨论。这一条也提示一个测试原则：**不可观察的东西难以测试**——如果一个行为没有日志、没有界面反馈、没有接口返回，你怎么验证它发生了？

第四条，**类比、隐喻**。遇到陌生系统，找熟悉系统类比："这个像银行的核心账务"、"这个像电商的订单中心"。类比能快速建立直觉，但要注意**类比的边界**——哪里像、哪里不像，不像的地方往往是缺陷高发区。

第五条，**收集足够的信息或数据**。分析不是空想，要有材料：需求、设计、数据、历史缺陷、用户反馈。信息不足时的分析，本质是猜。

五条合起来是一个意思：**分析不是天赋，是一套可以练的动作**。你按这五条走一遍，至少不会漏得太离谱。

我特别推荐第三条，因为它同时解决两个问题：自己想不清，和跟别人说不清。

> **板书：**五条竖排：① 简单化・抽象化（剥到骨架）② 结构化・层次化（业务→功能→模块→接口）③ 可视化・可观察（画出来＋留痕迹）④ 类比・隐喻（找熟悉系统，注意边界）⑤ 收集信息与数据（不猜）。底部："不可观察的，就难以测试"。

> **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"所以要在需求阶段就要求加日志"——很专业。可测试性（testability）本身就是需求的一部分，很多团队在测试阶段才发现"没法验证"，只能靠猜。

- 追问收口：所以"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点。

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，类比只提供起点，边界处必须实证；④ 案例题常考"系统无法验证某行为时怎么办"，答"要求补日志/埋点，或改用间接验证"。

过渡： 下手有途径了，接下来看几种常用的分析模型，它们各有各的地盘。

【10.24】常见分析模型比较（3 分钟）

这一页是一张对比表，四种分析模型，每一种都标了建模标准、优点和短板。这张表考试爱考，因为它能出选择题，也能出"选型理由"的简答题。我一个一个讲。

跨职能流程图。它的建模标准是**商业建模标准**——也就是说它首先面向业务人员，不是面向程序员的。优点是**复用性强、用户最容易接受**。为什么用户容易接受？因为它的画法就是"泳道"：一条泳道一个角色，业务人员一眼就能看出"这步该我干"。它的短板是**并行、异步支持差**——泳道图天生偏向"顺序往下走"，遇到并行分支、异步消息就画不清楚。

活动图。UML 家族，优点是**语义最丰富**，能表达分支、合并、并发、泳道、对象流；而且它**强调行为流、强调活动内容**。适合干什么？适合把一段业务流程拆到"每一步做什么"的粒度。短板是容易画得过大，一张图几十个节点，谁也看不懂。

时序图。也是 UML，**强调行为流、强调协作，技术类系统更常用**。为什么技术类系统爱用它？因为技术系统里"谁调谁"是关键——一笔交易经过网关、鉴权、账务、风控、消息队列，时序图一眼看清调用顺序和消息内容。业务类系统用得少，因为业务人员看不懂对象之间的消息。

数据流图。它的建模标准是 **IDEF**，特点是**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"，而是"数据从哪来到哪去、经过哪些加工、口径怎么变"。

四张表放一起，选型逻辑就出来了：**面向业务沟通选泳道图，要把流程拆细选活动图，要看调用协作选时序图，要盯数据加工选数据流图**。

再补一句考试技巧：这类题的答案永远落在"模型的表达能力与你的分析目的匹配"，不要背"哪个最好"——没有最好的模型，只有最合适的模型。

板书： 四行表格：跨职能流程图｜商业建模标准｜复用性强・用户最易接受｜并行异步支持差；活动图｜UML｜语义最丰富・强调行为流与活动内容｜易画过大；时序图｜UML｜强调行为流与协作｜技术类系统

> 常用；数据流图｜IDEF｜强调数据流｜未表示谁执行・计费类最适用。底部："业务沟通选泳道，拆细流程选活动，看调用选时序，盯数据选 DFD"。

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，因为计费的本质是数据加工和口径"——正确。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"——时序图能看清调用，但看不清"一笔话单经过五次加工后，金额字段被谁改成了什么"，这恰恰是计费的核心。

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

- 追问收口：所以选型题的正确姿势是"**主模型＋辅助模型**"，并说明各自负责回答什么问题。

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 四个短板要记住：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，下一页看工具。

【10.25】分析工具（2 分钟）

这一页很朴素，列了一串工具：业务流程图、UML 图（用例图、活动图等）、建模工具、思维导图、Excel……

我先说结论：**工具不重要，重要的是可追溯**。工具选哪个，取决于团队已有的习惯；但如果分析结果不能被追溯——测试项追不到需求、缺陷追不到测试项——那用什么工具都白搭。

我把工具按用途分三类。

第一类，画流程与结构的：业务流程图、UML 图、专门的建模工具（如各类 UML 建模软件）。这类工具解决"复杂关系可视化"。

第二类，梳理思路的：思维导图。思维导图在测试分析里非常好用，因为它天然是"分解"的形状：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 里写三级标题快得多，而且能一眼看出哪一

枝长得太粗（说明那块复杂，要重点测）。

第三类，管数据的：Excel。别小看它。测试项清单、需求映射矩阵、优先级表、风险清单、WBS 估算表——工业界绝大多数测试资产最初都长在 Excel 里。为什么？因为**能排序、能筛选、能透视、能共享**。

还有一类隐形的工具：**缺陷库和历史数据**。它们不是画图工具，但它们是最好的分析输入——上一版哪里出问题最多，这一版就该重点测哪里。

给你一个选工具的实用标准：**这个工具产出的东西，能不能直接变成下游要用的表格**。画得漂亮但不能导出成清单的图，往往只能当汇报材料。

再补一条：工具要能共享。测试资产是团队资产，一个人电脑里的 Excel 等于不存在；放在共享位置、有版本记录，才算数。

板书：三类工具：画图（业务流程图・UML 图・建模工具）｜理思路（思维导图：中心→一层层长出功能项）｜管数据（Excel：测试项清单・映射矩阵・优先级・风险・WBS）。右侧一句："工具不重要，可追溯才重要"。底部："选型标准：产出能不能直接变成下游要用的表"。

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，但不是主因。

- 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。

- 追问收口：所以"会用 Excel 建一张好的测试项表"，是测试岗的一项基本手艺，别嫌它土。

易错点 / 考点：① 工具的三类用途划分；② 高频判断："分析必须使用专业建模工具"——错，工具从属于目的；③ 记住"可追溯"是分析质量的核心标准：需求↔测试项↔用例↔缺陷四级追溯；④ 案例题常问"如何保证可追溯"，答映射矩阵＋编号体系。

过渡：讲了三页方法，来看一个具体的分析示例。

【10.26】示例（2 分钟）

这一页课件放了一个分析示例，没有配文字说明，我按它的路子带你走一遍，你把动作学会就行。

分析的**第一个动作**：把一段业务描述读成"角色＋动作＋对象＋规则"。比如一段"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"，读出来就是四个角色、五个动作、两条规则。

第二个动作：把动作串成流程，标出分支和异常。哪些步骤有分支？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集的地方**，因为正常路径开发会反复自测，异常路径常常没人走。

第三个动作：把流程里的每一步变成候选测试项，并写判据。比如"资料完整性校验"这一项，判据是：必填项缺失时给出明确提示且不允许提交；附件格式不符时拒绝并说明原因。

第四个动作：把测试项归类到功能项与非功能项两条线上。上面那个例子里，除了功能项，还应该派生出"并发提交同一申请会不会重复建档"（数据一致性）、"复审超时的时限配置是否可调"（可配置性）、"客户资料是否脱敏展示"（安全性）。

四个动作做完，一段几百字的业务描述就变成了十几条测试项。这就是分析的全部手艺：**读—串—拆—归**。

我想强调第四个动作，因为它是学生作业里最缺的一环：大家很容易把业务描述拆成一堆功能点就收工，而把非功能那条线忘掉。记住：**每读完一段业务，都要追问一次"这里有没有质量属性上的要求"**。

> **板书：**四步动作：读（角色＋动作＋对象＋规则）→ 串（流程＋分支＋异常）→ 拆（每步变测试项＋写判据）→ 归（功能线／非功能线）。底部："异常分支＝缺陷密集区"；"每读完一段业务，追问一次质量属性"。

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了缺陷分布。

- 答"因为异常处理的需求常常没写清"——也很对：需求对异常的表述往往只有一句"系统应妥善处理"，"妥善"是什么，没人定义。

- 追问收口：所以分析时要专门列一张"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："测试项只从正常业务流程产生"——错；④ 案例题常给一段业务描述要求列测试项，注意功能

与非功能都要有。

过渡：测试项有了，接下来是最考验判断力的一步——排优先级。

【10.27】明确测试项的优先级（2 分钟）

这一页讲优先级怎么定，课件给了五个依据。

第一，**产品质量风险的识别**。风险高的先测、深测。风险来自哪？复杂度、变更量、历史缺陷、技术新颖度、依赖外部系统。

第二，**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——**这个功能坏了，业务会不会停**。

第三，**风险分析-风险矩阵**。把"可能性"和"影响"两个维度做成矩阵，落在"高可能×高影响"那一格的，就是最高优先级。下一页会展开。

第四，**经验与历史数据**。上一版哪里问题多、生产上哪里报障多，这一版就往那些地方加权。这是最便宜也最有效的一招，因为它不需要额外分析成本。

第五，**80/20 原则**。百分之二十的功能承载百分之八十的风险或使用量。找出那 20%，先把它们测透，边际收益最高。

五个依据不是并列关系，我建议的用法是：**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**。粗排定大方向，加权修正偏差，矩阵检查有没有明显误判。

一个实操建议：优先级**不要超过三级**。P0/P1/P2 够了，分五级最后没人分得清。而且每一级要有明确定义，比如 P0＝不通过就不能发布，P1＝影响主流程但有绕行方案，P2＝影响体验不影响业务。

最后提醒：优先级不是给测试组自己看的，它是**和业务方对齐期望的工具**。时间不够时你砍掉 P2，得让业务方知道并同意。

补一个操作细节：优先级要写进测试项表，不能留在脑子里。写下来才能被评审、被质疑、被调整，也才能在时间被砍时说得清砍了谁。

板书：五依据竖排：质量风险识别｜功能重要性排序｜风险矩阵（可能性×影响）｜经验与历史数据｜80/20 原则。右侧写用法："粗排（重要性＋80/20）→ 加权（风险＋历史）→ 校验（矩阵）"。底部："优

