《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）

> 演示前一晚，你把缺陷单交上去。开发回你四个字：我这儿是好的。

>

> 你截图，你复述，他那边还是好的。问题不在你说谎，在于你交出去的信息里，缺了他能站到你这边的落脚点。这种事在岗位上，比写错代码常见得多。

>

> 前面十二章都在做准备。第 3 章教怎么设计用例。第 4 章排流程、定退出准则。第 11 章反复讲用例要能追溯。第 12 章把环境、容器、自动部署搭起来。全是准备。

>

> 准备够了，项目照样出事。出事的地方常在准备之后。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，开发看一眼就退回来。退回来的单子没人跟，最后不知落在谁手里。

>

> 第 13 章就管这三件。怎么把测试跑得有序，包括执行前的检查、冒烟、执行顺序、进度监控。怎么把发现的问题写成开发能直接动手的信息。怎么让每条缺陷有始有终，包括生命周期、状态流转、跟踪工具、缺陷分析与度量。

>

> 课件把整章切成五块。执行与跟踪，缺陷的描述，缺陷的跟踪和分析，质量评估与度量，评估与报告。前四块干活，最后一块收口。

>

> 这一章的分量不在术语，在产出。你交出去的缺陷单和报告，就是别人判断这支测试队伍的全部材料。

> **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，说明步骤不够精确，或者环境信息没给。

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

- 答"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：这三句话正好对上本章三块：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 五块内容的名称与顺序要能默写：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，它还写给管理者（判断严重程度与排期）、写给后来的接手人（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，后者一个阶段或一个版本一份阶段总结。

> **过渡：**先把上一章那份家底收一收，看它为什么决定这一章能不能开跑。

【13.2】第 12 章回顾（1 分钟）

> 一条缺陷是真是假，有时候不由软件决定，由环境决定。

>

> 你机器上跑得通，测试环境跑不通，生产上又是第三种样子。三条路三个结果，谁也不敢说哪一个是软件的真相。所以第 12 章那四块，得先收一遍。

>

> 第一块，重要性。很多所谓的缺陷，追到最后是环境不一致。环境不统一，缺陷的真假都判不了，测试的可信度更谈不上。

>

> 第二块，要素。硬件、网络、操作系统、中间件、数据库、测试数据、工具链。缺一样就不算一套完整的环境，缺口会出现在最不起眼的那个环节。

>

> 第三块，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几套环境就能拉起几套。

>

> 第四块，自动部署。环境不靠人手搭，靠脚本、靠流水线，一条命令还原到已知状态。

>

> 四块合起来只有一个目的：让结果可信。测试执行最怕的不是跑得慢，是跑出来的东西没法复现、没法解释。

>

> 所以讲执行要点时，第一条实践就是严格审查测试环境。那不是形式主义，是在保后面所有数据的可信度。

>

> 两章的分工也说一句。第 12 章是把环境做对，第 13 章是在环境里把事做对。前者是条件，后者是纪律。没有可信的环境，纪律无从谈起；没有纪律，再好的环境也只是摆设。这套条件备齐了，才开始用条件干活。活干得好不好，看每道工序有没有记录。

提问：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，因为能对齐到同一个已知状态。

- 答"让他上我的机器看"——偶尔可行，不可复制，人多就跑不通。

- 收口：环境信息的价值不是"证明我没错"，是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，容器保证运行环境一致，数据、配置、依赖服务版本仍然要管；③ 记这条因果链：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：** 环境这层立住了，就能看整章的结构。

【13.3】13.1 软件测试执行与跟踪（1 分钟）

> 有人学完整章，还是说不清这五块叫什么、排什么顺序。

>

> 第一块，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、进度管理的办法、过程管理工具。这是动作层。

>

> 第二块，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。往下是生命周期、严重性和优先级、其它属性、完整信息、基本要求，还有一好一坏两个示例。这是表达层，也是本章分量最重、出题最多的一块。

>

> 第三块，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，怎么从趋势和分布看出项目处在什么阶段。这是流程层。

>

> 第四块，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。

>

> 第五块，13.5 测试的评估与报告。测试过程评估、充分性评估、报告怎么写。这是结论层。

>

> 五块连成一条链：跑起来、写清楚、追到底、算明白、说得出结论。哪一步断掉，后面都接不上。

>

> 课件排这个顺序，不是按难易，是按依赖。没跑起来，没有缺陷可写。没写清，谈不上跟踪。没有跟踪数据，度量是空算。度量不成立，报告只能写感觉。

>

> 以后带项目，这五步一步都省不得。省掉哪一步，下一个项目就在哪一步重来一次。

> **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，算起来麻烦，算出来还不好看。

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

- 答"13.5 报告"——也有，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一一对应，选择题爱拿"缺陷描述"和"缺陷跟踪"互换来设陷阱；② 本章主链：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，先看测试执行要抓哪几个要点。

【13.4】13.1 软件测试执行与跟踪（1 分钟）

> 一进项目组，最先被问的常是：咱们用哪个工具？

>

> 这个问法把顺序弄反了。13.1 这一节分三段，正好回答它。

>

> 第一段，13.1.1 测试执行过程的要点。先讲不同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、NOB 曲线法。这一段解决现在跑到哪了、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。这些解决的是拿什么来装。工具是最后一步，不是第一步。

>

> 顺序有讲究。先定动作标准，再定监控手段，最后选工具承载。反过来干的团队不少：先买工具，再想流程，最后工具里跑的全是糊涂活儿。

>

> 三段之间也是依赖。要点不清，进度没法量。进度不管，工具里只剩一堆没意义的执行记录。

>

> 还有一句要放在这节开头。测试执行不是点用例，是在受控的环境里、按优先级、带着监控地往前推。点用例谁都会，受控和带监控才是本事。

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。缺陷单是两边最主要的沟通载体，沟通不畅，缺陷就被反复退回。

- 答"跟技术没关系，可以略过"——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：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 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，还要看缺陷的打开与关闭趋势（NOB），用例跑完不等于质量达标；③ S 曲线和 NOB 曲线各用什么数据、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）

同一套用例，单元测试阶段跑得挺顺，搬到验收阶段，没人认这个结论。

>

判据变了。测试执行拆成四件事，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，重点是接口之间的约定和数据流，环境得能拼起来。系统测试阶段才是测试团队的主场，看业务流程、看非功能指标。验收测试阶段，主角换成用户或业务方，判据是能不能接受，不是符不符合设计。

>

同一批人、同一套用例，换个阶段判据就变。这是最容易糊掉的地方。

>

第二件，测试用例执行。两条纪律。按用例跑，逐条记结果，通过、失败、阻塞、未执行，四种状态如实记。偏离要留痕：临时改步骤、换数据、跳某一步，都得写下来。为什么？三天后你要判断这个模块测没测透，靠的是这些记录，不是记忆。

>

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，信息要有一个共同入口。具体做法下面一批经验条目里有。

>

第四件，测试执行结束。不是用例跑完就完。要确认退出准则满不满足，这是第 4 章讲过的概念。要处理遗留缺陷：哪些必修、哪些带到下个版本、哪些接受风险。最后出报告，把结论说清楚。

>

> 四件事收成一句：开跑前有标准，跑的过程有记录，跑完有结论。三样缺一样，这一轮测试就说不清。

> **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"——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，要马上上报；后者是进度问题。

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

- 追问：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 用例执行四种状态（通过／失败／阻塞／未执行）常考，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，看课件给的那批具体做法，都是从项目里捡回来的经验条目。

【13.6】测试执行实践（2 分钟）

> 七条做法，短的一句就说完，长的一条得配一套动作。

>

> 按三段看：开跑前、跑的过程、出偏差。

>

> 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、数据干不干净、依赖服务起没起。这两件事花半天，能省后面一周的返工。

>

> 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。抽查两个字是关键。不是放下用例乱点，是

> 在用例之外专门去高风险处试：支付、权限、并发、边界。目的是检验用例有没有真盖住这个区域。

>

> 过程中三条。第三条，交叉互换测试人员所测的模块，可以发挥互补作用。一个人测久了有盲区，换人来看往往一眼就看出问题，这就是交叉测试。

>

> 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，让整个测试进程在控制轨道上发展。审的是单子质量：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。动作顺序要紧：不是少测点，是按优先级重划范围。

