《软件测试课程设计》讲稿（第 13 章）

第 15 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 13 章　测试执行、缺陷报告与跟踪

第 15 周 · 2 学时（90 分钟）

《软件测试课程设计》第 13 章 讲稿（第 15 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 13 章 测试执行、缺陷报告与跟踪 —— 缺陷写不清楚，等于没发现
>
> 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
>
> 配套课件：逐页还原版 **68 页**（`lecture13/`，云端 http://111.231.0.226:2345/lecture13/index.html）
>
> 教案对应：第 15 次课「测试执行、缺陷报告与跟踪」
>
> 本讲重点：测试执行的组织与监控；缺陷报告七要素与好标题写法；缺陷生命周期与状态流转；严重程度与优先级；缺陷跟踪与争议处理；缺陷度量与分析；回归与关闭准则；从缺陷反推预防
>
> 本讲难点：严重程度与优先级的区别；"不可复现"缺陷怎么处理；度量为什么不能用来追责
>
> **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 能说出缺陷报告的七个要素；② 能区分严重程度与优先级并各举两例；③ 能画出缺陷生命周期并说明各状态含义；④ 能说出四项缺陷度量指标及用途；⑤ 能说明关闭缺陷的准则
能力	① 能把一条口头抱怨改写成一条合格的缺陷报告；② 能对"不可复现"缺陷给出规范处理流程；③ 能读一份缺陷趋势图并说出结论与建议
素质（思政）	① 如实描述＝职业诚信；② 客观判断严重度与优先级；③ 争议对事不对人；④ 度量用于改进不用于追责；⑤ 从缺陷反推过程改进＝复盘与担当

本讲要建立的一个总印象：**一条缺陷报告的质量，决定它能不能被修好**。写不清环境、步骤、预期与实际，开发只能猜——而"猜"是效率最低的协作方式。

二、时间分配总表

> **容量提示**：本章课件 **68 页**，讲稿为素材全量版（口播约 3.1 万字，≈130 分钟），课时 90 分钟。请按下表取用：
>
> 1. **慢讲**：执行组织、缺陷七要素、生命周期、严重度与优先级、度量、关闭准则。
> 2. **快带**：工具截图页与示例罗列页（★）。
> 3. **当堂动手**：现场改写一条缺陷报告；给三个缺陷定级。

时段	页码	内容	分钟	备注
第 1 节	13.1–13.20	测试执行的组织、准备、监控	24	执行准备必讲
第 1 节	13.21–13.40	缺陷报告要素、好标题、可复现性、示例对照	26	**七要素与标题必讲**

——	——	**课间休息 10 分钟**	——	——
第 2 节	13.41–13.56	生命周期、严重度与优先级、跟踪与争议	22	**生命周期必讲**
第 2 节	13.57–13.68	度量与分析、回归与关闭准则、预防、小结与作业	18	**度量用途必讲**

> **压缩优先级**：先压缩工具截图与长篇示例 → 再压缩单一状态的细节 → 最后才压缩七要素、生命周期、严重度优先级、度量、关闭准则。

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（13.1–13.68）：http://111.231.0.226:2345/lecture13/index.html
>
> **口径提醒**：本章的"关闭准则"与第 4 章"退出准则"、第 10 章"测试准则"是同一套判据思想，讲课时可明确呼应。

【13.1】软件测试方法和技术（1 分钟）

各位同学，我们进入第 13 章——**测试执行、缺陷报告与跟踪**。

到这一章，前面十二章的准备工作全部要落地了。第 3 章教你怎么设计用例，第 4 章教你排测试流程、定退出准则，第 11 章反复强调用例要能追溯，第 12 章把环境、容器、自动部署这些基础设施搭起来。可这些说到底都还是"准备"。

真正让一个项目出问题的地方，常常在准备之后。环境没核对就开跑，跑出来的结果没人敢信；跑出来的问题没人写清楚，开发看一眼就退回来。这三件事，恰恰是课本上讲得最少、岗位上用得最多的。

这一章就解决它们：**怎么把测试跑得有序**，包括执行前的检查、冒烟、执行顺序和进度监控；**怎么把发现的问题写成开发能直接动手的信息**，也就是缺陷报告；**怎么让每一个缺陷都有始有终**，包括生命周期、状态流转、跟踪工具、缺陷分析与度量。

一句话记住这一章的定位：它是测试工作的**显影液**——你这个团队测试做得怎么样，不看你说得多辛苦，看

> 你交出去的缺陷单和报告长什么样。

> **板书：**中间写"第 13 章 测试执行・缺陷报告・跟踪"；左侧写"前面：准备（用例／流程／追溯／基础设施）"；右侧写"本章：跑起来 → 写清楚 → 追到底 → 算明白"。

> **提问：**你交上去一份缺陷单，开发第一句话最常说的是什么？

> **预设回答：**

- 答"我这儿复现不了"——最常见，说明步骤写得不够精确，或者环境信息没给。

- 答"这不是 Bug，是需求就这样"——第二常见，说明期望结果的依据没写清。

- 答"知道了，等这个版本完了再看"——说明严重性和优先级没谈拢，或者压根没定标准。

- 收口：这三句话，正好对应这一章的三块内容：报告的可复现性、严重性优先级的判定、缺陷的跟踪与决策。

> **易错点 / 考点：**① 本章四块内容的顺序与名称要能默写：测试执行与跟踪 → 缺陷的描述 → 缺陷跟踪和分析 → 产品质量评估与度量 → 测试的评估与报告；② 常考判断："缺陷报告是给开发看的"——不完整，它同时给管理者（判断严重程度与排期）、给后来的接手人（三个月后自己还看得懂）；③ 别把"缺陷报告"和"测试报告"混了，前者一条缺陷一张单，后者一个阶段/一个版本一份总账。

> **过渡：**先把上一章那笔账收一下——基础设施搭好了，为什么这一章还要先讲"执行前的检查"。

【13.2】第 12 章回顾（1 分钟）

> 这一页把第 12 章的四块内容收一收，因为它直接决定这一章能不能开跑。
>
> 第 12 章讲的是**测试基础设施**。第一，它的**重要性**：很多所谓的"缺陷"，追到最后是环境不一致造成的——你机器上是好的，测试环境是坏的，生产上又是另一个样。环境不统一，缺陷的真假都判断不了。第二，它的**要素**：硬件、网络、操作系统、中间件、数据库、测试数据、工具链，缺一样就不算完整。第三，**虚拟机、容器及其集群管理**：容器解决的是"同一份镜像，到哪都跑出一样的结果"，集群管理解决的是"要多少套环境就能拉起多少套"。第四，**自动部署**：环境不该靠人手搭，靠脚本、靠流水线，一条命令还原到已知状态。
>
> 这四条合起来只有一个目的：**让"结果"可信**。因为测试执行最怕的不是跑得慢，是跑出来的结果没法复现、没法解释。所以下一页讲执行要点时，第一条实践就是"严格审查测试环境"——那不是形式主义，那是在保护你后面所有数据的可信度。

最后补一句方法上的话：第 12 章讲的是"把环境做对"，这一章讲的是"在环境里把事做对"。前者是基础设施，后者是执行纪律。**没有可信的环境，执行纪律无从谈起；没有执行纪律，再好的环境也只是摆设。**你现在应该能感到这两章是接着的：第 12 章把地基打完，第 13 章开始在地基上盖楼，盖得好不好，不看图纸画得多漂亮，看每一道工序有没有人签字。

板书：四行：基础设施的重要性（环境不统一 → 缺陷真假难辨）→ 要素（硬件・网络・OS・中间件・DB・数据・工具）→ 虚拟机／容器／集群（一致性・可复制）→ 自动部署（一条命令还原）。底部一句："环境可信 → 结果可信"。

提问：如果开发说"我本地是好的"，你手上最能压住这句话的证据是什么？

预设回答：

- 答"截图给他看"——截图只能证明现象，证明不了环境差异。
- 答"把我的环境信息、版本号、构建包号一起给他"——这才是硬证据，因为可以对齐到同一个已知状态。
- 答"让他上我的机器看"——偶尔可行，但不是可复制的方法，人多就跑不通。
- 收口：环境信息的价值不在"证明我没错"，而在"让双方站到同一个坐标系里说话"。

易错点 / 考点：① 基础设施四块内容要能对上号；② 常考判断："容器化之后就不需要测试环境管理了"——错，容器保证的是运行环境一致，数据、配置、依赖服务版本仍然要管；③ 记住这条因果链：**环境不一致 → 缺陷不可复现 → 缺陷被退回 → 测试可信度下降**，它是后面"可复现性"那一节的理论根据。

过渡：环境这一层立住了，我们看这一章的整体结构。

【13.3】13.1 软件测试执行与跟踪（1 分钟）

这一页是本章的五块地图，我按顺序点一遍，你先把结构记住。

13.1 软件测试执行与跟踪——测试真正跑起来这一段怎么组织：执行过程的要点、进度的管理方法、过程管理工具。这是"动作层"。

13.2 软件缺陷的描述——发现的问题怎么写成一条合格的信息：生命周期、严重性和优先级、其它属性、完整信息、描述的基本要求、正反两个报告示例。这是"表达层"，也是本章分量最重、考试最常出题的一块。

13.3 软件缺陷跟踪和分析——一条缺陷从提交到关闭这一路怎么走，团队怎么处理争议，怎么用趋势和分布

看出项目处于什么阶段。这是"流程层"。

13.4 产品质量评估与度量——用缺陷数据反算质量：缺陷密度、清除率、逃逸率、种子公式、质量度量模型。这是"数据层"。

13.5 测试的评估与报告——收口：测试过程评估、充分性评估、测试报告怎么写。这是"结论层"。

五块连起来是一条完整的链子：**跑起来 → 写清楚 → 追到底 → 算明白 → 说得出结论**。你以后带项目，这五步一步都不能少。

> **板书：**竖排五块：13.1 执行与跟踪（跑起来）→ 13.2 缺陷描述（写清楚）→ 13.3 跟踪与分析（追到底）→ 13.4 质量度量（算明白）→ 13.5 评估与报告（说得出结论）。右侧一行："动作 → 表达 → 流程 → 数据 → 结论"。

提问：这五块里，哪一块最容易在真实项目里被跳过？

预设回答：

- 答"13.4 度量"——确实常被跳过，因为算起来麻烦，而且算出来不好看。

- 答"13.2 缺陷描述"——这个更常见。大家觉得"发现了就行"，随手写两句，结果返工的时间比写清楚的时间长得多。

- 答"13.5 报告"——也会有，做完了不总结，下一个项目从零开始。

- 收口：被跳过的那一块，往往就是下一次踩同一个坑的原因。

易错点 / 考点：① 五个小节编号与标题要能一一对应，选择题常拿"缺陷描述"和"缺陷跟踪"互换来设陷阱；② 记住本章的**主链**：执行 → 记录 → 跟踪 → 度量 → 报告；③ 13.4 的度量不是"额外工作"，它是 13.5 报告的数据来源，两者是因果不是并列。

过渡：从第一块开始——测试执行到底要抓哪几个要点。

【13.4】13.1 软件测试执行与跟踪（1 分钟）

这一页是 13.1 的分隔，我用一分钟把这一节的路线画出来。

这一节分三段。**第一段 13.1.1 测试执行过程的要点**：先讲清不同测试阶段的执行要点不一样，再讲用例怎么

执行、团队怎么建设与沟通、执行结束要做什么。这一段解决的是"怎么把活干对"。

第二段 13.1.2 测试项目进度的管理方法：讲进度和质量、进度和成本这两组关系，再给两个能落地的监控工具——S 曲线法和 NOB 曲线法。这一段解决的是"怎么知道现在跑到哪了、还来不来得及"。

第三段 13.1.3 测试过程管理的工具：AgileTC、Jira、MeterSphere、PractiTest、TestLink 这一批，解决"这些事用什么装起来"。

顺序是有讲究的：**先定动作标准（要点），再定监控手段（进度），最后选工具承载（工具）**。反过来的团队很多——先买工具，再想流程，最后工具里跑的是糊涂账。

记住这一节的一句话：**测试执行不是"点用例"，是"在受控的环境、按优先级、带着监控地推进"**。

> **板书：**三段横排：要点（阶段差异・用例执行・团队沟通・执行结束）→ 进度（质量关系・成本关系・S 曲线・NOB 曲线）→ 工具（AgileTC・Jira・MeterSphere・PractiTest・TestLink）。底部："先定标准 → 再定监控 → 最后选工具"。

提问：为什么课件把"团队建设与沟通"放在测试执行的要点里，这不是人事的事吗？

预设回答：

- 答"因为测试要和开发打交道"——对。缺陷单是测试和开发之间最主要的沟通载体，沟通不畅，缺陷就会被反复退回。

- 答"跟技术没关系，可以略过"——恰恰相反，行业里大量"测试推不动"不是技术问题，是信息不对称和对抗情绪。

- 追问：你在实训里遇到过"提了缺陷没人理"吗？那通常不是缺陷不对，是**没有约定的处理节奏**。13.3 会讲怎么建这个节奏。

- 收口：测试执行一半是技术，一半是协作。

易错点 / 考点：① 13.1 三个子节的名称与顺序；② 常考判断："测试进度管理就是看还剩多少用例没执行"——不完整，还要看缺陷的打开/关闭趋势（NOB），因为用例跑完不等于质量达标；③ S 曲线和 NOB 曲线分别用什么数据、看什么结论，是本节最可能出简答的地方。

过渡：先看 13.1.1——不同阶段的执行要点到底差在哪。

【13.5】13.1.1 软件测试过程的要点（2 分钟）

这一页把测试执行拆成四件事，我们一件一件说。

第一件，**不同测试阶段的执行要点不一样**。单元测试阶段，执行的主体往往是开发自己，重点是覆盖分支、跑通断言，速度要快，最好每次提交都跑；集成测试阶段，重点是接口之间的契约和数据流，环境要能拼起来；系统测试阶段，才是测试团队的主场，要看业务流程、非功能指标；验收测试阶段，主角变成用户或业务方，判据是"能不能接受"，不是"符不符合设计"。同一套用例、同一批人，换个阶段判据就变了——这是很多同学最容易糊掉的地方。

第二件，**测试用例执行**。这里有两条纪律：一是**按用例跑，逐条记录结果**，通过、失败、阻塞、未执行，四种状态都要如实记；二是**偏离要留痕**——你临时改了步骤、换了数据、跳过了某一步，必须写下来。为什么？因为三天后你要判断"这个模块到底测没测透"，靠的就是这些记录，不是靠记忆。

第三件，**团队建设与沟通**。测试不是一个人的活。要有人对接开发、有人对接产品、有人管环境，信息要有一个共同的入口。课件在下一页给了一批具体做法，比如每周例会、和开发及时沟通。

第四件，**测试执行结束**。这不是"用例跑完就完了"。要确认退出准则是否满足——这是第 4 章讲过的概念；要处理遗留缺陷，哪些必须修、哪些可以带到下个版本、哪些接受风险；最后要出测试报告，把结论说清楚。

四件事合起来，就是"**开跑前有标准，跑的过程有记录，跑完有结论**"。

> **板书：**四行：① 阶段差异（单元：开发自测・覆盖率｜集成：接口契约・数据流｜系统：业务流程・非功能｜验收：用户判据）；② 用例执行（四种状态：通过／失败／阻塞／未执行 ＋ 偏离留痕）；③ 团队与沟通（共同入口・例会・对接人）；④ 执行结束（退出准则・遗留缺陷处理・测试报告）。

提问：用例执行记录里，"阻塞"和"未执行"有什么区别？为什么要分成两种状态？

预设回答：

- 答"没区别，都是没跑"——这是典型错答。**阻塞是"想跑但跑不了"**，比如前置功能没做完、环境挂了；**未执行是"还没轮到"**。前者是风险信号，要立刻上报；后者是进度问题。

- 答"阻塞要报，未执行不用报"——方向对了。把两者混在一起，管理者就分不清"进度慢"和"质量卡住"。

- 追问：那如果阻塞一直不解除呢？它就变成退出准则里必须写明的遗留风险。

- 收口：状态分得越细，后面的判断才越准。

易错点 / 考点： ① 四个要点的名称；② 用例执行的四种状态（通过/失败/阻塞/未执行）常考，注意"阻塞"不等于"失败"；③ 常考判断："用例执行率 100% 就可以结束测试"——错，还要看通过率、缺陷收敛、退出准则；④ 阶段的判据差异是案例分析题的高频点。

过渡： 要点讲完，看课件给的一批具体做法——这些是从项目里总结出来的经验条目。

【13.6】测试执行实践（2 分钟）

这一页是八条实战经验，我按"开跑前—跑的过程—出偏差怎么办"分三组讲。

开跑前两条。第一条，**执行前开一个动员会，严格审查测试环境**。动员会不是喊口号，是把范围、时间、责任人、判定标准当众对齐一遍；环境审查是逐项核对：版本对不对、数据是不是干净的、依赖服务起没起。这两件事花半天，能省掉后面一周的返工。

第二条，**抽查性质的探索式测试，验证高风险区域的测试质量**。注意"抽查"两个字：不是让你放弃用例乱点，而是在用例之外，专门去高风险的地方试——支付、权限、并发、边界。目的是检验"我们的用例是不是真把这个区域盖住了"。

跑的过程中三条。第三条，**交叉互换测试人员所测试的模块，可以发挥互补作用**。一个人测久了会有盲区，换个人来看，往往一眼就看出问题。这就是行业里说的"交叉测试"。

第四条，**良好的沟通，比如每周例会，以及和开发人员的及时沟通**。例会解决"节奏"，及时沟通解决"个案"。

第五条，**常规的缺陷审查，及时发现问题、纠正问题，使整个测试进程在控制轨道上发展**。缺陷审查审的是"缺陷单的质量"：步骤清不清楚、严重性判得对不对、有没有重复。审得勤，返工就少。

出偏差两条。第六条，**测试时间被压缩时，做测试策略的优化、计划调整，回到测试需求的优先级、调整测试范围**。注意这个动作顺序：不是"少测点"，而是"按优先级重新划范围"，先保高优先级需求。

第七条，**阶段性结果分析，保证阶段性测试任务得到完整的执行并达到预定的目标**。每个阶段收口时对一次账：该做的做了没有、做到什么程度、能不能进下一阶段。

第八条虽然课件没单列，但它藏在第六条里：压缩时间必须**书面记录放弃了什么**，否则到期没人说得清"这段到底测没测"。

板书： 三组。开跑前：动员会＋环境审查｜探索式抽查高风险区。过程中：交叉互换｜周例会＋及时沟通｜

> 缺陷审查。出偏差：时间压缩 → 调优先级 → 调范围（记录放弃项）｜阶段性结果分析。旁边写："先保高优先级，再谈砍范围"。

提问：项目只剩三天，负责人说"用例能跑多少跑多少"。你作为测试负责人，第一件事做什么？

预设回答：

- 答"加班跑，能跑多少算多少"——态度可取，但没解决"跑哪些"的问题，很可能把时间花在低风险模块上。

- 答"按需求优先级重排用例，先跑核心业务流程和高风险模块"——正确做法，这就是课件第六条讲的"回到测试需求的优先级、调整测试范围"。

- 答"直接跟老板说做不完"——说结论没错，但要带着方案说：给出"三天内可保证的范围 + 放弃的范围 + 相应风险"。

- 收口：压缩时间时，测试负责人的价值不在"多跑几条"，在"**把有限的时间放在最该测的地方，并且让所有人知道放弃了什么**"。

易错点 / 考点：① 八条实践要能说出至少五条；② 高频考点："时间被压缩时正确的应对顺序"——优化策略 → 调整计划 → 按需求优先级调范围，**不是**直接砍用例数量；③ 常考判断："探索式测试就是不写用例随便点"——错，它是有目标、有侧重、面向高风险区域的测试活动；④ "交叉测试"的目的答"互补、减少个人盲区"。

过渡：执行的动作标准有了，接下来要回答一个更硬的问题：现在到底跑到哪了？

【13.7】13.1.2 测试项目进度的管理方法（2 分钟）

这一页讲进度管理，课件给了两组关系：**进度与质量**、**进度与成本**。这两组关系想清楚了，你才知道为什么进度管理不能用"催"来解决。

先说**进度与质量**。测试时间是质量的函数，但**不是线性的**。多给一倍时间，不等于多找出一倍缺陷——因为容易发现的缺陷在前面就被发现了，越往后越是隐蔽的、需要构造复杂条件才能触发的。这就是缺陷发现的"长尾"。反过来说，压缩时间也不是"少发现一点"，而是**把缺陷推到后面去发现**：研发阶段省下的时间，会在生产环境上加倍还回来。所以"进度 vs 质量"本质上是"现在付还是以后付、付多少"的选择。

再说**进度与成本**。这里有个业内反复验证的现象：项目后期加人，往往更慢。为什么？新人要学习业务、要

熟悉代码、要占用老人的时间带他，沟通路径还随人数平方增长。所以进度落后时，第一反应不该是"加人"，而该是"缩范围"或"调优先级"。

那怎么管？靠数据，不靠感觉。课件给了两个工具：**S 曲线法**看"进度是不是按计划走"，**NOB 曲线法**看"缺陷是不是在收敛"。一个管"做了多少"，一个管"还剩多少问题"，两个一起看，才敢判断"能不能按期交付"。

一句话：**进度管理的核心不是催进度，是尽早看见偏差，并且有权做取舍。**

补一句实操提醒：这两组关系要在测试计划里就写成"假设与约束"，而不是等进度落后了才拿出来讨论。事先写明白"如果时间压缩到某个点，我们放弃哪些范围、由谁批准"，比事后争论有用得多。

板书：两列。左"进度 vs 质量"：时间→缺陷发现非线性；压缩时间＝把缺陷推到生产。右"进度 vs 成本"：后期加人更慢（学习成本＋沟通路径）。底部："管进度靠数据：S 曲线（做了多少）＋ NOB 曲线（还剩多少）"。

提问：项目进度落后两周，老板说"我给你加两个人"。你怎么回应？

预设回答：

- 答"太好了，人多力量大"——典型的直觉错答。测试工作有大量"必须先懂业务才能测"的门槛，新人短期产出极低。

- 答"加人可以，但请给我两周让他们熟悉，实际缓解要到下个阶段"——比较专业的回答，把期望值拉回现实。

- 答"不如砍范围，先保核心流程"——这就是课件第六条讲的取舍思路，通常比加人更有效。

- 收口：进度落后时，**调整范围**往往比**增加资源**更快见效，因为范围是你自己能决定的。

易错点 / 考点：① 两组关系的名称；② 常考判断："测试时间越长，发现的缺陷一定按比例增加"——错，边际递减；③ 常考简答："为什么压缩测试时间会把成本推高"——缺陷发现越晚，修复成本越高（越靠后要改的东西越多）；④ 记住 S 曲线与 NOB 曲线分别回答"做了多少"和"还剩多少"。

过渡：先看第一个工具——S 曲线法，它是怎么画出来的。

【13.8】测试进度的 S 曲线法（2 分钟）

S 曲线法，名字来自曲线的形状——**两头平、中间陡，像个躺着的 S**。

它怎么用？课件说得很清楚：**通过对计划中、尝试的与实际的三者对比来实现**。也就是画三条线。第一条是**计划的进度线**：按计划，第几天应该完成多少。第二条是课件说的"尝试的"——可以理解为你**预期能达到的**那条线，也就是在资源、人力确定的条件下，团队自己估的可行线。第三条是**实际完成的**线。

三条线摆在一起，结论就出来了：实际线落在计划线下面，说明进度落后；实际线贴着"尝试线"走，说明团队已经按自己的能力上限在跑，落后不是态度问题，是计划本身不现实；实际线高于计划线，说明计划保守，可以考虑提前收口或者加测。

它的**基本数据是测试用例或测试点的数量**。为什么用"用例数/测试点数"而不是"发现的缺陷数"？因为缺陷数受软件质量影响——质量好缺陷就少，你不能说"缺陷少所以进度快"。而用例数、测试点数是**工作量单位**，它反映的是"干了多少活"，这才是进度。

为什么要用 S 形而不是直线？因为测试工作的节奏本来就这样：前期要读需求、搭环境、准备数据，产出很慢；中期跑起来以后速度最快；后期剩下的都是难啃的、要反复复现的，速度又降下来。所以**形状本身就是信息**：如果你的实际曲线该陡的时候不陡，说明卡在某个环节了。

用的时候记住一句：**S 曲线看的是"斜率"和"位置"**——斜率告诉你速度，位置告诉你差距。

> **板书：**画三条 S 形曲线：计划（虚线）／尝试（点线）／实际（实线）。横轴"时间"，纵轴"累计执行的用例数（或测试点数）"。旁注："斜率＝速度；位置＝差距；贴尝试线＝计划不现实"。

> **提问：**如果实际曲线一直贴在"尝试线"上、但明显低于"计划线"，问题出在谁身上？

> **预设回答：**

- 答"测试团队不努力"——错。"尝试线"本身就是团队按自己能力估的可行线，贴着它跑说明已经到上限了。

- 答"计划定得不现实"——正确。这时候该做的是改计划、调范围，而不是继续压团队。

- 答"可能是阻塞太多"——补得好，如果曲线斜率突然变平，要去看是不是环境、前置功能卡住了。

- 收口：S 曲线最大的价值，是**把"谁的责任"这个问题变成"哪条线错了"这个问题**。

> **易错点 / 考点：**① 三条线的名称（计划／尝试／实际）与含义；② 基本数据是**测试用例或测试点的数量**，这是选择题高频点，别答成缺陷数；③ 常考判断："S 曲线是直线上升的"——错，S 形反映测试工作的三个阶段节奏；④ 简答要能说出"斜率看速度、位置看差距"。