> 先级不超过三级，每级要有定义"。

> **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"——容易和业务方对立。

- 答"请他回答一个具体问题：上线当天如果只能保证三件事不出错，是哪三件？"——这是最有效的办法：**用资源约束逼出真实排序**。

- 追问收口：排序的本质是取舍，取舍需要一个约束条件才能谈；给他一个"只能选三个"的场景，答案自然出来。

> **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："优先级一旦确定不再调整"——错，随风险变化滚动调整；④ 案例题常考"如何说服业务方接受优先级"，答"用资源约束场景化提问"。

> **过渡：**接着上一页的示例，看优先级排完之后是什么样子。

【10.28】示例 – 续（2 分钟）

这一页是上一页示例的延续，课件用图把分析结果往下推了一步。我讲它示范的动作：**把测试项排成一条有先后的队列**。

排队的第一个原则：**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——从入口到出口走一遍，确认这条路是通的。为什么？因为主干不通，后面所有细节测试都是在流沙上做实验，测出来的缺陷会被主干缺陷掩盖。

第二个原则：**高风险模块早测**。早测的价值在于"早发现早改"——同一个缺陷，在设计阶段改和在上线前一天改，代价差着数量级。这是缺陷成本随阶段上升的基本规律。

第三个原则：**依赖关系决定顺序**。A 功能依赖 B 接口，那 B 必须先测。排优先级不能只看重要性，还要看**拓扑顺序**：底层的、被依赖的先测。

第四个原则：**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。

排完你会得到一张表：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分析结果。

提醒一个常被忽略的细节：**同一个测试项，在不同阶段可能有不同优先级**。比如"批量导入"在功能测试阶段是 P1，在性能测试阶段可能升到 P0，因为导入是最容易压垮系统的操作。优先级是带场景的，不是绝对的。

还有一个实用做法：把"依赖"单列一栏，排期时先按依赖画拓扑，再按优先级微调。否则你排出来的顺序，执行时天天被阻塞，优先级排得再准也跑不动。

板书：四原则：先主干后端到端→再钻细节｜高风险早测（缺陷成本随阶段上升）｜依赖决定顺序（拓扑）｜为回归留位置（保护在用的东西）。底部表格列头："测试项｜优先级｜依赖｜工作量｜责任人"；再写一句："优先级带场景，不同阶段可不同"。

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"——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"因为用户往往在特定时点集中导入（比如月初），形成峰值"——这就把业务特征也说清了，很有价值。

- 追问收口：所以优先级的判断要区分"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、阶段都要考虑；④ 案例题常考"给一组测试项排序并说明理由"，理由要落到四条原则上。

过渡：把前面的结果汇总起来，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）

这一页是分析结果的汇总形态。课件没有铺文字，但产出物的样子是很固定的，我给你讲清"一份合格的分析结果长什么样"。

它通常包含四张表。

第一张，测试项清单：编号、测试项名称、所属功能/特性、来源需求编号、优先级、备注。编号体系很关键，

建议用"业务域-功能-序号"的结构，比如 `TRF-001`、`ACC-003`，这样一眼能看出归属。

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。它的三个用途前面讲过——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：特性类别（性能/安全/兼容/可靠/易用）、指标口径、测试方法、依赖条件。

四张表之间靠**编号**打通。这是很多同学作业里最弱的地方：表与表之间没有编号关联，看起来齐全，实际无法追溯。

再补一个容易被忽视的字段：**"不测项"清单**。哪些需求本期不测、为什么、谁确认的。这张表在评审时最能体现你的专业度，因为它证明你想过边界。

判断标准很简单：**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，就是合格的分析结果；不能，说明还缺东西。

再补一个字段：每条测试项的判据。哪怕只写一句"通过条件"，也比只写名称强得多，因为它逼你把"测到什么算过"想清楚。判据缺失的测试项，到了执行阶段一定会变成"看着差不多就行"。

> **板书：**四张表：① 测试项清单（编号/名称/所属/来源需求/优先级）② 需求-测试项映射矩阵 ③ 风险清单（可能性・影响・严重性・措施・责任人）④ 非功能测试项表（类别・指标・方法・依赖）。加一张"不测项清单"。底部："表与表靠编号打通"；"交给外人能否据此排计划"。

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"因为评审时最容易出分歧的就是边界：有人以为测了，有人以为没测"——最本质。边界不清是项目扯皮的第一来源。

- 追问收口：所以边界不是写给自己看的，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（正向覆盖、反向查漏）；③ 判断陷阱："分析结果就是一份测试项清单"——错，还含映射、风险、非功能与不测项；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。下一页专门讲：造成分析不准确的风险有哪些。

【10.30】造成测试分析不准确的风险（3 分钟）

这一页列了一串会造成分析不准确的原因，我按"范围—需求—复杂度—可测性—数据—代码—环境"七组归一下，最后讲课件右侧那条更重要的线索。

范围类：项目范围不清楚、项目之间耦合性强。耦合是最麻烦的——你以为改的是自己家，其实动了别人家的地基。

需求类：产品业务不够清晰或没说明清楚；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**，于是分析变成一次次重做。

复杂度类：系统复杂性导致分解比较困难。分解不动的时候，人容易退回"按界面点"，这是能力不足时的本能反应。

可测性类：系统特性的不可测试性。没有日志、没有接口、状态不可观测，你的测试项只能写成"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。数据的组合爆炸，是分析漏项的重灾区。

代码类：代码质量差、代码结构不好。这会直接影响测试的成本与有效性——同样的用例，在烂代码上跑出来的现象可能无法解释。

环境类：环境配置复杂、用户配置项很多。配置项多意味着组合多，而配置组合恰恰是最难穷尽的部分。

现在看课件右侧那条线索，它比左边七条更有价值：它把系统分成三段——**新开发、继承特性、平台/维护特性**，并标出"问题高发区"与"测试重点"。意思是：**不同来源的代码，缺陷分布规律不同**。新开发的问题集中在"没做全、没做对"；继承特性的问题集中在"改坏了、兼容性破了"；平台与维护特性的问题集中在"配置、部署、环境适配"。

这条线索的用法是：**按代码来源调整测试重点**，而不是平均用力。这是把风险意识落到具体动作上的典型例子。

板书：左侧七组：范围（不清・耦合强）｜需求（不清・变更频繁）｜复杂度（难分解）｜可测性（不可观

> 测）｜数据（种类多・环节多）｜代码（质量差・结构差）｜环境（配置复杂・配置项多）。右侧三段：新开发（问题高发）｜继承特性（问题高发）｜平台/维护特性（问题高发）→ 箭头指向"测试重点"。底部："按代码来源调整测试重点，不平均用力"。

> **提问：**为什么"继承特性"（老代码）是问题高发区？老代码不是最稳定吗？

> **预设回答：**

- 答"老代码稳定，不会出问题"——这是最大的误解。老代码稳定是"在没人动它的前提下"，一旦改动，它的问题就出来了。

- 答"因为改的人不熟、文档缺失、没人敢动"——准确。改动老代码的典型风险是"改一处、坏一片"，而坏的这片往往没有用例保护。

- 答"因为它跟新功能耦合，边界模糊"——也很到位，尤其在新老系统并存的场景。

- 追问收口：所以回归测试的重点，恰恰是那些"看起来不会坏"的老功能。

> **易错点 / 考点：**① 七组原因要能答出至少五组；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 案例题常考"需求频繁变更如何应对"，答"短周期、可追溯、滚动更新测试项"。

> **过渡：**分析做到这一步，可以收口了。下一页用六个基本问题做小结。

【10.31】小结：从基本问题出发完成测试分析（2 分钟）

> 这一页是 10.2 的小结，也是我认为整章最该背下来的一页。它把分析的全部成果归成五条，又把分析的全部出发点归成六个问题。
>
> 先看五条成果。**第一，分解了产品质量目标**——把"质量要好"拆成一条条可验证的目标。**第二，基于风险对特性进行了分类**——哪些是高风险特性、哪些是常规。**第三，确定了测试深度和广度以及测试优先级**——深度是测多细，广度是测多宽。**第四，确定了测试的总体框架，知道了如何安排各种测试活动**——什么阶段做什么类型。**第五，预估了测试结果（缺陷趋势）**——你甚至能预判"大概会出多少缺陷、集中在哪"。
>
> 再看六个基本问题，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？测试的重点和难点是什么？测试的深度和广度？先测试什么，再测试什么？评价测试的效果？**

这六问的价值在于：它们是**任何测试项目的通用提问模板**。你拿到任何一个陌生的系统，把这六问逐条答一遍，就已经完成了一半的测试分析。

六问与五条成果的对应关系也很有意思：前两问对目标与范围，三四问对深度广度与重点难点，第五问对优先级，第六问对评估。**问题的顺序就是分析的顺序**。

我建议你把六问抄在笔记本第一页。以后每次做课程设计、做实习项目，先答这六问，再动手写任何文档。

再补一句：六问也是评审别人计划的提问模板。你拿着一份计划逐条问这六问，很快就能发现它哪一格是空的。

板书：上排五条成果：分解质量目标｜按风险分类特性｜定深度广度与优先级｜定总体框架与活动安排｜预估缺陷趋势。下排六问：对象与范围？目标？重点难点？深度广度？先后顺序？如何评价效果？中间画双向箭头："六问 → 五条成果"。

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，因为它最容易写成口号。

- 答"评价测试的效果"——也很常见。很多人到项目结束都没想清楚"怎么证明这轮测试是有效的"。

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

- 追问收口：凡是需要判断力的问题都容易被写成空话；对策是**每答一问，都逼自己给出一个可核对的对象或数字**。

易错点 / 考点：① 六个基本问题是本章高频简答题，必须能默写；② 五条成果的名称；③ 判断题陷阱："测试分析结束的标志是文档写完"——错，标志是能指导计划与设计；④ 记忆抓手：六问＝对象・目标・重难・深广・先后・评价。

过渡：分析完成，接下来要把"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）

进入 10.3，测试估算与进度安排。这一节解决一个很俗但躲不开的问题：**要多少人、多少天**。

四个小节：**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**。

这四节的逻辑是：先算总量（估算方法），再算细账（WBS），然后配人配物（资源），最后排成时间表（里程碑与进度）。你会发现这是一条**从数字到日历**的路径。

为什么要单列一节讲估算？因为在测试项目里，估算是**唯一一个所有人都盯着、又所有人都靠拍脑袋**的数字。拍高了被砍，拍低了天天加班，而且事后很难说清"到底估错了没有"。

这一节会给你四种估算方法，它们精度不同、成本不同。学会它们的目的不是让你算出精确数字，而是让你的数字**有出处、可解释、可修正**。

带着一个问题听：**如果领导问"这个系统测完要几天"，你答的数字是怎么来的？**

板书：四小节横排成流程：工作量估算（算总量）→ WBS（算细账）→ 资源安排（配人配物）→ 里程碑与进度（排成日历）。右侧一句："从数字到日历"；底部："数字要有出处、可解释、可修正"。