>

> 第七条，阶段性结果分析，保证阶段性测试任务完整执行并达到预定目标。每个阶段收口核一遍：该做的做了没有、做到什么程度、能不能进下一阶段。

>

> 再补一条实操，课件没有单列。压缩时间必须书面记下放弃了什么。不然到期那天，没人说得清这段到底测没测。

> **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，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，先跑核心流程和高风险模块"——正确做法，就是第六条讲的回到需求优先级、调整范围。

- 答"直接跟老板说做不完"——结论没错，但要带方案说：三天内能保证的范围、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 七条实践要能说出至少五条；② 高频考点"时间被压缩时的应对顺序"——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、面向高风险区域的活动；④ 交叉测试的目的答互补、减少个人盲区。

> **过渡：** 动作标准有了，接下来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）

> 进度落后，老板第一反应常是加人。这个反应经常帮倒忙。

>

> 课件给两组关系：进度与质量、进度与成本。想清楚这两组，才知道进度管理为什么不靠催。

>

> 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。容易发现的在前面就被发现了，越往后越是隐蔽的、要构造复杂条件才触发的。缺陷发现是一条长尾。

>

> 反过来，压缩时间也不是少发现一点，是把缺陷推到后面去发现。研发阶段省下的时间，会在生产环境上还回来，还得加倍。所以进度和质量的选择，实质是现在付还是以后付、一次付多少。

>

> 再说进度与成本。业内反复验证过一个现象：项目后期加人，往往更慢。新人要学业务、要熟悉代码、要占老人的时间带他，沟通路径还随人数涨得很快。所以进度落后时，第一反应不该是加人，该是缩范围或者调优先级。

>

> 那怎么管？靠数据，不靠感觉。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，才敢说能不能按期交付。

>

> 这两组关系要写进测试计划的假设与约束里，不要等落后了才拿出来讨论。事先写明时间压到某个点就放弃哪些范围、由谁批，比事后争有用。

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，把期望值拉回现实。

- 答"不如砍范围，先保核心流程"——这就是取舍思路，通常比加人有效。

- 收口：进度落后时，调整范围往往比增加资源见效快，因为范围是你自己能定的。

> **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 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 曲线是怎么画出来的。

【13.8】测试进度的 S 曲线法（2 分钟）

先看名字。S 曲线为什么是 S 形，不是一条直线？

>

> 课件说得很清楚：进度 S 曲线法通过对计划中、尝试的与实际的三者对比来实现。也就是三条线。

>

> 第一条，计划的进度线：按计划，第几天该完成多少。第二条，课件说的尝试的，理解为预期能达到的那条线。资源人力定下来以后，团队自己估出来的可行线。第三条，实际完成的线。

>

> 三条摆一起，结论自己会出来。实际落在计划下面，进度落后。实际贴着尝试线走，说明团队已经按能力上限在跑，落后不是态度问题，是计划本身不现实。实际高于计划，说明计划保守，可以提前收口或者加测。

>

> 它的基本数据是测试用例或测试点的数量。为什么不用缺陷数？缺陷数受软件质量影响，质量好缺陷就少。你没法说缺陷少所以进度快。用例数、测试点数才是工作量单位，反映的是干了多少活。

>

> 为什么用 S 形？因为测试工作的节奏本来就这样。前期读需求、搭环境、备数据，产出慢。中期跑起来最快。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：该陡的时候不陡，说明卡在某个环节。

>

> 看这条线只抓两件事。斜率告诉你速度，位置告诉你差距。

> **提问：**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。尝试线就是团队按自己能力估的可行线，贴着它跑说明已到上限。

- 答"计划定得不现实"——正确。该做的是改计划、调范围，不是继续压团队。

- 答"可能是阻塞太多"——补得好。斜率突然变平，要去看环境、前置功能是不是卡住了。

- 收口：S 曲线最大的价值，是把"谁的责任"换成"哪条线错了"。

> **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，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，做得对不对、问题还剩多少，它管不了。

【13.9】测试进度的 NOB 曲线法（2 分钟）

> NOB 三个字母，展开是 Number of Open Bug，打开状态的缺陷数。

>

> 它画的不是干了多少活，是还欠多少没解决。

>

> 形状通常是先升后降。前期测试刚开始，缺陷一个接一个被提出来，开发还在写别的功能、来不及修，打开数快速上升。到了中后期，开发集中修，修复速度超过新发现速度，曲线掉头向下。发布前降到很低，甚至接近零，这就叫收敛。

>

> 这条线看三件事。

>

> 第一，是不是在下降。临近发布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。

>

> 第二，下降的速度。降得快，修复有效。降得慢甚至反弹，要看是不是修复引入新问题，改一个坏两个。13.3.2 的累积趋势分析专门讲这个反弹。

>

> 第三，和 S 曲线对着看。执行率上去了、NOB 也降了，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，那是测出来了修不动。NOB 降了但执行率没上去，那是测得不够。

>

> 还有一层，NOB 只看打开状态，所以它受状态管理影响。团队习惯把没修的随手标成延期或者无法解决，NOB 就会假性下降。这就是 13.2 要严抓状态定义的原因：统计口径决定你看到的真相。

>

> 最后一条，NOB 要按严重性分层看。全部在降，严重及以上的没降，真实风险其实没降。分层看，才不会被总数骗过去。

> **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。先问这 70 个是怎么没的。修完、验证通过才算真降；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。每天还在新增 20 个，那只是提得慢，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，只说明没有打开状态的缺陷，可能被延期、被拒绝或没被发现；④ 与 S 曲线配合使用是简答得分点。

> **过渡：** 两个监控工具都会看了，接着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）

> 五个名字摆在这儿，不考你会不会点按钮，考定位。

>

> 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，面向用例的组织、评审、执行跟踪。适合用例驱动、迭代节奏快的团队。

>

> Jira，Atlassian 公司开发的项目管理工具，课件明确写它常常用于缺陷管理。这个措辞要留意：常常用于，说明它不是专门的缺陷系统，是靠工作流配出来的。强项是自定义工作流和看板，弱项是要配。

>

> 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。关键词是一站式：用例、接口、性能、报告都在一个平台里。适合想把工具链收成一个的团队。实验 10 的综合实验就建在它上面。

>

> PractiTest，端到端的测试管理系统。它提供用例管理、缺陷状态管理，有可定制的仪表板，附详细报告。定位偏管理和汇报。

>

> TestLink，开源的测试过程管理工具，用于项目管理、缺陷跟踪和用例管理。老牌，教学里常用，部署简单、概念清晰。实验 9 用的 MantisBT，两者常配套。

>

> 课件末尾还写了一句"更多工具详见此公众号文章"。意思是这行工具很多，别死记名单。选型就问四个问题。要不要管接口和性能，要就 MeterSphere 这类。要不要强定制工作流，要就 Jira。要不要轻量开源，要就 TestLink。要不要给管理层出报告，要就 PractiTest 这类。

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，经典组合，代价是维护两套账号和关联。

- 答"MeterSphere"——也可以，一站式省心，功能更多，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，第 11 章的追溯靠手工表很难维持。

- 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，尤其 Jira＝缺陷管理、TestLink＝用例加缺陷跟踪、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、要不要接口和性能），再给匹配，别直接报工具名。

过渡：工具只是容器，决定质量的是你写进去的那一条条缺陷。

【13.11】13.2 软件缺陷的描述（1 分钟）

整章最该花时间的一块，在这儿。

>

13.2 分六小节，顺序本身就是逻辑。

>

13.2.1 软件缺陷的生命周期。先看一条缺陷一辈子要经过哪些状态，你才知道单子该填哪些字段。

>

> 13.2.2 严重性和优先级。两个最容易混、也最容易吵的属性。

>

> 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 缺陷报告的示例。一好一坏，对着看。

>

> 这个顺序不是排版。先看清一条缺陷要走哪些状态，才知道该设哪些字段。字段定了，才谈得上填全、写准。反过来先背写作要求，你会背六条，却不知道它写在哪一栏。

>

> 为什么值得单独成节？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，是唯一会被反复阅读的东西。写得好的，开发十分钟定位；写得差的，来回三天还在问你到底在说什么。