> **过渡：**S 曲线管"做了多少"，可"做得对不对、问题还剩多少"它管不了，这就要看 NOB 曲线。

【13.9】测试进度的 NOB 曲线法（2 分钟）

> NOB，**Number of Open Bug**，直译就是"打开状态的缺陷数"。这条曲线画的不是"干了多少活"，而是"**还欠多少账**"。
>
> 它的形状通常是这样：项目前期，测试刚开始，缺陷一个接一个被提出来，而开发还在写别的功能、来不及修，所以**打开的缺陷数快速上升**；到了中后期，开发开始集中修缺陷，修复速度超过新发现速度，曲线**掉头向下**；理想情况下，发布之前它会降到很低，甚至接近零。这就是"收敛"。
>
> 那这条曲线要看什么？看三件事。
>
> 第一，**是不是在下降**。如果临近发布，NOB 还在往上走或者横着不动，说明修复跟不上发现，这个版本不能发。这是最硬的一条红线。
>
> 第二，**下降的速度**。降得快，说明修复有效；降得慢甚至反弹，要看是不是修复引入新问题——也就是"改一个坏两个"，后面 13.3.2 的累积趋势分析会专门讲这个"反弹"。
>
> 第三，**和 S 曲线对着看**。用例执行率上去了、NOB 也降下来了，两个都好看，才能说"这个版本差不多了"。只有执行率上去、NOB 不降，那是"测出来了但修不动"；NOB 降了但执行率没上去，那是"测得不够"。
>
> 提醒一句：NOB 只看"打开"的，所以它天然受"状态管理"影响——如果团队习惯把没修的缺陷随手标成"延期"或"无法解决"，NOB 就会假性下降。这就是为什么 13.2 要严抓状态定义：**你统计的口径，决定了你看到的真相**。
>
> 再提醒：NOB 曲线要**按严重性分层**看。全部缺陷的 NOB 在下降，但"严重及以上"的 NOB 没降，那这个版本的真实风险其实没降——分层看，才不会被总数骗过去。

> **板书：**一条先升后降的曲线，纵轴"打开缺陷数（NOB）"，横轴"时间"。标三个阶段：上升（发现＞修复）→ 拐点（修复赶上）→ 下降（收敛）。旁注："发布前 NOB 不降＝不能发"；"状态乱标＝NOB 假降"。

> **提问：**临近发布，NOB 从 80 降到 10，是不是可以放心发了？

> **预设回答：**

- 答"可以，降得这么快"——要警惕。得先问：这 70 个是怎么没的？修完了、验证通过了，才算真降；被标成"延期""无法解决""不是 Bug"，就是假降。

- 答"还要看剩下 10 个的严重性"——很专业。10 个微小的和 10 个致命的，结论完全不同。

- 答"还要看新发现的速率"——对，如果每天还在新增 20 个，那只是"提得慢"而不是"修得好"。

- 收口：**NOB 是结果指标，必须用状态数据去校验它**。

> **易错点 / 考点：**① NOB 全称 Number of Open Bug，只统计"打开"状态的缺陷；② 典型形状是"先升后降"，拐点意味着修复速度追上了发现速度；③ 常考判断："NOB 归零说明没有缺陷了"——错，只说明没有处于打开状态的缺陷，可能被延期、被拒绝或未被发现；④ 与 S 曲线配合使用是简答的得分点。

> **过渡：**两个监控工具都会看了，接下来看用什么系统把这些数据装起来。

【13.10】13.1.3 测试过程管理的工具（2 分钟）

> 这一页列了五个工具，我不要求你背界面，要求你**记住每个工具的定位**，因为选型考的就是定位。
>
> 第一个，**AgileTC**，滴滴开源的一套**敏捷的测试用例管理平台**。关键词是"用例管理"和"敏捷"——它面向的是用例的组织、评审、执行跟踪，适合用例驱动、迭代节奏快的团队。
>
> 第二个，**Jira**，Atlassian 公司的项目管理工具，课件明确说它**常常用于缺陷管理**。注意这个措辞："常常用于"，说明它不是专门的缺陷管理系统，而是通过工作流配置把缺陷管起来。它的强项是自定义工作流和看板，弱项是要配。
>
> 第三个，**MeterSphere**，一站式开源持续测试平台，**涵盖测试管理、接口测试、性能测试、团队协作**。关键词"一站式"：用例、接口、性能、报告在一个平台里，适合想把工具链收敛成一个的团队。后面的实验 10 就是基于它做的综合实验。
>
> 第四个，**PractiTest**，端到端的测试管理系统，**提供测试用例管理、缺陷状态管理，具有可定制的仪表板，并附有详细报告**。关键词"可定制仪表板 + 详细报告"，它偏管理和汇报。
>
> 第五个，**TestLink**，开源的测试过程管理工具，用于**项目管理、缺陷跟踪和测试用例管理**。它是老牌工具，教学里常用，因为部署简单、概念清晰，实验 9 用的是 MantisBT，两者常配套。
>
> 课件还注明"更多工具详见公众号文章"，意思是这个领域工具很多，别死记名单。选型的时候按四个问题问自己：**要不要管接口和性能（要 → MeterSphere 这类）？要不要强定制工作流（要 → Jira）？要不要轻量开源（要 → TestLink）？要不要给管理层出报告（要 → PractiTest 这类）？**

> **板书：**五行对照：AgileTC＝敏捷用例管理（滴滴开源）｜Jira＝项目管理，常作缺陷管理（Atlassian）｜

> MeterSphere＝一站式持续测试（管理＋接口＋性能＋协作）｜PractiTest＝端到端，可定制仪表板＋详细报告｜TestLink＝开源过程管理（项目＋缺陷跟踪＋用例）。底部："选型四问：接口性能？工作流定制？轻量开源？管理报告？"

提问：一个十人的小团队，只想把用例和缺陷管起来，预算为零，你推荐哪个组合？

预设回答：

- 答"Jira + TestLink"——可行，TestLink 管用例，Jira 管缺陷，是经典组合，代价是要维护两套系统的账号和关联。

- 答"MeterSphere"——也可以，一站式省心，功能更多但部署要求也更高。

- 答"用 Excel 就行"——短期能跑，一旦要追溯"哪条需求被哪些用例覆盖、哪些缺陷影响了哪些用例"，Excel 就撑不住了。第 11 章讲的追溯，靠手工表很难长期维持。

- 收口：小团队选型的第一原则是**能坚持用下去**，不是功能最全。

易错点 / 考点：① 五个工具与定位的对应关系，尤其 Jira＝缺陷管理、TestLink＝用例＋缺陷跟踪、MeterSphere＝一站式；② 常考判断："Jira 是专门的缺陷跟踪系统"——它是通用项目管理工具，缺陷管理靠工作流配置实现；③ 选型题的答法：先给约束（规模、预算、要不要接口/性能），再给匹配，别直接报工具名。

过渡：工具只是容器。真正决定质量的，是你在里面写下的那一条条缺陷——这就要进入本章最重的一块：13.2 缺陷的描述。

【13.11】13.2 软件缺陷的描述（1 分钟）

这一页是 13.2 的分隔，也是本章最该花时间的一块。

它分六小节，我按逻辑给你串一下，你就明白为什么是这个顺序。**13.2.1 缺陷的生命周期**——先看一条缺陷一辈子要经过哪些状态，你才知道一张单子该填哪些字段。**13.2.2 严重性和优先级**——两个最容易被混为一谈、也最容易吵架的属性。**13.2.3 缺陷的其它属性**——ID、类型、频率、来源、原因这些分类维度。**13.2.4 完整的缺陷信息**——一张合格的单子到底要有哪些内容。**13.2.5 缺陷描述的基本要求**——六条写作纪律。**13.2.6 缺陷报告示例**——一好一坏两个真实例子对照着看。

为什么这一块值得单独成节？因为**缺陷报告是测试人员最重要的产出物**。你写了多少用例、加了多少班，别人看不见；你交出去的缺陷单，是唯一能被反复阅读东西。一份写得好的缺陷单，开发十分钟能定位；写得差，来回三天还在问"你到底在说什么"。

有句话我说在前面：**缺陷报告写的不是"我发现了什么"，写的是"你要怎么才能看到我看到的东西"**。带着这个视角，这一节每一条要求你都会觉得理所当然。

板书：六小节竖排：生命周期 → 严重性与优先级 → 其它属性 → 完整信息 → 基本要求 → 示例（好/坏）。
右侧一句："缺陷报告＝让别人看到你看到的东西"。

提问：一条缺陷单，你觉得最该写清的是哪一项？

预设回答：

- 答"重现步骤"——最常见也最正确的答案之一，因为它决定开发能不能复现。

- 答"严重程度"——重要，但它是判断，不是事实；没有步骤支撑的严重程度只是个人意见。

- 答"标题"——也很关键，标题是唯一会被列表页展示的东西，标题烂，单子就容易被忽略。

- 收口：这几项其实构成一条链：**标题决定要不要点开，步骤决定能不能复现，期望与实际决定算不算缺陷，严重性优先级决定什么时候修**。

易错点 / 考点：① 六小节名称与顺序；② 常考判断："缺陷描述的核心是把现象说清楚"——不完整，核心是**让别人能复现**；③ 记住"缺陷报告是测试的主要交付物"这个定位，简答常考。

过渡：先想一个问题——我们关心一条缺陷的哪些方面？课件用四个问句把这六小节串起来了。

【13.12】对缺陷会关心哪些问题？（2 分钟）

这一页只有四个问句，但它其实是整个 13.2 的纲。我们一条一条看，每条背后都对应一类信息。

第一个问题，**描述？**——这条缺陷到底是个什么事。对应的是标题、步骤、期望结果、实际结果。它回答"是什么"。

第二个问题，**是否严重？**——对应严重性和优先级。它回答"多要紧"。注意这里课件把"是否严重"放在第二位，说明在真实项目里，**能不能复现是技术问题，多严重是决策问题**，两个都要，但顺序上先弄清事实再谈判断。

第三个问题，**是否需要修正？**——这个问题很关键，因为它的答案可以是"不修"。有些缺陷是设计如此，有些是第三方组件的问题，有些修的成本远高于影响。所以缺陷的终态不只有"关闭"，还有"延期""无法解决""不是 Bug"。这一点在下一页的生命周期图里会看到。

第四个问题，**当前状态？**——对应生命周期。它回答"这事进行到哪了"。没有状态，就没有跟踪；没有跟踪，就没有进度判断。

你把这四个问题记牢，以后拿到任何一条缺陷，都可以用它自查：**说清了吗？多要紧？要不要修？到哪一步了？**四个问题有一个答不上来，这条单子就不合格。

补一句：这四个问题其实是给**三类读者**准备的。开发只关心第一个（怎么复现）；测试组长和项目经理关心第二个和第三个（排期与取舍）；三个月后的你自己关心第四个，也关心第一个——因为那时候你已经忘了当时在干什么。

板书：四个问句竖排：描述？（是什么）｜是否严重？（多要紧）｜是否需要修正？（修不修）｜当前状态？（到哪了）。旁注三类读者：开发→描述；组长/PM→严重性・是否修；未来的你→状态・描述。

提问：为什么"是否需要修正"这一问的答案可以是"不需要"？

预设回答：

- 答"因为有些不是真的 Bug"——对，比如需求就是这么定的，或者用户操作本身违规。

- 答"因为修不起"——也是真实答案。影响极小、改动风险极大的缺陷，常常选择记录在案、不修。

- 答"发现了就必须修，不然提它干什么"——这是最典型的新手想法。缺陷记录的价值不只是"修"，还包括**留下证据、支撑决策、供以后复盘**。

- 收口：一条缺陷被标成"不修"，不是测试白干了，是**测试把风险显性化并交给了决策者**。

易错点 / 考点：① 四个问句要能背，它们是 13.2 六小节的纲；② 常考判断："所有缺陷都必须修复后才能发布"——错，允许延期、拒绝、无法解决，但必须有记录和决策；③ 简答常考"缺陷报告的读者有哪些"，答开发、管理者、后续维护者三类。

过渡：先解决第一个问题的底层——一条缺陷从被发现到关闭，要经过哪些状态。

【13.13】13.2.1 缺陷生命周期（2 分钟）

缺陷生命周期，讲的是**一条缺陷从生到死要经过哪些状态、每个状态下谁在干活**。这一页先给最简版，四个状态。

第一步，**发现—打开**：测试人员找到软件缺陷，并将缺陷提交给开发人员。注意"打开"这个动作的含义：它不是"我知道了"，而是正式把责任交接给开发——所以打开时必须带齐信息，否则开发接不住。

第二步，**打开—修复**：开发人员**再现、修复**缺陷，然后提交给测试人员验证。这里有两个动词要划重点。"再现"是前提：开发复现不出来，就没法确定改哪一行；"修复"之后要提交，说明修复必须落到可验证的构建上，不是"我改好了"一句话。

第三步，**修复—关闭**：测试人员**验证修复过的软件，关闭已不存在的缺陷**。注意这个措辞："关闭已不存在的缺陷"——关闭的依据是"缺陷现象不再存在"，而不是"开发说改好了"。

这四个状态构成了主干。但你一听就能发现，这个模型太理想了：万一开发复现不出来呢？万一验证没通过呢？万一这个版本不打算修呢？所以下一页会给出完整版，把分支补上。

现在先把主干记住，并且记住每个状态的**责任人**：发现和关闭是测试，打开和修复是开发。这条责任线的意义在于——**谁改状态，谁对这一步负责**。

再补一句：四个状态是一条最短闭环，但它有一个前提——**每一跳都要有明确的责任人和可验证的交付物**。发现交出去的是缺陷单，打开交出去的是复现确认，修复交出去的是构建包，关闭交出去的是验证结论。缺一样，这一跳就是空的。

> **板书：**四个方框横排：发现（测试）→ 打开（测试提交给开发）→ 修复（开发再现、修复、提交验证）→ 关闭（测试验证后关闭）。每框下标责任人。

提问：为什么"关闭"这一步必须由测试来做，不能让开发自己关？

预设回答：

- 答"因为要验证"——正确。开发验证的是"我改的代码逻辑对不对"，测试验证的是"缺陷现象是否消失、有没有引入新问题"。

- 答"谁提的谁关，公平"——这只是形式上的公平，理由是验证职责。

- 答"开发自己关更快"——快，但会把"没真修好"的缺陷藏起来，最后在生产上暴露。

- 收口：状态流转的本质是**责任交接**，交接点必须有独立验证。

> **易错点 / 考点：** ① 四个状态的名称与顺序；② 每个状态的**责任人**是高频考点：发现和关闭归测试，打开和修复归开发；③ 常考判断："修复即关闭"——错，中间必须经过验证；④ 名词辨析："打开"是提交动作，不是"开启排查"。

> **过渡：** 把分支补上，就是完整的缺陷生命周期。

【13.14】完整的缺陷生命周期（3 分钟）

这一页是完整版，比刚才多了分支和回路。我按"主干 + 四个岔路 + 两条回路"讲，你跟着走一遍就清楚了。

主干是这样：**创建**之后进入**激活状态**，系统发邮件通知开发。开发收到后先判断一件事：**这个缺陷是否清楚、可再现？**如果清楚可再现，进入**已处理状态**——开发开始查、开始改，课件在旁边标了 `Unit test, code review`、`Check in CVS` 这些动作，意思是修复不是随手改一行，要走单元测试和代码评审，然后入库。改完之后进入**已修正状态**，系统发邮件通知测试。测试做**验证**：通过就进入**关闭状态**；不通过就打回，重新激活。

现在看四个岔路，都从"是否清楚、可再现"这个判断点分出去。

第一，**不能再现 / 缺少信息**——开发复现不出来，或者信息不全。这时候不是简单退回，而是要跟测试确认；如果确认后仍无法复现，可能进入**缺陷评审**。

第二，**缺陷评审**。当双方对"这是不是缺陷"有分歧时，交给评审。评审的出口有几个：**不是 Bug**（关掉）、**增强设计**（其实是需求改进，转成需求或改进项）、**延期**（确实是缺陷，但这个版本不修，标"下一个版本"）、**无法解决**（技术上做不到或成本不可接受）。

第三，**需要处理**——评审结论是"要修"，回到已处理状态。

第四，**延期**这条路特别要说明：课件标注了"下一个版本"。意思是延期的缺陷必须挂到具体的版本号或日期上，不能就这么飘着，否则它会永远消失。

两条回路：一条是"验证不通过 → 重新激活"；另一条是"延期 → 下个版本 → 重新打开"。这两条回路是缺陷管理最容易出问题的地方——**回路没有记录，缺陷就变成了黑洞**。

一句话总结这一页：**四个主干状态负责流转，四个岔路负责定性，两条回路负责兜底。**

> **板书：**主干：创建 → 激活（邮件通知 DEV）→ 判断"清楚、可再现？"→ 已处理（Unit test・Code review・Check in）→ 已修正（邮件通知 QA）→ 验证 → 关闭。岔路：不能再现/缺信息 → 缺陷评审 → {不是 Bug｜增强设计｜延期（下个版本）｜无法解决｜需要处理}。回路：验证不通过 → 重新激活；延期 → 下个版本 → 重开。

> **提问：**开发把一条缺陷标成"无法解决"，测试该接受吗？

> **预设回答：**

- 答"接受吧，他说解决不了"——太软。"无法解决"是个强结论，必须给出理由：技术不可行、依赖第三方、成本远超收益。

- 答"不接受，必须修"——太硬。有些缺陷确实是第三方组件或平台限制，硬顶没有意义。

- 答"要求写明原因，并评估影响和替代方案，再由评审确认"——正确做法。这个状态可以存在，但不能是"一句话就关闭"。

- 收口：**分支状态不是逃生门，是要付出说明成本的**。

> **易错点 / 考点：**① 主干四个状态与岔路四个出口（不是 Bug／增强设计／延期／无法解决）；② 邮件通知的两个节点：激活通知开发、已修正通知测试；③ 常考判断："延期的缺陷不需要再跟踪"——错，必须挂版本号或日期，下个版本要重新打开；④ 简答高频："缺陷评审的出口有哪些"。

> **过渡：**这套流程放到真实项目里是什么样？课件给了一个 14 步的实例。

【13.15】实例（2 分钟）

> 这一页把上一页的流程图落到一条时间线上，**14 个步骤，从 open a bug 到 change bug status to close**。我按三段念，你感受一下一条缺陷要走多少道手续。
>
> 第一段，**提出与受理**：`Open a bug`——测试提交；`Dev checks mail & Review bug`——开发收邮件、看单子；`Duplicate the bug`——查重，判断是不是已有同类缺陷。注意"查重"是开发做的第三件事，说明重复缺陷是常见现象，也说明**提交前自己先查一遍**是好习惯。
>
> 第二段，**修复与集成**：`Debug` 定位；`Check out code` 取代码；`Fix bug` 修改；`Code Review` 代码评审；`Unit test` 单元测试；`Check in code` 入库；`Build a package` 打包；`Upload package` 上传

构建包。

第三段，**验证与关闭**：`Installation/configuration` 安装配置环境；`Verify fixed bugs` 验证缺陷；`Change bug status to close` 关闭。

数一下：真正"改代码"只有 `Fix bug` 一步，其余十三步全是**沟通、确认、评审、构建、部署、验证**。这就是软件工程的真相——**修一个 Bug 的时间，大部分花在保证"改对了、没改坏、能交付"上**。

再注意中间那段"打包—上传—安装配置"。这说明一个重要事实：**验证缺陷必须基于一个明确的构建包**，不能基于"开发本地的代码"。所以完整的缺陷信息里要有"构建包跟踪"这个字段，下一页会讲到。

我还想让你注意一个容易被忽视的角色：`Build a package` 这一步通常不是开发个人做的，而是流水线做的。这说明成熟的团队里，**缺陷修复要经过持续集成，而不是"我本地编译一下发给你"**。为什么这件事重要？因为一旦跳过流水线，你验证的就不是"将要发布的那个东西"，而是"某个人机器上的一个版本"。这两者的差别，往往就是"测试都通过了、上线还是坏的"这类事故的根源。

所以这 14 步真正教你的不是流程，是**可追溯**：每一步都留下记录，事后能回答"这个修复是什么时候、由谁、经过哪些检查，进到哪个包里去的"。

板书：三段横排。提出受理：Open a bug → Dev 收邮件 review → Duplicate 查重。修复集成：Debug → Check out → Fix → Code Review → Unit test → Check in → Build package → Upload package。验证关闭：Installation/configuration → Verify → Change status to close。底部标"14 Steps"，旁注："改代码 1 步，保证改对 13 步"。

提问：这 14 步里，哪一步最容易被省略，省略之后会出什么事？

预设回答：

- 答"Code Review"——常见。省了之后风险是改一处坏一处，而且这种坏往往几周后才暴露。

- 答"Unit test"——也常被省，尤其在赶版本时。

- 答"查重（Duplicate）"——最容易被忽略。省了之后缺陷库里同一问题有五个单子，统计出来的缺陷数、缺陷密度全是虚的，13.4 的度量就失真了。

- 收口：被省掉的那一步，往往正是**保证数据可信**的那一步。

易错点 / 考点：① 14 步的顺序要能说出关键节点（提交→查重→修复→评审→单测→入库→打包→部署→验证→关闭）；② 常考判断："验证缺陷用开发本地的代码即可"——错，必须用**构建包**并记录版本；③ 记一句

答题模板："缺陷修复是流程活动，不是编码活动"。

过渡：流程走通了，接下来是全网最容易吵架的两个属性：严重性和优先级。

【13.16】13.2.2 严重性和优先级（3 分钟）

这一页只有两个概念，但它们是缺陷管理里吵得最多的两个词。先把定义钉死。

严重性（severity）：衡量缺陷**对客户满意度的影响程度**。注意"客户满意度"这五个字——它不是"代码错得多离谱"，是"用户有多难受"。课件分了四级：**致命的（fatal）、严重的（critical）、一般的（major）、微小的（minor）**。四级怎么区分？我给一个能落地的判据：致命是"系统不能用、数据丢了、钱算错了、安全被击穿"；严重是"主流程走不通，但有绕行办法"；一般是"次要功能出错，或主功能有瑕疵但可用"；微小是"界面文字、样式、提示语这类不影响使用的"。

优先级（Priority）：指缺陷**被修复的紧急程度**。注意它衡量的是"多急"，不是"多坏"。

这两个词为什么必须分开？因为它们的判断依据完全不同。严重性看**影响**，优先级看**排期**。一个错别字出现在企业官网首页的标语上——技术影响是微小的，但它是品牌门面，优先级可能很高，当天就得改。反过来，一个只在极端并发下才触发的崩溃——技术上致命，但如果这个功能下个季度才上线，优先级可以先放低。

所以你会看到四种组合，每种都有现实对应：**高严重高优先**（核心功能挂了，马上修）、**高严重低优先**（罕见路径的崩溃，先记录）、**低严重高优先**（门面问题，先改）、**低严重低优先**（慢慢排队）。

关键的一句话：**严重性是客观判断，优先级是主观决策**。客观判断要的是标准统一，主观决策要的是有人负责。这两件事混在一起谈，就会变成"我觉得很严重"对"我觉得不严重"的口水仗。

板书：两列对照。左"严重性 Severity＝对客户满意度的影响"：fatal 致命／critical 严重／major 一般／minor 微小（各配一句判据）。右"优先级 Priority＝修复的紧急程度"。底部四象限：高严高优／高严低优／低严高优／低严低优。中间一句："严重性＝客观判断，优先级＝主观决策"。

提问：一个只在特定分辨率下错位的按钮，严重性怎么定？优先级怎么定？

预设回答：

- 答"都是微小，都往后排"——可能错。如果这个分辨率是主力机型的标准分辨率，那它影响面很大，优先级要高。

- 答"严重性微小，但优先级看影响面"——这就是正确思路：严重性按"客户满意度受损程度"，优先级按"什么时候必须好"。

- 答"看老板急不急"——真实但不专业。要有标准，标准里可以包含"营销节点""大客户诉求"这类因素，但必须事先写进判定标准。

- 收口：判定标准要**提前写好并公示**，事到临头再吵，永远吵不出结果。

> **易错点 / 考点：** ① 两个定义的准确表述：严重性＝对**客户满意度**的影响，优先级＝修复的**紧急程度**；② 四个严重性等级的英文与中文对应（fatal/critical/major/minor）；③ 常考判断："严重性高的缺陷优先级一定高"——错，两者可以背离；④ 案例题常给"错别字在首页"这类反例，考的就是两者的独立性。

> **过渡：** 既然优先级是决策，那有没有办法算？课件给了一个公式。

【13.17】优先级的计算（2 分钟）

这一页给了一个可以算的优先级公式，注意它是**经验公式**，不是物理定律，但它把"拍脑袋"变成了"有依据"。

公式是：**优先级 =（可重复性 + 可发生性）× 严重性**。三个因子，我们逐个看。

可重复性（Repeatability）：这个缺陷每次按步骤做都能出现吗？每次都出现，可重复性高；十次出现一次，低。它决定了开发能不能稳定地调试——**不能稳定复现的缺陷，修复成本会成倍上升**。

可发生性（Visibility）：用户遇到它的概率有多大。注意这两个词的区别：可重复性是"同一个人按同样步骤能不能复现"，可发生性是"在真实使用中撞上的机会有多大"。一个缺陷可能很容易复现（重复性高），但只有极少数用户会走到那条路径（可发生性低）。

严重性（Severity）：上一页的四级。注意公式里是**相乘**，不是相加——这意味着严重性是放大器：严重性低，另外两项再高也上不去；严重性高，另外两项稍高就会把优先级顶起来。

这个公式的教学价值在哪？在于它告诉你**优先级不是只看严重程度**。如果一个缺陷很严重，但基本没人会遇到、也不容易复现，那它确实可以先放一放；反过来，一个不太严重但人人都会撞上、而且必然出现的缺陷，往往要优先处理。

但要提醒：公式给的是**参考值**，最终排期还要考虑修复成本、版本节奏、依赖关系。公式解决的是"讨论有共同语言"，不解决"谁说了算"。

另外提醒一句：公式里的可重复性和可发生性通常也是分级打分的（比如都按 1～3 分），否则量纲不一致，相乘没有意义。

板书：写公式：**优先级 =（可重复性 + 可发生性）× 严重性**。三个因子各配一句：可重复性＝能否稳定复现（影响调试成本）；可发生性＝用户撞上的概率（影响暴露面）；严重性＝影响程度（放大器，相乘）。旁注："严重性相乘 → 放大器；不是相加"。