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，但确实是质疑的来源。

- 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"——这是本质原因。

- 答"因为测试没有可见的产出物"——部分对：测试的产出是证据和判断，不像代码那样一眼看得见。

- 收口：所以估算时要**把假设写出来**："按上一版缺陷密度估算""按版本按期交付估算"，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，它强依赖上游交付质量与进度；③ 记住估算的三条底线：有依据、有假设、有区间。

过渡：先看一个具体场景：拿到一张表，你能判断它是干什么用的吗？

【10.33】这是干什么的？（1 分钟）

这一页课件只放了一张图，配了一句问话："这是干什么的？"我先不揭答案，教你怎么读它。

读这类表，看三样东西。**第一看列头**：如果有"任务、工期、开始、结束、责任人、前置任务"，这是**进度表**；如果有"任务、工作量（人天）、单价"，这是**估算表**；如果有"任务、人力、时间占比"，这是**资源分配表**。列头决定表的身份。

第二看粒度：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？粒度决定它能不能用来管进度——太粗只能汇报，太细维护成本高。

第三看有没有依赖：有前置任务、有里程碑的，说明它是**排过序的计划**，不是一堆任务的堆砌。

我为什么要花一分钟讲"读表"？因为在项目里，你接手的往往不是"从零做计划"，而是"接手别人做了一半的计划"。**看懂现有计划的形状，比会画一张新表更常用**。

这一页的真正落点是：估算、进度、资源，三者最后都长成表；表的结构决定了它能不能被执行和跟踪。下面几页我们就把这三张表一张一张做出来。

板书：读表三看：① 看列头定身份（工期→进度表｜人天→估算表｜人力占比→资源表）② 看粒度定用途（粗→汇报｜细→管理）③ 看依赖定真假（有前置与里程碑＝排过序的计划）。底部："看懂别人的计划，比会画新表更常用"。

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"——很准确的区分：**汇报看结果，管理看责任与依赖**。
- 收口：所以做计划时，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（列头法）；② 高频考点：进度表必须包含的要素——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：工作量怎么算出来。

【10.34】测试工作量估算方法（2 分钟）

这一页给五种估算方法，我按"从粗到细"的顺序讲，并说清各自适用场合。

第一种，**功能点方法**。用软件规模（功能点）反推测试工作量。它的好处是有相对客观的规模基准，坏处是需要历史生产率数据才能真正换算。这是下一页要展开的。

第二种，**工作分解结构表（WBS）方法**。把测试工作拆成任务树，逐项估，再汇总。这是最常用、也最容易被接受的方法，因为**每一项都能说清为什么是这么多天**。10.3.2 专门讲。

第三种，**历史数据推算**，用相似规模、同类型的项目数据外推。它精度不错，前提是你有历史数据——这也是为什么成熟团队会建"项目度量库"。

第四种，**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往往比公式准，因为他把"这次需求特别糙""这个团队新人多"这些公式算不进去的因素都考虑进去了。

第五种，**综合方法**：几种方法各算一遍，取交集或加权。这是最稳的做法，也是工业界的常态——**没人只信一种方法**。

五种方法的取舍有个简单原则：**项目越新、越没历史数据，就越依赖经验法和 WBS；越成熟、数据越全，就越可以用功能点和历史推算**。

最后提醒一个陷阱：估算的结果应该是**区间**，不是点值。"18 到 22 人天"比"20 人天"更专业，因为它承认了不确定性。

还有一个常见做法：估两遍。第一遍自下而上按 WBS 加总，第二遍自上而下按经验给总数；两者差得太多，说明有一处判断出了偏差，回去查那一处。

> **板书：**五种方法竖排并标"粗→细"：功能点（规模反推）｜WBS（拆任务逐项估）｜历史数据推算（相似项目）｜经验法（团队/专家）｜综合法（多法交叉）。右侧："新项目靠经验＋WBS，成熟项目靠功能点＋历史"。底部："估算给区间，不给点值"。

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"——这正是原因。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。

- 追问收口：所以专业做法是"专家判断＋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 方法与功能点方法的区别（前者自上而下拆解，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 记住"估算给区间"这条专业习惯。

> **过渡：** 五种方法里，功能点方法最有"公式感"，先看它。

【10.35】软件规模估算：功能点方法（2 分钟）

这一页讲功能点方法，核心是一个公式和五类元素。

先看公式：**AFP（调整后功能点）＝ UFP（未调整功能点数目）× AF（影响因子）**。分两步走：先数出未调整的功能点，再用影响因子调整。

再看五类元素，它们是被"数"的对象：

外部输入数（EI，external input）——系统从外部接收的输入，比如录入表单、导入文件、接口请求。

外部输出数（EO，external output）——系统向外提供的输出，比如报表、导出、通知。

外部查询数（EQ，external query）——只读的查询，比如列表页、详情页。

内部逻辑文件（ILF，internal logical file）——系统内部维护的主要数据实体，比如客户表、订单表。

外部接口文件（EIF，external interface file）——本系统引用、但由别的系统维护的数据，比如从核心系统读到的账户信息。

这五类为什么这么分？因为它们在实现和测试上的复杂度不同：同样是"一个功能"，一个只读查询和一个带事务的输入，测起来差别很大。功能点方法用这五类把"规模"量化了。

对我们做测试的人来说，功能点方法的真正用途是：**它给出了一个和实现技术无关的规模尺子**。不管用 Java 还是 Python，功能点数是稳定的，所以它能用来横向比较项目、推算工作量。

但要说清它的局限：功能点衡量的是"功能规模"，不衡量"质量难度"。一个高安全要求的模块和一个普通模块，功能点可能一样，测试工作量却差几倍。所以功能点只能当**起点**，还要用风险与质量要求去修正。

再补一句实操：功能点法的五类元素，恰好也是一张很好的测试项检查表——每一类都问一句"这类东西我们测了吗"，能发现不少遗漏。

板书：公式：AFP = UFP × AF。五类元素：EI 外部输入｜EO 外部输出｜EQ 外部查询｜ILF 内部逻辑文件｜EIF 外部接口文件。底部："规模尺子与技术无关"；"功能点不含质量难度，需用风险修正"。

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"还要看是不是老系统改造，改造比新建难测"——很好的补充，继承特性测试成本更高。
- 追问收口：所以功能点给出的是"基数"，工作量＝基数 × 一堆修正系数。

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与中文名称对应，选择题高频；③ 高频辨析：ILF（内部维护）与 EIF（外部维护）的区别；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：公式给了基数，接下来看真正影响测试工作量的是哪些因素。

【10.36】测试工作量估算（3 分钟）

这一页讲测试工作量到底由什么决定。课件列了六条影响因素，最后给了一句结论，我把结论放到最后讲，因为它最重。

第一，**测试任务由质量需求、测试目标决定**。目标定得高，任务就多。要"验一遍主流程"和要"证明系统能扛住峰值"，工作量差着量级。

第二，**测试范围由产品（新）功能特性或测试任务决定**。范围是工作量的第一乘数——范围翻倍，工作量基本翻倍。

第三，**代码质量越低，测试工作量越大**，比如回归测试次数与频率加大。这一条特别实在：烂代码的缺陷密度高，修一轮带出新的，回归就得反复做。**质量差的代码，会让测试陷入"永远回归不完"的循环**。

第四，**处在不同的开发阶段测试工作量不同**。同样是测一个功能，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的成本。

第五，**自动化程度高，测试工作量就越低**。注意是"越低"不是"为零"——自动化把重复执行的成本降下来，但脚本的开发与维护成本要算进去。所以自动化对**高频回归**最划算，对一次性验证不划算。

第六，**针对不同的应用领域、技术、编程语言，其估算方法不同**。金融、电信这类高可靠领域，测试深度要求高；嵌入式、移动端还多出设备与兼容矩阵的成本。

现在看课件那句结论：**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，测试范围和测试任务是测试工作量估算的主要依据。** 这句话是本节的标准答案：范围＋任务＝主要依据。

我要补一句实践提醒：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，这些往往占到一半以上。只按"跑用例"估出来的数，一定不够。

> **板书：**六因素竖排：质量需求与目标｜测试范围（第一乘数）｜代码质量（越差越多回归）｜开发阶段｜自动化程度（降重复成本，非零成本）｜领域与技术。底部结论句照抄："测试工作量由测试范围、策划任务和开发阶段确定，范围与任务是主要依据。"再补一行："别只算执行，准备常占一半以上"。

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，尤其数据构造困难时。

- 答"缺陷复测和回归"——很重要：一个缺陷从提出到关闭，平均要经过提出、修复、复测、回归、关闭几轮，每轮都要人。

- 追问收口：所以正确的估法是按**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 课件结论句是简答标准答案，务必背准；③ 高频判断："自动化程度越高，测试工作量越低，所以自动化一定划算"——错，要看回归频次；④ 记住"准备成本常被低估"这一实践要点。

> **过渡：**估算最常用的工具是 WBS，下一页专门讲它。

【10.37】工作分解结构表方法 WBS（2 分钟）

这一页讲 WBS，工作分解结构表方法。课件给了三步，我逐条讲，再讲它为什么准。

第一步，**列出本项目需要完成的各项任务**。注意是"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，**对每个任务进一步细分，可进行多层次的细分，直到不能细分为止**。这一步是 WBS 的灵魂。什么叫"不能细分"？标准是**能直接估算、能指派给一个人、能判断完成没有**。三个条件都满足，就不再往下拆。

第三步，**根据任务的层次给进行编号，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"一目了然，也让后面的估算汇总、进度跟踪、责任分配都能挂在编号上。

为什么 WBS 准？课件给了答案：**测试工作量的估算依赖于测试任务的细化，对每项测试任务进行分解，然后根据分解的子任务进行估算。通常分解粒度越小，估算精度越高。** 这句话的道理很简单：估"测整个系统要多久"没法估，估"给这张表加五个字段校验的用例要半天"就能估。

但要注意一个反向的边界：**粒度不是越细越好**。拆到"打开浏览器"这一级，管理成本会超过估算收益。工程上的经验是拆到"半天到两天"的量级最合适——短到能估准，长到值得管。

还有一个常被忽略的用法：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。

> **板书：**三步：列任务 → 逐层细分（细到能估算・能指派・能判断完成）→ 编号成表。底部两句："粒度越小，估算精度越高"；"拆到半天到两天最合适"；右侧："同一棵 WBS：挂人＝责任表，挂日期＝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，它是一个大任务"——作为**层级名**可以，作为**可估算的任务**不行，因为它没法指派给一个人、也没法判断完成。

- 答"不能，得拆到具体活动"——对，比如拆成"编写用例、搭建环境、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：WBS 的上层是分类，下层才是任务。