>

> 有一句放在前面。缺陷报告写的不是"我发现了什么"。写的是"你要怎么做，才能看到我看到的东西"。带着这个视角，下面每条要求你都觉得理所当然。

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，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"——也关键。标题是唯一在列表页露面的东西，标题烂，单子容易被忽略。

- 收口：这几项是一条链：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 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。

【13.12】对缺陷会关心哪些问题？（2 分钟）

> 课件抛四个问句，后头还挂一串省略号。省略号是留给你的。

>

> 这四个问句是 13.2 的纲，一条一条对。

>

> 描述？这条缺陷到底是个什么事。对应标题、步骤、期望结果、实际结果。它回答是什么。

>

> 是否严重？对应严重性和优先级，回答多要紧。课件把它放第二位，说明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，先弄清事实再谈判断。

>

> 是否需要修正？这一问的关键是答案可以为不修。有些缺陷设计如此，有些是第三方组件的问题，有些修的成本远高于影响。所以终态不只关闭一个，还有延期、无法解决、不是 Bug。完整的流程图里能看到这几种出口。

>

> 当前状态？对应生命周期，回答这事走到哪一步了。没有状态就没有跟踪，没有跟踪就没有进度判断。

>

> 省略号补什么？还能补缺陷类型、来源、影响范围这些。问到哪一层，看你写给谁看。

>

> 四个问题记牢，拿到任何一条缺陷都能自查：说清了吗？多要紧？要不要修？到哪一步了？四条有一条答不上来，这张单子就不合格。

>

> 这四问还替三类读者分好了工。开发只看第一问。组长和项目经理看第二、第三问。三个月后的你看第四问，也看第一问，因为那时你已经忘了当时在干什么。

> **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，常常记录在案、不修。

- 答"发现了就必须修"——最典型的新手想法。缺陷记录的价值不只修，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，是测试把风险摆到桌面上，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）

> 一条缺陷的一生，主干只有四个状态。

>

> 第一步，发现到打开。测试人员找到缺陷，把它提交给开发。打开这个动作的含义要弄清：不是我知道了，是正式把责任交给开发。所以打开时必须带齐信息，否则开发接不住。

>

> 第二步，打开到修复。开发再现、修复，然后提交给测试验证。两个动词要划出来。再现是前提，开发复现

> 不出来，就定不了改哪一行。修复之后要提交，说明改动必须落到可验证的构建上，不是一句我改好了。

>

> 第三步，修复到关闭。测试人员验证修过的软件，关闭已经不存在的缺陷。课件这个措辞很讲究：关闭已不存在的缺陷。依据是现象不再出现，不是开发说改好了。

>

> 三个动作四个状态，就是主干。责任线跟着走：发现和关闭是测试，打开和修复是开发。谁改状态，谁对这一步负责。

>

> 主干先立住，分支才有地方挂。真实项目里还有打回、延期、拒绝这些出口，都从这条主干上分叉出去，不是另起一套。

>

> 这模型一听就太理想。开发复现不出来呢？验证没通过呢？这个版本根本不打算修呢？把分支补上，就是完整版。

>

> 每个状态之间还得有可验证的交付物。发现交出去的是缺陷单，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。

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。开发验的是我改的逻辑对不对，测试验的是现象消没消失、有没有带出新问题。

- 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：状态流转的实质是责任交接，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，不是开始排查。

> **过渡：**把分支补上，就看完整的缺陷生命周期。

【13.14】完整的缺陷生命周期（3 分钟）

> 主干走完是顺利的情况。真实项目里，一半时间耗在岔路上。

>

> 主干是这样。创建之后进入激活状态，系统发邮件通知开发。开发先判断一件事：这个缺陷是否清楚、可再现？清楚可再现，进已处理状态，开始查、开始改。课件在旁边标了 Unit test、code review、Check in CVS。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。改完进已修正状态，系统发邮件通知测试。测试做验证，通过就进关闭状态，不通过就打回，重新激活。

>

> 四个岔路都从这个判断点分出去。

>

> 第一，不能再现或者缺少信息。复现不出来，或者信息不全。这不是简单退回，要跟测试确认。确认后仍复现不了，可能进缺陷评审。

>

> 第二，缺陷评审。双方对这是不是缺陷有分歧，就交给评审。出口有几个。不是 Bug，直接关掉。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。无法解决，技术做不到，或者成本不可接受。

>

> 第三，需要处理。评审结论是要修，回到已处理状态。

>

> 第四，延期这条要单说。课件标了下个版本。延期的缺陷必须挂到具体版本号或者日期上，不能飘着。飘着就永远消失。

>

> 两条回路。验证不通过，回到重新激活。延期到下个版本，再重新打开。这两条最容易出问题：回路没有记录，缺陷就掉进黑洞。

>

> 三样齐了流程才闭合：四个主干状态管流转，四个岔路管定性，两条回路管兜底。

> **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，硬顶没意义。

- 答"要求写明原因，评估影响和替代方案，再由评审确认"——正确。这个状态可以存在，但不能一句话就关掉。

- 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：**这套流程放进真实项目是什么样，课件给了一条 14 步的时间线。

【13.15】实例（2 分钟）

> 课件给了一条时间线，数下来 14 步。从 open a bug 到 change bug status to close。

>

> 分三段念。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。

>

> 查重排在开发的第三步。重复缺陷很常见，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 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。Change bug status to close 关闭。

>

> 数一下，真正改代码只有 Fix bug 一步，其余十三步是沟通、确认、评审、构建、部署、验证。修一个 Bug 的时间，大半花在保证改对了、没改坏、能交付上。

>

> 中间那段打包、上传、安装配置，说明验证必须基于明确的构建包。不能基于开发本地的代码。所以完整缺陷信息里要有构建包跟踪这个字段。

>

> 还有个角色容易被忽视。Build a package 通常不是开发个人做的，是流水线做的。成熟团队里修复要过持续集成，不是本地编译一下发给你。跳过流水线，你验的就不是将要发布的那个东西，是某个人机器上的一个版本。这两者的差，往往就是测试都通过了、上线还是坏的事故根源。

>

> 所以这 14 步教的不是流程，是可追溯。事后能回答这个修复什么时候、由谁、过了哪些检查、进了哪个包。

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，而且往往几周后才暴露。

- 答"Unit test"——也常被省，尤其赶版本的时候。

- 答"查重（Duplicate）"——最容易被忽略。省了之后缺陷库里同一个问题挂着五个单子，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 步的关键节点要能排出来：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 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。

【13.16】13.2.2 严重性和优先级（3 分钟）

> 评审会上最容易吵起来的两个词，严重性和优先级。

>

> 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。五个字要看清楚：客户满意度。它不是代码错得多离谱，是用户有多难受。课件分四级：致命的 fatal、严重的 critical、一般的 major、微小的 minor。

>

> 四级怎么分？给一条能落地的判据。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，但有绕行的办法。一般的，是次要功能出错，或者主功能有瑕疵但可用。微小的，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。它衡量的是多急，不是多坏。

>

> 两个词为什么必须分开？因为判断依据完全不同。严重性看影响，优先级看排期。企业官网首页标语上的错别字，技术影响微小，可它是门面，优先级可能很高，当天就得改。一个只在极端并发下才触发的崩溃，技术致命，但这个功能下季度才上线，优先级可以先放低。

>

> 于是有四种组合，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，排队慢慢来。

>

> 一句话：严重性是客观判断，优先级是主观决策。客观判断要标准统一，主观决策要有人负责。两件事混着

> 谈，就会变成我觉得很严重对你觉得不严重。

>

> 所以判定标准得提前写、公开写，不要留到评审会上临时定。

> **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。这个分辨率要是主力机型的标准分辨率，影响面就很大，优先级得往前提。

- 答"严重性微小，优先级看影响面"——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。标准里可以含营销节点、大客户诉求，但要事先写进判定标准。

- 收口：标准要提前写好并公示，事到临头再争，永远争不出结果。

> **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 四个等级的中英文对应（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"首页错别字"这类反例，考的就是两者的独立性。