提问：一个致命缺陷，但一年只出现一次、且要极特殊的操作序列才能触发，优先级该怎么排？

预设回答：

- 答"致命就必须最高优先级"——直觉，但没算。按公式，可重复性低、可发生性低，乘上高严重性，结果往往不算最高。

- 答"排中等，但要留详细记录和监控"——这是成熟做法：不马上修，但要能第一时间知道它有没有在生产上出现。

- 答"先修了保险"——如果修复成本低，这也没错。公式是参考，成本也要算进去。

- 收口：公式帮你算"该多急"，成本帮你算"值不值得现在修"，两个一起看。

【思政落点 2】：这一页讲的是"客观判断"。严重性和优先级都是判断，判断就有立场。同一个缺陷，开发觉得"影响不大"，用户觉得"天塌了"，测试夹在中间很容易被情绪带走。真正专业的做法是：把判据提前写进团队的标准文档，用可重复的事实（能不能复现、多少人会遇到、坏了什么功能）说话，而不是用嗓门说话。这背后是一种职业素养——**手里有标准，就不靠情绪办事；心里有用户，就不靠自我安慰过关。**

易错点 / 考点：① 公式要能默写：优先级＝（可重复性＋可发生性）× 严重性；② **相乘不是相加**，这是判断题的常见陷阱；③ 可重复性与可发生性的区别是名词辨析高频点；④ 常考简答："为什么优先级不能只看严重性"。

过渡：严重性和优先级是最重要的两个属性，但一条缺陷的属性远不止这两个。

【13.18】13.2.3 缺陷的其它属性（2 分钟）

这一页列了五类"其它属性"，课件注明详见教材 P.361～362 的表格。我讲每一类**为什么要有它**，因为属性不是填着玩的，每一个都对应一种后续用途。

第一，**缺陷标识（ID）**。唯一编号。它的作用不是"好看"，是**可引用**——开会时说"第 1024 号那个问题"，比说"那个登录的问题"精确一万倍。

第二，**缺陷类型（type）**，比如功能、UI、性能、文档。分类是为了**分布分析**：如果某个版本的缺陷 60% 都是 UI，那说明前端规范或设计评审出问题了；如果大部分是功能，说明需求理解和逻辑实现有问题。13.3.3 的缺陷分布分析，就是建立在这些分类上的。

第三，**缺陷产生可能性（frequency）／可再现的概率**。这就是上一页公式里的可重复性。它决定了修复难度，也决定了测试要不要花时间去做更细的复现尝试。

第四，**缺陷来源（source）**：需求、设计、编码。这个属性极其重要，因为它是**过程改进的入口**。缺陷来自需求，说明需求评审不到位；来自设计，说明设计评审不到位；来自编码，说明编码规范和单元测试不到位。13.3 后面讲"从 Bug 中学习"，靠的就是这个字段的统计。

第五，**缺陷原因（cause）**：数据格式、计算错误、接口参数、变量定义与引用等等。这比"来源"更细一层——来源说"哪一阶段引入的"，原因说"具体错在哪一类"。有了它，才能定出针对性的编码规则。

你看这五个属性，其实分成两组：**ID 和类型是管理用的**（找得到、统计得了），**来源和原因是改进用的**（知道该改什么）。这也解释了一个常见现象：只填必填项、不填来源和原因的团队，永远在原地打转，因为他们的数据只能用来"数缺陷"，不能用来"减缺陷"。

板书：五行：ID（可引用）｜类型（功能・UI・性能・文档 → 分布分析）｜产生可能性/可再现概率（→ 修复难度）｜来源（需求・设计・编码 → 过程改进）｜原因（数据格式・计算错误・接口参数・变量定义引用 → 编码规则）。括号里标"管理用"和"改进用"两组。

提问：如果团队只统计缺陷数量，从不填"来源"，会失去什么能力？

预设回答：

- 答"不知道缺陷从哪来"——对，也就无法判断该加强需求评审还是代码评审。

- 答"没法考核"——这个想法要小心，缺陷来源不该用来追责，否则大家都会填"编码"，数据立刻失真。

- 答"数量还是能看的"——能看，但只能看"多不多"，看不出"为什么多"，也就做不出改进。

- 收口：**只数不析，等于白测**；属性填得越完整，后面的分析和改进才越有抓手。

易错点 / 考点：① 五类属性要能列举；② 缺陷**来源**（需求/设计/编码）与缺陷**原因**（数据格式/计算错误/接口参数/变量定义与引用）的区别是高频辨析题；③ 常考判断："缺陷属性只是记录用，与分析无关"——错，

13.3、13.4 的所有分析都依赖这些属性；④ 教材 P.361～362 有完整属性表，考试若考细则以教材为准。

过渡：属性是"字段"，接下来要看一张合格的缺陷单到底要填哪些内容——也就是完整的缺陷信息。

【13.19】13.2.4 完整的缺陷信息（3 分钟）

这一页讲三个核心要素，我认为是整章的"技术核心"：**步骤、期望结果、实际结果**。三个缺一个，缺陷就不成立。

先说**步骤**。课件的原话很讲究：**提供了如何重复当前缺陷的准确描述，应简明而完备、清楚而准确。这些信息对开发人员是关键的，视为修复缺陷的向导**。三个词要拆开：**准确**，是指能照着做；**简明**，是指不写无关内容；**完备**，是指不缺关键前提。最后那句"视为修复缺陷的向导"最重要——步骤不是给测试自己看的记录，是给开发指路的。开发照着你的步骤走一遍，能看见同样的现象，他才有可能定位到代码。

再说**期望结果**。课件说它**与测试用例标准或设计规格说明书或用户需求等一致，达到软件预期的功能。是验证缺陷的依据**。这里的关键是"**依据**"两个字：你凭什么说这是缺陷？凭它违背了某个书面来源——用例、规格说明书、需求。这就解释了为什么缺陷单里经常要写"依据：需求文档第 3.2 节"。没有依据的"期望结果"就是个人偏好。

最后说**实际结果**。课件说它**是实际执行测试的结果，不同于期望结果，从而确认缺陷的存在**。注意"不同于期望结果"这个判据——**缺陷的定义就是"实际 ≠ 期望"**。所以写实际结果的时候，只写事实：看到了什么、报了什么错、界面变成什么样，不要写"我觉得应该是……"，那是期望结果那一栏的事。

三个要素连起来是一条完整的论证：**照我的步骤做（前提）→ 你应该看到 A（期望）→ 你实际会看到 B（实际）→ 因为 A≠B，所以这是缺陷**。这条论证链写完整，开发就没有反驳的余地。

板书：三行。步骤：准确・简明・完备 → "修复的向导"。期望结果：依据＝用例／规格说明书／需求 → "验证缺陷的依据"。实际结果：客观事实 → "实际≠期望 ⇒ 缺陷成立"。底部画论证链：步骤 → 期望 A → 实际 B → A≠B ⇒ 缺陷。

提问：学生写缺陷单常写"实际结果：显示不正常"。这句话的问题在哪？

预设回答：

- 答"太笼统，没说哪里不正常"——对。要写具体的：是乱码、是空白、是错位、还是报错弹窗。

- 答"没给依据"——也对，没说是对照哪条需求判定"不正常"的。

- 答"能看懂就行"——恰恰不行。开发看到"不正常"，只能回你一句"我这儿正常"，一轮沟通就浪费了。

- 收口：**实际结果要写成"可被第三方核对的事实"，不是"感受"**。

> **易错点 / 考点：**① 三要素名称与各自定位（向导／依据／确认存在）；② 期望结果的依据来源：测试用例标准、设计规格说明书、用户需求——三个都要能答；③ 常考判断："实际结果可以写主观判断"——错；④ 简答高频："为什么说步骤是修复缺陷的向导"。

> **过渡：**步骤、期望、实际都有了，但真到了开发手上，往往还差几样东西。

【13.20】还需要什么重要的信息？（2 分钟）

> 这一页问"还需要什么"，课件给了五样：**产品、版本信息、图片、Trace Log、录制这个操作过程**。这五样东西有一个共同点——它们都是**让"我看到的"变成"你也能看到的"**的手段。
>
> 第一，**产品**。哪个产品、哪个端。同一个公司可能有 App、小程序、Web 后台，同一功能在三处表现不同，不写清就是白写。
>
> 第二，**版本信息**。这是最常被漏、后果最严重的一项。缺陷是跟版本走的：这个版本有、下个版本可能就修了；你在旧版本上提的缺陷，开发在新版本上当然复现不了。所以版本号、构建包号必须写。13.3.5 讲的缺陷跟踪系统里专门有"构建包跟踪"字段，就是为这个。
>
> 第三，**图片**。截图的价值在于**一眼抵一百字**。界面错位、乱码、弹窗，一张图就说清了。
>
> 第四，**Trace Log**。日志是给"看不见的问题"用的：崩溃、超时、接口报错、性能异常，这些现象在界面上可能只有一句"操作失败"，真正的原因在日志里。会看日志、会给日志，是测试人员从初级往中级走的分水岭。
>
> 第五，**录制这个操作过程**。当缺陷涉及多步操作、时序、动画、偶发问题时，光靠文字步骤很难还原，一段录屏最有效。
>
> 我把这五样归成一句话：**能截图就别只用文字，能给日志就别只说"报错了"，能录屏就别指望别人脑补**。缺陷报告的成本是测试的十分钟，收益是开发的两小时——这笔账非常划算。
>
> 再补一句实操建议：截图要截**有用**的部分——把错误提示、版本号、时间一起截进去，比截一整屏桌面有价值得多。

> **板书：**五样竖排：产品（哪个端）｜版本信息（版本号・构建包号）｜图片（截图）｜Trace Log（崩溃・超时・接口错误）｜录制操作过程（多步・时序・偶发）。旁边一句："让'我看到的'变成'你也能看到的'"。

> **提问：**一个偶发的崩溃，界面只提示"操作失败"，你最该补哪两样信息？

> **预设回答：**

- 答"日志和录屏"——正确组合。日志给原因线索（异常栈、错误码），录屏给触发路径。

- 答"截图就行"——不够。截图只能证明"失败了"，证明不了"为什么失败"。

- 答"复现步骤多写几遍"——如果是偶发，重复步骤也未必复现，要靠日志里的时间点和上下文去定位。

- 收口：**偶发缺陷拼的是证据密度**，证据越全，越可能被修掉。

> **易错点 / 考点：**① 五样信息要能列举；② **版本信息**是最易漏又最关键的一项，常考；③ 常考判断："有截图就不需要日志了"——错，二者证明的东西不同；④ 简答可答"提高可复现性、降低沟通成本"。

> **过渡：**把这些全加起来，就是一张完整缺陷单的字段清单——课件用一张表列了出来。

【13.21】缺陷完整的信息（3 分钟）

这一页是字段总表，教材 P363 表 13-7 有完整版。课件把字段分成三组，我按组讲，你按组记。

第一组，问题本身的描述：ID、标题、前提、环境、操作步骤、期望结果、实际结果、频率。这八个是"这件事是什么"。其中三个要特别说：**标题**——它是列表页唯一显示的东西，决定了别人点不点开；**前提**——很多人漏，比如"必须先登录管理员账号"，漏了开发就复现不出来；**频率**——就是可重复性，是"必现"还是"偶现"，直接决定排查策略。

第二组，管理属性：严重程度、优先级、类型、缺陷提交人、缺陷指定解决人、来源、产生原因、构建包跟踪。这八个是"这件事归谁管、多要紧、从哪来"。注意**提交人**和**指定解决人**必须明确——没有指定解决人的缺陷，就是没人认领的孤儿单，会一直躺在库里。

第三组，时间与归属：版本跟踪、提交时间、修正时间、验证时间、所属项目/模块、产品信息、状态。这七个是"这件事什么时候发生、现在在哪"。**三个时间戳**（提交、修正、验证）是后面算"缺陷年龄""平均修复时长"的基础数据；**所属项目/模块**是缺陷分布分析的维度；**状态**是生命周期跟踪的核心。

三组加起来二十多个字段，实际项目不会全填。但你要知道**哪些是不能省的**：标题、环境、版本、步骤、期

望、实际、严重性、优先级、提交人、指定解决人、状态——这十一个缺任何一个，这条缺陷的可用性都会明显下降。

最后提醒一句：字段不是越多越好。字段太多、必填项太多，测试人员就会敷衍填，数据质量反而下降。**必填项要少而关键，选填项要全而方便。**

> **板书：**三组竖排。① 描述组：ID・标题・前提・环境・操作步骤・期望结果・实际结果・频率。② 管理组：严重程度・优先级・类型・提交人・指定解决人・来源・产生原因・构建包跟踪。③ 时间归属组：版本跟踪・提交时间・修正时间・验证时间・所属项目/模块・产品信息・状态。底部："必填十一项，缺一不可"。

提问：为什么"前提"这一栏经常被漏掉，漏了会怎样？

预设回答：

- 答"因为大家觉得步骤里已经包含了"——常见原因。步骤是"做什么"，前提是"在什么状态下做"，两者不同。
- 答"漏了开发就复现不出来，然后退单"——对，这是最直接的后果。
- 答"前提不重要，反正开发会自己试"——这个想法会让你持续收到"无法复现"的回复。
- 收口：**前提是复现的必要条件**，尤其是权限、数据状态、配置开关这类隐藏条件。

易错点 / 考点：① 三组字段的划分逻辑；② 三个时间戳（提交/修正/验证）的用途是高频考点；③ 常考判断："字段越多越好"——错，要考虑填写成本与数据质量；④ 简答："一条缺陷单至少应包含哪些信息"——按描述组＋严重性优先级＋责任人＋状态来答。

过渡：字段知道了，怎么写才算合格？课件给了六条基本要求。

【13.22】13.2.5 缺陷描述的基本要求（3 分钟）

这一页是六条写作纪律，课件最后还问了一句"哪点比较关键？"，我们逐条讲完再回答这个问题。

第一条，**单一准确，每个报告只针对一个软件缺陷**。这是最容易被违反的一条。很多同学一条单子里写五个问题。结果开发修了三个、忘了两个，你验证的时候没法关，这条单子就永远开着。**一个缺陷一条记录**，是缺陷管理的铁律。

第二条，**可以再现，提供缺陷的精确操作步骤**。"可以再现"是要求，也是筛选：如果一个缺陷你自己都再现不了，要么继续挖，要么在单子里明确写"偶现，尝试 N 次出现 1 次，附日志"，让开发知道这是概率问题。

第三条，**完整统一，提供完整、前后统一的软件缺陷的步骤和信息**。"统一"这个词有意思——它指的是术语、版本号、模块名要跟团队一致。你在步骤里一会儿叫"用户中心"、一会儿叫"个人主页"，开发就得猜。

第四条，**短小简练，包括使用关键词**。这条和第三条看似矛盾，其实不矛盾：**信息要完整，表达要精炼**。能用"点击【保存】"就不用"用鼠标左键单击一下那个保存按钮"。关键词的作用是**可搜索**——以后有人搜"支付超时"，能搜到你这张单子。

第五条，**特定条件一定要注明**。特定条件包括：特定浏览器、特定分辨率、特定账号权限、特定数据量、特定网络环境。这一条和"前提"字段是一回事，但它强调"条件"——**缺陷往往藏在条件下**。第 23、24 页那两个示例，差别就在这一点上。

第六条，**不做评价**。不要写"开发太粗心了""这个功能做得真烂""这么简单的错误也能犯"。原因有两层：一是这些信息对定位问题毫无帮助；二是它会引发对抗，让技术讨论变成人身争论。

那"哪点比较关键"？我的答案是：**第一条和第二条最关键**。单一准确保证了缺陷可关闭，可以再现保证了缺陷可修复。其余四条都是为这两条服务的。

板书：六条竖排：① 单一准确（一缺陷一记录）② 可以再现（精确步骤）③ 完整统一（术语版本一致）④ 短小简练（关键词・可搜索）⑤ 特定条件注明（环境・权限・数据量）⑥ 不做评价（对事不对人）。圈出 ① ②，旁注"最关键：可关闭 + 可修复"。

提问：为什么"不做评价"被写进技术要求里，而不是"个人修养"里？

预设回答：

- 答"因为评价没用"——对了一半。评价不提供任何定位信息，属于噪声。

- 答"因为会引起矛盾"——这是实际后果。开发看到"这么简单都能错"，第一反应是防御，不是排查。

- 答"写评价显得测试很生气"——情绪可以理解，但缺陷单会被长期保存、被多人阅读，情绪化表达会一直留在库里。

- 收口：缺陷单是**工作文档**，不是情绪出口；写事实，让事实自己说话。

易错点 / 考点：① 六条要求要能列举；② **"一个缺陷一条记录"**是最高频考点，案例题常给"一条单子写多个问题"让学生判断；③ 常考判断："缺陷描述越详细越好"——要"完整而不冗长"，信息完整、表达精炼；④ "不做评价"的理由要能说出两条（无助于定位、引发对抗）。

过渡：要求讲完了，看两个真实报告——一个好的，一个散漫的，对照着看差异在哪。

【13.23】13.2.6 缺陷报告示例（3 分钟）

这是课件给的优秀缺陷报告示例，场景很朴素：一个文字编辑软件，把文字改成 Arial 字体后变成乱字符。我们看它好在哪。

看它的**重现步骤**，四步：打开编辑软件并创建一个新文档；在这个文档里随意录入一两行文字；选中这几行文字，通过 Font 菜单选择 Arial 字体；文字变成了无意义的乱字符。

这四步的质量非常高。第一步给了**起点**（新建文档，而不是打开已有文档）；第二步给了**数据**（一两行文字）；第三步给了**精确操作**（选中 → Font 菜单 → Arial）；第四步是**现象**。四步没有任何冗余，也没有任何缺失，别人照着做一定能看到。

再看**期望结果**："当用户选择已录入的文字并改变文字格式的时候，文本应该显示正确的文字格式，不会出现乱字符显示。"这句话是可判定的——不是"应该正常"，而是"应该显示正确的格式、不出现乱字符"。它有明确的判据。

最有价值的是**实际结果**那一段。它没有停在"显示乱字符"，而是继续做了三件事：第一，**定位了范围**——"它是字体格式的问题"；第二，**隔离了变量**——"如果在改变格式之前先保存文件，缺陷不会出现"，这说明"保存"这个动作会影响结果，是一条极有价值的线索；第三，**划定了平台边界**——"缺陷仅仅发生在 Windows 98，改变成其它字体格式时文字显示正常"。这三条信息，等于帮开发把排查范围缩小了一大半。

最后它附了图片链接。文字 + 截图，双证据。

这份报告为什么优秀？因为它不只是"报告现象"，它做了**测试人员该做的分析**：什么条件下有、什么条件下没有、影响哪些平台、影响哪些字体。这就是初级测试和成熟测试的差距——**成熟测试交出去的不是问题，是问题的边界**。

> **板书：**四步步骤（新建文档 → 录入文字 → 选中改 Arial → 出现乱字符）。期望：可判定（显示正确格式、无乱字符）。实际三件宝：定位范围（字体格式问题）＋隔离变量（先保存则不出现）＋平台边界（仅 Windows 98；其它字体正常）＋附件图片。底部："交出去的不是问题，是问题的边界"。

> **提问：**这份报告里，哪一句信息对开发最有价值？

> **预设回答：**

- 答"出现乱字符"——这是现象，开发从现象开始排查，但它本身不含定位信息。

- 答"先保存文件就不出现"——这一句价值最高，它直接指向了"未保存状态下的渲染路径"这一类可疑代码。

- 答"仅 Windows 98 出现"——也极有价值，把平台相关的字体渲染差异圈了出来。

- 收口：**"什么情况下不出现"往往比"什么情况下出现"更有诊断价值**——这是排除法的力量。

> **【思政落点 1】**：这一页讲的其实是"如实"。这份报告最难得的地方，是它写了"在其它字体格式下显示正常""在 Solaris 和 Mac 上没有这个问题"——这些话没有夸大缺陷的严重性，也没有隐瞒它只影响一个平台。缺陷报告是职业诚信最直接的考场：夸大，会让团队把资源投到不该投的地方；隐瞒，会让风险在发布后爆掉。测试人员手里握着"真相的定义权"，**如实描述、不夸大不缩小，是这个岗位最基本的职业底线**。

> **易错点 / 考点：**① 优秀报告的四个特征：步骤精确、期望可判定、实际含边界分析、有附件；② 常考简答："好缺陷报告与差缺陷报告的区别"——差在是否隔离变量、是否限定条件；③ 记住"期望结果必须可判定"这条判据；④ 案例题常要求"改写一段缺陷描述"，评分点就在步骤精确度与条件限定。

> **过渡：**看完好的，再看一个反面教材——同一类问题，写成什么样会让人抓狂。

【13.24】散漫的缺陷报告的示例（3 分钟）

> 这一页是反面示例，还是那个字体乱码的问题，但报告写成了这个样子。我们一段一段拆，看它犯了几种错。
>
> 它的重现步骤是这样的：在 Windows 11 上打开编辑软件并编辑已存在的文件；文件字体显示正常；我添加了图片，图片显示正常；在此之后我创建了一个新文档；在这个文档里随意录入大量文字；录入之后选择几行文字，通过 Font 菜单改成 Arial；有三次我重现了这个缺陷；我在 Solaris 上运行这些步骤没有问题；我在 Mac 上运行这些步骤没有问题。
>
> 问题一堆。**第一，混入了大量与缺陷无关的操作**：添加图片、图片显示正常——这些跟字体乱码有什么关系？**第二，把"平台验证"写进了重现步骤**。Solaris 和 Mac 的结果有价值，但属于"影响范围"，不该塞进"按这个步骤做"的列表——那一步是给人执行用的，不是给人读结论用的。
>
> **第三，变量没有隔离**。"编辑已存在的文件"和下一页优秀示例里的"新建文档"是不同的起点，而它后面又"创建了一个新文档"——同一个报告里出现了两个起点，到底哪个是触发条件？说不清。
>
> **第四，数据量描述含糊**："大量文字"到底是多少？

第五，最致命的一点：它把"重现了三次"写成了结论，但**没说三次都是必现还是偶然**。频率信息写了，但没写成可用信息。

再看它的期望结果——这部分其实和优秀示例差不多，写得还行。但**实际结果**那一段就跑偏了："我试着选择少量的不同的字体格式，但是只有 Arial 字体格式有软件缺陷，不论如何，它可能会出现在我没有测试的其它的字体格式"。前一句是好信息（只有 Arial 有问题），后一句是**猜测**。这就违反了第六条的精神——**把猜测当结论写进事实栏**。

对比下来，两份报告的核心差别只有一句话：**优秀的那份，交出去的是"缺陷的边界"；散漫的那份，交出去的是"我的操作日记"**。

板书：左侧列错误清单：① 混入无关操作（加图片）② 平台验证写进步骤 ③ 起点混乱（打开已有文件 vs 新建文档）④ 数据量含糊（"大量"）⑤ 频率无意义（"三次"没说明必现/偶现）⑥ 实际结果夹带猜测。右侧一句："要交缺陷的边界，不要交操作日记"。

提问：这份报告里"我在 Solaris 和 Mac 上没问题"这句话，该怎么处理才不浪费？

预设回答：

- 答"删掉"——可惜了，这是有价值的平台边界信息。

- 答"放到环境或备注里，写成'影响范围：仅 Windows 系列，Solaris／Mac 未复现'"——正确。信息保留，位置归位。

- 答"写在步骤最后也行"——形式上不算错，但会让人误以为这是一步操作。

- 收口：**信息没有废的，只有放错位置的**；写缺陷报告一半是收集，一半是归位。

易错点 / 考点：① 散漫报告的典型毛病要能指出至少四条；② 常考判断题："重现步骤里可以写自己的排查过程"——不可以，排查结论应放到备注/影响范围；③ "实际结果中不得包含推测"是高频评分点；④ 案例题常要求"两份报告对比，指出优劣并说明理由"。

过渡：缺陷写得再好，也只是单条记录。要让几十上百条缺陷有序流转，就得靠 13.3 的跟踪与分析。

【13.25】13.3 软件缺陷跟踪和分析（1 分钟）

13.2 解决的是"一条缺陷怎么写"，13.3 解决的是"一堆缺陷怎么管、怎么读"。

这一节五个小节，我按逻辑串一遍。**13.3.1 软件缺陷处理技巧**——人在处理缺陷时的动作规范：审阅、拒绝、分配、验证、重开、关闭、暂缓，以及沟通纪律。**13.3.2 缺陷趋势分析**——把缺陷数量画成时间曲线，看项目在什么阶段、能不能收敛。**13.3.3 缺陷分布分析**——把缺陷按模块、类型、严重性切开，看问题集中在哪。**13.3.4 缺陷跟踪方法**——Bug dashboard、缺陷报告的种类、Bug bash 活动，以及"必须做"和"不要做"的清单。**13.3.5 软件缺陷跟踪系统**——用工具把上面这些装起来。

你会发现这一节的中心词是"**跟踪**"。跟踪的意思是：每一条缺陷都有人在管、每一个状态变化都有记录、每一次决策都有依据。反过来，没有跟踪的团队会呈现三种症状：缺陷单开了没人认领、修了没人验证、关了不知道是谁关的。

这一节还有一条暗线，我要先点出来：**缺陷跟踪不只是"管缺陷"，它还是在管项目的节奏**。因为决定能不能发布的，往往不是"还剩多少用例没跑"，而是"还剩多少缺陷没关、关的速度够不够快"。

板书：五小节竖排：13.3.1 处理技巧（动作规范）→ 13.3.2 趋势分析（看时间）→ 13.3.3 分布分析（看空间/模块）→ 13.3.4 跟踪方法（看工具与活动）→ 13.3.5 跟踪系统（落地载体）。旁边："跟踪＝有人管・有记录・有依据"。

提问：一条缺陷三个月没人动，最可能是哪个环节出了问题？

预设回答：

- 答"没人认领"——很常见，说明"指定解决人"没填，或者填了但没通知到。

- 答"优先级太低被忘了"——延期缺陷没有挂版本号，就会永久沉底。

- 答"状态乱标，其实已经修了"——也有可能，状态管理不严，数据就不可信。

- 收口：这三种情况，都能靠本节讲的"处理技巧 + 跟踪系统"解决，关键是有没有人定期看那张列表。

易错点 / 考点：① 五个小节名称；② 常考判断："缺陷跟踪只是记录状态"——不完整，还包括推动处理、支撑决策、支撑发布判断；③ 趋势分析与分布分析的区别：**趋势看时间维度，分布看属性维度**，辨析题高频。

过渡：先讲人的动作——处理缺陷有哪些技巧。

【13.26】13.3.1 软件缺陷处理技巧（3 分钟）

这一页九条技巧，是团队协作层面的经验。我分成三组：让缺陷"动起来"、让判断"有标准"、让信息"看得见"。

第一组，让缺陷动起来。第一条，**确保被发现的缺陷能够被及时处理**。这是总目标。缺陷躺在库里不动，测试的工作就等于没做。第二条，**及时沟通，如 reopen bug 前，事先线下沟通**。这一条特别实在：你要重开一条缺陷，先跟开发说一声"我这边还是不行，我把日志发你看看"，比直接点"重新打开"再抄送一堆人要有效得多——后者在开发眼里等于"当着全组的面打我脸"。第三条，**只有测试人员有关闭缺陷的权限**，这条是制度保障，后面还会强调。

第二组，让判断有标准。第四条，**明确定义优先级和严重级衡量标准**。注意"明确定义"四个字——标准要写下来、要公示、要事先定。事后吵"这个到底算不算严重"，成本高得多。第五条，**注释清楚、达成一致意见**。缺陷单里的备注是团队共识的载体，改状态要写原因，改优先级要写理由。

第三组，让信息看得见。第六条，**周期性的会诊 Bug**——就是定期开会集体过一遍未解决的缺陷，下一页后面有专门的"三国会议"。第七条，**实时监控 Bug Dashboard**——看板要天天看，不能等周报。第八条，**不要试图隐藏 Bug**。这一条看着像道德要求，其实是技术要求：隐藏的缺陷不会消失，只会在生产上以更高成本出现，而且会毁掉整个度量体系——你藏了 10 个，缺陷密度、逃逸率全都不准了。第九条，**收集缺陷数据，进行必要的缺陷分析**。数据不分析就是垃圾，分析了才是资产。

九条讲完，你发现它们其实是三个层次：**制度（权限、标准）、习惯（沟通、看板、例会）、文化（不隐藏、做分析）**。工具能解决第三个层次的一部分，前两个层次只能靠团队自己。

> **板书：**三组。动起来：及时处理｜reopen 前先线下沟通｜测试独占关闭权。有标准：明确定义严重级与优先级标准｜注释清楚达成一致。看得见：周期性会诊｜实时看 Dashboard｜不隐藏 Bug｜收集数据做分析。底部三个层次：制度・习惯・文化。

提问：为什么"reopen 之前先线下沟通"会被写成一条技巧?