- 追问收口：所以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： 看一个 WBS 的实际样子。

【10.38】示例：WBS 方法（2 分钟）

这一页是 WBS 的示例。课件用一张图展示分解结果，我给你讲清一份测试 WBS 的常见骨架，你照着就能画出自己的。

顶层通常是"某项目测试工作"，往下第一层按**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：比如"测试执行"下面分单元测试、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：比如"系统功能测试"下面分账户、交易、对账、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。

编号随之生成：`4.3.2.1` 表示第四块（执行）下第三类（系统功能）的第二组（交易）的第一项活动。你一看编号就知道它在树上的位置。

我要强调这张图的三个用法。**用法一，估算**：自下而上汇总人天，得到总量；再和自下而上的经验值对照。**用法二，找漏**：把 WBS 和测试项清单对照，有没有测试项落不到任何任务上？有，说明计划漏了活。**用法三，排期**：同一层的兄弟任务往往可以并行，父子任务必须串行。

最后提醒一个真实场景里的大坑：**WBS 里最容易被漏掉的是"沟通与评审"类任务**。用例评审、计划评审、缺陷评审、日报周报，这些活不写进 WBS 就不算工作量，但它们实际吃掉了大量时间。

最后补一句：WBS 要有人维护。任务拆了不更新，两周后它就和现实脱节，变成一张只能挂在墙上的图。

板书： 画一棵四层树：项目测试工作 → ① 计划与分析 ② 设计与用例 ③ 环境与数据 ④ 执行 ⑤ 缺陷与回归 ⑥ 总结与报告；第四层落到"编写/评审/执行/记录"。右侧编号示例：`4.3.2.1`。底部三用法："估算・找漏

・排期"；再写："别漏沟通与评审类任务"。

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：每一项测试都要有对应的设计、执行任务。

- 答"用测试项的数量来分配 WBS 的工作量"——也很实用：某个模块 60 个测试项，就大致能推出它占多少工作量。

- 追问收口：两者对齐了，计划才是"有内容的计划"，而不是"有格式的表格"。

易错点 / 考点：① 测试 WBS 的常见四层结构；② 高频考点：WBS 编号的层级含义；③ 判断陷阱："WBS 做完就等于进度排完"——错，还要加依赖、资源、日历；④ 记得把沟通评审类任务写进去，这是实践加分点。

过渡：估算和分解有了，接下来要给测试加上"风险"这个维度。

【10.39】10.4 测试风险与测试策略（1 分钟）

进入 10.4，测试风险与测试策略。这一节是本章里"最像管理、其实最考验判断"的一节。

三个小节：**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。第一，**哪些地方可能出事，出多大的事，怎么办**——这是风险。第二，**在资源不够、时间不够的情况下，怎么打这场仗**——这是策略。

为什么把风险和策略放在同一节？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**。没有风险识别，策略就是平均用力；没有策略，风险清单就只是一张吓人的表。

这一节有个很实在的价值：它是测试人员**向上沟通的武器**。你能说清"这三块风险最高，如果只给我两周，我只能覆盖前两块"，你就不再是"要人要时间的那个部门"，而是"能给出取舍方案的专业角色"。

带着一个问题听：**如果时间被砍掉一半，你会砍掉什么、保住什么？依据是什么？**

板书：中间写"10.4 风险与策略"；左侧"风险：哪里会出事・多大・怎么办（识别→分析→防范）"；右侧"策略：资源不够怎么打（取舍）"；底部一句："风险说要往哪用力，策略说怎么用力"。

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"——忙不能证明价值。

- 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，管理层确实更关心后果而不是过程。

- 收口：所以风险清单的写法要落到**后果**上，而不是"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）

这一页给测试风险下定义并划范围。课件先讲清风险的来源，再讲风险管理怎么干。

先说来源，课件列了几条：**软件系统的复杂性**、**需求频繁变更**、**测试范围的分析不到位（存在未知区域）**、**测试的充分性（覆盖率）不足**、以及**某些不确定性**。这五条要背下来，因为它们是"为什么测试总有风险"的标准答案。

我解释一下其中两条。**"测试范围分析不到位，存在未知区域"**——这一条最值得警惕，因为它不是"已知没测"，而是"不知道自己没测"。已知的缺口可以沟通、可以补；未知的缺口只能等它上线后自己暴露。

"测试的充分性不足"——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，这是先天属性，不是工作失误。

再看风险管理的三个环节：**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"采取什么措施"。三个环节的顺序不能颠倒：没识别出来就无从分析，没分析就无法决定防到什么程度。

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。请注意"降低或缓解"这个措辞——风险不能被消灭，只能被管理。任何声称"风险已消除"的测试报告，都值得你

多看一眼。

最后给你一个视角的转换：风险不是"坏消息"，风险是**决策依据**。管理层要决定"能不能发版"，靠的就是你这份风险清单。

板书：来源五条：复杂性｜需求频繁变更｜范围分析不到位（未知区域）｜覆盖率不足｜不确定性。三环节：识别 → 分析 → 防范。底部两句："风险不能消灭，只能降低或缓解"；"风险清单＝发版决策依据"。

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"——责任的角度的确更重，但风险的可控性反而更高。

- 答"不知道更危险，因为无法应对"——从风险管理角度这是更准的判断：**未知风险无法定价，也无法沟通**。

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

- 追问收口：所以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，第48 页会讲。

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 记住"未知区域"这个提法，案例题常考。

过渡：三环节里的第一步是识别，它有具体做法。

【10.41】风险识别（2 分钟）

这一页讲怎么"找风险"。课件给了方法，也给了产出要求。

第一句是关键：**风险识别的有效方法就是建立风险项目检查表**。为什么是检查表？因为人脑在开会时容易只想到最近发生的事、最容易想到的事，检查表能把"该想的都摆到桌面上"，避免遗漏和偏见。

检查表怎么来？课件的答案是**历史资料、Brainstorming（头脑风暴）等**。历史资料包括：上一版的生产事故、缺陷库里的高频模块、行业里的经典事故案例。头脑风暴则是把开发、测试、运维、业务拉到一起，各说各的担忧——注意，不同角色担心的东西完全不同，这正是头脑风暴的价值。

一个成熟团队会把检查表沉淀成资产：几十条常见风险，每次项目开局逐条过，命中就打勾、评估、定措施。

这就是"组织级过程资产"的价值。

然后是一个很容易被忽略的要求：**风险识别并确定其程度，给出预防或处理措施**。也就是说，光列风险不算完成，还要**定级＋给措施**。只列不定的风险清单，在评审会上会被问死："这条风险有多大？你打算怎么办？"

我给你一个检查表的常见分类骨架，便于你课后自己建：**需求类**（不清晰、变更频繁、验收标准缺失）、**技术类**（新技术、遗留系统、第三方依赖）、**资源类**（人不够、环境不稳、数据难造）、**进度类**（上游延期、窗口太短）、**质量类**（代码质量差、历史缺陷多）、**组织类**（职责不清、沟通不畅）。

最后提醒：风险识别不是一次会议，它是**贯穿项目的过程**。每个阶段开始前都要重扫一遍，因为风险会变。

板书：方法：风险项目检查表（来源：历史资料＋头脑风暴）。要求：识别 → 定程度 → 给措施（三件事都做才算完成）。分类骨架六类：需求・技术・资源・进度・质量・组织。底部："只列不定＝没做风险识别"。

提问：为什么必须"不同角色一起"做头脑风暴？

预设回答：

- 答"人多想法多"——表面理由。

- 答"因为不同角色担心的东西不一样：开发担心技术难点，运维担心部署与监控，业务担心规则边界"——准确。测试最怕的是"只从测试视角看风险"。

- 答"还能顺便对齐期望，让大家都知道风险在哪"——这是额外收益：风险共识比风险清单更值钱。

- 追问收口：所以风险识别的产出不只是清单，还有**共识**；没有共识的清单，执行时没人配合。

易错点 / 考点：① 检查表法的地位（"有效方法"）；② 三个动作：识别・定级・措施；③ 高频考点：头脑风暴的跨角色价值；④ 案例题常给一个项目背景要求列风险，按六类骨架答不会漏。

过渡：识别出风险之后，要给它算一个"严重程度"。课件给了两个公式。

【10.42】风险分析常用的模板（2 分钟）

这一页给两个计算模板和一张矩阵，是本节最"像数学"的地方，但其实很好记。

第一个模板：**风险严重性 = 可能性 × 影响**。两个维度相乘，而不是相加。为什么用乘？因为它表达的是"两

个都小才是真的小"：可能性再低，影响极大（比如资金损失）也不能忽略；影响再小，可能性极高（天天发生）也很烦人。

第二个模板：**风险危害 = 可能性 × 后果严重程度 × 范围**。比第一个多了一个"范围"因子。这个加法很有讲究：**同样一个缺陷，影响一个用户和影响全部用户，危害差着量级**。范围是测试里特别该问的一句："这问题会影响多少人、多少笔业务？"

再看那张矩阵：横轴是**影响程度**（低、中、高），纵轴是**发生的可能性**（低、中、高），交叉出九个格子，标着很低、低、中等、高、很高。这是最经典的三乘三风险矩阵。

矩阵的用法有两个。**用法一，定级**：落在"高可能×高影响"那一格的，是必须优先处理的；落在"低可能×低影响"那一格的，可以接受或监控。**用法二，沟通**：把风险画进格子里，比写一段文字更容易让非技术人员理解——"这个在最高那一档"，谁都听得懂。

这里有个应试要点：矩阵的对角线是"同等级"，但**不对称**。可能性和影响哪个更重要？在测试里通常**影响优先**：影响极大的风险，即使可能性低也要有应对方案（比如资金类、合规类）。这一条在案例题里很吃分。

最后一句话提醒：风险分析的结果要**定期更新**。同一个风险，随着版本推进，可能性和影响都会变，三个月前的红色风险现在可能已经变成绿色。

板书：两个模板：风险严重性＝可能性×影响；风险危害＝可能性×后果严重程度×范围。右侧画 3×3 矩阵：纵轴可能性（低中高）、横轴影响（低中高），右上标"很高"，左下标"很低"。底部："乘法不是加法：两个都小才真小"；"影响优先于可能性"。

提问：一个"每千次操作出错一次、但一出错就导致对账差一分钱"的问题，风险算高还是低？

预设回答：

- 答"可能性低，算低风险"——典型的只看可能性。

- 答"影响也不大，一分钱"——这就是不懂业务：**一分钱的差错会污染整批对账，导致全量重跑甚至资金挂账**。

- 答"要按危害公式算：可能性低，但后果严重、范围大（整批对账），综合应为高风险"——正确思路。

- 追问收口：所以风险定级前必须问清"后果"和"范围"，这两项不能靠测试人员单独判断，要拉业务方一起定。

易错点 / 考点：① 两个公式的因子，特别是第二个多出的"范围"；② 高频考点：3×3 矩阵的读法与定级；③ 判断陷阱："可能性低就不用管"——错，影响优先；④ 案例题常给若干风险要求定级并给措施，答案要写清可能性、影响、范围三个判断依据。