> **【思政落点 1】**：同一个缺陷，开发觉得影响不大，用户觉得天塌了，测试夹在中间最容易被情绪带走。把判据提前写进团队的标准，用可复现的事实说话：能不能复现、多少人会遇到、坏了哪个功能。手里有标准，就不靠嗓门办事；心里装着用户，就不拿自我安慰过关。这是测试岗最基本的职业操守。

> **过渡：**优先级既然是个决策，那能不能算出来。

【13.17】优先级的计算（2 分钟）

> 拍脑袋定优先级是常态。课件给了个公式，让这个动作变成算一遍。

>

> 公式是：优先级等于（可重复性加可发生性）乘严重性。三个因子逐个看。

>

> 可重复性，Repeatability。这条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。它决定开发能不能稳定调试。不能稳定复现的缺陷，修复成本成倍上升。

>

> 可发生性，Visibility。用户撞上它的概率有多大。两个词要分清：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会不会撞上。一条缺陷可能很容易复现，但只有极少数用户会走到那条路径。

>

> 严重性，上一页的四级。公式里是相乘，不是相加。严重性是放大器：它低，另外两项再高也顶不上去；它高，另外两项稍高就把优先级抬起来。

>

> 公式的价值在这——优先级不只看严重程度。一条缺陷很严重，但基本没人会遇到、也不容易复现，它确实可以先放一放。反过来，不太严重但人人会撞、而且必然出现的，往往要优先处理。

>

> 公式给的是参考值。最终排期还要看修复成本、版本节奏、依赖关系。它解决讨论有共同语言，不解决谁说了算。

>

> 还有一条实操：可重复性和可发生性通常也分级打分，比如都按 1 到 3 分。不打分就直接相加，量纲不一致，乘出来没意义。

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

> **预设回答：**

- 答"致命就必须最高优先级"——直觉，但没算。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"——成熟做法。不马上修，但要能第一时间知道它在生产上出现过没有。

- 答"先修了保险"——如果修复成本低，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。

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，判断题常见陷阱；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：**属性远不止这两个，下面把剩下的分类维度补齐。

【13.18】13.2.3 缺陷的其它属性（2 分钟）

教材 361 到 362 页那几张表，五类属性。考试不考全表，考用途。

>

> 逐类讲为什么要有它。属性不是填着玩的，每一条都对着后面的用法。

>

> 第一，缺陷标识 ID。唯一编号。作用不是好看，是可引用。开会说 1024 号那个问题，比说那个登录的问题精确得多。

>

> 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。某个版本 60% 的缺陷是 UI，说明前端规范或者设计评审出问题了。大部分是功能，说明需求理解和逻辑实现有问题。13.3.3 的分布分析就建在这些分类上。

>

> 第三，缺陷产生可能性 frequency，或者叫可再现的概率。上一页公式里的可重复性。它决定修复难度，也决定测试要不要再花时间做更细的复现尝试。

>

> 第四，缺陷来源 source：需求、设计、编码。这条是过程改进的入口。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 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。它比来源细一层。来源说哪一阶段引入的，原因说具体错在哪一类。有了它，才定得出针对性的编码规则。

>

> 五个属性分两组。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。

>

> 只填必填项、不填来源和原因的团队，永远在原地打转。他们的数据能数缺陷，不能减缺陷。

> **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，否则大家都填编码，数据立刻失真。

- 答"数量还是能看的"——能看，但只看得出多不多，看不出为什么多，做不出改进。

- 收口：只数不析，等于白测。属性填得越全，后面的分析和改进越有抓手。

> **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，考试考细则以教材为准。

> **过渡：**属性是字段，接下来看一张合格的缺陷单要填哪些内容。

【13.19】13.2.4 完整的缺陷信息（3 分钟）

> 三个要素，缺一个，缺陷就不成立。这是本章技术含量最高的地方。

>

> 先说步骤。课件原话很讲究：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。

>

> 三个词拆开看。准确，能照着做。简明，不写无关内容。完备，不缺关键前提。最后那句视为修复缺陷的向导最重要：步骤不是给自己留的记录，是给开发指路。开发照你的步骤走一遍，看见同样的现象，才有可能定位到代码。

>

> 再说期望结果。课件说它要与测试用例标准、设计规格说明书、用户需求等一致，达到软件预期的功能。它是验证缺陷的依据。关键词是依据。你凭什么说这是缺陷？凭它违背了某个书面来源。这就解释了缺陷单里为什么常写依据：需求文档第 3.2 节。没有依据的期望结果，只是个人偏好。

>

> 最后说实际结果。课件说是实际执行测试的结果，与期望结果不同，缺陷由此成立。注意这个判据：缺陷的定义就是实际不等于期望。写实际结果只写事实，看到什么、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论证。照我的步骤做，这是前提。预期是看到 A，这是期望。实际会看到 B，这是实际。A 不等于 B，所以这是缺陷。这条链写完整，开发就没有反驳的余地。

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：乱码、空白、错位，还是弹窗报错。

- 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 期望结果的依据有三个来源：测试用例标准、设计规格说明书、用户需求，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：三样都有了，可单子到了开发手上，往往还差几样东西。

【13.20】还需要什么重要的信息？（2 分钟）

设想开发照你的步骤走了一遍，回你一句复现不了。你手上还缺什么？

>

> 课件给了五样：产品、版本信息、图片、Trace Log、录制这个操作过程。五样有个共同点，都是把我看到的变成你也能看到的。

>

> 产品。哪个产品、哪个端。同一家公司可能有 App、小程序、Web 后台，同一功能在三处表现不同，不写清等于白写。

>

> 版本信息。最常被漏，后果最重。缺陷跟着版本走：这个版本有，下个版本可能已经修了。你在旧版本上提，开发在新版本上当然复现不了。版本号、构建包号必须写。13.3.5 的缺陷跟踪系统里专门有构建包跟踪字段，就是为这个。

>

> 图片。截图的价值是一眼抵一百字。界面错位、乱码、弹窗，一张图就说清。截图要截有用的部分，把错误提示、版本号、时间一起截进去，比截一整屏桌面值钱。

>

> 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，界面上可能只有一句操作失败，原因都在日志里。会看日志、会给日志，是测试人员从初级往中级走的一道坎。

>

> 录制这个操作过程。缺陷涉及多步操作、时序、动画、偶发，光靠文字步骤很难还原，一段录屏最有效。

>

> 五样收成一句：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。花这十分钟，省的是开发的两小时。

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"——偶发的，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，两者证明的东西不同；④ 简答可答提高可复现性、降低沟通成本。

> **【思政落点 2】**：写清版本、附上日志、留下录屏，这些动作没人检查，偷懒也看不出来。可一条缺陷能不能被修，往往就卡在这几步。它考的不是技术，是肯不肯把证据给足。别人要花半天猜的东西，你多写三行就说清了——对下一个接手的人负责，也对自己签下的结论负责。

> **过渡：** 把这三要素和这五样加起来，就是一张完整缺陷单的字段清单。

第 13 章讲稿（13.21–13.40，v3）

> 锚点与课件一一对应，不增删页。取消板书。口播按四拍：起 → 推 → 收。每页开场句式各不相同。

【13.21】缺陷完整的信息（3 分钟）

> 二十多个字段，一条缺陷单真要全填？大多数人第一遍就跳过去了。教材 P363 表 13-7 是完整版，一共二十多项。课上按三组记，比死背强。

>

> 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，八项。这八项回答的是"这件事是什么"。里头有三个要单独说。标题是列表页唯一显示的东西，别人点不点开就看它。前提最容易漏，比如"必须先以管理员身份登入"，漏了开发就复现不出来。频率是可重复性，必现还是偶现，直接决定怎么排查。

>

> 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，也是八项。这八项管的是归谁、多要紧、从哪来。提交人和指定解决人必须写清。没有责任人的单子就是孤儿单，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，七项。三个时间戳是算缺陷年龄、平均修复时长的原料。所属项目和模块是分布分析的分组依据。状态是生命周期的骨架。

>

> 三组加起来二十多项，真实项目不会全填。哪些省不了？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，十一个。缺任何一个，这张单子的可用性都要掉一截。

>

> 反过来也得说一句，字段不是越多越好。必填项太多，测试人员就开始敷衍，数据质量反而下来。必填要少而关键，选填要全而方便。

>

> 二十多个字段，能省的和不能省的，今天就分这一刀。

> **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"步骤里已经写了"——步骤写的是做什么，前提写的是在什么状态下做，两件事。