预设回答：

- 答"避免吵架"——对，直接重开在很多人眼里是"否定我的工作"，容易触发防御。

- 答"因为重开会增加计数，影响考核"——这个想法暴露了一个管理问题：如果重开率被用来扣分，大家就会想办法不重开，数据立刻失真。这正好是后面 13.4 讲的"度量不能用于追责"。

- 答"没必要，系统里有记录就行"——记录是给事后的，沟通是给当下的，两者都要。

- 收口：**技术问题用技术证据解决，人的情绪也要照顾**——这两件事不冲突。

易错点 / 考点：① 九条技巧要能说出五条以上；② **"只有测试人员有关闭缺陷的权限"**是最高频考点；③ 常考

判断："隐藏部分缺陷可以让指标更好看"——错，且会破坏度量可信度；④ 简答："缺陷处理技巧中，制度层面有哪些"（权限、标准定义）。

过渡：技巧是原则，具体到操作层面，课件给了八个基本动作。

【13.27】一些缺陷处理的基本操作（2 分钟）

这一页讲四个基本操作，都发生在缺陷"刚提交、还没进入修复"这个阶段，属于**受理环节**。

第一个，**审阅**。课件说：可以由**测试管理员、项目管理员或其他人**来进行，审阅缺陷报告的**质量水平**。注意审阅的对象是"报告的质量"，不是"缺陷的重要性"。审的是：步骤清不清楚、能不能复现、有没有重复、字段填全了没有。这一关的作用是**在缺陷流到开发手里之前，先把质量兜住**。

第二个，**拒绝**。课件说得很清楚：如果审阅者决定需要对一份缺陷报告进行**重大修改**，应该**和测试人员一起讨论，由测试人员纠正缺陷报告，然后再次提交**。这里有两个细节值得强调：一是"**和测试人员一起讨论**"——不是直接打回去让他自己猜；二是"**由测试人员纠正**"——提交人要对内容负责，别人替改会导致责任模糊。

第三个，**完善**。课件原文是"完整地描述了问题的特征并将其分离，那么审查者就会肯定这个报告"。这里的"**分离**"是关键——把混杂在一起的多问题拆开，一个缺陷一条记录，这正是 13.2.5 第一条要求。分离之后报告就"完善"了，可以放行。

第四个，**分配**。分配给适当的开发人员；**如果不知道具体开发人员，应分配给项目开发组长，由开发组长再分配给对应的开发人员**。这条很实用：很多缺陷卡在"不知道该给谁"，于是没人管。兜底方案就是给组长，让组长分。核心原则是——**缺陷必须有一个明确的责任人，宁可给大一级，也不能悬空**。

四个操作串起来是一条受理流水线：**审阅 → 不合格就拒绝并协助整改 → 合格就完善（拆分、补全）→ 分配责任人**。

板书：四步流水：审阅（管理员看报告质量）→ 拒绝（重大修改：一起讨论 → 测试自己改 → 再提交）｜完善（拆分多个问题、描述完整 → 肯定）→ 分配（给具体开发；不确定 → 给开发组长再分）。底部："缺陷不能悬空：宁可给大一级"。

提问："拒绝"和"关闭"有什么区别？为什么不能把不合格的报告直接关掉？

预设回答：

- 答"拒绝是退回去改，关闭是结束"——正确。拒绝是流程内的返工，关闭是终态。

- 答"都能让它消失"——这个区别至关重要：拒绝会留下记录，说明"这条单子曾被认为不合格"，还能再提交；关闭意味着"这事处理完了"，容易掩盖真实问题。

- 答"直接关掉省事"——省的是当下的事，代价是这个问题以后没人再提。

- 收口：**状态用错，等于把问题藏进了流程里**。

> **易错点 / 考点：**① 四个操作的名称与含义；② 拒绝的正确做法：与测试人员讨论 → 由测试人员纠正 → 再次提交（**不是审阅者代改**）；③ 分配的兜底规则：不确定具体人时给开发组长；④ 常考判断："审阅是审缺陷的严重程度"——错，审的是报告质量。

> **过渡：**受理完了，缺陷进入修复和验证。这一段的四个操作，是最容易产生争议的地方。

【13.28】一些缺陷处理的基本操作 – 续（3 分钟）

这一页接着讲四个操作，都在缺陷的后半段：**验证、重新打开、关闭、暂缓**。这四个动作直接决定缺陷的终态，也是团队摩擦最集中的地方。

第一个，**验证**。课件原文：缺陷的修复需要得到测试人员的验证，**同时还要进行回归测试，检查这个缺陷的修复是否会引入新的问题**。"同时还要"这四个字是重点。很多团队验证只做一件事：把当初失败的那个步骤再走一遍，通了就关。这是不够的，因为**修改本身就是风险的来源**——改一处，可能影响调用它的十处。所以验证 = 原缺陷验证 + 回归测试，两步都做完才算验证通过。

第二个，**重新打开**。课件说：重新打开一个缺陷，**需要加注释说明、电话沟通等**，否则会引起"打开—修复"多个来回，造成测试人员和开发人员**不必要的矛盾**。注意"不必要的矛盾"这个措辞——课件没有说"不许重开"，而是说"重开要有配套动作"。这是很务实的态度：重开是测试的权利，但行使权利要带证据、要带沟通。

第三个，**关闭**。课件再次强调：**只有测试人员有关闭缺陷的权限，开发人员没有这个权限**。这一条本章出现了两次，必考。它背后的逻辑是"**验证权与修复权分离**"——谁改的谁不能判合格，这是质量控制最基本的一条原则。

第四个，**暂缓**。课件说：如果**每个人都同意**将确实存在的缺陷移到以后处理，应该**指定下一个版本号或修改的日期**。**一旦新的版本开始时，这些暂缓的缺陷应该重新被打开**。三个要点：一是"每个人都同意"，暂缓是

共识不是单方决定；二是必须挂版本号或日期，不能空口说"以后再说"；三是新版本开始要**主动重开**——这一条最容易漏，漏了暂缓就等于删除。

四个动作合起来，讲的是同一个道理：**缺陷的终态必须是"有依据的终态"**。验证有回归、重开有说明、关闭有权限、暂缓有版本。

板书：四行。验证＝原缺陷验证 ＋ **回归测试**（修复可能引入新问题）｜重新打开＝加注释＋沟通（否则多来回 → 矛盾）｜关闭＝**仅测试有权限**｜暂缓＝全员同意＋挂版本号/日期＋新版本主动重开。底部："验证权与修复权分离"。

提问：开发说"这个改动很小，就一行，不用回归了"。你怎么回应？

预设回答：

- 答"行吧，看着确实小"——风险就在这里。"一行改动"造成大面积故障是行业里的经典事故类型，改动越小越容易让人放松警惕。

- 答"还是要回归，至少覆盖调用它的相关功能"——正确。回归范围可以按影响面裁剪，但不能为零。

- 答"那你写个说明，出问题你负责"——把技术问题变成了责任划分，不解决问题。

- 收口：**回归测试的规模可以谈，能不能不做不能谈**。

【思政落点 3】：这一页几乎每一句都在讲"怎么不吵架"。重开要沟通、暂缓要共识、关闭要权限——这些规则的本质，是**把人和事分开**。开发被你重开五次，不代表他能力差；你提的缺陷被判"不是 Bug"，也不代表你不专业。真正成熟的团队有一个共同特征：争论的时候争的是事实和标准，散会之后不记仇。**对事不对人，不是一句口号，是一套可以写进流程的动作**——加注释、给证据、当面说、按标准判。做到这四条，团队里的技术分歧就不会变成人际损耗。

易错点 / 考点：① 四个操作的名称与要点；② **验证必须包含回归测试**，这是高频考点；③ **关闭权限只归测试**（全章出现两次，必考）；④ 暂缓的三个要求：共识、挂版本/日期、新版本重开；⑤ 案例分析常给"开发自行关闭缺陷"让学生指出违规点。

过渡：把这八个操作放到项目层面看，缺陷处理和跟踪到底要达成什么目标？课件给了三条。

【13.29】软件缺陷的处理和跟踪（2 分钟）

这一页把前面所有操作收成三个目标，我认为是 13.3 的"总纲"。

第一个目标，**确保每个被发现的缺陷都能够被解决**。注意紧接着的那句解释：**"解决"的意思不一定是被修正，也可能是其他处理方式（例如，延迟到下一个版本中修正或者由于技术原因不能被修正）**。这句话非常重要，它纠正了一个普遍误解——很多同学以为"解决 = 修好"。实际上缺陷管理的目标是"**每条缺陷都有一个明确的、被记录的处理结论**"，这个结论可以是修复、延期、拒绝、无法解决。真正要杜绝的是"**没有结论**"——那条缺陷一直挂着，谁也不知道该怎么办。

第二个目标，**收集缺陷数据并根据缺陷趋势曲线识别测试处于测试过程中的哪个阶段**。这句话点出了趋势曲线的诊断功能：缺陷发现率还在高位，说明测试还在早期、功能还在大量暴露问题；发现率下降、修复率上升，说明进入收敛期；发现率很低但修复也停滞，可能是测试力度不够或者团队在等版本。**曲线是测试过程的"体温计"**。

第三个目标，**决定测试过程是否结束**。课件说：**通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式**。为什么有效？因为缺陷收敛是"质量趋于稳定"的间接证据。用例跑完了不代表质量稳定——用例可能设计得不够，或者跑得太浅；但缺陷的发现和修复都趋于收敛，说明"该暴露的暴露得差不多了，该修的修完了"。

这三个目标连起来是一条逻辑：**保证每条缺陷有结论 → 用数据看过程在什么阶段 → 用趋势判断能不能结束**。第 4 章讲的"退出准则"，在这里有了一个可量化的抓手。

> **板书：**三行。① 每条缺陷都有"解决"（＝修复／延期／拒绝／无法解决，**不等于修好**，但不能没结论）。② 用趋势曲线识别测试阶段（体温计）。③ 用趋势曲线决定测试是否结束（收敛＝可结束）。底部连到第 4 章："退出准则的可量化抓手"。

提问：一条缺陷被标成"延期到下个版本"，算不算"被解决了"？

预设回答：

- 答"不算，没修就是没解决"——这是最常见的理解，但课件明确说不修也可以是解决方式之一。

- 答"算，只要有明确结论和版本号"——正确。关键在"有结论、有归属"。

- 答"算，但要在报告里披露"——补得好，延期缺陷是发布风险的一部分，必须写进测试报告。

- 收口：**缺陷管理的目标不是"零缺陷"，是"零未决"**。

易错点 / 考点：① "解决"的定义（不一定是修正）是必考判断；② 趋势曲线有两大用途：识别阶段、决定是否结束；③ 常考判断："测试结束的标志是用例全部执行完毕"——不完整，还要看缺陷收敛与退出准则；④ 简答："如何用缺陷数据判断测试是否可以结束"。

过渡：既然要集体做决策，那这个"集体"是谁、怎么开？课件给了一个很生动的名字。

【13.30】集体 review 缺陷："三国会议"（2 分钟）

这个词很形象——"三国会议"，指项目经理和开发、测试三方坐在一起，集体过缺陷。为什么要三方？因为缺陷的去留，本质上是一个**三方利益平衡**的问题：开发想少改、想稳；测试想全修、想保险；项目经理要按期交付。谁单独决定都会偏。

看它的**参加者**：项目经理和开发组长、测试组长。注意是"组长"级别，不是全员——会议要能拍板，人多了拍不了板。

它做五件事。第一，**通过 Bug 数据库评估每个未解决的 Bug**。注意"通过数据库"——不是凭印象，是一条一条看记录。第二，**决定 Bug fix 优先级**。这就是 13.2.2 讲的优先级，在这里被集体确认。第三，**可否等到下个里程碑或版本解决？**——这是在决定"延期"这个出口。第四，**谁来解决**——落实到人。第五，**预测项目实际进度和发布时间**。这一条最有价值：缺陷会议的产出不只是"哪些修、哪些不修"，还包括"**基于当前缺陷状况，我们什么时候能发**"。

你会注意到，这五件事跟 13.3.1 那九条技巧、13.2 的优先级判定完全对得上——**三国会议是把前面所有规则落到一次会议里的机制**。

会议怎么开才有效？我给三条实践建议：一是**带着数据开**，别空口讨论；二是**只讨论有分歧的**，能按标准自动判定的不用上会；三是**每个结论都要落到字段上**——改优先级就改字段，延期就填版本号，别只在会议纪要里写一句"待定"。

还有一个细节：这个会要有**固定的输入和输出**。输入是排序后的未解决缺陷列表加趋势图，输出是"每条缺陷的结论＋责任人＋时间点"。没有输出的会，等于没开。

板书：中间写"三国会议＝PM ＋ 开发组长 ＋ 测试组长"。五件事：评估每个未解决 Bug → 定 fix 优先级 → 能否延到下个里程碑/版本 → 谁来解决 → 预测进度与发布时间。底部三条实践：带数据开・只议分歧・结论落到字段。

> **提问：**为什么这类会议要"组长"参加，而不是让每个开发和测试都来？

> **预设回答：**

- 答"人少好决策"——对，这类会议的核心是取舍，取舍需要有权的人在场。

- 答"组长更懂全局"——也对，组长知道排期和资源约束。

- 答"应该全员参加才透明"——透明度有价值，但全员会容易变成辩论赛，效率极低。透明可以靠会议纪要公开来实现。

- 收口：**决策会要小，信息要公开**——这两件事不矛盾。

> **易错点 / 考点：**① 三方参加者（PM、开发组长、测试组长）；② 五件事要能列举，尤其"预测进度与发布时间"；③ 常考判断："三国会议只决定修不修"——还包括优先级、责任人、进度预测；④ 与 13.3.1"周期性会诊 Bug"是同一件事的两种表述，别当成两个概念。

> **过渡：**会上要看的核心材料，就是缺陷趋势曲线。我们从这里进入 13.3.2。

【13.31】13.3.2 缺陷趋势分析（3 分钟）

趋势分析，就是**监控打开、关闭、已修正的缺陷随时间的变化**。三条线画在一张图上，横轴时间，纵轴缺陷数。

课件给了三句判断，每一句都值得拆开讲。

第一句，**产品开发质量情况取决于累积打开/关闭曲线的趋势**。注意"累积"两个字——累积打开曲线只增不减（因为它是历史上打开过的总数），累积关闭曲线也是只增不减。两条曲线的**间距**就是"当前还欠着多少缺陷"。间距在缩小，说明修复速度超过发现速度，质量在收敛；间距在扩大，说明发现快于修复，越欠越多。

第二句，**项目进度取决于累积关闭/打开曲线起点的时间差**。这句话有点绕，我翻译一下：两条曲线开始明显上升的时间点之间的间隔，反映了"从开始发现问题到开始解决问题"这段滞后。滞后短，说明团队响应快；滞后长，说明发现和修复脱节，进度会被拖住。

第三句，**开发人员、测试人员的工作进度、效率也能得到反映**。曲线的斜率就是效率：测试的斜率反映发现速度（也反映测试投入），开发的斜率反映修复速度。

这三个判断合起来，趋势曲线回答三个问题：**质量在变好还是变坏（间距）、团队响应快不快（起点差）、**

谁快谁慢（斜率）。

我要提醒一句方法论上的事：**趋势分析最怕"数据被污染"**。如果状态乱标（把没修的标成延期）、重复提交（同一问题五个单子）、批量关闭（版本末一次性关掉几十个），曲线就会画出漂亮的假象。所以趋势分析的可信度，取决于 13.2 那套字段和 13.3.1 那套纪律执行得怎么样。

板书：画三条曲线：累积打开（只增）、累积关闭（只增）、已修正。标三句判断：① 质量看**间距**（欠多少）；② 进度看**起点时间差**（响应滞后）；③ 效率看**斜率**（谁快谁慢）。底部："数据被污染 → 曲线画假象"。

提问：累积打开曲线和累积关闭曲线的间距越来越大，说明什么问题？

预设回答：

- 答"缺陷越来越多"——准确说是"欠的缺陷越来越多"，发现速度超过修复速度。

- 答"测试太猛了"——有可能，但更常见的是开发修复资源不足，或者版本前期功能还在大量提交。

- 答"要赶紧加测试"——方向反了。间距扩大时该做的是**加强修复**，不是加强发现。

- 收口：**间距扩大是修复侧的警报，不是发现侧的表扬**。

易错点 / 考点：① "累积"曲线的性质（只增不减）；② 三句判断的对应关系：质量→间距、进度→起点时间差、效率→斜率；③ 常考判断："缺陷趋势分析只看打开数"——错，要三条一起看；④ 数据质量对趋势分析的影响是简答得分点。

过渡：把趋势画成图，最直观的一种就是累积曲线图。

【13.32】累积缺陷趋势分析（2 分钟）

这一页是累积缺陷趋势的图示，你要从图里看出四件事。

第一件，**两条线的形状不一样**。累积打开曲线通常前陡后缓——前期功能陆续提交，问题集中暴露；后期新功能少了，发现速度自然下降。累积关闭曲线通常起步晚、上升稳——因为它要等缺陷被发现、被分配、被修复、被验证，天然滞后。

第二件，**两条线的开口方向**。开口（间距）先扩大后收窄，是一个健康的形状：前期欠账是正常的，因为测试在集中发现；关键是后期要收窄到足够小。如果开口一直扩大，这个版本不能发。

第三件，**交叉与平行**。真实项目里，累积关闭曲线几乎不可能追上累积打开曲线（因为总有一部分缺陷被拒绝或延期）。所以判断标准不是"交叉"，而是"**间距缩小到可接受范围**"。这一点很多同学会想错，以为两条线要相交。

第四件，**斜率的变化点**。曲线上任何一次明显的折点都有原因：斜率突然变陡，可能是新模块进入测试；突然变平，可能是环境挂了、或者测试资源被抽调。**折点就是故事**，看到折点要去问"那几天发生了什么"。

用这张图的时候记住一句话：**累积曲线看趋势，不看绝对值**。绝对值受项目规模影响，趋势才反映过程是否受控。

> **板书：**画两条累积曲线（打开、关闭），标出：前陡后缓的打开线、滞后上升的关闭线、开口先扩后收、折点。旁注四条：形状・开口方向・不必相交（看间距）・折点即故事。

提问：累积关闭曲线要不要"追上"累积打开曲线，才算测试可以结束？

预设回答：

- 答"要，追上了才说明都修完了"——理想化，现实中总有被拒绝、被延期的缺陷，追不上是常态。
- 答"不用，看间距是否缩小到可接受范围"——正确。
- 答"看剩余缺陷的严重性"——补得好，间距里如果剩下的都是微小的，风险就小。
- 收口：判断测试能否结束，看的是"**剩余缺陷的结构**"，不只是数量。

易错点 / 考点：① 累积曲线的性质与两条线的形状特征；② 常考判断："两条曲线必须相交才能结束测试"——错；③ 折点的诊断价值是简答亮点；④ 与 NOB 曲线的区别：NOB 是**存量**（打开数），累积曲线是**总量**（历史累计），这个辨析常考。

过渡：图上还有一个关键位置要单独讲——两条曲线最接近的那个点。

【13.33】收敛点（2 分钟）

这一页专门讲"收敛点"，课件配了一张横轴是日期、纵轴是 Bug 数量的图，图上有两条曲线：**报告的 Bug** 和**解决的 Bug**，交汇的位置被标了出来。

什么叫收敛点？直观地说，就是**报告的 Bug 数和解决的 Bug 数趋于接近、曲线走平的那个位置**。在这个点上，团队基本追上了欠账：新发现的速度和修复的速度大致平衡，剩余未解决的缺陷数量稳定在一个较低水平。

为什么收敛点重要？因为它是**"可以进入发布评估"的信号**。注意是"进入评估"，不是"可以发布"——收敛只说明过程稳定了，还要看剩余缺陷的严重性结构、还有没有高风险模块没测。

图上还有一个容易被忽略的细节：**两条曲线在收敛点之后不应该再明显张开**。如果收敛之后又张开（报告数又上去了），说明出现了新的问题源——可能是新功能合入、可能是回归引入、也可能是测试范围扩大。这种情况要重新评估。

另外提醒一点统计口径：这张图上的"解决"通常指的是"已修正/已验证"，不是"已关闭"。因为关闭还要经过测试验证，滞后更多。口径不说清，同一个项目两个人能画出两张不同的图——这也是为什么 13.2.1 要把状态定义钉死。

用一句话概括这一页：**收敛点不是一个精确的点，是一个"过程稳定"的区间；它的意义是告诉你"什么时候可以开始讨论发布"。**

板书：画两条曲线（报告的 Bug／解决的 Bug），标交汇处＝收敛点。旁注：收敛点＝新发现速度 ≈ 修复速度，剩余存量稳定；收敛后不应再张开；口径：解决＝已修正/已验证（非已关闭）。

提问：项目已经收敛了，是不是就可以安排发布了？

预设回答：

- 答"可以，都收敛了"——太急。收敛只说明过程稳定，还要看剩余缺陷的严重性、覆盖率、退出准则。

- 答"还要看剩下的缺陷都是什么级别"——正确，10 个微小和 10 个致命完全不是一回事。

- 答"还要看有没有回归引入的新问题"——对，收敛后张开往往就是回归问题。

- 收口：**收敛是必要条件，不是充分条件**。

易错点 / 考点：① 收敛点的含义（发现速度与修复速度平衡、存量稳定）；② 常考判断："收敛即可发布"——错；③ 统计口径问题（解决 vs 关闭）是易错点；④ 与 13.5 测试结束准则的联系常出综合题。

过渡：理想情况是平滑收敛，可现实中的收敛往往是"反弹着"完成的。

【13.34】不断反弹之中收敛（2 分钟）

这一页的标题就是结论：**在不断反弹之中收敛**。意思是真实的缺陷曲线不是平滑下降的，它一路上会反复抬头。

为什么会反弹？我给四个常见原因。

第一，**修复引入新缺陷**。改 A 影响了 B，这是最典型的反弹来源。这也是为什么 13.3.1 强调"验证要带回归测试"。

第二，**新功能合入**。敏捷迭代里，每个迭代都会提交新代码，新代码就带来新缺陷。所以在持续交付的团队里，缺陷曲线永远不可能真正归零，只能在每个版本末期收敛。

第三，**测试范围扩大**。前期只测主流程，后期开始测异常路径、兼容性、性能，一扩大范围就会冒出新的一批缺陷。这不是质量变差，是**视野变宽**。

第四，**环境或数据变化**。换了浏览器版本、加了数据量、改了配置，原来不报的问题开始报。

这四种反弹的性质完全不同：第一种是**质量信号**（要警惕），第二种是**正常代价**（要接受），第三种是**测试深化的结果**（要肯定），第四种是**环境问题**（要排查）。**看到反弹，第一件事是分清属于哪一种**，而不是一律当成"质量不行"。

由此得出一个很重要的判断方法：**看趋势不要看一天的抖动，要看一段时间的包络**。把曲线的高点连起来看，如果包络在下降，就是在收敛；如果包络也在上升，那才是真问题。

这一页给我们的启示是：**缺陷管理要有耐心，也要有分辨力**。耐心是指接受反弹，分辨力是指分清反弹的原因。

> **板书：**中间写"反弹之中收敛"。四个原因竖排：修复引入新缺陷（质量信号）｜新功能合入（正常代价）｜测试范围扩大（视野变宽，应肯定）｜环境/数据变化（要排查）。底部："看包络不看抖动"。