> **过渡：**公式是工具，来看一份真实的测试风险示例。

【10.43】示例：测试风险（2 分钟）

这一页给了一份测试风险示例，课件还追问了一句："还有什么高风险的项？"我们就按这个思路，把一张典型的测试风险表补全。

示例里通常会有这几类：**需求变更频繁**导致用例反复重写；**关键模块代码质量差**导致缺陷密度高、回归没完；**第三方接口不可控**导致联调阻塞；**测试环境不稳定**导致结论不可靠；**测试数据难构造**导致某些场景根本测不了；**人员经验不足**导致高风险模块测不深。

我补充几条经验上最该盯、却最常被漏掉的：

第一，上游交付延期。测试的时间窗口是被上游压出来的，这是测试项目最普遍的进度风险。

第二，性能与容量的未知。功能测完了，但从没在接近生产的数据量下跑过——这是上线首日事故的经典成因。

第三，数据迁移与历史数据兼容。凡是涉及老数据、老配置、老版本的，几乎必出问题。

第四，权限与安全边界。越权、信息泄露这类问题，功能测试天然覆盖不到。

第五，回滚方案不可用。上线出问题时能不能退回去？回滚本身也要测，这一点经常整条被忘掉。

判断"什么算高风险项"有个简单公式：**新（没做过）＋复杂（依赖多）＋关键（坏了业务停）＋历史多（老出问题）**，四个里占两个以上，就该进高风险清单。

再提醒一句写法：风险条目要写成"**原因 → 后果**"的句式，比如"第三方支付接口联调窗口只有两天，若延期将导致支付相关 30 个测试项无法执行"。这样评审的人一眼能判断要不要给资源。

> **板书：**示例风险清单：需求变更频繁｜关键模块代码质量差｜第三方接口不可控｜环境不稳定｜数据难构造｜人员经验不足。补充五项：上游延期｜性能容量未知｜数据迁移与老版本兼容｜权限与安全边界｜回滚方案不可用。公式："新＋复杂＋关键＋历史多，占两条即高风险"。写法："原因 → 后果"。

> **提问：**"回滚方案"为什么也属于测试风险？

> **预设回答：**

- 答"回滚是运维的事，跟测试无关"——最常见的误解。回滚脚本、数据回退、版本切换，全都要验证。

- 答"因为回滚没测过，真出事时可能退不回去"——准确，而且后果极严重：故障时长被放大数倍。

- 答"回滚也算一次变更，当然要测"——很专业：回滚是"反向变更"，同样会带来副作用，正是第 8 页讲的"变更实施不得带来负面效应"。

- 追问收口：所以高风险清单的检查角度之一，是"**除了正常路径，还有哪些异常处置路径没验证过**"。

> **易错点 / 考点：** ① 常见测试风险的类别；② 高频考点：高风险项的判断公式（新・复杂・关键・历史多）；③ 判断陷阱："回滚属于运维，不在测试范围"——错；④ 风险条目的写法要求：原因＋后果，可评估。

> **过渡：** 识别和分析完了，接下来是"怎么办"——风险控制。

【10.44】风险常用控制方法（2 分钟）

这一页给五条风险控制方法，我按"事前—事中—事后"的顺序重排一下，更好记。

事前两条。第一，**做好风险管理计划，制定有效的风险应急处置方案**。注意是"应急处置方案"，不是"注意事项"：真出事的时候，你要知道第一步打给谁、第二步切哪个开关。第二，**计划时，对于估算资源、时间、预算留有余地**。这就是我们常说的缓冲（buffer）。留余地不是偷懒，是承认不确定性——**没有缓冲的计划，等于假设一切顺利**。

事中两条。第三，**采取措施避免可以避免的风险**。能绕开就绕开：环境不稳就提前搭两套，数据难造就早点开始造。第四，**高风险转移为低风险**。转移不是甩锅，是改变风险的结构：把一个大版本拆成两次小发布，把一次大迁移拆成分批迁移，风险就被摊薄了。

事后一条。第五，**设法降低不可避免的风险所带来的损失**。有些风险躲不掉，那就把损失控制在小范围：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，我认为最被低估的是第二条"留余地"。现实中的计划往往是"刚好排满"，任何一点意外都会导致全线延期，而延期又会挤压测试时间，形成恶性循环：**越没缓冲，越容易牺牲测试；越牺牲测试，后期事故越多；事故越多，越没时间**。

给你一句可以直接用在评审会上的话："这个排期没有缓冲，任何一天的上游延期都会直接压缩测试窗口，请确认是否接受这个风险。"

> **板书：**五条按三段排：事前（风险管理计划＋应急处置方案｜估算留余地/缓冲）｜事中（避免可避免的风险｜高风险转低风险：拆分、分批）｜事后（降低不可避免风险的损失：灰度・开关・限流・回滚）。底部："没有缓冲＝假设一切顺利"；"把全量失败变成局部失败"。

提问："把高风险转移为低风险"，具体怎么转移？举一个测试场景。

预设回答：

- 答"买保险"——那是财务转移，测试里更常见的是**结构转移**。

- 答"把大版本拆成两次发布，第一次只上低风险功能"——正确。风险被摊薄，每次的爆炸半径都变小。

- 答"把全量数据迁移改成分批迁移，每批验证后再继续"——也很典型。

- 追问收口：所以"转移"的本质是**改变交付结构**，让风险落在可控的粒度上；这也是第 49 页"两段论"能成立的原因。

易错点 / 考点：① 五条控制方法；② 高频考点：缓冲（buffer）的意义；③ 判断陷阱："风险转移就是推给别人"——错，是改变风险结构；④ 案例题常考"排期无缓冲怎么办"，答"书面提示风险＋给取舍方案"。

过渡：风险讲完了，接下来讲策略。先从"策略"这个词本身说起。

【10.45】为什么要制定测试策略？（2 分钟）

这一页问的是一个看起来很简单的问题：为什么要有测试策略？课件给了两段英文定义，我们从定义入手。

第一段：**Strategy：the art of devising or employing plans or stratagems toward a goal**——策略是"为实现目标而设计或运用计划与谋略的艺术"。注意两个词：**art（艺术）** 和 **toward a goal（朝向目标）**。它是艺术，说明它没有唯一解；它朝向目标，说明它必须服务于目标，脱离目标的策略是花架子。

第二段：**an adaptation that serves an important function in achieving the goals**——策略是一种"适应"，在达成目标中起重要作用。**adaptation（适应）** 这个词是关键：它意味着策略是对**环境约束**的回应。没有约束，就不需要策略——什么都给足，照着标准流程做就行了。

所以这两段定义合起来回答了这一页的问题：**因为资源永远不够、环境永远有约束、目标又必须达成，所以才需要策略。** 策略是在"必须达成"和"做不到全做"之间搭的那座桥。

我给你一个更直白的说法：**测试策略就是"在打不赢全面战争的时候，怎么打赢关键战役"的作战方案。**

这也解释了为什么学生作业里最缺策略：因为课程设计往往没有真实的时间压力，什么都测一遍就行了——**没有约束，自然长不出策略**。所以这一节的案例，你要带着"资源不够"的假设去读。

再补一个反例帮你记住"约束"这两个字：如果没有约束，测试只需要回答一个问题——还有什么没测。有了约束，测试才要回答另一个问题——在这么多没测的东西里，先测哪个、放弃哪个。后者才是策略。

板书：两段定义关键词：art（艺术，无唯一解）｜toward a goal（服务目标）｜adaptation（对约束的适应）。中间写一句："资源不够＋目标必须达成 → 才需要策略"。底部："策略＝打不赢全面战争时，怎么打赢关键战役"。

提问：如果时间、人力、环境全都充足，还需要测试策略吗？

预设回答：

- 答"不需要，全测一遍就行"——理论上如此，但现实中不存在这种项目。

- 答"还是需要，因为'全测'本身也测不完"——很准：即使资源充足，测试的穷尽性也不成立，还是要选。

- 答"需要，因为策略还包括方法选择、层次划分"——补充得好，策略不只是"砍什么"，还包括"怎么测"。

- 追问收口：所以策略有两层：**选择测什么（取舍）** 和 **决定怎么测（方法）**，前者因约束而生，后者因目标而生。

易错点 / 考点：① 策略定义里的两个关键词（art、adaptation）；② 高频判断："策略就是砍需求"——错，还包括方法与资源的安排；③ 案例题常考"资源被压缩时如何调整策略"，答题要给出取舍依据；④ 记住"没有约束就没有策略"这句话。

过渡：定义之后，看策略具体包含哪些内容。

【10.46】测试策略及其内容（3 分钟）

这一页给测试策略下了完整定义，并列出四项内容，最后落到一个词：trade-offs，取舍。

先读定义：测试策略是**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；针对不同测试阶段或不同的测试对象，其测试的范围、方法、类型等进行必要的调整**。

这句话有三个要害。**"受到条件限制、不利因素"**——策略的前提是约束；**"为了实现目标而不得不"**——策略是被逼出来的，不是想出来的；**"针对不同阶段、不同对象做调整"**——策略是分对象、分阶段的，一份策略管不

了全项目。

再看四项内容。第一，**针对风险（工作量、时间等压力）采取对策，如标准取舍、测试任务的优先级等**——也就是"压力来了怎么排"。第二，**更好地执行测试用例以及后续的回归测试**——策略要保证可执行、可回归，否则再漂亮也落不了地。第三，**选定使用测试技术和工具**——用什么手段测。第四，**考虑影响资源分配的特殊情况**——比如某个模块需要专用设备、某类测试需要外部专家。

最后是那个词：**取舍，trade-offs，有得有失**。这是整页的灵魂。测试策略的本质就是"我明确地放弃什么，以换取什么"。**不肯放弃的策略不是策略，是愿望。**

我给大家一个写策略的句式，课后作业可以直接用：**"鉴于〔约束〕，我们选择〔做法〕，放弃〔什么〕，代价是〔风险〕，并由〔谁〕确认。"** 这句话写得出，你的策略就站得住；写不出，说明你还在许愿。

补一句考试提醒：策略题最容易失分的地方是只写"要做什么"，不写"放弃什么"。阅卷老师看的就是你有没有取舍意识。

板书： 定义三要害：受条件限制｜为达目标不得不｜分阶段分对象调整。四项内容：对风险采取对策（标准取舍・优先级）｜保障用例执行与回归｜选定技术与工具｜考虑资源分配的特殊情况。底部大字："trade-offs 取舍，有得有失"；句式："鉴于〔约束〕，选择〔做法〕，放弃〔什么〕，代价是〔风险〕，由〔谁〕确认"。