- 答"开发试一下不就知道了"——他试的时候状态不对，只能回你一句"无法复现"。

- 答"前提不重要，反正开发会自己摸"——这条单子会一直挂着，直到你自己也复现不出来。

- 收口：权限、数据状态、配置开关，这三类是前提里最容易被省掉的。

> **易错点 / 考点：**三组的划分逻辑要能说清；三个时间戳（提交、修正、验证）的用途是算缺陷年龄和平均修复时长；判断"字段越多越好"——错，要算填写成本；简答"一条缺陷至少包含哪些信息"，按描述组加严重性优先级加责任人加状态来答。

> **过渡：**字段齐了，下一件事是别在一张单子里塞五个问题。

【13.22】13.2.5 缺陷描述的基本要求（3 分钟）

> 你们组交上来的单子，一半以上一条里塞着三四个问题。看着写得挺满，开发看完不知道从哪下手。

>

> 第一条，单一准确，每份报告只针对一个缺陷。这是最容易被违反的一条。一条单子写五个问题，开发修了三个、忘了两个，你验证时关不掉，这条就永远开着。

>

> 第二条，可以再现，给出精确的操作步骤。自己都再现不了的缺陷，要么继续挖，要么写清"偶现，试 N 次出 1 次，日志见附件"。让开发知道这是概率问题，别让他拿五次尝试去赌。

>

> 第三条，完整统一，步骤和信息前后一致。统一指的是术语、版本号、模块名跟着团队走。你在步骤里一会儿叫"用户中心"、一会儿叫"个人主页"，开发只能猜。

>

> 第四条，短小简练，包括使用关键词。这条和上一条不打架：信息要全，表达要省。能写"点击【保存】"，就不写"用鼠标左键单击那个保存按钮"。关键词是为了能搜，以后有人搜"支付超时"，能搜到你这条。

>

> 第五条，特定条件一定要注明。浏览器、分辨率、权限、数据量、网络环境，都算条件。缺陷常藏在条件下，换个环境就没了。第 23、24 页那两份示例，差别就在这一条上。

>

> 第六条，不做评价。不写"开发太粗心""这么简单也能错"。两个原因：对定位没用；会把人推到防御的位置上，技术讨论变成吵架。

>

> 回到课件底部那句问话，哪点比较关键？我答前两条。单一准确让缺陷能被关闭，可以再现让缺陷能被修复。后面四条都在服务这两条。

提问：为什么"不做评价"写进技术要求，而不只是个人修养？

预设回答：

- 答"评价没信息量"——对一半，它不提供任何定位线索。

- 答"会引发对抗"——这是实际后果。开发看到"这么简单也能错"，第一反应是解释，不是排查。

- 答"测试当时在气头上"——情绪能理解，可单子会长期保存、被很多人读，情绪会一直留在库里。

- 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：**六条要能列举；"一个缺陷一条记录"是最高频考点，案例题常给"一条写多个问题"让你判；判断"描述越详细越好"——错，要完整不冗长；"不做评价"要能说两条理由：无助于定位、引发对抗。

> **过渡：**要求清楚了，下面拿两份真报告对着看。

【13.23】13.2.6 缺陷报告示例（3 分钟）

把文字改成 Arial 字体，字就变成一堆乱码。这个场景你照着做，一定能看到。

>

> 先看重现步骤，四步。打开编辑软件并新建一个文档。在这个文档里随意录入一两行文字。选中这几行，通过 Font 菜单选 Arial。文字变成无意义的乱字符。

>

> 这四步为什么好？第一步给了起点，是新建文档，不是打开已有文件。第二步给了数据，一两行文字。第三步是精确操作，选中、菜单、Arial。第四步是现象。四步不冗也不缺，别人照着做能看到。

>

> 再看期望结果：当用户选中已录入的文字并改变格式时，文本应显示正确的格式，不出现乱字符。这句话能判定。它没写"应该正常"，它给了判据。

>

> 最值钱的是实际结果那一段。它没停在"显示乱字符"，而是接着做了三件事。第一，定位范围，这是字体格式的问题。第二，隔离变量：改格式之前先保存文件，缺陷不出现，说明"保存"这个动作会影响结果。第三，划平台边界：只在 Windows 98 出现，换成别的字体格式显示正常。

>

> 这三条等于替开发把排查范围砍掉一大半。报告末尾还附了图片链接，文字加截图，两份证据。
>
> 重读这四步还有个用处。你自己照着它走一遍，走到第三步就出不来，说明你写的步骤里少了一个条件。这

> 张单子要改的是你，不是开发。

>

> 所以这份报告强的地方不是描述得细，是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。交出去的不是一个问题，是这个问题的边界。

提问：这份报告里哪一句对开发最有用？

预设回答：

- 答"出现乱字符"——这是现象，是起点，但它本身不含定位信息。

- 答"先保存文件就不出现"——这句最值钱，它直接指向未保存状态下的渲染路径。

- 答"只在 Windows 98 出现"——也很有用，把平台相关的字体渲染差异圈出来了。

- 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。

易错点 / 考点：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；简答"好报告和差报告差在哪"——差在有没有隔离变量、有没有限定条件；"期望结果必须可判定"是判据；案例题常让改写一段描述，评分点在步骤精确度和条件限定。

【思政落点 3】：这份报告最难得的地方，是它写了"换成别的字体格式显示正常""Solaris 和 Mac 上没这个问题"。它没夸大严重性，也没隐瞒只影响一个平台。缺陷报告是职业诚信最直接的考场：夸大，团队会把资源投到不该投的地方；隐瞒，风险就在发布后爆掉。测试人员手里握着"问题怎么被描述"的权力，如实、不夸大不缩小，就是这个岗位的底线。

过渡：好的看完了，再看一份让人抓狂的。

【13.24】散漫的缺陷报告的示例（3 分钟）

同一件事，换个人写，能写成一份让人抓狂的报告。

>

> 还是字体乱码，报告写成这样。在 Windows 11 上打开软件编辑已有文件。文件字体显示正常。我添加了图片，图片显示正常。之后新建了一个文档。在文档里随意录入大量文字。选中几行改成 Arial。有三次我重现了这个缺陷。我在 Solaris 上跑这些步骤没问题。我在 Mac 上跑也没问题。

>

> 第一处毛病，混进了无关操作。加图片、图片显示正常，跟字体乱码有什么关系？一步都不该在。

>

> 第二处，把平台验证写进了重现步骤。Solaris 和 Mac 的结果有价值，可它属于影响范围，不该塞进"照着做"的列表。那个列表是给人执行的，不是给人读结论的。

>

> 第三处，变量没隔离。开头是"编辑已存在的文件"，后面又"新建了一个文档"，同一份报告里两个起点，到底哪个是触发条件？说不清。

>

> 第四处，数据量含糊。"大量文字"到底多少行？

>

> 第五处最要命。它写了"重现三次"，可没说这三次是必现还是碰巧。频率写了，没写成能用的信息。

>

> 期望结果那一段其实还行，和优秀示例差不多。
>
> 为什么散的报告会长成这样？多半是作者一边试一边记，把手边做过的事全抄了上去。记录不是报告，报告是给下一个人照着做的。

实际结果就跑偏了：只有 Arial 有问题，这句是好信息；"它可能出现在我没测的别的字体格式里"，这是猜测。把猜测写进事实栏，等于让开发替你去验证你的想象。

>

> 两份报告差一句话。一份交的是缺陷的边界，一份交的是操作日记。

> **提问：**"我在 Solaris 和 Mac 上没问题"这句，怎么处理才不浪费？

> **预设回答：**

- 答"删掉"——可惜，这是有价值的平台边界。

- 答"放到影响范围里，写成仅 Windows 系列，Solaris／Mac 未复现"——对，信息保留，位置归位。

- 答"写在步骤最后"——形式上不算错，可读的人会以为那也是一步操作。

- 收口：信息没有废的，只有放错地方的。写报告一半是收集，一半是归位。

> **易错点 / 考点：**散漫报告的典型毛病至少能指出四条；判断"重现步骤里可以写排查过程"——不可以，结论放备注或影响范围；"实际结果里不能有推测"是高频评分点；案例题常让两份对比说优劣并讲理由。