提问：缺陷曲线在版本末期又抬头了，你怎么判断这是好事还是坏事？

预设回答：

- 答"抬头就是坏事"——不一定，如果是扩大测试范围导致的，反而是好事，说明在补漏洞。

- 答"看新增缺陷集中在哪"——专业。集中在刚测的模块＝范围扩大；集中在刚修的模块＝修复引入；分散全系统＝可能是环境问题。

- 答"看严重性"——也对，新增的都是微小问题，风险有限。

- 收口：**反弹要做归因，归因之后才有结论**。

易错点 / 考点：① 四种反弹原因及其性质区分；② 常考判断："缺陷曲线应严格单调下降"——错，反弹是常

态；③ "看包络不看抖动"是可写入答题的技巧句；④ 与 13.3.2 趋势分析结合出案例分析的概率较高。

过渡：趋势看的是时间轴，如果换个维度——按模块、按类型切开来，就是分布分析。

【13.35】13.3.3 缺陷分布分析（2 分钟）

分布分析的定义，课件给得很干脆：**缺陷分布报告，是缺陷数量与缺陷属性的函数。如测试需求和缺陷状态、严重性的分布情况等。**

这句话要拆开理解。"缺陷数量与缺陷属性的函数"——意思是**把缺陷按某个属性分组，然后数每组的个数**。属性可以是模块、类型（功能/UI/性能/文档）、严重性、状态、来源（需求/设计/编码）、发现阶段、提交人。换一个属性，就是一张新的分布图。

它和第 31 页的趋势分析是一对：**趋势分析把缺陷按时间切，分布分析把缺陷按属性切**。一个回答"什么时候出问题"，一个回答"哪里出问题"。

分布分析最典型的用途是找**聚集**。课件在 13.4.1 里有一句问话："缺陷分布是否符合正态分布？是否过于集中在 1-2 个模块？"——这就是分布分析的判据。如果缺陷高度集中在少数模块，通常意味着这几个模块**复杂度高、改动频繁、或者开发质量差**，它们就是下一轮测试和代码评审的重点。

反过来，如果分布非常均匀地散落在所有模块，也不是好消息——可能说明测试深度不够，只测到了表面的共性问题，没测到各模块特有的深层问题。

几个常用的分析角度我列一下：**按模块**（找热点）、**按严重性**（判断风险结构）、**按类型**（判断是功能设计问题还是实现问题）、**按来源**（判断哪个阶段引入最多）、**按发现阶段**（判断缺陷是在哪一层被拦住的）。最后一个角度直接连到 13.4 的"缺陷清除率"。

记住一句：**分布分析的价值不在"数清楚"，在"看出该改哪里"**。

板书：定义：缺陷数量 = f(缺陷属性)。常用维度竖排：模块（找热点）｜严重性（风险结构）｜类型（功能/UI/性能/文档）｜来源（需求/设计/编码）｜发现阶段（哪层拦住）。判据："是否过于集中在 1–2 个模块？"底部："趋势看时间，分布看属性"。

提问：如果 80% 的缺陷集中在两个模块，你的第一反应应该是什么？

预设回答：

- 答"这两个模块的开发水平不行"——最容易想到，但这是**对人归因**，风险很大，而且往往不是真相。

- 答"这两个模块最复杂、改动最频繁"——更可能。复杂度高、变更多，缺陷自然多，这是正常规律。

- 答"这两个模块的测试最充分"——这个可能性必须考虑！有时候不是那两个模块差，是**只有那两个模块被认真测了**。这个反直觉的答案最能体现思维深度。

- 收口：分布集中至少要区分三种原因：**复杂度、变更频率、测试投入不均**，别急着下结论。

易错点 / 考点： ① 分布分析的定义（缺陷数量与缺陷属性的函数）；② 与趋势分析的辨析（时间 vs 属性）；③ 常考判断："缺陷集中在少数模块一定是开发质量问题"——错，可能是复杂度、变更频率或测试投入不均；④ 常用分析维度要能列举四个以上。

过渡： 分布分析长什么样？课件给了一张示例图。

【13.36】示例：缺陷分布分析（2 分钟）

这一页是分布分析的示例图。图我没有逐格念的必要，你要学会的是**看这类图的三个动作**。

第一个动作，**先看横纵轴各是什么**。分布图的横轴通常是分类属性——模块、类型、严重性、状态；纵轴是缺陷数量或占比。这两者一确定，图的含义就定了。如果一张分布图没标清轴，它就没有分析价值。

第二个动作，**看形状**。分布的形状基本只有三种，每种对应一个结论。第一种，**集中型**：某一两个分类明显高出其它，形成尖峰——去找那几个模块/类型的共性问题，通常是复杂度或变更热点。第二种，**均匀型**：各分类数量差不多——要么系统各部分质量均衡，要么测试深度不足、只摸到表面。第三种，**长尾型**：少数分类很高、大量分类很低——这是最常见的真实形状，重点抓头部即可。

第三个动作，**看它跟上一版的变化**。单张分布图只能说明"现在哪里多"，两张对比才能说明"改进有没有效果"。如果上一版集中在那两个模块，这一版还是那两个，说明上一轮的改进措施没落地；如果转移到了别的模块，说明那两个修好了，问题在别处。

我还要强调一个常见误用：**不要用分布图去评价个人**。按"提交人"切分缺陷数，看起来很像绩效排名，但缺陷数受分配的任务难度、测试投入、模块复杂度影响极大。这种图一旦用来考核，数据立刻失真——大家会少提缺陷、或者把缺陷提给别人。这是 13.4 会专门讲的"**度量会改变行为**"。

板书： 三个动作：① 先看轴（分类属性 × 数量/占比）② 看形状（集中型／均匀型／长尾型 → 三种结论）

③ 看上版对比（改进有没有效）。红框警示："按提交人切分做绩效 → 数据失真"。

提问：两张分布图对比，上一版缺陷集中在支付模块，这一版集中在订单模块，说明什么？

预设回答：

- 答"支付修好了，订单变差了"——可能，但也可能是订单模块这次改动大。

- 答"要结合变更量看"——专业。缺陷密度＝缺陷数/规模，不看规模只看数量会误判。

- 答"可能是测试重点转移了"——也要考虑，这一版可能重点测了订单。

- 收口：**分布变化必须结合"变更量"和"测试投入"一起解读**，否则容易误伤。

易错点 / 考点：① 分布图的三个观察动作；② 三种形状对应的结论；③ 常考判断："缺陷数多的模块就是质量差的模块"——错，要看缺陷密度（除以规模）；④ "用缺陷数据考核个人会导致数据失真"是重要考点，为 13.4 埋伏笔。

过渡：分布和趋势都是"看现象"。再往深一层，是从缺陷里挖出原因——这才是真正的价值所在。

【13.37】缺陷分析：从 Bug 中学习（3 分钟）

这一页是 13.3 里思想性最强的一页，标题就叫"**从 Bug 中学习**"。它给了三个词和一个方法。

先看三个词：**People、Process、Technology**——人、过程、技术。这是缺陷根因的三个层面。一个缺陷反复出现，你要问：是人的能力/态度问题？是流程缺失（没有评审、没有回归）？还是技术限制（框架缺陷、工具不足）？**大部分团队习惯把问题归到"人"头上，而真正可改的往往是"过程"和"技术"。**

再看三个方法。第一个，**根因分析 Root Cause Analysis**。注意是"根因"，不是"直接原因"。直接原因是"这里少了一个判空"，根因可能是"代码规范里没有强制要求边界检查，评审清单里也没这一项"。改直接原因只能修一个 Bug，改根因能修一类 Bug。

第二个，**案例分析与学习：Case Study**。把典型缺陷整理成案例，在团队里讲——不是批判谁写的，而是讲"这类问题长什么样、怎么早发现"。这就是组织级的经验积累。

第三个，**缺陷预防：代码规则、设计 Checklist**。这是根因分析的**产出物**。分析完了不做规则，下次还会犯；做成了规则，就能在编码和评审阶段拦住。

页面上还有三个问句，是根因分析的固定套路：**What's problem?**（问题是什么）**What's root cause?**（根因是什么）**What's solution?**（对策是什么）。这三个问句看着简单，但顺序不能乱：没弄清问题就找根因，会找到假根因；没找到根因就给对策，对策往往是"加强重视""提高意识"这种没用的东西。

我给这个方法配一条判据：**如果一个"对策"无法验证是否执行、也无法验证是否有效，它就不是对策，是口号。**"加强代码评审"是口号，"在评审清单里增加'所有外部输入必须判空'一项，并在每次评审记录里勾选"才是对策。

板书：三个层面：People／Process／Technology（提示：多数问题可改的是后两个）。三个方法：根因分析 RCA｜案例分析 Case Study｜缺陷预防（代码规则・设计 Checklist）。三问：What's problem? → What's root cause? → What's solution?（顺序不可乱）。

提问："这次事故是因为某某同学粗心"——这句话作为根因分析的结论，问题在哪？

预设回答：

- 答"太主观，没有证据"——对，而且"粗心"不可改、不可验证。

- 答"没有分析流程原因"——关键。为什么一个粗心能造成事故？是不是缺少评审、缺少自动化检查、缺少回归？这些才是可改的。

- 答"归因到人会影响团队氛围"——也是实际后果，以后没人愿意暴露问题。

- 收口：**好的根因分析，结论一定指向"可以被改变的东西"**——流程、规则、工具，而不是人的性格。

【思政落点 5】：这一页讲的是"复盘与担当"。一个团队敢不敢做根因分析，取决于它把缺陷当成什么：当成"某人的罪证"，就没人说真话，分析会变成甩锅会；当成"过程的反馈"，大家才愿意把细节摊开。这背后是一种组织品格——**出了问题先想"我们的流程哪里漏了"，而不是"谁该背这个锅"**。对个人也一样：一个测试人员愿意把自己漏掉的缺陷写进复盘，愿意承认"这条我本该想到"，他的成长速度会远快于那些只会说"这不是我的问题"的人。担当不是背锅，是把问题变成改进的起点。

易错点 / 考点：① 三个层面（People/Process/Technology）与三个方法；② 三问的顺序不能乱；③ 常考判断："根因分析就是找出犯错的代码行"——错，那是直接原因；④ 简答高分点："如何从缺陷反推过程改进"——根因分析 → 规则/Checklist → 前移到评审与自动化。

过渡：怎么把这些分析结果天天摆在团队眼前？靠看板。

【13.38】13.3.4 缺陷跟踪方法 – Bug dashboard（2 分钟）

Bug dashboard，缺陷看板。这一页是图，你要从这类看板里看出"**它是给谁看的、看什么、看完做什么**"。

先说给谁看。看板有三类读者：**测试人员**看自己的待办和验证队列；**开发组长**看修复队列和积压；**项目经理**看整体收敛情况和风险。所以一个好的看板不是把所有数据堆上去，而是**按角色分区**。

再看常见的几块内容。**缺陷总量与未解决量**——回答"还欠多少"。**按严重性/优先级的分组**——回答"欠的是不是要命的"。**按状态的分布**——回答"卡在哪个环节"（比如大量缺陷停在"已修正待验证"，说明验证是瓶颈）。**按模块的分布**——回答"哪里最痛"。**新增与修复的日/周速率**——回答"追得上追不上"。**缺陷年龄**——回答"有没有陈年老账"。

看完做什么？这是看板最关键的价值：**看板不是展示，是驱动动作**。看到大量缺陷停在"待验证"，就要加验证人力；看到某模块缺陷年龄普遍偏长，就要开专项会诊；看到 NOB 连续三天上升，就要去找原因。

我要提醒一个反模式：**"装饰性看板"**——数据很好看，但没人根据它做决定。判断一个看板有没有用，只需问一句："**上周你看板上的哪个数字，让你做了什么决定？**"答不上来，这个看板就是墙纸。

另外，看板要**实时或准实时**。课件在 13.3.1 说"实时监控 Bug Dashboard"，在 13.3.2 说"实时缺陷趋势分析"——两次强调"实时"。周报式的滞后数据，等你看出来已经晚了。

板书：三类读者：测试（待办・验证队列）｜开发组长（修复队列・积压）｜PM（收敛・风险）。六块内容：总量/未解决・严重性分布・状态分布（找瓶颈）・模块分布・新增/修复速率・缺陷年龄。底部判据："上周哪个数字让你做了什么决定？"

提问：看板上显示大量缺陷停在"已修正待验证"，这说明什么？该做什么？

预设回答：

- 答"缺陷没修好"——不对，已修正说明开发改完了，问题是验证没跟上。

- 答"验证是瓶颈，要加验证人力或简化验证流程"——正确。

- 答"说明测试提得多、修得快"——只对了一半，卡在验证环节意味着发布被拖住。

- 收口：**状态分布图的最大价值，就是找出流程里的瓶颈环节**。

易错点 / 考点：① 看板的核心内容与读者；② 常考判断："看板只要数据全就行"——错，要能驱动动作、要按角色分区；③ "已修正待验证积压 → 验证瓶颈"是典型的状态分布判读题；④ "实时"这个要求在课件里出现

两次，属于考点。

过渡：看板展示的是一张张具体的报表。课件把常用的缺陷报告归了四类，我们一类一类看。

【13.39】示例（2 分钟）

这一页是看板的实例，我借它把"看板怎么读"这件事再落一遍地。

拿到任何一张缺陷看板截图，我建议按固定的四步走，形成肌肉记忆。

第一步，找总量和趋势。先看未解决缺陷的总数，再看这两天的走向。总数告诉你"欠多少"，走向告诉你"在变好还是变坏"。这两个数字是看板的地基，其它都是细节。

第二步，找瓶颈。看状态分布，哪一列的柱子最高，瓶颈就在哪里。停下来最久的那一堆，就是流程里最堵的地方。常见的三个瓶颈：待分配（没人认领）、待验证（验证人力不足）、待修复（开发排期靠后）。

第三步，找热点。看模块分布，找出缺陷最集中的模块。这个模块接下来要重点测、重点评审——这就是 13.3.3 分布分析的落地。

第四步，找异常。看严重性分布和缺陷年龄：有没有致命/严重的缺陷挂了很多天？有没有一批缺陷的年龄超过了两周？这些是风险的集中体现，必须当场上报。

四步走完，你看板就"读完"了，而且读出了动作。

我还要说一句看板之外的话：**看板上的数字是给团队用的，不是给上级检查的**。一旦看板变成"检查工具"，团队就会开始"管理看板"而不是"管理缺陷"——把单子改得好看一点，把不好看的关掉。这个现象在行业里非常普遍，你们以后带团队时一定要留意。

板书：读板四步：① 总量＋趋势（欠多少・变好变坏）② 瓶颈（状态分布最高的那列）③ 热点（模块分布最高）④ 异常（致命缺陷挂多久・年龄超期）。底部："看板是团队工具，一旦变成检查工具，就会被'管理'"。

提问：如果一张看板上所有指标都很漂亮，你会相信吗？

预设回答：

- 答"会，说明质量好"——要谨慎。指标全好通常只有两种解释：质量真的好，或者**数据被"管理"过**。

- 答"要抽查几条原始缺陷单"——专业做法，抽 5 条看步骤是否完整、状态是否合理，就能判断数据可信度。

- 答"要看跟上一版比有没有突变"——好思路，指标突然全面转好往往不正常。

- 收口：**看板的第一层能力是读图，第二层能力是判断数据可不可信**。

> **易错点 / 考点：** ① 读板四步的顺序；② 三个常见瓶颈（待分配、待验证、待修复）；③ 常考判断："看板指标越好，质量越好"——要注意数据质量；④ "抽查原始记录验证数据可信度"是可写入答题的方法。

> **过渡：** 看板上那些图，其实是四类缺陷报告的组合。我们逐类看它们的定义。

【13.40】缺陷报告（3 分钟）

这一页讲四类缺陷报告，是本节最"可考"的一页，因为每类都有明确定义。我们逐条过。

第一类，**缺陷分布报告**。课件原文：允许将缺陷计数作为**一个或多个缺陷参数的函数**来显示，生成缺陷数量与缺陷属性的函数。如测试需求和缺陷状态、严重性的分布情况等。关键词是"**函数**"和"**属性**"——横轴是属性，纵轴是数量。它的用途是找热点、看结构。

第二类，**缺陷趋势报告**。按**各种状态**将缺陷计数作为**时间的函数**显示。趋势报告**可以是累计的，也可以是非累计的**。这里"累计/非累计"的区分很重要：累计曲线只增不减，看长期走向；非累计曲线（每天新增多少、每天关闭多少）看短期节奏。两个一起看，既有全局又有当下。

第三类，**缺陷年龄报告**。显示缺陷**处于活动状态的时间**，展示一个缺陷处于某种状态的时间长短，从而了解处理这些缺陷的**进度情况**。这一类最容易被忽视，但它最实用——因为它直接暴露"卡住了"。一个团队缺陷总数不多，但平均年龄 30 天，问题比"数量多但三天内清完"严重得多。

第四类，**测试结果进度报告**。展示测试过程在被测应用的**几个版本中的执行结果以及测试周期**。注意它跨版本：它回答的是"这几个版本，测试执行得怎么样"，介于测试报告和缺陷报告之间。

四类报告的分工我给你一句话：**分布看"哪里"，趋势看"何时"，年龄看"多久没动"，进度看"测到哪了"**。四个问题合起来，就是一个完整的项目视图。

> **板书：** 四行。分布报告＝缺陷数 = f(属性)（测试需求・状态・严重性）。趋势报告＝缺陷数 = f(时间・状态)，**可累计/非累计**。年龄报告＝缺陷处于活动状态（某状态）的时间 → 处理进度。测试结果进度报告＝跨多个版本的执行结果与测试周期。底部："哪里・何时・多久没动・测到哪了"。

提问：缺陷年龄报告能发现什么问题，是趋势报告发现不了的？

预设回答：

- 答"哪些缺陷挂太久"——对。趋势报告看总量，会掩盖"总量正常但有几条挂了两个月"这种情况。

- 答"哪个环节最拖"——也能，因为年龄报告按状态看时长，能定位到"卡在验证"还是"卡在修复"。

- 答"跟趋势报告差不多"——不一样。趋势是流量视角，年龄是**存量时长**视角。

- 收口：**总量健康不代表结构健康**，年龄报告就是照结构的镜子。

易错点 / 考点：① 四类报告的定义要能对上号，尤其"分布＝属性函数、趋势＝时间函数"；② 趋势报告分**累计**与**非累计**两种，是高频细节；③ 缺陷年龄报告的定义（处于活动状态的时间）常考；④ 常考判断："测试结果进度报告就是测试报告"——不是，它只是测试报告的数据来源之一。

过渡：除了日常跟踪，行业里还有一种集中清理缺陷的活动，值得了解一下。

【13.41】“Bug bash” 活动（2 分钟）

Bug bash，直译"缺陷大扫除"，课件给的注释就是"**缺陷大扫除活动**"。它是一个集中式的、跨角色的找缺陷活动。

它的典型形态是这样：在版本发布前的某个时间点，把产品、开发、测试、甚至运营和市场的人聚在一起，给一段时间（常见半天到一天），所有人放下手上的活，专门去用这个产品、找问题，找到就当场提交，然后集中评估、集中修。

它为什么有效？三个原因。

第一，**视角多样**。测试人员天天测，会形成路径依赖，总走那几条熟路；产品经理会按业务场景走；开发会按实现细节走；市场的人会按真实用户那种"乱点"的方式走。**不同的使用习惯，撞出不同的缺陷**。

第二，**成本低、见效快**。半天时间，几十个人，能扫出一批"显而易见但没人报"的问题——错别字、文案不通、按钮文案和功能不符、明显的界面错位。这些单条不重要，但堆起来就是"产品不精致"的观感。

第三，**它是一次团队共识活动**。大家一起看到产品现在的样子，对"能不能发"会有更一致的判断。

但要注意它的局限：**Bug bash 找不出深层缺陷**。它适合扫表面问题，不适合替代系统测试、性能测试、安全

测试。所以它通常放在"主要测试已经做完、准备发布"这个阶段，作为补充。

组织上还有两个要点：一是**要有明确的记录方式**（用同一个系统提交、统一标签，比如都打上"bugbash-2026xx"），否则事后没法统计；二是**要当场去重和分诊**，几十个人同时提，重复率极高，必须有人实时合并。做完之后要出一份小结：提了多少、去重后多少、修了多少、哪些延期——这才是一次完整的活动。

> **板书：**中间写"Bug bash＝缺陷大扫除"。三优点：视角多样（测试路径依赖 vs 产品业务视角 vs 开发实现视角 vs 市场用户视角）｜成本低见效快（扫表面问题）｜团队共识。局限：找不出深层缺陷（不能替代系统/性能/安全测试）。组织要点：统一提交与标签・当场去重分诊・事后小结。

提问：Bug bash 应该安排在项目的哪个阶段？为什么？

预设回答：

- 答"一开始就搞"——效果差，功能还不完整，扫出来的都是"没做完"。

- 答"发布前，主要测试做完之后"——正确。这时产品形态稳定，扫的是"打磨类"问题。

- 答"随时都可以"——形式上可以，但性价比在发布前最高。

- 收口：**它补的是"打磨"，不是"兜底"**——深层问题还是得靠专业测试。

易错点 / 考点：① Bug bash 的中文含义（缺陷大扫除）与组织形式；② 它的价值与局限要能各说两条；③ 常考判断："Bug bash 可以替代系统测试"——错；④ 组织要点（统一标签、当场去重）是实操题的得分点。

过渡：把这一节的实践收成清单，课件先给了"必须做的"五条。

【13.42】必须做的…（2 分钟）

这一页五条"必须做"，是全章的实践清单，我逐条讲它为什么"必须"。

第一条，**尽早定义和推广 Bug 管理流程**。关键词"尽早"和"推广"。流程不是出事之后补的，是项目第一天就该有的；"推广"意味着要让所有人都知道并且会用——流程写在文档里没人执行，等于没有。

第二条，**明确定义优先级和严重级衡量标准**。这一条在 13.2.2 和 13.3.1 都出现过，第三次出现说明它真的重要。标准要具体到"什么样的情况算致命"，最好带例子。没有标准，每次判定都是一场谈判。

第三条，**清晰设置 Bug 提交必须的信息**。就是 13.2.4 那些必填字段。设置为必填，是从系统层面保证数据质

量——靠自觉是靠不住的。

第四条，**周期性的会诊 Bug**。就是三国会议。周期性意味着有固定节奏，比如每周一次，而不是"想起来才开"。

第五条，**周期性的检查 Bug 列表**。注意它和第四条不同：会诊是**集体决策**，检查是**个人/责任人自查**——看自己名下有没有该处理没处理的、有没有信息不全的、有没有长期挂着的。

五条合起来，其实是一条完整的"制度链"：**流程（第一条）→ 标准（第二条）→ 字段（第三条）→ 决策机制（第四条）→ 巡检机制（第五条）**。有了这五条，缺陷管理才能自我运转，而不是靠某个人天天催。

我给一个自检方法：拿这五条去问一个团队，只要有两条答"没有"，这个团队的缺陷管理基本靠人治，一旦核心成员离开就会崩。

补一条判断标准：这五条里只要缺两条，团队就还是"人治"——靠某个负责任的人天天催，人一走就散。

> **板书：**五条竖排：① 尽早定义并推广 Bug 管理流程 ② 明确定义优先级/严重级标准 ③ 清晰设置提交必填信息 ④ 周期性会诊 Bug（集体决策）⑤ 周期性检查 Bug 列表（责任自查）。底部链条："流程 → 标准 → 字段 → 决策 → 巡检"。

提问："周期性会诊"和"周期性检查列表"，为什么需要两个都做？

预设回答：

- 答"一个是开会一个是看表，重复了"——不重复。会诊解决"有分歧的、要取舍的"；检查解决"遗漏的、没人管的"。

- 答"会诊管集体，检查管个人"——精确。

- 答"只做会诊就够了"——会诊频率低（比如每周），一周之内新挂的单子没人看；检查可以每天做。

- 收口：**决策要集体，巡检要高频**——两者节奏不同，不能互相替代。

易错点 / 考点：① 五条要能列举；② "尽早定义并推广"中的"推广"是易漏点；③ 会诊（集体决策）与检查（个人自查）的区别是辨析题；④ 常考简答："如何建立可持续的缺陷管理机制"，按这五条答即可。

过渡：知道该做什么，也要知道不该做什么——课件专门列了四条"不要做"。

【13.43】不要做的…（2 分钟）

这一页四条"不要做"，每一条背后都有一个真实的坑。

第一条，**把 I18N Bug 和 L10N Bug 混为一谈**。I18N 是国际化（internationalization），指**产品本身要支持多语言、多地区**的能力问题——比如代码里把文字硬编码了、日期格式写死了、货币符号不支持。L10N 是本地化（localization），指**针对某个具体地区做的适配**——比如某个国家的日期习惯、法规文案、计量单位。前者是**架构和代码问题**，改起来伤筋动骨，要在设计阶段解决；后者是**内容和配置问题**，改起来相对轻。混在一起，会导致两个后果：把架构问题当成翻译问题，一直修不好；或者把内容问题上升成架构问题，白花大力气。

第二条，**试图隐藏你的 Bug**。这一条前面讲过，这里再说一层：隐藏的动机通常来自考核压力。如果缺陷数跟绩效挂钩，隐藏就成了理性选择。所以治这个现象，靠喊口号没用，得**改考核口径**。

第三条，**解决和关闭你自己的 Bug**。这是权限问题，13.3.1 已经强调"只有测试人员有关闭权限"。让开发自己关，等于取消了验证环节。补充一句：**测试人员也不该"关闭自己发现的、自己修的问题"**——原则是"谁改谁不判"。

第四条，**未经诊断发布一个补丁**。这条最要命。"未经诊断"是指还没搞清原因就上线一个改动，常见于生产事故的紧急处理。后果是：原来的问题可能没解决，还引入了新问题，而且因为情况紧急、没有回归，新问题的破坏力更大。正确做法是：**先止血（回滚、限流、降级），再定位，再修，再验证，再发布**。