提问： 一份测试策略里如果没有出现"不做什么"，它算合格的策略吗？

预设回答：

- 答"算，写了要做什么就够了"——这只是任务清单，不是策略。

- 答"不算，没有取舍就没有策略"——对。策略的价值在于"在约束下做选择"，不写放弃什么，等于宣称资源无限。

- 答"看情况，如果资源够就不用写"——可以接受，但要在文档里写一句"本次资源满足全范围测试，无取舍"，把"没有取舍"也变成明确结论。

- 追问收口：所以"没有取舍"也要写出来，因为它同样是一个需要被确认的判断。

易错点 / 考点： ① 定义里的三个要害；② 四项内容的名称；③ 高频考点：trade-offs 的含义，简答常问"为什么说测试策略的本质是取舍"；④ 判断陷阱："策略一旦确定不再变化"——错，它随阶段和风险滚动调整。

过渡： 理论说完，来一道真实的压力题：时间砍掉一半，你怎么测？

【10.47】问题：减少 50%时间，如何测试？（2 分钟）

这一页是一道讨论题，也是我每年课堂上争论最激烈的一页：**如果测试时间减少百分之五十，你怎么测？**

我先给出错误答案，都是现场真实会出现的：第一种，"加班，把时间补回来"——短期可行，长期不可持续，而且加班会降低判断力，漏测率上升；第二种，"按原计划测一半的用例"——听起来公平，实际最糟：随机的"一半"会把关键路径也砍掉；第三种，"跟老板说做不到"——态度诚实，但没有给出方案。

正确的思路是**分层做取舍**，我给你一个可以照着说的五步。

第一步，重排优先级，砍尾部。 按上一节讲的优先级，只保 P0 和一部分 P1，P2 明确写入"本轮不测"。

第二步，缩深度不缩广度。 优先保证"每个关键功能都走一遍"，而不是"少数功能测到极致"。因为广度不足会带来未知风险，深度不足只带来残余缺陷。

第三步，提效率。 把高频回归用例自动化，把环境一键部署，把数据准备脚本化——这些一次性投入在压缩周期里回报最高。

第四步，改结构。 分批发布、灰度上线，把"一次全量验证"变成"多次小范围验证"，用生产环境的小流量替代部分测试环境的大验证。

第五步，把风险说清。 输出一份"未覆盖清单＋对应风险"，让决策者签字。这一条不是甩责任，是让取舍变成**组织的决定**而不是测试组的私自妥协。

五步里最重要的是第二步的判断：**宁可测浅，不可测漏**。这条原则在时间被压缩时几乎总是对的。

> **板书：**三道错答划掉：加班补时间｜随机砍一半用例｜只说不做。五步正解：① 重排优先级砍尾部 ② 缩深度不缩广度 ③ 提效率（自动化・一键环境・数据脚本）④ 改结构（分批・灰度）⑤ 风险清单＋确认。底部："宁可测浅，不可测漏"。

提问：为什么"缩深度不缩广度"比"缩广度保深度"更安全？

预设回答：

- 答"因为深度不好判断，广度好判断"——有道理：广度有清单可核对，深度很难说"够深了"。

- 答"因为漏掉一整块功能的风险远大于某块测得不细"——准确。整块漏测意味着完全未知，细度不足意味着已

知残余。

- 答"因为用户会先撞到主流程，深度问题往往在边界才暴露"——补充得好，符合缺陷暴露的实际顺序。

- 追问收口：所以压缩时间时，先保"都碰到"，再谈"挖多深"。

> **易错点 / 考点：** ① 五步取舍法；② 高频考点："缩深度不缩广度"的原则与理由；③ 判断陷阱："时间不够就全部用例减半执行"——错，应按优先级砍；④ 案例题标准答案要素：重排优先级、自动化提效、分批灰度、风险清单与确认。

> **过渡：** 有了取舍思路，来看几个具体的、已经在用的测试策略。

【10.48】示例：测试策略（2 分钟）

> 这一页是策略示例，课件连问了三个问题：**之前的回归测试策略，还记得吗？Pairwise／正交试验法隐含了什么策略？还有什么常用的测试策略？**
>
> 第一问，**回归测试策略**。常见有三类：**完全回归**——把全部用例再跑一遍，最保险也最贵；**选择性回归**——按影响分析挑出受影响的用例跑，性价比最高，但依赖影响分析的质量；**随机回归**——按一定比例抽样，适合用例量巨大且变更局部的场景。选哪一类，本身就是一次取舍：时间换风险。
>
> 第二问，**Pairwise／正交试验法隐含了什么策略？** 我的答案是**"采样代表全体"**。组合爆炸时你测不完所有组合，于是假设"多数缺陷由两个参数的交互引起"，用少量组合覆盖所有两两配对。这个假设本身就是一种策略：**用已知的缺陷统计规律，换取可执行的用例规模**。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？** 课件给了两个例子：**交叉测试策略**——不同人测别人写的模块，用"不熟悉"来对抗"思维定势"；**ST ＋ ET 测试策略**，也就是**测试执行的两段论**——先脚本化测试打底，再用探索式测试去挖脚本想不到的地方。
>
> 把这三问放一起看，你会发现**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。
>
> 学这一页的目标不是记住几个名词，而是建立一个反射：**每当你做一个测试决定，都问一句"我这是在用什么策略、放弃了什么"**。这个反射建立起来，你就从"执行者"变成"设计者"了。

> **板书：** 三问三答：回归测试策略（完全・选择性・随机）｜Pairwise/正交＝"采样代表全体"（假设两两交

> 互，换取可执行规模）｜其他（交叉测试：用不熟悉对抗定势；ST＋ET 两段论）。底部："每个测试决定背后都有一个策略，别忘了问放弃了什么"。

提问：选择性回归最依赖什么？它最大的风险是什么？

预设回答：

- 答"依赖用例库大小"——不是关键。

- 答"依赖影响分析的准确性；风险是分析漏了，该回归的没回归"——准确。选择性回归是"用分析质量换执行成本"。

- 答"还依赖变更记录的完整性，改了什么得说得清"——很好的补充，变更粒度不清时，影响分析无从下手。

- 追问收口：所以选择性回归的配套动作是"变更清单＋影响矩阵"，两者缺一，选择性回归就退化成"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 高频考点：Pairwise 的隐含假设（多数缺陷源于两参数交互）；③ 判断陷阱："完全回归永远最优"——错，成本可能不可承受；④ 记住 ST＋ET 两段论的组合思路。

过渡：第二个例子值得展开——测试执行的两段论，它是本章最实用的策略模型。

【10.49】两段论（2 分钟）

这一页讲"两段论"——把测试执行分成两段，各自有不同的目标和手段。这张图信息量大，我按阶段顺序讲。

图的左半段是**第一段：发现缺陷，强调效率**。它包含：单元/集成测试、新功能验证、**自动化每日验证（SVT/BVT）**、非功能性的系统测试。这一段的特点是**越早越好、越快越好**：目标是把缺陷尽快挖出来，把质量底数摸清。为什么这一段强调效率？因为缺陷发现得越早越便宜，而且这一段的问题最多、最需要快速迭代。

图的右半段是**第二段：验证功能，降低风险**。它包含：功能验证测试、剩余缺陷验证、确认测试、回归测试、探索式测试、针对有风险区域的功能测试。这一段的目标变了——不再是"挖"，而是"**确认**"：确认修好了、确认没带坏、确认风险可控。所以它强调的不是效率，是**可信度**。

中间那条分界线标着"**关键／里程碑**"。这就是两段论的精髓：**在某个里程碑之前，你在探索质量；过了这个**

> **里程碑，你在证明质量。** 两段的判据、节奏、心态完全不同。
>
> 如果混淆这两段会发生什么？最常见的事故是：**里程碑之后还在大改代码**——这时测试已经进入"证明"模式，改动会不断打破已验证的结论，回归永远追不上，最后只能带风险上线。所以两段论不只是测试策略，它其实是**对项目节奏的约束**：过了里程碑，改动必须有门槛。
>
> 还有一个细节值得注意：**探索式测试被放在第二段**。很多人以为探索式是"早期乱点"，其实它更常用于**后期针对高风险区域的定向挖掘**，因为那时你才知道风险在哪。

> **板书：** 画一条横线，中间标"关键/里程碑"。左侧（第一段：发现缺陷，强调效率）：单元/集成 → 新功能验证 → 自动化每日验证 SVT/BVT → 非功能性系统测试。右侧（第二段：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 → 高风险区定向测试。底部："之前探索质量，之后证明质量"。

> **提问：** 为什么"探索式测试"通常放在第二段，而不是项目一开始就做？

> **预设回答：**

- 答"因为早期没东西可测"——不准确，早期也有可测对象。

- 答"因为后期才知道风险在哪，探索才有方向"——准确。探索式测试的价值取决于探索者的假设质量，而假设来自前面积累的信息。

- 答"因为它能发现脚本覆盖不到的问题，正好补第二段'证明'模式的盲区"——很到位：第二段容易陷入"只验证已知"，探索式负责"发现未知"。

- 追问收口：所以两段论的深层逻辑是"**先广撒网，再定点挖**"，探索式在两个阶段都能用，但用法的目的不同。

> **易错点 / 考点：** ① 两段各自包含的活动，特别是 SVT/BVT 属于第一段；② 高频考点：两段的目标差异（发现缺陷 vs 验证功能）；③ 判断陷阱："过了里程碑还能随意改代码"——错，这正是两段论要防的事故；④ 记住"关键/里程碑"是分界线。

> **过渡：** 策略受什么影响？课件列了八个因素。

【10.50】测试策略影响因素（2 分钟）

> 这一页列了影响测试策略的八个因素，我逐条讲，并告诉你每一条会把你推向哪种策略。

第一，**测试方式**：静态还是动态、要不要用探索式方式。偏静态的策略更省成本但覆盖行为少；偏探索式的策略更灵活但难度量。

第二，**测试方法**：黑盒还是白盒。有源码、有覆盖率工具，就能用白盒；系统是外购的，只能黑盒。

第三，**测试层次**：单元、集成、系统。层次决定你能在哪个粒度上发现问题——单元层最容易定位，系统层最接近真实。

第四，**测试人员**：责任、能力、独立性。这一条最常被忽略。**人员能力直接决定策略可行性**：团队里没人会性能调优，你写一套复杂的性能测试策略也执行不了。独立性指的是"测试的人是不是写代码的人"，独立测试更容易发现盲区。