> **过渡：**单条写好了，还得让上百条有序流转。

【13.25】13.3 软件缺陷跟踪和分析（1 分钟）

> 一条缺陷怎么写，前面讲完了。接下来要管一堆。

>

> 13.2 管的是单条，13.3 管的是一堆怎么流转、怎么读。这一节五个小节。

>

> 13.3.1 软件缺陷处理技巧，讲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，还有沟通纪律。

>

> 13.3.2 缺陷趋势分析，把缺陷数量画成时间曲线，看项目走到哪一段、会不会收敛。

>

> 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。包括 Bug dashboard、缺陷报告的种类、Bug bash 活动，还有该做和不该做的清单。

>

> 13.3.5 软件缺陷跟踪系统，用工具把上面这些装起来。
>
> 为什么先讲人的动作，再讲数据？因为数据是动作留下的痕迹。人怎么填、怎么分、怎么关，决定曲线长什么样。动作一乱，后面的图就没有读的价值。

>

> 五个小节串起来，中心词是跟踪。跟踪的意思是三条：每条缺陷有人在管，每次状态变化有记录，每个决策有依据。

>

> 反过来看，没有跟踪的团队有三个症状。单子开了没人认领，修了没人验证，关了不知道谁关的。
>
> 这三个症状有个共同后果：项目到底能不能发，谁也说不清。测试说还早，开发说都修了，两边手里没有同一份记录可以对。

>

> 还有一条暗线得先点出来。跟踪管的不只是缺陷，它管的还是项目节奏。决定能不能发的，常常不是还剩多少用例没跑，而是还剩多少缺陷没关、关得够不够快。所以这一节读的时候，别看它是流程，它其实一直在回答"现在能不能发"。

提问：一条缺陷三个月没人动，最可能是哪个环节出的问题？

预设回答：

- 答"没人认领"——常见，指定解决人没填，或者填了没通知到。

- 答"优先级太低被忘了"——延期缺陷不挂版本号，就会永久沉底。

- 答"状态乱标，其实早修了"——也可能，状态不严，数据就不可信。

- 收口：三种都能治，关键是有人定期去看那张列表。

易错点 / 考点：五个小节名称；判断"缺陷跟踪只是记录状态"——不完整，还包括推动处理、支撑发布判断；趋势看时间、分布看属性，辨析题常考。

过渡：先从人的动作讲起，看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）

一条缺陷在库里躺三个月没人动，是谁的错？

>

> 课件给九条，我按三组讲。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。缺陷躺着不动，测试的工作等于没做。

>

> 第二条，及时沟通，reopen 之前先线下说一声。你要重开，先告诉开发"我这边还是不行，日志发你看看"，比直接点重开再抄送一堆人有效。后者在开发眼里就是当众打脸。

>

> 第三条，只有测试人员有关闭缺陷的权限。这是制度层面的保障。

>

> 第二组，让判断有标准。第四条，明确定义优先级和严重级的衡量标准。"明确定义"四个字，意思是写下来、公示、事先定。事后再吵"这算不算严重"，成本高得多。

>

> 第五条，注释清楚、达成一致。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，周期性会诊，定期开会集体过一遍没解决的缺陷。第七条，实时看 Bug dashboard，不能等周报。第八条，不要试图隐藏缺陷。这条看着像道德要求，其实是技术要求。藏起来的缺陷不会消失，只会在生产上更贵地出现。而且你藏十个，缺陷密度、逃逸率全都不准了。第九条，收集数据做分析。数据不分析就是垃圾，分析了才是资产。

>

> 九条归到三个层次。制度管权限和标准，习惯管沟通和看板，文化管不隐藏和做分析。工具能解决第三层的一部分，前两层只能靠团队自己。

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他的工作。

- 答"重开会增加计数，影响考核"——这个想法本身暴露了管理问题。重开率一旦用来扣分，大家就不重开了，数据立刻失真。

- 答"系统里有记录就行"——记录给事后，沟通给当下，两样都要。

- 收口：技术问题用技术证据解决，人的情绪也得照顾，这两件事不冲突。

> **易错点 / 考点：**九条至少能说出五条；"只有测试人员有关闭权限"是最高频考点；判断"隐藏部分缺陷能让指标好看"——错，还会毁掉度量的可信度；简答"制度层面有哪些"，答权限和标准定义。

> **过渡：**原则说完了，落到操作层面是八个动作。

【13.27】一些缺陷处理的基本操作（2 分钟）

缺陷刚提交、还没人动手，这一刻有一道关。

>

> 四个动作，都在受理环节。第一个，审阅。由测试管理员、项目管理员或别的人来做，审的是报告的质量水平。审什么？步骤清不清楚、能不能复现、有没有重复、字段全没全。这一关的作用是在单子流到开发手里之前先把质量兜住。

>

> 第二个，拒绝。如果审阅者认为这份报告需要重大修改，要和测试人员一起讨论，由测试人员自己改，改完再提交。这里有两个细节。一是"一起讨论"，不是打回去让他猜。二是"由测试人员改"，提交人对内容负责，别人代改，责任就糊了。

>

> 第三个，完善。原文说，完整地描述了问题的特征并把它分离，审查者就会肯定这个报告。"分离"是关键：把混在一起的问题拆开，一条一个。拆开之后报告才叫完善，可以放行。

>

> 第四个，分配。分给合适的开发人员。不知道具体是谁，就分给项目开发组长，由组长再分下去。很多单子卡在"不知道该给谁"上，于是没人管。兜底就是给组长。原则是缺陷必须有一个明确的责任人，宁可给大一级，不能悬空。

>

> 四个动作连起来是一条流水线。审阅，不合格就退回去协助整改；合格就拆分补全；然后分配责任人。

> 四个里最容易被跳过的是审阅。小团队觉得反正就三个人，提了直接改。可一旦没人看质量，返工就从开发那边挪回你自己这边，来回两趟比当场补一步贵得多。

> **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"——对，拒绝是流程内的返工，关闭是终态。

- 答"都能让它消失"——区别就在这。拒绝留下记录，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，真实问题就被盖住。

- 答"直接关掉省事"——省的是当下，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：**四个动作的名称和含义；拒绝的正确做法是讨论、由测试人员改、再提交，不是审阅者代改；分配的兜底是给开发组长；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，缺陷进入修复和验证，这一段最容易吵起来。

【13.28】一些缺陷处理的基本操作 – 续（3 分钟）

> 改一行代码，你敢不敢不回归？

>

> 四个动作。第一个，验证。原文说，缺陷的修复需要得到测试人员的验证，同时还要做回归测试，看这个修复会不会引入新问题。"同时还要"四个字是重点。很多团队验证只做一件事：把当初失败的步骤再走一遍，通了就关。不够。改一处可能影响调用它的十处，修改本身就是风险的来源。所以验证是原缺陷验证加回归测试，两步都做完才算过。

>

> 第二个，重新打开。重开要加注释说明、电话沟通，否则"打开—修复"来回几轮，测试和开发就会产生没必要的矛盾。课件没说"不许重开"，说的是"重开要带配套动作"。重开是测试的权利，行使权利要带证据、带沟通。

>

> 第三个，关闭。只有测试人员有关闭权限，开发没有。这一条在本章出现两次，说明它重要。背后的逻辑是验证权和修复权分开：谁改的，谁不能判合格。

>

> 第四个，暂缓。如果每个人都同意把一个确实存在的缺陷放到以后处理，就要指定下一个版本号或者修改的日期。新版本一开始，这些暂缓的缺陷要重新打开。三个要点。暂缓是共识，不是单方决定。必须挂版本号或日期，不能空口说"以后再说"。新版本要主动重开，这条最容易漏，漏了暂缓就等于删除。

>

> 四个动作讲的是同一件事。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。

提问：开发说"这个改动很小，就一行，不用回归了"。你怎么回？

预设回答：

- 答"行吧，确实小"——风险就在这。一行改动造成大面积故障，是行业里的经典事故类型，越小越让人放松警惕。

- 答"还是要回归，至少覆盖调用它的相关功能"——对。回归范围能按影响面裁，不能裁到零。

- 答"那你写个说明，出事你负责"——把技术问题变成了责任划分，问题还在。

- 收口：回归的规模可以谈，做不做不能谈。

易错点 / 考点：四个动作的要点；验证必须包含回归测试，这是高频考点；关闭权限只归测试，全章出现两次，必考；暂缓的三个要求是共识、挂版本或日期、新版本重开；案例常给"开发自己关缺陷"让你指出违规点。