这四条有个共同点：**它们都是在"省一步"**。省掉区分、省掉暴露、省掉验证、省掉诊断。缺陷管理里省下的每一步，最后都会以更高的代价还回来。

> **板书：**四条竖排：① I18N（国际化能力：架构/代码）≠ L10N（本地化适配：内容/配置）② 不隐藏 Bug（动机多来自考核）③ 不自行解决并关闭自己的 Bug（谁改谁不判）④ 未经诊断不发补丁（先止血→定位→修→验证→发布）。底部："省下的每一步，都会加倍还回来"。

提问：生产出了故障，最快的止血动作是什么？为什么不是"马上改代码发补丁"？

预设回答：

- 答"回滚"——最常见的正确止血，代价最小、最可预测。

- 答"限流/降级"——也对，适用于回滚成本高的场景。

- 答"赶紧改代码修"——这正是"未经诊断发布补丁"的坑：改动本身是新的风险源，而且此时你还没搞清原因。

- 收口：**止血和修复是两个动作**，先恢复可用性，再解决问题。

> **易错点 / 考点：**① I18N 与 L10N 的区别是本节最可能出题的点（国际化能力 vs 本地化适配）；② 常考判断："开发可以关闭自己修的缺陷"——错；③ "先止血再定位"的应急处置顺序是案例题得分点；④ 记住"隐藏 Bug 的根因常在考核口径"。

> **过渡：**这些"必须做"和"不要做"，最终要靠一个系统来承载——13.3.5 缺陷跟踪系统。

【13.44】13.3.5 软件缺陷跟踪系统（2 分钟）

这一页给了缺陷跟踪系统的六项作用，我把它归成三类能力：**沟通能力、分析能力、管控能力**。

先看沟通能力。第一，**推动团队内部的有效沟通**。系统是一个共享的事实来源——有争议时不用互相回忆"当时谁说的什么"，打开单子看记录。第六，**报告任何一个 Bug 的当前状态**。这一条是沟通的基础：任何人任何时刻都能查到"这事到哪一步了"，不用去问人。

再看管控能力。第三，**按照优先级排列重要 Bug**。系统能按优先级、严重性排序和筛选，让团队先看要命的。第四，**跟踪任何一个 Bug 的整体生命周期**。从新建到关闭，每一步都在。第五，**记录任何跟 Bug 有关的操作**。这是审计能力——谁改的状态、什么时候改的、改之前的备注是什么，全留痕。这一条在出争议时价值极大。

最后是分析能力。第二，**提供报表和分析**。系统把数据聚合成趋势、分布、年龄这些报表，支撑 13.3.2、13.3.3 和 13.4 的所有分析。

六项能力里，我认为最容易被低估的是第五项——**记录操作**。很多团队只看重"能不能提缺陷"，不看重"有没有审计日志"。可一旦出现"这条缺陷是谁关的、为什么关"的争议，没有日志就说不清；一旦要做过程改进，没有日志就没法复现当时的决策过程。

一句话：**缺陷跟踪系统的价值不在于"存",而在于"推"**——推动沟通、推动处理、推动决策。

再补一层：这六项能力其实构成一个成熟度阶梯——先能"存"（记录），再能"看"（报表），再能"推"（流程与提醒），最后能"学"（分析与改进）。多数团队停在第二步。

> **板书：**六项归三类。沟通：推动团队沟通・报告当前状态。管控：按优先级排列・跟踪全生命周期・**记录所有操作（审计）**。分析：提供报表与分析。底部："价值不在'存'，在'推'"。

> **提问：**为什么"记录任何跟 Bug 有关的操作"这项能力这么重要？

预设回答：

- 答"方便追责"——能追，但主要价值不是追责，而是**还原决策过程**。

- 答"出争议时能说清"——对，这是审计价值。

- 答"做过程改进时能复盘"——也对，日志是复盘的一手材料。

- 收口：**没有日志的系统只是电子表格，有日志的才是管理系统**。

易错点 / 考点：① 六项作用要能列举；② 常考判断："缺陷跟踪系统就是缺陷数据库"——不完整，还包括流程推动、报表分析、操作审计；③ 三类能力的归类方式便于记忆；④ 与 13.1.3 测试过程管理工具的区别：前者管测试过程，后者专管缺陷。

过渡：看一个真实的跟踪系统长什么样。

【13.45】示例（2 分钟）

这一页是缺陷跟踪系统的界面示例。看这类界面，我建议你专门找五样东西，因为它们对应上一页的六项能力。

第一样，**缺陷列表与筛选条件**。列表页通常能按项目、状态、严重性、优先级、指派人、时间筛选。筛选器就是"管控能力"的入口——能不能一键筛出"我名下所有未处理的严重缺陷"，决定了这个系统好不好用。

第二样，**一条缺陷的字段区**。对照 13.2.4 那张字段表看：有没有环境、版本、前提、步骤、期望、实际、频率、严重性、优先级、指派人。字段残缺的系统，会逼着大家在备注里补信息，最后数据无法统计。

第三样，**状态流转按钮**。看它能从当前状态走到哪些状态。按钮就是流程——如果界面上有"开发直接关闭"的按钮，那"只有测试能关闭"这条制度就是废的。**制度要靠系统强制，不能靠自觉**。

第四样，**操作历史/变更记录**。看它记不记"谁在什么时候把状态从 A 改成 B、备注是什么"。这就是上一页说的审计能力。

第五样，**关联关系**。看缺陷能不能关联到用例、需求、构建包、其它缺陷（重复/相关）。第 11 章讲的追溯，在这里体现为"缺陷→用例→需求"的链路，它能支撑"哪些需求因为缺陷被阻塞"这类判断。

五样看完，你对这个系统的评价就有依据了：**字段决定数据质量，按钮决定流程刚性，日志决定可信度，关联决定分析深度**。

> **板书：**五样：① 列表与筛选（管控入口）② 字段区（对照 13.2.4）③ 状态流转按钮（制度强制）④ 操作历史（审计）⑤ 关联关系（缺陷↔用例↔需求↔构建包）。底部："字段→数据质量；按钮→流程刚性；日志→可信度；关联→分析深度"。

提问：如果系统里允许开发直接把缺陷状态改成"已关闭"，你会怎么处理？

预设回答：

- 答"跟大家说一声别这么干"——靠自觉，撑不过一个赶版本的深夜。

- 答"改系统权限配置，去掉开发的关闭权限"——正确，制度要靠系统强制。

- 答"无所谓，反正测试能看出来"——看不出来，关闭的缺陷不会出现在待验证列表里，很容易被漏掉。

- 收口：**能被绕过的流程，等于没有流程**。

易错点 / 考点：① 五样要素与能力的对应；② 常考判断："流程写进制度文档就会被执行"——错，要靠系统强制；③ 缺陷与用例、需求的关联是第 11 章追溯的延续，综合题常考；④ "字段残缺 → 数据无法统计"是简答可用的因果链。

过渡：具体到一个开源工具，我们从 MantisBT 看起——它也是实验 9 的主角。

【13.46】MantisBT 主要功能（2 分钟）

这一页是 MantisBT 的功能界面。MantisBT 是一款经典的开源缺陷跟踪系统，实验 9 就要求你装上并用起来，所以我按"你会用到什么"讲。

第一块，**缺陷的提交与字段配置**。MantisBT 允许管理员自定义字段：哪些必填、哪些显示、下拉选项有哪些。这就直接对应"清晰设置提交必须的信息"那条实践——你可以把"环境""版本"设成必填，从系统层面保证数据质量。

第二块，**状态与工作流**。它有比较完整的状态机：新建、已确认、已分配、已解决、已关闭，还有反馈、挂起这类中间态。管理员可以配置"谁能从哪个状态改到哪个状态"——这就是"只有测试能关闭"这条制度的落地点。

第三块，**过滤器与视图**。可以保存常用筛选（比如"我负责的未解决缺陷"），也可以订阅——缺陷有变化时自动发邮件。这个功能让"实时监控"变成可能，不用天天手动刷。

第四块，**报表与图表**。内置按状态、严重性、模块、时间等的统计图表，也有汇总报表。这就是"提供报表和分析"。

第五块，**权限与角色**。报告者、开发者、管理员等角色权限不同，配合工作流实现职责分离。

第六块，**通知与集成**。邮件通知、与版本控制系统（如 Git/SVN）关联，提交代码时可以引用缺陷编号，这样"哪个缺陷由哪次提交修复"就有据可查——这正好补上 13.2.4 里"构建包跟踪"这一环。

你上实验 9 的时候，别只做到"能提交一条缺陷"就交差。建议按这条线验收：**配好字段 → 配好工作流（尤其关闭权限）→ 建好过滤器 → 看一次报表 → 走一遍完整生命周期**。走完这五步，你才算真的会用缺陷跟踪系统。

顺便说一句题外话：这些老牌开源项目今天多数已不再是主流选择，但它们的设计思想——状态机、权限、通知、报表——被后来所有工具继承下来了。**学工具要学它的模型，不要只学它的按钮。**

板书：六块：① 提交与字段配置（必填项＝数据质量）② 状态与工作流（谁能改到哪＝制度落地）③ 过滤器与订阅（实时监控）④ 报表与图表（分析）⑤ 权限与角色（职责分离）⑥ 通知与集成（引用缺陷号＝构建包跟踪）。实验验收五步：字段→工作流→过滤器→报表→全生命周期。

提问：MantisBT 里为什么要把"谁能把缺陷改成已关闭"做成可配置项，而不是写死在代码里？

预设回答：

- 答"方便灵活"——是，但更重要的是不同团队职责划分不同，工具要能适配流程，而不是逼流程适配工具。

- 答"默认配置通常已经限制开发关闭"——对，但也需要管理员理解这条配置的意义，否则容易被改松。

- 答"配置太麻烦，不如默认就行"——默认往往偏宽松，一松就等于没有制度。

- 收口：**工具的默认值，往往决定团队的实际流程**——这是选型和配置时最该注意的一点。

易错点 / 考点：① MantisBT 的核心能力（字段配置、工作流、过滤器、报表、权限）；② 常考判断："缺陷跟踪系统的工作流是固定的"——多数可配置；③ 实验 9 的验收标准要能说出"走完一次完整生命周期"；④ 与版本控制集成的意义（缺陷↔提交↔构建包）是加分点。

过渡：开源世界里这类工具不止一个，课件列了一批，我们过一下名字和定位。

【13.47】开源缺陷跟踪系统（2 分钟）

这一页列了六个开源缺陷跟踪系统，我按"你可能遇到"的顺序讲。

Mantis（mantisbt.sourceforge.net）——就是刚讲的 MantisBT，PHP 写的，部署简单、概念清晰，教学和小团队最常见，实验 9 用它。

Bugzilla（mozilla.org/projects/bugzilla/）——Mozilla 出品，历史非常悠久，功能强大、查询能力强，很多大型开源项目用它。它偏重"海量缺陷的查询与统计"，但界面和配置相对传统。顺带一提，Jira 里说的"bug"这个词的用法，和 Bugzilla 的普及有很大关系。

Bugzero（bugzero.findmysoft.com）——商业化产品加开源版本，特点是支持多项目和邮件集成，配置较轻。

Scarab（scarab.tigris.org）——源于 Eclipse 社区体系的项目，强调可定制性，适合想深度改造的团队。

TrackIT（trackit.sourceforge.net）——轻量级，功能聚焦，适合只需要基本跟踪能力的场景。

Itracker（itracker.org）——基于 J2EE 的老牌开源项目，企业环境里曾有较多使用。

六个名字，你不必都记住，但要记住**选型的判断维度**：技术栈是否匹配（PHP/Java）；部署与运维成本；工作流可定制程度；报表能力；是否还在维护（这一条最实际——很多老项目已经多年不更新，装上去有安全风险）；社区活跃度。

还有一句实在话：今天很多团队不再自建，而是用 SaaS 或综合平台（比如前面讲的 Jira、MeterSphere）。**开源工具的价值，一半在功能，一半在"你能看懂它的流程"**——对于学习来说，装一套开源系统，把状态机、权限、过滤器都摸一遍，比看十页文档都管用。

再补一句：这些老牌开源项目今天多数已不再是主流选择，但它们的设计思想——状态机、权限、通知、报表——被后来所有工具继承下来了。**学工具要学它的模型，不要只学它的按钮。**

> **板书：**六个名字＋一句定位：Mantis（PHP・教学常用・实验 9）｜Bugzilla（Mozilla・查询强・大型开源项目）｜Bugzero（多项目・邮件集成）｜Scarab（可定制）｜TrackIT（轻量）｜Itracker（J2EE・老牌）。选型维度：技术栈・部署成本・工作流定制・报表・**是否还在维护**・社区活跃度。

提问：选缺陷跟踪系统时，为什么"是否还在维护"这一条这么关键？

预设回答：

- 答"不维护就没新功能"——是小事。真正的问题是**安全漏洞没人修**，而缺陷系统里存的是产品和客户的敏感信息。

- 答"出问题没人支持"——也对，自建团队要自己扛。

- 答"能用就行，反正只是内部系统"——内部系统被攻破，往往比外部系统更致命，因为它连着内网和代码库。

- 收口：**选型时"活着"比"功能多"更重要**。

> **易错点 / 考点：**① 六个工具的名称与大致定位（Mantis、Bugzilla 是必记的两个）；② 选型维度要能说出四条以上；③ 常考判断："开源工具一定比商业工具差"——看场景；④ 课件注明"更多工具详见公众号文章"，说明这里的名单不是穷举，考点在**选型方法**而非名单。

> **过渡：**到这里，13.3 讲完了"怎么管"。接下来 13.4 要回答一个更硬的问题：**用这些数据，怎么算质量**。

【13.48】13.4 产品质量评估与度量（1 分钟）

13.4 是本章最"数学"的一节，核心只有一句话：**质量不能靠感觉，得靠算**。

四个小节是这样排的。**13.4.1 基于缺陷的质量度量**——用缺陷数据算质量：缺陷密度、缺陷清除率、缺陷逃逸率，还有趋势和分布这两个判据。**13.4.2 经典的种子公式**——一个很聪明的估算方法：往程序里"种"已知缺陷，看你能捞出多少，反推还剩多少。**13.4.3 基于缺陷清除率的估算方法**——用功能点、研发期缺陷、发布后缺陷，算质量、注入率、清除率。**13.4.4 软件质量的度量**——把多个质量因素加权合成一个综合分。

这一节的逻辑是递进的：**先算单指标（密度、清除率）→ 再估总量（种子公式）→ 再估整体质量（清除率估算）→ 最后做综合评价（加权模型）**。

我要提前打个预防针：这一节公式看着多，但重点不在推导，而在**每个指标回答什么问题、什么时候会骗人**。比如缺陷密度低可能是质量好，也可能是测试太浅；清除率高可能是修得好，也可能是发现得少。**任何单一指标都会骗人，指标必须成组看**。

记住这句，13.4 就通了：**度量的目的不是打分，是发现问题、支撑决策。**

> **板书：**四小节递进：13.4.1 单指标（密度・清除率・逃逸率・趋势・分布）→ 13.4.2 估总量（种子公式）→ 13.4.3 估整体（清除率估算）→ 13.4.4 综合评价（加权模型）。底部："单一指标会骗人，指标要成组看"。

> **提问：**如果一个项目的缺陷密度很低，你会说它质量好吗？

> **预设回答：**

- 答"会，密度低说明缺陷少"——可能是质量好，也可能是**测试太浅没测出来**。

- 答"要看测试覆盖率和逃逸率"——正确，成组看才有意义。

- 答"要看跟历史基线比"——也好，课件后面专门有一页讲"缺陷度量-基线"。

- 收口：**分母不动、分子小，可能不是好事**；要看它是在什么测试强度下得到的。

> **易错点 / 考点：**① 四小节名称与逻辑递进；② 常考判断："缺陷密度越低质量越好"——不成立，需结合覆盖率/逃逸率；③ 记住"指标要成组看"这个方法论，简答常考。

> **过渡：**先看度量的整体框架——度量分几类、按什么过程做。

【13.49】软件度量的分类与过程（2 分钟）

这一页是度量的"元问题"：度量分几类、按什么步骤做。先分类。

课件给三类：**软件过程度量、项目度量、产品质量度量**。三者的区别用一句话说清：**过程度量看"活干得怎么样"**（评审有效性、缺陷注入率、测试效率）；**项目度量看"项目进展如何"**（进度偏差、工作量、成本）；**产品质量度量看"东西好不好"**（缺陷密度、可靠性、复杂度）。本章 13.4 主要讲第三类，但会用前两类的数据。

再看过程，五步。**第一步，识别目标**：分析出度量的工作目标和列表，并由管理者审核确认。注意"审核确认"——度量目标要有人认账，否则做出来的数据没人用。这一步最关键的问题是：**我们为什么要量这个？量出来准备做什么决定？**答不上来就别量。

第二步，定义度量过程：定义收集要素、收集过程、分析、反馈过程、IT 支持体系，为具体的收集活动、分析、反馈活动和 IT 设备/工具开发提供指导。说白了就是**把"怎么收、怎么算、怎么反馈"写成规范**，否则每个人算法不一样，数据不可比。

第三步，搜集数据：应用 IT 工具进行数据收集工作，并按指定的方式审查和存储。关键词"审查"——数据进来要校验，脏数据进去，结论一定错。

第四步，数据分析与反馈：根据数据收集结果，按已定义的分析方法进行数据分析，完成规定格式的图表，进行反馈。注意"**反馈**"是个动作，不是"发个邮件"——要有人看、有人讨论、有人据此决策。

第五步，过程改进：根据度量的分析报告，管理者基于度量数据做出决策。**这是整个循环的目的**——如果度量不产生改进动作，那五步全白走。

这五步是一个闭环：目标 → 定义 → 收集 → 分析反馈 → 改进，然后改进之后重新定目标。它和 13.3 的缺陷跟踪是同一套思路，只是一个管"缺陷"，一个管"度量"。

> **板书：**分类三类：过程度量（活干得怎么样）／项目度量（进展如何）／产品质量度量（东西好不好）。过程五步：识别目标（管理者审核确认）→ 定义度量过程（收集要素・收集过程・分析・反馈・IT 支持）→ 搜集数据（工具・审查・存储）→ 分析与反馈（规定格式图表・真反馈）→ 过程改进（决策）。底部："量的目的：支撑决策；不产生改进的度量＝浪费"。

提问：如果领导说"我们把这些指标都量起来吧"，你该先问什么？

预设回答：

- 答"量哪些指标"——顺序反了，先问目的。

- 答"量出来准备做什么决定"——正确。这就是第一步"识别目标"。

- 答"有没有工具"——是后面的问题，工具服务于目标。

- 收口：**没有决策场景的度量，最后都会变成填表负担**。

易错点 / 考点：① 三类度量的名称与区别；② 五步过程要能按顺序说出，尤其"识别目标"在最前；③ 常考判断："度量就是收集数据"——不完整，是五步闭环，终点是改进；④ "数据分析与反馈"里的"反馈"是动作，是易漏点。

过渡：框架清楚了，进入第一个具体指标组——基于缺陷的质量度量。

【13.50】13.4.1 基于缺陷的质量度量（3 分钟）

这一页给了五个指标，是本章最实用的一组。我逐个讲定义、用途和坑。

第一，**代码的缺陷密度**：**每千行代码（KLOC）的缺陷数（Bug#/KLOC）**。用途是横向比较不同模块、纵向比较不同版本。坑在哪？**它受语言、代码风格、注释率影响**——同样功能，Java 和 Python 的行数差一倍，密度就没法直接比。所以比较要在同语言、同团队、同统计口径下做。

第二，**缺陷清除率**：**各个阶段的缺陷清除率和总的缺陷清除率**。它的含义是"在某阶段之前，我们清掉了多少

比例的问题"。分阶段看很有价值：如果需求阶段的清除率很低，说明需求评审形同虚设，问题都漏到后面去了。

第三，**缺陷逃逸率**：课件定义得很明确——**未在研发阶段发现的缺陷意味着逃逸出去，即线上发现的缺陷（生产环境发现的缺陷）**。公式是：**缺陷逃逸率 = 线上发现的缺陷数 /（线上发现的缺陷数 + 研发环境下发现的缺陷数）**。这个指标我认为是**最能反映测试有效性**的一个：它不看你测了多少，只看你漏了多少。而且它的分母包含了线上和研发两侧，比单看线上数量更公平（大项目线上缺陷多，可能只是因为它大）。

第四，**缺陷趋势是否良好，是否收敛**。这是把 13.3.2 的趋势分析作为质量判据——收敛说明过程稳定。

第五，**缺陷分布是否符合正态分布？是否过于集中在 1-2 个模块？**这是把 13.3.3 的分布分析作为判据。注意课件用了问句，说明这是"判据"不是"定律"——高度集中要警惕，绝对均匀也未必好。

五个指标的关系我这样归纳：**密度看"浓不浓"，清除率看"拦得住吗"，逃逸率看"漏了多少"，趋势看"稳不稳"，分布看"偏不偏"**。前三个是数字，后两个是形状。

> **板书：**五行。缺陷密度＝Bug#/KLOC（坑：语言・风格・口径）。缺陷清除率＝各阶段＋总的（低 → 评审失效）。**缺陷逃逸率＝线上/(线上＋研发)**（最能反映测试有效性）。趋势：是否收敛。分布：是否过于集中 1–2 模块。底部："浓不浓・拦得住吗・漏多少・稳不稳・偏不偏"。

提问：缺陷逃逸率的分母为什么要包含"研发环境下发现的缺陷"，只算线上缺陷数不行吗？

预设回答：

- 答"行，线上缺陷数少就是好"——不行。一个大项目线上 50 个、小项目线上 5 个，直接比数量对小项目不公平。

- 答"因为要算比例才有可比性"——正确。分母把"发现的总量"考虑进来，得到的是"漏出去的比例"。

- 答"分母大说明测试强，逃逸率就低"——这个逻辑对，但要注意：分母大也可能是"提得多但都是小问题"，所以要结合严重性看。

- 收口：**比率指标的意义在于可比**，绝对数量只能看自己。

易错点 / 考点：① 缺陷密度的单位（Bug#/KLOC）与局限；② **缺陷逃逸率的公式必须能默写**（线上 /（线上＋研发））， 是本章最高频计算考点之一；③ 清除率与逃逸率的方向相反（越高越好 vs 越低越好），易混；④ 常考判断："缺陷密度可以直接跨语言比较"——错。

过渡：有了指标，还得有个"参照系"才知道好坏——这就是基线。

【13.51】缺陷度量-基线（2 分钟）

这一页讲"基线"。什么叫基线？简单说就是**一个被认可的参照值**，用来判断"这次的数字是好还是坏"。

为什么必须要基线？因为指标本身没有好坏。缺陷密度 2 个/KLOC 是好是坏？不知道——如果你的历史基线是 1.5，那 2 就是退步；如果历史基线是 4，那 2 就是大幅改善。**没有基线的数字，只能看不能判**。

基线怎么来？通常三条路。第一条，**历史数据**：把过去几个版本的同口径指标算出来，取一个区间。这是最常用的。第二条，**行业基准**：参考同类型产品的公开数据或行业报告。这条路要谨慎，因为口径差异大。第三条，**项目目标**：由项目组和管理层事先约定，比如"逃逸率控制在 5% 以内"。这一条最有效，因为它直接连着决策。

基线怎么用？关键在"**同口径**"。如果上一个版本按"提交数"算缺陷，这个版本按"去重后数"算，两个数字没法比。所以我建议团队把**度量口径文档化**：分子是什么、分母是什么、什么算缺陷、什么不算、统计截止到哪一天。这份文档比任何工具都重要。

还有一个容易被忽略的点：**基线要随项目成熟度调整**。一个刚起步的团队缺陷密度高是正常的；随着评审、自动化、规范建立起来，基线应该逐步收紧。基线一成不变，就等于没有目标。

一句话：**基线是度量的"尺子"，尺子不准，量出来全是错的。**

> **板书：**中间写"基线＝被认可的参照值"。来源三条：历史数据（最常用）｜行业基准（慎用，口径差异）｜项目目标（最有效，连着决策）。使用要点：**同口径**（分子・分母・什么算缺陷・截止时间）→ 文档化。底部："基线随成熟度收紧"。

提问：两个版本缺陷密度分别是 3.2 和 2.1，能不能直接说"质量提升了 34%"？

预设回答：

- 答"能，降了就是提升"——要小心。口径一样吗？代码量统计方式一样吗？测试强度一样吗？

- 答"要先确认统计口径一致"——正确，这是前提。

- 答"还要看测试深度，测得浅密度自然低"——关键补充。密度下降可能是"测得少"而不是"错得少"。

- 收口：**比数字之前，先比口径和测试强度**。

易错点 / 考点： ① 基线的定义与三种来源；② 常考判断："指标可以直接跨版本/跨项目比较"——需同口径；③ "口径文档化"是可写入答题的实践建议；④ 基线随成熟度调整是加分点。

过渡： 如果连"总共有多少缺陷"都不知道，还能估质量吗？课件给了一个很巧的方法。

【13.52】13.4.2 经典的种子公式（3 分钟）

这一页是本章最有意思的一个方法——**种子公式**。它解决一个很实际的问题：**你怎么知道还有多少缺陷没被发现？**

思路是这样的：往程序里**人为植入一批已知的缺陷，叫"种子"**。然后让测试照常去测。测完之后，你数两个数：**测出了多少个种子（s）**，以及**测出了多少个非种子缺陷（n）**。同时你知道**总共植入了多少种子（S）**。

逻辑就出来了：如果测试能捞出 s/S 比例的种子，那它应该也能捞出同样比例的真实缺陷。所以：

$$N = S \times n / s$$

其中 N 是程序里**全部非种子缺陷的估计总数**。这就是课件的公式。推导很好理解：捕到种子的比例 s/S 应该等于捕到真实缺陷的比例 n/N，交叉相乘就得到 N = S · n/s。

课件接着给了一个判据：**如果 n = N，说明所有的 Bug 已找出来，说明做的测试足够充分**。也就是说，当"测出的真实缺陷数"等于"估计的真实缺陷总数"时，说明基本捞干净了。