第五，**测试用例选择/优化**：用例有没有优先级。有优先级，你才能做"时间不够时砍尾部"的取舍；没有优先级，只能全跑或随机跑。

第六，**测试环境**：设置是否简单、能不能自动部署。环境能不能一键起来，直接决定你能跑多少轮回归——**环境能力就是测试频率的上限**。

第七，**测试工具**：能不能用工具、用起来简单与否。工具的可获得性和易用性，会改变你整个测试的性价比结构。

第八，**质量标准**：采用国内标准还是美国 DO-178C 这类严格标准。标准决定了"什么叫达标"，是策略的天花板——DO-178C 那类航空标准，对追溯性和覆盖的要求会重塑整个测试流程。

八条里，我要特别点出第四、第六两条：**人和环境是策略的两个硬约束**，很多策略失败不是因为想错了，是因为人和环境撑不住。

> **板书：**八因素竖排：测试方式｜测试方法｜测试层次｜测试人员（责任・能力・独立性）｜用例选择与优化｜测试环境（能否一键部署）｜测试工具｜质量标准（国内标准／DO-178C）。右侧标注两条硬约束："人员能力""环境能力"。底部："环境能力＝测试频率的上限"。

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"——现象层面的正确。

- 答"因为每轮回归都要部署、造数据、清理，环境不行这些时间就把轮次吃掉了"——准确。

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

- 追问收口：所以第 12 章要专门讲"部署测试基础设施"——它不是杂活，它决定你策略的上限。

易错点 / 考点： ① 八个因素的名称；② 高频考点：人员与环境的约束作用；③ 判断陷阱："策略只由测试目标决定"——错，八因素共同约束；④ 记住 DO-178C 这个例子，它是"标准影响策略"的典型。

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）

进入 10.5，也是本章的收口：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**。

这三节的顺序很有意思：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"。你会发现它和一般文档课的讲法不一样——一般课先讲模板，这里把"过程"和"有效性"放进了同一节。

为什么？因为课件想强调一件事：**测试计划的价值不在文档本身，而在编制它的过程**。一份从模板复制、填填空格交上去的计划，格式再标准也没有用；一份经过讨论、有假设、有取舍、有风险应对的计划，哪怕只有三页纸，也是好计划。

这一节还有两个具体落点要记住：一是**进入/退出标准**，它是质量的闸门；二是**计划的评审与滚动维护**，它决定计划是"活的"还是"死的"。

带着一个问题听：**你写过的计划，后来有人看过第二遍吗？**

板书： 三小节：内容（写什么）→ 计划过程（怎么做出来）→ 制定有效计划（怎么做好）。底部两句："价值在编制过程，不在文档本身"；"两个落点：出入标准、评审与滚动维护"。

提问： 为什么"填模板"式的测试计划没有价值？

预设回答：

- 答"因为大家都不看"——现象。

- 答"因为模板里的空填满了，但判断都没做：范围为什么这么划、风险为什么这么排，全没有"——准确。计划的核心是判断，模板只能提供位置。

- 答"因为模板让计划看起来完整，掩盖了没想清楚的部分"——很尖锐：**形式完整是思考缺失最好的伪装**。

- 收口：所以评审一份计划，不要看它有几页，要看它的每个数字后面有没有理由。

易错点 / 考点： ① 三小节名称；② 高频判断："测试计划的格式比内容重要"——错；③ 记住本节两个落点：出入标准、评审与维护。

过渡： 先看一份测试计划到底包含哪些条目。

【10.52】测试计划内容（2 分钟）

这一页给测试计划的内容清单。课件的定义是：**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。** 这句话本身就是简答题答案。

具体条目，我按逻辑分组讲，比死记顺序好。

身份与背景组：测试计划标识符（文档编号）、项目总体情况简介。编号不是形式，它保证版本可追溯——计划改了七八版，你得知道评审的是哪一版。

对象与判据组：测试项（test item，含需要测试的功能）、测试项通过/失败的标准、测试中断和恢复的规定。注意"中断和恢复"这一条很少有人写，但很实用：环境挂了、版本回退、关键人请假，测试怎么停、怎么续。

方法与策略组：方法、策略。

交付组：测试完成需提交的材料。

执行安排组：测试任务与进度表、测试环境要求、人员安排与职责、培训需求。

风险组：潜在的问题和风险。

其他：课件最后留了一个兜底项。

课件提示详细内容见教材附录 B，考试若考细节以教材为准。

这里我要把本章和第 4 章接上：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**，正好就是这份清单的骨架。角色对人员职责，准入对进入准则，输入对测试项与需求，活动对方法与任务，输出对提交材料，验证对评审，准出对通过标准，度量对进度与覆盖率。你按八要素检查一遍计划，

漏项一眼就能看出来。

最后给一个实操建议：计划的第一页应该是一页纸的"摘要"——目标、范围、关键风险、里程碑、需要的支持。**决策者只看这一页**，后面几十页是给执行者看的。

> **板书：**定义句照抄（纲领性文件：目标・范围・测试项与优先级・方法・风险与策略・资源・任务与进度）。条目分六组：身份与背景｜对象与判据（含中断与恢复）｜方法与策略｜交付物｜执行安排（进度・环境・人员・培训）｜风险。右侧写第 4 章八要素对照："人・门・料・活・果・验・出口・尺"。底部："第一页写摘要，决策者只看这一页"。

提问："测试中断和恢复的规定"这一条，为什么值得写进计划？

预设回答：

- 答"因为可能出事"——太笼统。

- 答"因为测试被打断后，最难的是'从哪接着测'：哪些结论还有效、哪些要重跑"——准确。恢复规定的核心是**结论的有效性判定**。

- 答"还涉及版本回退：回退到旧版本后，之前的测试记录怎么处理"——很实际，这在真实项目里天天发生。

- 追问收口：所以这一条要写清三件事：什么情况算中断、中断期间结论如何标记、恢复后从哪一步继续。

易错点 / 考点：① 计划的定义句与主要条目，简答高频；② 高频考点：与第 4 章八要素的对应关系；③ 判断陷阱："测试计划只需写测试组内部的事"——错，还包括环境、培训、交付物、跨组依赖；④ 附录 B 是细节依据，考试以教材为准。

过渡：清单里有一条被课件单独拎出来讲，就是进入/退出标准。

【10.53】进入/退出标准（2 分钟）

这一页专门讲进入标准和退出标准，因为它是整份计划里**最有约束力**的两条。

先看定义。**进入准则**：对软件测试切入点的确立，即满足什么条件，才启动测试。**退出标准**：满足某个阶段结束／里程碑达到的事先定义的要求。

我把这两个词翻成大白话：进入准则是"**门票**"，退出准则是"**放行条**"。没有门票就开始测，是在残破版本上空转；没有放行条就宣布测完，是在赌运气。

课件还有一句总结：**软件测试每个阶段需要控制进入/退出标准，以保障测试的质量**。注意"每个阶段"——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。

这两条标准为什么必须"事先定义"？因为事后定义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"；这种调整在技术上叫口径变化，在管理上叫**移动球门**，它会彻底摧毁标准的公信力。

再讲一个实操要点：标准要**可判定**。"代码质量良好"不是标准，"编译通过、冒烟用例 100% 通过、无未解决的阻塞级缺陷"才是。判断方法和前面讲测试目标时一样——第三方能不能核对。

最后说清它的管理价值：进入/退出标准是测试组**最硬的谈判筹码**。它把"我觉得还不能上线"变成了"未满足退出标准第 3 条"，后者是事实，前者只是意见。

再补一句：进入准则和退出标准最好成对写。一个阶段有门票就该有放行条，只写一半的标准，执行时一定会被含糊过去。

板书：进入准则＝门票（满足什么条件才启动）；退出标准＝放行条（满足什么条件算结束）。中间写："每个阶段都要有，且必须事先定义"。底部："可判定才算标准"；"未满足退出标准第 X 条 ＞ 我觉得还不能上线"。

提问：项目已经延期，老板要求"降低退出标准先上线"，你怎么应对？

预设回答：

- 答"照做，老板说了算"——可以做，但必须留下书面记录：哪些标准被豁免、对应什么风险、谁批准的。

- 答"坚决不同意"——立场清楚但缺少方案，容易被绕过。

- 答"给三个选项：延期三天补测关键项／带风险上线并准备回滚／缩减范围上线"——最专业：把"降标准"变成"选方案"，让决策者承担决策责任。

- 追问收口：所以测试人员的专业性，体现在**把质量决策翻译成可选项**，而不是简单地同意或反对。

易错点 / 考点：① 进入准则与退出标准的定义，名词解释高频；② 高频考点：标准必须事先定义、必须可判定；③ 判断陷阱："退出标准可以随进度调整"——错，调整必须走变更并留下记录；④ 案例题常考"被要求降标准怎么办"，答"书面记录＋给选项＋明确风险"。

过渡：退出标准具体长什么样？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）

这一页给的是系统测试结束标准的具体例子，分两档，非常适合背下来当模板用。

第一档，对于非严格系统，可以采用"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%**。

注意这两个数字的差别，它是有讲究的。功能性要求 100%——功能不对，业务直接不能用，没有妥协空间。非功能给到 95%——因为非功能项往往受环境、数据、时段影响，存在客观上难以稳定复现的情况，留 5% 的余量是现实的。

但我要提醒一句：**"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"，如果分母本身不全，通过率再高也是假的。所以用这条标准时，必须同时规定"用例执行率"。

第二档，对于严格系统，如金融、航空这类，要补充"基于缺陷密度"等准则：一是**连续几个小时或几天"测试发现的缺陷密度"低于某阈值**——这叫缺陷收敛，说明系统在趋于稳定；二是**代码的缺陷密度要高于某阈值（如上一个版本的平均值）**——这一条乍看很怪，为什么要求缺陷密度**高**于阈值？

因为它的逻辑是"**测出来的缺陷太少，可能是测得不深，而不是质量好**"。如果这一版的缺陷密度明显低于历史平均，先别高兴，要怀疑覆盖不足。这是很有经验的一条判据，也是我最想让你们记住的一条。

两条准则合起来看：**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"**。前者是结果判据，后者是过程判据。严格系统两条都要有。

> **板书：**非严格系统（基于用例）：功能性通过率 100%｜非功能性通过率 95%（另需规定用例执行率）。严格系统（补充基于缺陷密度）：连续数小时/数天缺陷密度低于阈值（收敛）｜代码缺陷密度高于上版平均值（防测得太浅）。底部："通过率答'对不对'，缺陷密度答'测得够不够'"。

提问：为什么"缺陷密度低于历史平均"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"因为可能是测得不够深：用例没设计到、执行不彻底、环境跑不通"——准确。