过渡：八个动作放到项目层面，要达成什么？课件给了三条目标。

【13.29】软件缺陷的处理和跟踪（2 分钟）

缺陷管理走到最后，你希望它是零缺陷，还是零未决？

>

> 第一个目标，确保每个被发现的缺陷都能够被解决。紧接着有一句解释。"解决"不一定是被修正，也可能是

> 别的处理方式。比如延到下一个版本，或者因为技术原因不能修。

>

> 这句话纠正一个普遍误解。很多人以为解决等于修好。其实目标是每条缺陷都有一个明确的、被记录的处理结论。结论可以是修复、延期、拒绝、无法解决。真要杜绝的是没有结论，那条一直挂着，谁也不知道该拿它怎么办。

>

> 第二个目标，收集数据，根据趋势曲线识别测试处在哪一段。发现率还在高位，说明测试还早，功能在大量暴露问题。发现率降、修复率升，说明进入收敛期。发现率很低但修复也停了，可能是测得不透，也可能是团队在等版本。曲线是测试过程的体温计。

>

> 第三个目标，决定测试能不能结束。课件说，用趋势曲线来判断是常用而且比较有效的方式。为什么有效？缺陷收敛是质量趋稳的间接证据。用例跑完不代表质量稳，可能用例设计得不够，或者跑得太浅。可发现和修复都收敛，说明要暴露的暴露得差不多、要修的修完了。

>

> 三条连起来：每条有结论，用数据看处在哪一段，用趋势判断能不能停。
>
> 这三条不是三个部门各自的事。测试定结论，开发出数据，项目经理拿曲线排期。谁手里都只有一段，合起来才是一条完整的链。

第 4 章讲的退出准则，在这里有了一个能量化的抓手。

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，没修就没解决"——最常见的理解，可课件明确说不修也是一种解决方式。

- 答"算，只要有明确结论和版本号"——对，关键在有结论、有归属。

- 答"算，但要写进测试报告"——补得好，延期缺陷是发布风险的一部分。

- 收口：零缺陷是理想，零未决是纪律。

易错点 / 考点："解决"的定义是必考判断；趋势曲线有两个用途，识别阶段和决定是否结束；判断"测试结束

的标志是用例全跑完"——不完整，还要看缺陷收敛和退出准则；简答"怎么用缺陷数据判断能不能结束"。

> **过渡：**既要集体决策，那这个集体是谁、怎么开？课件给了个很形象的名字。

【13.30】集体 review 缺陷："三国会议"（2 分钟）

> 三个组长关一间屋子，把没解决的缺陷一条条过。这个会叫三国会议。

>

> 为什么是三方？项目经理和开发组长、测试组长。因为缺陷的去留是个利益平衡的事。开发想少改、想稳；测试想全修、想保险；项目经理要按期交付。谁单独定都会偏。

>

> 参加者是组长级，不是全员。会要能拍板，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缺陷，不是凭印象，是一条条看记录。第二，决定 fix 优先级。第三，判断能不能等下一个里程碑或版本再解决，这就是在定"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。

>

> 第五条最值钱。会议的产出不只是哪些修哪些不修，还包括"按现在的缺陷状况，我们什么时候能发"。
>
> 三方坐一起还有个副作用。开发当场听到测试为什么把这个判成严重，测试当场听到开发为什么说这个改不动。这种交换写在单子上你说不清，会上三句话就说清了。

>

> 这五件事和前面九条技巧、和 13.2 的优先级判定对得上。三国会议就是把那些规则落到一次会议里的机制。

>

> 怎么开才有效？三条。带着数据开，别空口讨论。只议有分歧的，能按标准自动定的不用上会。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，别只在纪要里写"待定"。

>

> 还有一点，这个会要有固定的输入和输出。输入是排好序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。

> **提问：**为什么这类会要组长来，而不是每个开发和测试都到场？

> **预设回答：**

- 答"人少好决策"——对，这类会核心是取舍，取舍要能拍板的人在场。

- 答"组长更懂全局"——也对，他知道排期和资源约束。

- 答"应该全员参加才透明"——透明有价值，可全员会容易变成辩论赛。透明靠纪要公开就够。

- 收口：决策会要小，信息要公开，两件事不矛盾。

> **易错点 / 考点：**三方参加者；五件事要能列举，尤其预测进度和发布时间；判断"三国会议只决定修不修"——还包括优先级、责任人、进度；它和 13.3.1 的"周期性会诊"是同一件事的两种说法，别当成两个概念。

> **过渡：**会上要看的核心材料是趋势曲线，从这里进 13.3.2。

【13.31】13.3.2 缺陷趋势分析（3 分钟）

> 一张图，横轴时间，纵轴缺陷数，上面三条线。它回答的问题比你想的多。

>

> 趋势分析监控的是打开、关闭、已修正的缺陷随时间怎么变。

>

> 课件给了三句判断。第一句，产品开发质量情况取决于累积打开和累积关闭曲线的趋势。"累积"两个字要咬住。累积打开只增不减，因为它是历史上打开过的总数。累积关闭也是只增不减。两条线的间距，就是现在还欠着多少。

>

> 间距在缩小，说明修得比发现快，质量在收敛。间距在扩大，说明发现快过修复，越欠越多。

>

> 第二句，项目进度取决于累积关闭和累积打开曲线起点的时间差。这句话绕，翻一下。两条线开始明显上升

> 的时间点之间隔多远，反映的是从开始发现问题到开始解决问题这段滞后。滞后短，团队响应快；滞后长，发现和修复脱节，进度就被拖住。

>

> 第三句，开发和测试的工作进度、效率也能看出来。斜率就是效率。测试的斜率反映发现速度，也反映投入；开发的斜率反映修复速度。

>

> 三句合起来，趋势曲线回答三个问题。质量在变好还是变坏，看间距；团队响应快不快，看起点差；谁快谁慢，看斜率。

>

> 还有一句方法上的话。趋势分析最怕数据被污染。状态乱标，把没修的标成延期；同一个问题提五张单；版本末一次性关掉几十条。这些都会让曲线画出漂亮的假象。曲线不会骗人，往里头灌的数据会。

> **提问：**累积打开和累积关闭的间距越来越大，说明什么？

> **预设回答：**

- 答"缺陷越来越多"——准确说是欠的越来越多，发现快过修复。

- 答"测试太猛了"——有可能，更常见的是修复资源不够，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。间距扩大该加强的是修复，不是发现。

- 收口：间距扩大是修复侧的警报，不是发现侧的成绩。

> **易错点 / 考点：**累积曲线只增不减；三句判断的对应关系是质量看间距、进度看起点差、效率看斜率；判断"趋势只看打开数"——错，三条一起看；数据质量对趋势的影响是简答得分点。

> **过渡：**趋势画成图，最直观的一种就是累积曲线。

【13.32】累积缺陷趋势分析（2 分钟）

> 先分清一件事：累积曲线和每天新增，不是一张图。

>

> 累积缺陷趋势，图上看四件事。
>
> 为什么两条线要画在一张图上？单看打开数，你只知道发现了多少；单看关闭数，你只知道修了多少。两条放一起，才知道还差多少。

>

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

>

> 第二件，开口的方向。间距先扩大再收窄，是健康的形状。前期欠着是正常的，测试在集中发现；关键是后期要收到足够小。开口一直扩大，这个版本不能发。

>

> 第三件，交叉和平行。真实项目里，累积关闭几乎追不上累积打开，因为总有一部分缺陷被拒绝或被延期。所以判据不是"交叉"，是"间距缩小到可接受范围"。很多人会想成两条线必须相交，那是想错了。

>

> 第四件，斜率的变化点。曲线上每次明显折弯都有原因。突然变陡，可能新模块进测试了；突然变平，可能环境挂了，或者测试资源被抽走。折点就是故事，看到折点要去问那几天发生了什么。