它还给了**信心指数 C**，用一个百分比表示，**值越大说明对产品质量的信心越高，最大值为 1**：

当 $n > N$ 时，$C = 1$；当 $n \leqslant N$ 时，$C = S/(S-N+1)$。

第一个式子好理解：测出来的比估计的还多，说明估计偏保守，信心拉满。第二个式子是课件给的形式，你可以把它理解成"基于种子回收情况的一个保守估计"——注意这个式子里 S、N 的量纲要一致，代值时别把"总数"和"测出数"混用。

这个方法的**优点**是：不需要知道真实缺陷总数，也能估出量级；**局限**也要说清：一是种子要"像真的"——如果种子都是显眼的低级错误，测出来比例虚高，N 就被低估；二是植入种子会改动代码，需要严格管理，测完要移除干净；三是它假设"测试对种子和真实缺陷的发现能力相同"，这在现实中只是近似。

用一句话记住它：**用已知的漏，推未知的漏**——这就是所有估算方法的共同智慧。

> **板书：**左：已知量 s（测出的种子）／n（测出的非种子）／S（全部种子）／N（全部非种子，待估）。右：**N = S × n / s**。判据：n = N → 全找出 → 测试充分。信心指数 C：n>N → C=1；n≤N → C=S/(S−N+1)，最大 1。底部："用已知的漏，推未知的漏"。

提问：植入 100 个种子，测出 60 个种子、同时测出 90 个真实缺陷。估计真实缺陷总数是多少？

预设回答：

- 答"按公式 N = 100 × 90 / 60 = 150"——正确，直接代公式。

- 答"还剩 60 个没找到"——也对，150 − 90 = 60，这是"估计剩余量"的用法，很实用。

- 答"应该等于 90，因为测出来多少就是多少"——这是常见误解，把"已发现"当"全部"，正是这个公式要纠正的。

- 收口：这个公式最大的价值，是让你知道"**没找到的那部分大概有多大**"——而这恰恰是发布决策最需要的信息。

易错点 / 考点：① 公式 **N = S • n/s** 必须能默写并会代值计算；② 四个符号的含义不能混（s 测出种子、n 测出非种子、S 全部种子、N 全部非种子）；③ 信心指数分段定义（n>N → 1；n≤N → S/(S−N+1)）；④ 常考判断："种子公式能精确算出缺陷总数"——是**估算**，有前提假设。

过渡：种子公式要动代码，还有个不动代码的办法——从清除率反推质量。

【13.53】13.4.3 基于缺陷清除率的估算方法（3 分钟）

这一页给三个公式，用的都是同一组输入量。先把符号钉死，后面就不会乱。

课件定义：**F 为描述软件规模用的功能点；D1 为在软件开发过程中发现的所有缺陷数；D2 为软件发布后发现的缺陷数；D 为发现的总缺陷数，D = D1 + D2**。

注意 D1 和 D2 的分界线是"**发布**"这个时间点。这个分界特别关键，因为所有"研发侧 vs 线上侧"的对比都靠它：D1 是"我们自己抓到的"，D2 是"用户帮我们抓到的"。

然后三个指标。

第一，**质量 = D2 / F**。含义是**每个功能点摊到多少发布后缺陷**。注意它的方向：**越小越好**。这个指标的好处是把"规模"消掉了——大项目发布后缺陷多很正常，除以功能点才有可比性。

第二，**缺陷注入率 = D / F**。含义是**每功能点注入了多少缺陷**（研发期 + 发布后的总和）。这个指标衡量的

是"**我们的开发过程有多容易产生缺陷**"，它反映过程能力而不是测试能力。注入率高，说明需求理解、设计、编码环节问题多，光靠测是救不回来的。

第三，**整体缺陷清除率 = D1 / D**。含义是**在所有被发现的缺陷里，有多少是在发布前被清掉的**。这个指标直接衡量测试与评审的有效性，**越大越好**。

三个指标的关系要讲清：**注入率看"产生多少"，清除率看"拦住多少"，质量（D2/F）看"漏到用户手里多少"**。它们不是孤立的：如果注入率高、清除率也高，说明"错得多但拦得住"，能发布但要治本；如果注入率低、清除率也低，说明"错得不多但漏得多"，问题在测试环节；最糟的是注入率高、清除率低，那就是"边错边漏"。

我给一条实践建议：这三个指标要和 13.4.1 的逃逸率一起看。逃逸率 = D2/(D1+D2)，而清除率 = D1/(D1+D2)，**两者相加恰好等于 1**（在口径一致的前提下）。所以你只要算一个，另一个就出来了——这个互补关系是理解这两个指标的最好方式。

板书：符号：F＝功能点；D1＝研发期发现缺陷；D2＝发布后发现缺陷；D＝D1+D2。三个公式：**质量 = D2/F**（越小越好）｜**缺陷注入率 = D/F**（过程能力）｜**整体缺陷清除率 = D1/D**（测试有效性，越大越好）。底部：清除率 + 逃逸率 = 1（同口径）。

提问：某项目 D1=180，D2=20，F=200。清除率、逃逸率、质量、注入率各是多少？

预设回答：

- 答"清除率 = 180/200 = 90%，逃逸率 = 20/200 = 10%"——正确，而且验证了互补关系。
- 答"质量 = 20/200 = 0.1（每功能点 0.1 个发布后缺陷）"——正确。
- 答"注入率 = 200/200 = 1（每功能点 1 个缺陷）"——正确。
- 追问：这个项目好不好？答"要跟基线比"——对，0.1 这个数孤立看没有意义。
- 收口：**公式不难，难在会解读、会对比**。

易错点 / 考点：① 四个符号的定义（尤其 D1/D2 以"发布"为界）；② 三个公式必须能默写与代值；③ 方向易混：质量（D2/F）越小越好、清除率越大越好、注入率反映过程能力；④ 清除率与逃逸率互补（和为 1）是高分点。

过渡：上面这些是"缺陷视角"的度量。再往上一层，是把多个质量因素合成一个综合分。

【13.54】13.4.4 软件质量的度量（2 分钟）

这一页讲的是"综合度量"。前面那些指标都是一个侧面，这一页要把多个侧面合成一个数。

先看课件列的四类度量：**软件可靠性度量、复杂度度量、缺陷度量和规模度量**。这四类基本覆盖了质量度量的主要方面：可靠性看"能不能稳定运行"（比如平均无故障时间、故障率）；复杂度看"结构有多难维护"（圈复杂度、耦合度）；缺陷度量就是前面讲的那一组；规模度量看"有多大"（代码行、功能点）。规模本身不是质量，但它是所有比率指标的分母。

再看合成公式：**Mi = c1×f1 + c2×f2 + … + cn×fn**。课件解释：**Mi 是一个软件质量因素（如 SQRC 层各项待计算值），fn 是影响质量因素的度量值（如 SQDC 层各项估计值），cn 是加权因子**。

这个公式的形式很朴素——**加权求和**。但里面有三个必须想清楚的问题。

第一，**选哪些 f**。选错了，合成出来的是"伪质量"。比如你只选"缺陷密度"和"代码行数"，那这个分数基本等于在奖励"代码写得少"。

第二，**权重 c 怎么定**。权重代表"你认为什么更重要"，这是价值判断，不是技术判断。所以权重必须由项目干系人共同确认，不能由测试一家定。

第三，**量纲要统一**。f 有的可能是"个数"、有的可能是"百分比"、有的可能是"秒"，直接加权相加是没有意义的，必须先**归一化**到同一量纲（比如都映射到 0–100）。这一点课件没有展开，但你在实际用的时候一定会遇到。

关于 SQRC、SQDC 这两个缩写，课件是作为示例提到的层级概念——一个是质量因素层，一个是度量值层。你抓住"**下层算度量值、上层算加权因素**"这个结构就够了，具体分层以教材为准。

一句话：**加权求和容易，选对因素和权重才是本事。**

> **板书：**四类度量：可靠性／复杂度／缺陷／规模。公式：**Mi = c1 · f1 + c2 · f2 + … + cn · fn**（Mi＝质量因素值，fn＝度量值，cn＝加权因子）。三个必须想清的问题：选哪些 f（别选成"奖励代码少"）｜权重谁定（干系人共识，价值判断）｜**量纲归一化**。

提问：为什么加权之前必须先做归一化？

预设回答：

- 答"因为单位不一样"——正确。缺陷密度是"个/KLOC"，可靠性可能是"小时"，直接相加等于把苹果和小时加

在一起。

- 答"归一化就是除以最大值"——是一种常见做法（极值归一），也可以是映射到区间、Z-score 等，方法可选，但必须有。

- 答"不归一化也能比大小"——不能，权重的意义会被量纲吞掉：量纲大的指标自动变成主导。

- 收口：**量纲不统一时，权重是假的**——你以为是 0.6 比 0.4，实际可能是 600 比 0.4。

> **易错点 / 考点：**① 四类度量的名称；② 公式 Mi = Σci · fi 的形式与三个符号含义；③ 常考判断："加权求和可以直接用原始值"——必须先归一化；④ 权重是价值判断、需干系人确认，这个观点常作为简答加分点。

> **过渡：**具体到缺陷，业界还有一套很细的错误分类，用来做质量统计。

【13.55】质量度量的统计方法（3 分钟）

这一页给了十二类错误分类，是一套经典的缺陷分类体系。我按"从上游到下游"的顺序讲，你会发现它其实是一条**缺陷产生链**。

最上游是**需求与沟通**：**IES——说明不完整或说明错误**；**MCC——与客户交流不够所产生的误解**。这两类都是"我们没搞清要做什么"，一个出在文档上，一个出在沟通上。它们是最贵的缺陷，因为一旦错了，后面全错。

第二层是**故意偏离**：**IDS——故意与说明偏离**。注意"故意"两个字，意思是明知规格这么写，实现时却按别的做（比如图省事、赶工期）。这一类性质最特殊，它不只是技术问题。

第三层是**编码规范与数据**：**VPS——违反编程标准**；**EDR——数据表示有错**（类型、精度、编码、格式）；**IMI——模块接口不一致**（参数个数、类型、顺序、语义）。后两个是集成阶段缺陷的主要来源。

第四层是**设计与逻辑**：**EDL——设计逻辑有错**。这是最"硬"的一类，往往是算法、流程、状态机的错误，改起来影响面大。

第五层是**测试与文档**：**IET——不完整或错误的测试**（测试本身有缺陷，漏测或测错）；**IID——不准确或不完整的文档**。IET 特别值得说：**测试也会产生缺陷**，一个错的用例会给你一个错的结论，甚至掩盖真问题。

第六层是**实现转换**：**PLT——将设计翻译成程序设计语言中的错误**。也就是"设计是对的，写代码时写错了"。

还有两类兜底：**HCI——不清晰或不一致的人机界面**；**MIS——杂项**。

十二类看着多，但它们的用途只有一个：**统计各类占比，找出主要矛盾**。如果 IES 和 MCC 占了 40%，说明该加强需求评审和用户沟通，而不是加强单元测试；如果 IMI、EDR 占大头，说明接口契约和数据规范要立起来。**分类的价值不在分类本身，在于它能指向改进动作。**

板书：六层链条竖排。需求沟通：IES（说明不完整/错误）·MCC（与客户交流不够的误解）→ 故意偏离：IDS → 编码规范与数据：VPS·EDR·IMI → 设计逻辑：EDL → 测试与文档：IET·IID → 实现转换：PLT → 兜底：HCI·MIS。底部："分类 → 占比 → 改进动作"。

提问：如果统计发现 IES 和 MCC 两类合计占了 40%，你该建议团队加强什么？

预设回答：

- 答"加强单元测试"——方向错了，这两类是需求侧的问题，测得再细也补不回来。

- 答"加强需求评审和用户沟通"——正确。

- 答"加强编码规范"——也不对，编码规范针对 VPS、PLT 那几类。

- 收口：**缺陷分类的第一用途是"指路"，告诉你资源该投在哪一环**。

易错点 / 考点：① 十二个缩写的含义要能对上（至少记 IES、MCC、EDL、IET、PLT、IMI 六个）；② 常考判断："测试不会产生缺陷"——错，IET 就是测试本身的错误；③ IES 与 MCC 的区别（文档不完整/错误 vs 沟通不足导致误解）是高频辨析；④ 分类到改进的映射是简答得分点。

过渡：这十二类在教材里还有更细的统计表，我们看一眼它的组织形式。

【13.56】质量度量的统计方法 – 续（2 分钟）

这一页是统计方法的续表，我借它讲"**怎么用这套分类做统计**"，这是比背缩写更有用的能力。

第一步，**建一张二维表**。行是缺陷分类（IES、MCC、IDS……），列是阶段（需求、设计、编码、测试、发布）。每个格子里填"这个阶段发现了多少这一类缺陷"。这张表一出来，信息量极大。

第二步，**看行**。某一类的总数很高，说明这类错误是主要矛盾——比如 EDL（设计逻辑有错）总数最高，那设计评审必须加强。

第三步，**看列**。某一阶段的总数很高，说明这个阶段"漏进来"的多；某一阶段数量很低甚至为零，要怀疑是不是**这个阶段根本没在查**——比如"需求阶段发现 0 个缺陷"，通常不是需求完美，而是需求评审没做或者做

了不算数。

第四步，**看对角线**。理想情况下，大部分缺陷应该在**引入它的那个阶段或紧邻的下一个阶段**被发现。如果EDR（数据表示有错）大量在"发布后"才被发现，说明数据相关的测试和评审缺失。

第五步，**算清除率**。每个阶段"本阶段发现/本阶段引入"的比例，就是分阶段清除率——这正好接上 13.4.1 的清除率指标和 13.4.3 的整体清除率。

你看，这套统计方法的价值是：**它把"缺陷"从一个笼统的数量，拆成了"哪一类、在哪个阶段、被谁发现"的三维结构**。有了这个结构，改进才有靶子。

提醒一句实操上的坑：**分类要互斥且穷尽**。同一个缺陷不能既算 IES 又算 MCC，否则统计出来会重复计数。所以团队要有一份"分类判定说明"，遇到模糊的先讨论、后固化。这跟 13.2 那些字段的口径问题是一回事——**口径不清，数据全废**。

板书：二维表：行＝缺陷分类（IES/MCC/IDS/VPS/EDR/IMI/EDL/IET/IID/PLT/HCI/MIS），列＝阶段（需求・设计・编码・测试・发布）。五步：看行（主要矛盾）→ 看列（哪阶段漏进/没查）→ 看对角线（是否在引入阶段被发现）→ 算分阶段清除率 → 指向改进。底部："分类要互斥且穷尽（口径说明）"。

提问："需求阶段发现 0 个缺陷"，这句话你信吗？

预设回答：

- 答"信，说明需求写得好"——概率很低。更可能是需求评审没做、或者发现了没记录。

- 答"不信，要查评审记录"——正确。零缺陷的阶段通常意味着"这个阶段没有质量活动"。

- 答"也可能需求阶段确实简单"——有可能，但要结合项目规模和复杂度判断。

- 收口：**统计里的"零"，往往比"高"更值得追问**。

易错点 / 考点：① 二维表的读法（行、列、对角线）；② 常考判断："某阶段缺陷数为零说明该阶段质量高"——错，可能是没有质量活动；③ 分类必须互斥且穷尽，是统计口径的基本要求；④ 与清除率计算的衔接是综合题常考点。

过渡：有了这张分类表，课件给了一套具体的计算公式——阶段错误度量和总体质量度量。

【13.57】质量度量计算（3 分钟）

这一页给两个公式，一个算阶段，一个算总体。公式看着复杂，但结构很清楚。

先看**阶段错误度量**：

PIi = Ws × (Si / Ei) + Wm × (Mi / Ei) + Wi × (Ti / Ei)

课件给的权重是 **Ws、Wm、Wi = 0.6、0.3、0.1**。三个比值共用同一个分母 Ei，我按"分母是什么、分子分三类"来解。

Ei 是这个阶段应该发现的错误总数（期望值）。三个分子按**严重程度**分档：Si 是严重（severe）类，Mi 是中等（medium）类，Ti 是轻微（trivial）类——这就是为什么权重递减：0.6、0.3、0.1。也就是说，**同样漏一个缺陷，漏严重的代价是漏轻微的六倍**。

这个设计的意图很明确：**不能只数缺陷个数，要给严重性加权**。一个阶段发现了 10 个轻微问题、漏了 1 个致命问题，用个数算它是"发现了 90%"，用这个公式算它会明显低于 90%。

再看**总体质量度量**：

Ep = Σ (i × PIi) / Ps

课件给的 i 取值是：**i = 1, 2, 3, 4, 5 分别代表需求分析、设计、编程、测试、发布**，同时括号里注明 **i = 1, 2, 5, 10, 100**。这两组数字不矛盾：前一组是**阶段编号**，后一组是**权重**。也就是说，越靠后的阶段，相同错误率的"代价系数"越大——发布阶段的权重是 100，需求阶段是 1。这正是"缺陷发现越晚，代价越高"的量化表达。Ps 是分母（各阶段权重之和或规模基准），按教材口径取用。

这个公式的教学价值不在算数，在于它把一句工程常识变成了数字：**同样一个错误，晚发现一百倍代价**。这就是为什么第 4 章反复强调"尽早测试"，为什么 13.4.3 强调"清除率"，为什么 13.3.1 强调"评审前置"。

用的时候注意两点：一是权重（0.6/0.3/0.1、1/2/5/10/100）是课件给的示例参数，实际项目应按自己的代价模型校准；二是所有量纲要一致，都是"个数"或都是"比率"，别混。

板书：左：**PIi = 0.6 • (Si/Ei) + 0.3 • (Mi/Ei) + 0.1 • (Ti/Ei)**（Ei＝应发现错误数；S/M/T＝严重/中等/轻微）。右：**Ep = Σ(i × PIi) / Ps**，i 阶段编号：1 需求 • 2 设计 • 3 编程 • 4 测试 • 5 发布；权重 1 • 2 • 5 • 10 • 100。底部："同一个错误，晚发现一百倍代价"。

提问：为什么"发布"阶段的权重给到 100，而"需求分析"只有 1？

预设回答：

- 答"因为发布后修复最贵"——正确。发布后要修，涉及定位、修复、回归、重新发布、用户沟通、可能的赔付和口碑损失。

- 答"因为发布阶段缺陷少"——不是理由，权重反映代价不是数量。

- 答"数字太大了不合理"——它是相对权重，只用于比较阶段之间的代价差异，绝对值没有意义。

- 收口：**权重是"代价的语言"**，它把"越晚越贵"这句常识变成了可计算的量。

易错点 / 考点： ① 两个公式的形式要能写出来；② 权重 0.6/0.3/0.1 与阶段权重 1/2/5/10/100 要能对上含义；③ 常考判断："缺陷数相同则质量相同"——错，要按严重性加权；④ 理解"晚发现代价呈数量级增长"这一核心结论，是本章的价值落点。

过渡： 到这里度量的算法都讲完了。但度量这件事有个更重要的前提——它会不会把团队带偏？

【13.58】度量及其指标（3 分钟）

这一页是 13.4 的收口，也是最该记住的一页。课件列了四个指标，然后给了两句判断。

四个指标：**Designed cases**（设计的用例数）、**Executed cases**（执行的用例数）、**Found bugs/Day**（每天发现的缺陷数）、**Missed bugs**（漏掉的缺陷数）。

这四个指标各有用途也各有陷阱，我一个个说。

Designed cases 看测试设计的工作量。陷阱：数量可以刷——把一条用例拆成五条，数字就上去了，覆盖的东西一点没多。

Executed cases 看执行进度。陷阱：跑了不等于跑对，也不等于跑得深。执行率 100% 可能只是把用例点了一遍。

Found bugs/Day 看发现效率。陷阱最大：这个指标一旦被当成 KPI，"每天必须提 N 个"，就会催生一堆凑数的、重复的、鸡毛蒜皮的缺陷，真正难的问题反而没人啃。而且它天然对**测试早期**有利、对**后期**不利——后期本来就该发现得少。

Missed bugs 看漏测。它是最有价值也最难量的一个——因为你不知道你漏了什么。实践中常用"逃逸到线上的缺陷"来近似，也就是 13.4.1 的逃逸率。

然后是两句话，是这一页的灵魂：**度量可以推动测试流程的改进；不良的度量也可以使测试流程的执行变形。**

"变形"这个词用得非常准。它会以这些形式出现：只提容易提的缺陷、把复杂问题拆成小问题凑数、用例只写不跑、跑用例只跑简单的、把缺陷延迟到下一版本以美化本期数据、把"阻塞"改成"通过"。**所有变形的共同原因是：指标被当成了评价人的工具，而不是改进流程的工具。**

所以用度量有一条铁律：**度量流程，不度量个人；用指标找问题，不用指标找责任人。**违反这一条，你得到的不是数据，是数据表演。

板书：四个指标＋陷阱：Designed cases（可拆用例刷量）｜Executed cases（跑了≠跑透）｜Found bugs/Day（易催生凑数缺陷・对后期不利）｜Missed bugs（最难量，用逃逸率近似）。两句判断：度量**推动改进**／不良度量使执行**变形**。铁律："**度量流程，不度量个人**"。

提问：如果公司规定"每人每天至少提 5 个缺陷"，会发生什么？

预设回答：

- 答"大家会很努力找缺陷"——短期可能，但很快会变味。

- 答"会提一堆重复的、无关紧要的"——必然。凑数是理性反应，不是道德问题。

- 答"真正难查的问题没人查了"——对，因为难查的问题耗时间、产出少，在"按件计酬"下最不划算。

- 收口：**你考核什么，就会得到什么——包括你不想要的那部分。**

【思政落点 4】：这一页讲的是"公正"。度量天然带着权力：谁定指标，谁就在定义什么叫"好"。如果指标被用来追责，团队会用最省力的方式应付它，数据就不再反映真实，管理者从此在假象上做决策——受害的是产品和用户。反过来，如果度量只用于发现流程问题、只对事不对人，团队才愿意暴露真实数据。这背后是一种职业操守：**手里有数据的人，要有不用数据伤人的自觉。**公正不是软弱，它是让数据长期可信的唯一办法。

易错点 / 考点：① 四个指标的英文与含义；② **"不良的度量使流程变形"**是本章最高频的简答/论述点，要能举出三种变形形式；③ 常考判断："缺陷发现数越多说明测试越有效"——片面，且可能诱导凑数；④ 铁律"度量流程不度量个人"要能展开说明理由。

过渡：13.4 讲完"怎么算"，最后一块是"怎么下结论"——13.5 测试的评估与报告。

【13.59】13.5 测试的评估与报告（1 分钟）

这是本章最后一块，也是整个测试过程的收口。

三个小节。**13.5.1 测试过程的评估**——回头看"我们是怎么测的"，测计划里说的事做到了没有、做得怎么样。**13.5.2 测试充分性的评估**——回答"测得够不够"，核心工具是覆盖率。**13.5.3 测试报告**——把结论写出来，对内外交付。

三者的关系是：**过程评估看"做得对不对"，充分性评估看"做得够不够"，测试报告是"把这两件事说清楚"**。

这一块很重要，因为它是很多团队的短板。前面 13.1 到 13.4 都是"做事"，13.5 是"交代"。不交代会怎样？三个后果：第一，**经验无法沉淀**——这个项目踩的坑，下个项目重踩；第二，**决策没有依据**——老板问"能不能发"，你只能说"我感觉差不多了"；第三，**责任无法界定**——出了问题，说不清当时测了什么、没测什么、风险有没有被披露。

测试报告的本质不是"汇报工作",而是"**风险披露**"。一份合格的测试报告，要让读的人清楚地知道：**测了什么、没测什么、发现了什么、还剩什么风险、我的结论和建议是什么**。

再记住一句：**测试报告不是给测试自己看的，是给要做发布决策的人看的。**

> **板书：**三小节：13.5.1 过程评估（做得对不对）→ 13.5.2 充分性评估（做得够不够，靠覆盖率）→ 13.5.3 测试报告（说清楚）。底部："测试报告的本质＝风险披露"，五要素：测了什么・没测什么・发现了什么・剩什么风险・结论建议。

提问：如果测试做得很充分，但报告写得很糟，会有什么损失？

预设回答：

- 答"白干了"——有点绝对，但确实会大打折扣。
- 答"决策者看不懂，可能做出错误决定"——正确，这是最直接的损失。
- 答"风险没披露，出事要担责"——也很现实。测了但没写"未覆盖范围"，等于没披露。
- 收口：**测试的价值要靠报告兑现**，没有报告，工作成果无法转化为决策依据。

易错点 / 考点：① 三个小节的名称与分工；② 常考判断："测试报告是工作总结"——本质是风险披露与决策依据；③ 报告五要素要能列举；④ 与第 4 章"退出准则"的衔接是综合题考点。

过渡：先看过程评估——它其实是拿一串问题去对着计划做体检。

【13.60】13.5.1 测试过程的评估（2 分钟）

这一页给了六个问题，是测试过程评审的固定问法。课件的说明是：**测试过程评审，结合测试计划来进行评审，相当于把计划的测试活动和实际执行的活动进行比较，了解测试计划执行的情况和效果**。

六个问题我逐条讲，每条都在查一件事。

第一，**是否完成了测试计划所要求的各项测试内容？**查"范围"。计划里说要测的，测了没有。

第二，**测试用例是否经过开发人员、产品经理的严格评审？**查"用例质量"。这条很多人会忽略：用例本身没评审，就可能漏测关键场景，后面跑得再认真也白搭。

第三，**系统测试是否包含了性能、兼容性、安全性等各项测试？**查"专项覆盖"。这是第 7 章的内容在这里的回响——功能测完了不等于测完了。

第四，**如果执行了，又是怎么进行的、结果如何？**查"执行质量"。这一问最狠：不是问你"做没做"，是问"怎么做的、结果怎么样"。"我们做了性能测试"和"我们用 500 并发压了 2 小时，TP99 是 1.8 秒"是两回事。

第五，**需要执行的测试用例是否百分之百地完成了？**查"执行率"。注意"需要执行的"这个限定——不是所有用例都必须执行（有些用例可能因需求变更作废），但"计划要执行的"必须完成，未完成的要有说明。