- 答"还可能是因为缺陷被漏报了，或者被合并统计了"——补充得好，数据口径也会骗人。

- 追问收口：所以看到"异常好"的数据，第一反应应该是**验证数据的来源**，而不是庆祝。

易错点 / 考点： ① 两档准则的具体数字（100% / 95%）是选择题高频；② 高频考点："代码缺陷密度要高于阈值"的原因；③ 判断陷阱："通过率 100% 就说明测完了"——错，还要看执行率与覆盖；④ 记住结果判据与过程判据的区分。

过渡： 最后两页，我们把"计划"这个东西重新认识一次。

【10.55】软件测试计划更是一个过程（2 分钟）

这一页只有一句话，但是本章最重要的一句观念转变：**计划不只是文档，而是一个过程，Planning。**

注意课件用英文动名词 Planning，而不是名词 Plan。名词是"那份文件"，动名词是"持续做计划这件事"。这一字之差，是两种完全不同的工作方式。

为什么计划必须是过程？三个理由。

第一，假设会变。 你计划时假设"需求两周内冻结"，第三周需求又变了；假设"环境下周就绪"，结果拖了一个月。假设一变，基于假设的计划就失效——**计划如果不变，说明它已经和现实脱节了**。

第二，认知会变。 测试做了一周，你对系统的理解远超第一周，风险排序会变，优先级会变，策略也该跟着变。

第三，计划本身是沟通工具。 每次更新计划，都是一次重新对齐：谁负责什么、什么时候交付、风险变没变。这个过程的价值，往往大于文档本身。

所以正确的做法是**滚动维护**：固定节奏（比如每周）更新一次，每次更新三件事——进度实际值、风险变化、下一步安排。别搞"一版到底"，那只是把计划变成历史文件。

还有一个配套动作课件在本章小结里点了名：**计划评审**。评审不是走过场，它是让计划"被承诺"的仪式：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，只是测试组的一厢情愿。**

这一页也是本章的思想落点：计划是对未来的承诺，而承诺需要不断被检验和修正。

板书： 中间写"Planning ＞ Plan"，左侧"为什么是过程"：假设会变｜认知会变｜计划是沟通工具。右侧"怎么做"：固定节奏滚动更新（进度实际值・风险变化・下一步）＋计划评审（让计划被承诺）。底部："没评审的计划＝一厢情愿"。

提问： 一份三个月没更新过的测试计划，能说明什么？

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"说明计划已经没人用了，大家在实际靠口头协调"——最可能的真相。

- 答"说明它已经变成一份归档文件，只在对审计时有用"——也很真实，很多项目的计划确实只活到评审那天。

- 追问收口：所以判断一个团队测试管理水平，最快的办法是问"你们的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 高频考点：滚动维护的三项更新内容；③ 判断陷阱："计划一旦批准就不能改"——错，变更要走流程但必须改；④ 记住计划评审的作用：把计划变成共同承诺。

【思政落点 4】：计划评审这件事，说到底是在练一种品质：**对自己许下的承诺负责**。写"10 天完成"很容易，难的是到期时你敢把实际进度摆出来说"我差了两天，原因是环境延期，我的补救方案是这样"。工程里最伤人的不是延期，是**隐瞒延期**——等到最后一刻才说"做不完"，所有人的排期一起崩。所以评审不是挑毛病，是让承诺公开、可检验、可修正。**敢承诺、敢报实情、敢改计划，这三件事合起来才叫靠谱。**

过渡：最后用一页把整章收起来。

【10.56】本章小结（2 分钟）

本章小结，五条，我逐条讲，并告诉你它对应前面哪一页。

第一，**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，而不是抄需求。

第二，**清楚测试上下文（目标、条件、环境等）**。这是 10.1 的内容：目标要可验证，准入要明确。为什么强调"上下文"？因为同一个测试项在不同上下文下的测法完全不同。

第三，**测试覆盖率是起点也是终点**。这句话最值得琢磨。"起点"是指分析阶段就要定覆盖口径——按需求覆盖还是按风险覆盖；"终点"是指评估阶段要用它判断够不够。**如果起点没定口径，终点就没法判断**，这是很多项目的通病。

第四，**风险识别、分析与防范**。这是 10.4 的三环节。记住风险不能消灭，只能降低或缓解。

第五，**有效制定测试计划，计划评审也不可忽视**。这是 10.5：计划是过程，评审是让它变成承诺的动作。

五条里，我认为第三和第五条最能体现专业度。覆盖率是**技术上的严谨**，评审是**管理上的严谨**，两者缺一，测试就只是"做了很多事"，而不是"能对结果负责"。

再给你一句话把整章串起来：**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。** 这句话能背下来，这一章就没白学。

再补一个复习方法：把这五条还原成五张表——测试项表、需求映射矩阵、WBS 估算表、风险清单、计划正文。表能画出来，说明你真会了；只能背文字，题目一变形就懵。

> **板书：**五条竖排：① 需求→测试需求逐层分析（业务・功能・用例・场景）② 清楚测试上下文 ③ 覆盖率是起点也是终点 ④ 风险识别・分析・防范 ⑤ 有效计划＋评审。底部一句串讲："分析定测什么，估算定花多少，风险定先测谁，策略定怎么测，计划写成承诺"。

提问："覆盖率是起点也是终点"，如果起点没定口径，会有什么后果？

预设回答：

- 答"测完不知道算不算测完"——对，这就是最直接的后果：没有口径就没有判据。

- 答"各方对'覆盖够了'的理解不一致，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，谁也说服不了谁。

- 答"最后只能靠感觉或靠时间到了来结束"——这是最常见的结局，也是最不专业的结局。

- 追问收口：所以覆盖口径要在**计划评审时就写死**，这比事后争论便宜一百倍。

易错点 / 考点：① 五条小结内容，简答高频；② 高频考点："覆盖率是起点也是终点"的含义；③ 判断陷阱："计划评审是形式"——错；④ 能把整章五节串成一句话，是案例题拿高分的关键。

过渡：理论讲完，下面这个实验就是把本章所有内容走一遍。

【10.57】实验 7：制定测试计划（2 分钟）

最后一个内容页是实验 7：制定测试计划。这是本章的落地作业，我把七个步骤讲清楚，也讲清每一步的易错处。

第一步，小组讨论，分析 SUT，重新明确测试目标，包括功能＋非功能性要求。 注意"重新明确"四个字——不是抄项目简介，是把目标写成可验证的句子。易错处：只写功能目标，忘了非功能。

第二步，分析哪些要测试、哪些可以不测，即确定测试的范围。 易错处：只写"要测的"，不写"不测的"和"为什么"。这一步的产出应该是双向清单。

第三步，对测试范围进行分解，列出测试项（包括功能测试项和专项测试项）。 易错处：按界面菜单切，而不是按业务能力切；专项测试项（性能、安全、兼容等）经常整块漏掉。

第四步，用 WBS 或敏捷中扑克牌估算法等来估算每一个测试项的工作量，并进行汇总。 易错处：只估执行不估准备；汇总后不和总工期对照。扑克牌估算法的好处是快速收敛、暴露分歧——两个人估同一个任务差了五倍，那个任务一定有没想清楚的地方。

第五步，针对这些测试内容，识别出其中的测试风险，并分析和列出主要的测试风险。 易错处：风险写成"可能存在进度风险"这类空话。要写成"原因→后果"。

第六步，一起讨论，看能否针对这些测试风险，找到相应的对策（策略）。 注意这一步的产出是"对策"，不是"注意事项"。

第七步，依据 10.5.1 所需的主要内容，将前面的内容汇总，形成一个简要的测试计划。 易错处：把前六步的内容直接粘贴，不做取舍和摘要——记住计划的第一页要是摘要。

我给一个交付要求：**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——它逼你做取舍，而取舍正是本章要教的核心能力。

板书：七步竖排：① 明确目标（功能＋非功能，可验证）② 定范围（含不测项与理由）③ 分解出测试项（功能＋专项）④ 估算（WBS／扑克牌，含准备成本）⑤ 识别风险（原因→后果）⑥ 找对策（策略）⑦ 汇总成计划（先写摘要）。底部交付要求："≤5 页＋三张表"。

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，但不是质量提升的原因。

- 答"因为它让分歧暴露出来：两个人估差五倍，说明有人没想清楚"——准确。分歧本身就是信息。

- 答"因为它避免了锚定效应：大家先独立出牌，不会被第一个说话的人带走"——很专业，这正是它相对"开会讨论出个数"的优势。

- 追问收口：所以估算的价值一半在数字，一半在**通过分歧发现理解偏差**。

易错点 / 考点： ① 七个步骤的顺序与要点；② 高频考点：估算要包含准备成本；③ 判断陷阱："实验只要交一份计划文档"——错，还要有可追溯的表；④ 记住"不测项＋理由"是必写内容。

过渡： 这一章就到这里。下一章我们把测试项变成一条条可执行的用例。

【10.58】感谢聆听（1 分钟）

第 10 章到这里结束。

回头看一眼这一章给了你什么：一套**把模糊需求变成可执行承诺**的方法。目标要可验证，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。

这一章没有教你怎么点界面、怎么跑工具，它教的是**在动手之前把事想清楚**。这部分工作最不出彩——没人会因为你写了一份好计划给你鼓掌；但它决定了后面所有工作是不是白干。

下一章第 11 章，我们把这一章的产物接过去：测试项变成测试用例，用例变成可维护的资产。

作业提醒：实验 7 的五页计划，按七步走，别忘了"不测项清单"和"风险的原因→后果"写法。下课。

板书： 一行字："第 10 章＝把模糊需求变成可执行承诺"。下方四组关键词：可验证（目标）｜有边界（范围）｜有依据（优先级・估算・风险）｜可维护（评审・滚动）。右侧："下一章：测试项 → 测试用例"。

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"——很好，抓住了 10.5 的核心。

- 答"风险不能消灭，只能管理"——也很好，抓住了 10.4。

- 答"没有判据的目标都是口号"——最锋利的一句，抓住了 10.1 到 10.2 的主线。

- 收口：这三句都对，因为它们其实是同一件事的三个侧面：**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（分析）＞ 10.5（计划内容与出入标准）＞ 10.4（风险与策略）＞ 10.3（估算）；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 4 处）

序号	所在页	落点主题
1	10.10	这一页最值得记的不是四个追问，而是它背后的态度：目标必须可验证。可验证意味着什么
2	10.11	准入和退出准则，本质上是规则意识。工程里最难的从来不是定规则，而是在压力下还守规
3	10.19	这一页的关键词是"干系人"。工程里很多事故不是技术不行，是该问的人没问：需求方以
4	10.55	计划评审这件事，说到底是在练一种品质：对自己许下的承诺负责。写"10 天完成"很容