第六，**所有严重的 Bug 都修正了？**查"遗留风险"。注意它问的是"严重的"，不是"所有的"——这符合现实：微小缺陷可以带着发布，严重的必须清掉或有明确处置。

六个问题合起来，覆盖了测试过程评估的五个维度：**范围、用例质量、专项覆盖、执行质量、遗留风险**。你以后写测试总结，可以就按这六问来组织，一个都不会漏。

> **板书：**六问竖排：① 计划内容完成了吗（范围）② 用例经开发/产品评审了吗（用例质量）③ 含性能/兼容/安全吗（专项）④ 怎么做的・结果如何（执行质量）⑤ 应执行的用例 100% 完成了吗（执行率）⑥ 严重 Bug 都修了吗（遗留风险）。底部五维度：范围・用例质量・专项・执行质量・遗留风险。

提问：第 4 问"如果执行了，又是怎么进行的、结果如何"，为什么这一问最容易被问倒？

预设回答：

- 答"因为很多人只做了没记录"——对，没记录就答不上"怎么做的"。

- 答"因为大家习惯回答'做了'"——这是形式化汇报的通病，做与做到位是两件事。

- 答"因为结果可能不好看"——也有，但要记住：**结果不好看比没有结果好**，前者能决策，后者只能猜。

- 收口：**能被追问细节的工作，才是真做了的工作**。

> **易错点 / 考点：**① 六问要能列举；② 评审的基准是**测试计划**（计划 vs 实际），这是定义性考点；③ 常考判断："所有用例都必须执行完毕"——是"需要执行的"100% 完成；④ "严重 Bug 都修了吗"体现遗留风险的处置逻辑。

> **过渡：**过程评估是"对着计划查"。可有些风险不在计划里——它们藏在过程本身。

【13.61】发现测试过程的风险（2 分钟）

> 这一页讲"发现测试过程的风险"，也就是前面六问之外，还要专门去找"过程本身可能出问题的地方"。
>
> 我先给一个分类，测试过程的风险通常来自五个方面。
>
> **第一，范围风险**：该测的没测。表现是需求覆盖不全、新功能没进测试范围、变更没有同步到用例。这类风险最隐蔽，因为"没测"不会报错。
>
> **第二，环境与数据风险**：环境不稳定、数据不真实、依赖服务是 mock 的。表现为"测试环境好的、生产坏的"，或者"数据量小的时候没问题，上量就崩"。
>
> **第三，执行风险**：用例跑了但跑得浅；回归范围不足；缺陷验证没带回归。这一类直接对应 13.3.1 那些操作规范。
>
> **第四，进度风险**：时间被压缩、人力被抽调、阻塞长期不解除。对应 13.1.2 的进度管理。
>
> **第五，协作风险**：缺陷被隐藏、状态乱标、争议不解决。对应 13.3 那一整套。
>
> 怎么发现这些风险？我给三个可操作的手段。
>
> 一是**评审**：拿过程记录去对计划，就是上一页的六问；同时做**同行评审**，让另一个测试看看你的用例和缺陷单。
>
> 二是**度量**：用 13.4 的指标找异常——执行率高但缺陷发现率极低，说明测得浅；缺陷清除率高但逃逸率也高，说明口径有问题或者数据不实。

三是**抽查**：随机抽几条缺陷单、几条用例，看质量。这个方法成本最低、发现问题的效率最高。

最后一句：**风险要被写下来**。写在测试报告里的风险叫"已披露风险"，出事时是尽职的证据；没写下来的风险，出事时就是"没想到"。**这两者的差别，就是专业和业余的差别。**

板书：五类风险：范围（该测没测）｜环境与数据（不稳定・不真实・mock）｜执行（跑得浅・回归不足・验证不带回归）｜进度（压缩・抽调・阻塞）｜协作（隐藏・乱标・争议）。三个手段：评审（对计划＋同行评审）｜度量（找异常组合）｜抽查（抽缺陷单与用例）。底部："风险要写下——披露 vs 没想到"。

提问：执行率 98%、但整周一个新缺陷都没提，你第一反应是什么？

预设回答：

- 答"质量太好了"——可能性存在，但概率低。

- 答"测得浅，或者没在测新东西"——正确方向。执行率高而发现率为零，最典型的解释是"在重复跑已通过的用例"。

- 答"可能是回归阶段，正常"——有可能，回归期发现率本来就低，但要确认是不是"只跑回归、没测新功能"。

- 收口：**指标的组合异常，比单个指标的绝对值更能说明问题**。

易错点 / 考点：① 五类过程风险的划分；② 三个发现手段（评审、度量、抽查）；③ 常考判断："没写进报告的风险不算风险"——反了，没写的才是最大的风险（未披露）；④ 指标组合异常（高执行率＋零发现）是典型案例分析点。

过渡：过程评估之后，还有一个更本质的问题：我们到底测"够"了没有？

【13.62】13.5.2 测试充分性的评估（2 分钟）

测试充分性，回答的是"**测得够不够、能不能停**"这个问题。

课件的定义：**测试结果分析的一项重要工作就是测试充分性的评估，通过了解测试覆盖率的值，可以知道测试是否充分、测试能否结束**。这句话把两件事绑在一起：覆盖率既是"充分性"的度量，也是"结束"的依据。这正好接上第 4 章的退出准则——**覆盖率是最常用的量化退出条件之一**。

第二句：**测试覆盖率是用来衡量测试完成程度、或评估测试活动覆盖产品代码的一种量化的结果，评估测试工作的质量，也是产品代码质量的间接度量方法**。注意这里有三层用途：衡量**测试完成程度**（做了多少）、

评估**测试工作质量**（做得好不好）、间接度量**产品代码质量**（覆盖不到的地方往往就是藏问题的地方）。

第三句最关键：**测试覆盖率可以看作由业务需求、系统特性功能/非功能特性和代码等多个层次的等构成**。这句话说明覆盖率**不是一个数**，而是一个立体结构——你在需求层算覆盖率，和你在代码行层算覆盖率，回答的是完全不同的问题。

我要提醒一个常见误解：**高覆盖率不等于高质量**。行覆盖 100% 只能说明每行代码都被执行过，不能说明每个分支、每个边界、每个异常路径都被验证过。这是第 5 章讲过的道理，在这里以"覆盖率"的形式再出现一次。

反过来也要说清：**低覆盖率几乎一定意味着测试不充分**。所以覆盖率的作用是"**下限判据**"——它不能证明你测得好，但能证明你测得不够。这个不对称性非常重要。

板书：三句定义：① 覆盖率 → 判断是否充分、能否结束（接第 4 章退出准则）② 三层用途：测试完成程度・测试工作质量・产品代码质量的间接度量 ③ 覆盖率是多层次的（业务需求・功能/非功能特性・代码）。底部："高覆盖 ≠ 高质量；低覆盖 ⇒ 不充分（下限判据）"。

提问：行覆盖率 100% 的项目，还会不会有严重缺陷？

预设回答：

- 答"不会，都覆盖了"——会。行覆盖只保证语句被执行，不保证分支组合、边界值、异常路径、并发时序被验证。

- 答"会，覆盖率只说明跑过，不说明测对"——正确。

- 答"要看是哪种覆盖率"——很专业，判定覆盖、条件组合覆盖、路径覆盖的强度差别很大。

- 收口：**覆盖率是必要条件，不是充分条件**——和第 5 章的逻辑覆盖准则是同一个道理。

易错点 / 考点：① 覆盖率的两大用途（判断充分性、判断能否结束）；② 覆盖率的多层次性（需求・特性・代码）是高频点；③ 常考判断："覆盖率高就说明测试充分"——错；"覆盖率低说明测试不充分"——对；④ 与第 4 章退出准则、第 5 章逻辑覆盖的衔接常出综合题。

过渡：既然覆盖率有层次，那具体分哪几个层次、从哪几个视角看？下一页画出来了。

【13.63】测试覆盖率分析（2 分钟）

这一页的要点是六个字：**三个层次、两种视角**。

先说**三个层次**，从外到内。

最外层是**业务需求层**：需求覆盖了多少、业务流程走了多少条主路径和异常路径。这一层最贴近用户，也最容易被忽略，因为很多团队直接从用例跳到代码，中间的需求覆盖没人算。

中间层是**系统特性层**，包括功能特性和非功能特性。功能特性就是"这个功能的各种场景测了没有"；非功能特性就是性能、兼容、安全、可靠性这些——第 7 章那一整套。这一层回答"除了功能对不对，其它指标达不达标"。

最内层是**代码层**：语句覆盖、判定覆盖、条件覆盖、路径覆盖等等。这一层最精确，但也最容易骗人，因为代码覆盖高不代表需求覆盖高——你可能把一段根本不该存在的代码测得很透。

再说**两种视角**。

第一种是**"测了多少"的视角**（正向）：我已经覆盖了什么。这是大家最熟悉的用法，用于汇报和判断进度。

第二种是**"还剩什么没测"的视角**（反向）：未覆盖的部分是哪些、为什么没覆盖、风险多大。这个视角更有价值，因为**风险永远在没覆盖的地方**。一个成熟的测试负责人看覆盖率报告，第一眼看的是"哪些没覆盖"，而不是"覆盖了多少"。

两个视角加上三个层次，就构成一个 3×2 的分析框架。你可以拿它做自检：**需求层我算了吗？特性层我算了吗？代码层我算了吗？我是不是只看了正向、没看反向？**

最后一句提醒：**覆盖率要跟缺陷分布一起看**。缺陷密集的模块，覆盖率必须高；覆盖率低又缺陷多的模块，就是最大的风险点——这两张图叠在一起，风险地图就出来了。

> **板书：**三层次（外→内）：业务需求 → 系统特性（功能・非功能）→ 代码（语句・判定・条件・路径）。两视角：正向"测了多少"｜**反向"还剩什么没测"（风险在未覆盖处）**。底部："覆盖率 × 缺陷分布 = 风险地图"。

提问：为什么说"看覆盖率要看未覆盖的部分"更有价值？

预设回答：

- 答"因为未覆盖的地方可能有缺陷"——正确，风险集中在没测到的地方。

- 答"因为已覆盖的部分已经证明了"——对，覆盖过至少说明执行过，虽然不保证测透。

- 答"看覆盖率高显得工作做得好"——这是汇报视角，容易自欺；反向视角才是质量视角。

- 收口：**覆盖率报告的正确读法是"找空白"，不是"看进度条"**。

易错点 / 考点：① "三个层次、两种视角"要能展开；② 三层次的名称与顺序（需求→特性→代码）；③ 常考判断："代码覆盖率高就说明测试充分"——错，需求层可能大片空白；④ "覆盖率×缺陷分布＝风险地图"是可写入答题的方法论。

过渡：所有的评估最后都要落到一份文件上——测试报告。国家标准对它是有要求的。

【13.64】13.5.2 测试报告（2 分钟）

这一页讲测试报告的内容要求。课件引的是国家标准：**GB/T 17544－1998（附录 E）对测试报告有了具体要求，对测试纪录、结果如实汇总分析，报告出来**。

注意这句话里最重要的四个字：**如实汇总**。报告的第一要求是真实，不是好看。

国标列了七项内容，我逐条讲它为什么必须在。

第一，**产品标识**——测的是哪个产品、哪个版本。没有标识，报告就没有对象。

第二，**用于测试的计算机系统**——在什么环境上测的。这条决定了结论的适用范围：在 8 核 16G 上测出来的性能，不能拿去承诺 2 核 4G 的部署。

第三，**使用的文档及其标识**——依据了哪些需求文档、设计文档、用例文档。这是"期望结果"的依据来源，也是可追溯的凭据。

第四，**产品描述、用户文档、程序和数据的测试结果**——这是报告的主体：测了什么、结果如何。注意它把"用户文档"单独列出来，说明文档测试也是测试内容的一部分。

第五，**与要求不符的清单**——就是遗留缺陷清单。这一项是报告的核心价值所在：**明确列出"哪些地方不满足要求"**。

第六，**针对建议的要求不符的清单，产品未作符合性测试的说明**——这一条最容易被忽略，也最能体现专业度：**你还要说明"哪些要求我们没测"**。测了没通过要写，没测也要写。这就是前面说的"风险披露"。

第七，**测试结束日期**——时间戳。它界定了结论的有效期：这份报告说的是"截至这一天"的状态。

七项内容，其实可以归成三块：**对象与环境（1、2）＋依据与结果（3、4）＋风险与时效（5、6、7）**。你以后写报告，按这三块组织就不会漏。

最后强调：国标用的是"**如实**"两个字。报告不是营销材料，不需要把结论写得好听；它要的是让读的人准确知道现状。

板书：GB/T 17544－1998 附录 E，七项：① 产品标识 ② 测试用计算机系统 ③ 使用的文档及其标识 ④ 产品描述/用户文档/程序/数据的测试结果 ⑤ 与要求不符的清单 ⑥ 建议的不符清单＋**未作符合性测试的说明** ⑦ 测试结束日期。归三块：对象与环境｜依据与结果｜风险与时效。底部圈"**如实**"。

提问：第 6 项要求写"产品未作符合性测试的说明"，为什么"没测什么"也要写进报告？

预设回答：

- 答"为了免责"——有这个作用，但把它理解成免责就浅了。

- 答"让决策者知道风险边界"——正确。不写"没测什么"，读的人会默认"报告没提就是没问题"。

- 答"显得报告不完整，不好看"——恰恰相反，写了才显得专业。**完整的报告一定包含"未覆盖范围"**。

- 收口：**报告的可信度，来自它敢写自己的边界**。

易错点 / 考点：① 国标编号与附录（GB/T 17544－1998 附录 E）；② 七项内容要能列举，**第 6 项"未作符合性测试的说明"是高频陷阱**；③ 常考判断："测试报告只写测试结果"——还必须写不符合项与未测项；④ "如实汇总"是报告的第一原则。

过渡：整章讲完了，我们把这条线收一下。

【13.65】本章小结（2 分钟）

这一页是本章小结，五句话，我按"做了什么、为什么这么做"给你重排一遍。

第一句，**讲解了如何有效地执行测试，包括测试进度控制等**。核心是：执行前有检查、执行中有记录、执行按优先级、进度靠 S 曲线和 NOB 曲线两个工具看。别忘了那句话——**时间被压缩时要调范围，而不是简单砍用例**。

第二句，**讲解了如何正确、规范地描述、分离、分类、记录和跟踪软件缺陷，以保证它们有效地、快速地被修复、最终得到解决**。这句话里五个动词很讲究：**描述**（写清步骤期望实际）、**分离**（一个缺陷一条记录）、**分类**（类型/来源/原因/严重性/优先级）、**记录**（完整字段）、**跟踪**（生命周期与状态流转）。五个动作缺一个，缺陷就修不快。

第三句，**需要建立软件缺陷跟踪数据库存储、搜索和分析软件缺陷，从而生成一系列的图表，分析项目的发展趋势，控制项目进度，并进行根因分析，找到问题所在，预防缺陷**。注意它的动词链：存储→搜索→分析→图表→趋势→进度→根因→预防。**终点是"预防"，不是"统计"**。

第四句，**讲解了如何评估产品质量和度量质量，包括基于缺陷的方法、经典的种子公式等**。关键指标：密度、清除率、逃逸率、趋势、分布；关键方法：种子公式 N=S・n/s、清除率估算、加权综合。**别忘了那句警告：不良的度量会让流程变形。**

第五句，**测试过程评审，测试结果分析以及测试报告**。收口三件事：对着计划评过程、用覆盖率评充分性、按国标写报告。

五句话其实就是一条完整的链：**执行 → 描述 → 跟踪 → 度量 → 报告**。这条链你记住，第 13 章就装进脑子了。

最后留一个问题给你自己：这五步里，你所在的团队（或你实训的小组）哪一步最弱？那个最弱的环节，就是你实习时最该补的课。

板书：五句缩成链：执行（检查・记录・优先级・S/NOB）→ 描述（描述・分离・分类・记录・跟踪）→ 跟踪（库→图表→趋势→根因→**预防**）→ 度量（密度・清除率・逃逸率・种子公式・加权）→ 报告（过程评审・覆盖率・国标七项）。底部大字："执行 → 描述 → 跟踪 → 度量 → 报告"。

提问：这五步里，哪一步最容易被做成"形式"？

预设回答：

- 答"度量"——常见，因为算起来麻烦，容易变成填表。

- 答"报告"——也常见，复制上一版模板改改日期。

- 答"跟踪"——如果只在版本末期集中关单一波，跟踪就是形式。

- 收口：**任何一步一旦只剩形式，整条链就断了**——因为下一环的输入是假的。

易错点 / 考点：① 五句话对应本章五块，可作为论述题框架；② 描述缺陷的五个动作（描述/分离/分类/记录/跟踪）是简答高分点；③ 跟踪的终点是"预防"这一点常被漏；④ 全章主线"执行→描述→跟踪→度量→报

告"要能默写。

> **过渡：**理论讲完，两次实验把这一章落到手上。先看实验 9。

【13.66】实验 9：安装和使用缺陷跟踪系统 MantisBT（2 分钟）

实验 9 是把这一章的前半部分落到工具上：**安装并使用 MantisBT**（课件给的地址是 mantisbt.org）。

这个实验的目标不是"装上能用"，而是**通过配置过程理解缺陷管理系统是怎么把制度固化下来的**。我按四步给你要求。

第一步，装起来并跑通。 MantisBT 是 PHP + 数据库的经典组合，安装的关键是三件事：Web 服务器（Apache/Nginx）、PHP 及扩展、数据库（MySQL/PostgreSQL）。装完之后先别急着提缺陷，先做一次**管理员初始化**：改默认管理员密码、建项目、建用户。

第二步，配字段和工作流。 这是本实验的重点。要做的配置包括：把"环境""版本""重现步骤""期望结果""实际结果"设为**必填**；调整状态机，确保**关闭权限只给测试角色**；配置严重性、优先级的选项集，写清每个级别的判定标准。做完之后，你的系统就体现了 13.3.1 和 13.2 的那些要求。

第三步，走一个完整的生命周期。 至少要造三条缺陷，分别走不同的路径：一条**正常流转**（新建→分配→修复→验证→关闭）；一条**验证不通过被重开**；一条**被判为"不是 Bug"或"延期"**。三条都跑完，你才算理解了 13.2.1 那张图。

第四步，出报表。 用内置的统计功能生成至少两张图：按状态/严重性的分布、按时间的趋势。然后在实验报告里写一句话结论：**从这张图里你看到了什么、会采取什么动作**。这句话是本实验最有价值的部分。

提交要求我建议这样：配置文件的关键截图（字段配置、工作流配置）＋三条缺陷的完整生命周期截图（含操作历史）＋两张报表＋一段 200 字以上的分析。**别只交界面截图，要交"你的配置决策和理由"。**

> **板书：**四步：① 安装跑通（Web 服务器・PHP・数据库・管理员初始化）② 配字段与工作流（必填项・**关闭权限只给测试**・严重性优先级选项与标准）③ 走三条生命周期（正常／重开／判非 Bug 或延期）④ 出两张报表＋写一句结论。提交：配置截图＋生命周期截图（含历史）＋报表＋200 字分析。

> **提问：**实验里为什么要求至少有一条缺陷"被判为不是 Bug 或延期"，而不是三条都走顺利路径？

> **预设回答：**

- 答"为了凑数量"——不是。顺利路径只能验证"主干"，验证不了"分支"。

- 答"因为真实项目里一定会遇到分歧"——正确。这两个分支恰恰是最需要流程支撑的地方：判"不是 Bug"要有依据，判"延期"要挂版本号。

- 答"三条都顺利不是更好吗"——顺利只说明系统能跑，不说明流程完整。

- 收口：**测流程要测分支，就像测代码要测异常路径**。

> **易错点 / 考点：**① 实验的四步目标（安装、配置、全生命周期、报表）；② **"关闭权限只给测试角色"的配置**是核心验收点；③ 三条缺陷要覆盖分支路径；④ 报告要求"配置决策与理由"，不是界面截图堆砌。

> **过渡：**实验 9 练的是缺陷管理一条线，实验 10 要把整章串起来做一遍。

【13.67】实验 10：基于 MeterSphere 的综合实验（2 分钟）

实验 10 是本章的综合大实验，用 **MeterSphere** 把"用例—接口—性能—缺陷—报告"整条线走一遍。课件列了五个任务，我逐个说要求。

第一，在线或离线安装 MeterSphere，并**创建项目、建立项目成员和基础测试配置环境**。这一步的关键是"成员和角色"——不同角色能做什么，直接决定了后面流程能不能走通。别用一个管理员账号走完全部流程，那样你体验不到权限设计。

第二，完成一个完整的接口测试：包括**接口调试与定义、接口测试用例的设计、基于真实场景的自动化测试**。注意三段递进：先调试通（能发出请求、看懂响应），再设计用例（正常、异常、边界、参数组合），最后做场景自动化（多个接口串成业务流，带变量传递）。

第三，完成一个完整的性能测试：包括**场景配置、压力配置、高级配置、结果分析**。这一段直接对应第 7 章：场景是"谁在用什么方式访问"，压力是"多少人、多久、什么节奏"，高级配置是集合点、思考时间、断言这些，结果分析要看响应时间分布、吞吐量、错误率、资源占用。**别只贴一张漂亮的 TPS 曲线，要写出瓶颈判断。**

第四，完成一个完整的功能测试流程：设计用例、用例评审、制定测试计划、**缺陷管理**、生成报告。这一段是本章的主线：从用例到计划到执行到缺陷到报告，一环不少。

课件还特别说明：**课程资源提供了详细的指导实验的视频，本实验可作为课程的大作业/实践考试**。这句话的

意思是——**这个实验的评分权重高，而且考的是"完整走通"，不是"某一步做得多漂亮"**。

我给一条建议：做这个实验时，**每一步都留下证据**（截图、导出文件、报告），最后按"任务一~任务四 + 一段总结"组织成一份文档。总结里至少回答三个问题：这条链里哪一步最花时间？哪一步最容易出错？如果重做一次你会怎么改？

板书：四个任务：① 安装＋建项目/成员/基础配置 ② 接口测试（调试与定义→用例设计→场景自动化）③ 性能测试（场景→压力→高级配置→结果分析，要写瓶颈判断）④ 功能测试全流程（用例→评审→计划→**缺陷管理**→报告）。底部："考的是完整走通，不是单点漂亮"＋留存证据。

提问：这个综合实验里，哪一步最能体现第 13 章的内容？

预设回答：

- 答"缺陷管理"——对，那是本章的核心。

- 答"生成报告"——也对，13.5 的落点。

- 答"用例评审"——补得好，这对应 13.5.1 第二问"用例是否经过严格评审"。

- 收口：其实整条链都在本章范围内——**13.1 执行、13.2 缺陷描述、13.3 跟踪、13.5 报告**，一个都不少。

易错点 / 考点：① 四个任务的名称与内容；② 接口测试的三段递进（调试定义→用例设计→场景自动化）；③ 性能测试四段（场景・压力・高级配置・结果分析）与第 7 章的衔接；④ 本实验可作大作业/实践考试，评分看**完整性**。

过渡：第 13 章到这里全部讲完。

【13.68】感 谢 聆 听（1 分钟）

第 13 章讲完了。

回顾一下我们走过的这条线：**测试怎么跑起来**（环境、冒烟、顺序、进度），**问题怎么写清楚**（步骤、期望、实际、严重性优先级），**缺陷怎么追到底**（生命周期、状态流转、争议处理、趋势与分布），**质量怎么算明白**（密度、清除率、逃逸率、种子公式、加权度量），**最后怎么交代**（过程评估、充分性评估、测试报告）。

这一章几乎不讲"测试技术"，全是"测试工作"，这些恰恰决定了你的用例能不能被执行、缺陷能不能被修掉、测试能不能被认可。**技术上你可能是第 5 章学得最好的人，但如果缺陷写得没人看得懂，你的价值就兑现不**

> **了。**
>
> 留三句话：第一，**缺陷报告写的不是"我发现了什么"，是"你怎么才能看到我看到的东西"**。第二，**度量流程，不度量个人**——数据一旦用来伤人，就再也不可信了。第三，**测试报告的本质是风险披露**——敢写"我没测什么"，才配说"我测了什么"。
>
> 第 14 章我们会走到测试的另一端：怎么把测试这件事**组织起来、持续下去**。这一章你手上有的，是单点的能力；下一章要给你的，是体系。
>
> 谢谢大家。

> **板书：**一条线：跑起来 → 写清楚 → 追到底 → 算明白 → 交代清楚。三句话：① 缺陷报告＝让别人看到你看到的 ② 度量流程不度量个人 ③ 报告本质是风险披露。下方预告："第 14 章：把测试组织起来"。

> **提问：**这一章里，你认为最难做到的一条是什么？

> **预设回答：**

- 答"如实写缺陷、不夸大不隐瞒"——最难的是"长期坚持"，因为压力总在。
- 答"不用度量去追责"——也很难，因为它逆着管理直觉。
- 答"敢在报告里写没测什么"——最难，因为它要承担当下的质疑。
- 收口：技术可以学，这几条靠的是职业习惯——**而习惯，是从第一份缺陷报告开始养成的**。

> **易错点 / 考点：**① 全章主线要能默写；② 三句总结性表述可作为论述题素材；③ 与第 14 章的衔接（从单点能力到体系）；④ 复习时按"13.1 执行—13.2 描述—13.3 跟踪—13.4 度量—13.5 报告"五块整理笔记效率最高。

> **过渡：**本章结束，下一章进入测试的组织与管理层面。

四、思政落点清单（共 5 处）

序号	所在页	落点主题
2	13.17	这一页讲的是"客观判断"。严重性和优先级都是判断，判断就有立场。同一个缺陷，开发

1 13.23 这一页讲的其实是"如实"。这份报告最难得的地方，是它写了"在其它字体格式下显示正

3 13.28 这一页几乎每一句都在讲"怎么不吵架"。重开要沟通、暂缓要共识、关闭要权限——这些

5 13.37 这一页讲的是"复盘与担当"。一个团队敢不敢做根因分析，取决于它把缺陷当成什么：当

4 13.58 这一页讲的是"公正"。度量天然带着权力：谁定指标，谁就在定义什么叫"好"。如果指