>

> 用这张图有一句话。累积曲线看趋势，不看绝对值。绝对值受项目规模影响，趋势才反映过程受不受控。
>
> 还有个常被问的点。累积曲线的起点不是项目第一天，是第一次记录缺陷的那天。起点选错，整张图的斜率和间距都跟着失真。

> **提问：**累积关闭线要不要追上累积打开线，测试才算能结束？

> **预设回答：**

- 答"要，追上了才说明都修完了"——理想化。总有被拒绝、被延期的，追不上是常态。

- 答"不用，看间距缩到可接受范围"——对。

- 答"看剩下的缺陷什么级别"——补得好。剩下的都是微小的，风险就小。

- 收口：能不能结束，看剩余缺陷的结构，不只看数量。

> **易错点 / 考点：**累积曲线的性质和两条线的形状特征；判断"两条线必须相交才能结束测试"——错；折点的诊断价值是简答亮点；和 NOB 曲线的区别是存量对总量，这个辨析常考。

> **过渡：**图上还有个位置要单独讲，就是两条线最接近的地方。

【13.33】收敛点（2 分钟）

> 两条曲线靠得最近的那个地方，有个名字。

>

> 图横轴是日期，纵轴是 Bug 数量。两条线分别是报告的 Bug 和解决的 Bug。交汇的位置被标出来了，叫收敛点。

>

> 什么叫收敛点？报告的数和解决的数趋于接近、曲线走平的那个位置。在这个点上，团队基本追上了欠下的量。新发现的速度和修复的速度大致平衡，剩下没解决的缺陷稳定在一个较低水平。

>

> 它为什么重要？因为它是"可以进入发布评估"的信号。
>
> 为什么偏偏这个位置值得单独命名？因为发布决策都压在这一段。曲线还在张开的时候讨论发不发，谁都没底气；走平之后再讨论，谈的是剩下的风险，不是还要不要等。

是进入评估，不是可以发布。收敛只说明过程稳了，还得看剩余缺陷的严重性结构、还有没有高风险模块没测。

>

> 图上还有个容易忽略的细节。收敛之后两条线不该再明显张开。又张开，说明出现了新的问题源，可能是新功能合入，可能回归引入，也可能测试范围扩大。这种情况要重新评估。

>

> 还有统计口径。这张图上的"解决"通常指已修正、已验证，不是已关闭。关闭还要过测试验证，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。

>

> 所以一句话。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。

> **提问：**项目已经收敛了，是不是就能安排发布？

> **预设回答：**

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

- 答"还要看剩下的都是什么级别"——对。十条微小和十条致命，不是一回事。

- 答"还要看有没有回归引入的新问题"——对，收敛后又张开，常常就是回归。

- 收口：收敛是必要条件，不是充分条件。

> **易错点 / 考点：**收敛点的含义是发现速度与修复速度平衡、存量稳定；判断"收敛即可发布"——错；统计口径问题（解决和关闭不一样）是易错点；和 13.5 测试结束准则的联系常出综合题。

> **过渡：**理想是平滑收敛，现实里的收敛常常是反弹着完成的。

【13.34】不断反弹之中收敛（2 分钟）

> 真实的缺陷曲线从来不光滑，它一路上反复抬头。

>

> 标题就是结论，在不断反弹之中收敛。为什么会反弹？四个常见原因。

>

> 第一，修复引入新缺陷。改 A 影响了 B，最典型的反弹来源。这也是 13.3.1 强调验证要带回归测试的原因。

>

> 第二，新功能合入。敏捷迭代里每个迭代都提交新代码，新代码带来新缺陷。所以持续交付的团队，曲线永远不可能真归零，只能在版本末期收敛。

>

> 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，一扩范围就冒出一批新缺陷。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，原来不报的开始报。

>

> 四种性质完全不同。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，别一律当成质量不行。

>

> 由此有个判断方法。看趋势不要看一天的抖动，要看一段时间的包络。把高点连起来，包络在降，就是在收敛；包络也在升，那才是真问题。

>

> 这一条给的启示是两层。> 有人会问，既然反弹是常态，那画曲线还有什么用？用处在于看方向。反弹还在，可一次比一次低，说明在收敛；一次比一次高，说明问题在积累。

>

> 这一条给的启示是两层。缺陷管理要有耐心，也要有分辨力。耐心是指接受反弹，分辨力是指分清原因。

提问：版本末期曲线又抬头了，怎么判断是好事还是坏事？

预设回答：

- 答"抬头就是坏事"——不一定。如果是扩了测试范围，反而是好事。

- 答"看新增集中在哪"——专业。集中在刚测的模块，是范围扩大；集中在刚修的模块，是修复引入；散在全系统，可能是环境。

- 答"看严重性"——也对，新增的都是微小问题，风险有限。

- 收口：先归因，再下结论，顺序反了就会误判。

易错点 / 考点：四种反弹原因和性质的区分；判断"缺陷曲线应严格单调下降"——错，反弹是常态；"看包络不看抖动"能写进答题；和 13.3.2 趋势分析结合出案例分析的概率较高。

> **过渡：**时间轴看完了，换个切法就是分布分析。

【13.35】13.3.3 缺陷分布分析（2 分钟）

同一个缺陷库，按模块切一刀，按类型再切一刀，答案完全不一样。

>

> 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，比如测试需求和缺陷状态、严重性的分布情况。

>

> 这句话拆开就是：把缺陷按某个属性分组，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。

>

> 它和趋势分析是一对。趋势按时间切，分布按属性切。一个回答什么时候出问题，一个回答哪里出问题。

>

> 分布最典型的用途是找聚集。13.4.1 里有一句问话：缺陷分布符不符合正态分布？是不是过于集中在 1 到 2 个模块？这就是判据。高度集中在少数模块，通常说明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，如果均匀散落在所有模块，也不是好消息。可能测得不深，只碰到表面的共性问题，没碰到各模块特有的深层问题。

>

> 常用角度几个。> 几个角度不是随便挑的。你要回答哪个问题，就切哪个属性。想知道下一轮测哪儿，切模块；想知道发布风险，切严重性；想知道过程哪一段漏了，切发现阶段。

>

> 常用角度几个。按模块找热点，按严重性看风险结构。按类型看是功能设计问题还是实现问题。按来源看哪

> 个阶段引入最多，按发现阶段看缺陷是在哪一层被拦住。最后一个直接连到 13.4 的缺陷清除率。

>

> 分布的价值不在数得清，在看出该改哪里。

> **提问：**80% 的缺陷集中在两个模块，你的第一反应是什么？

> **预设回答：**

- 答"这两个模块的开发水平不行"——最容易想到，也是最危险的对人归因，往往不是真相。

- 答"这两个模块最复杂、改得最频繁"——更可能。复杂度高、变更多，缺陷自然多。

- 答"这两个模块被测得最认真"——这个可能性必须留着。有时候不是它们差，是只有它们被认真测了。

- 收口：集中至少要分三种原因，复杂度、变更频率、测试投入不均，别急着定。

> **易错点 / 考点：**分布分析的定义是缺陷数量与缺陷属性的函数；和趋势分析的辨析（时间对属性）；判断"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；常用维度至少能列四个。

> **过渡：**分布分析长什么样？看一张示例图。

【13.36】示例：缺陷分布分析（2 分钟）

> 一张分布图摆在面前，先看哪儿？

>

> 图不用逐格念，要学会三个动作。

>

> 第一个动作，先看横纵轴各是什么。> 为什么先看轴再看形状？轴读错了，形状全白读。有人把占比当数量看，一个三成的小模块被当成最大的热点，结论正好反过来。

>

> 第一个动作，先看横纵轴各是什么。横轴通常是分类属性，模块、类型、严重性、状态；纵轴是数量或占比。这两头一确定，图的含义就定了。一张没标清轴的分布图，没有分析价值。

>

> 第二个动作，看形状。形状基本三种，每种一个结论。集中型，某一两个分类明显高出一截，形成尖峰，去找那几个模块的共性问题，通常是复杂度或变更热点。均匀型，各分类差不多，要么各部分质量均衡，要么测得不深、只摸到表面。长尾型，少数很高、大量很低，这是最常见的真实形状，抓头部就行。

>

> 第三个动作，看它跟上一版的变化。单张只能说明现在哪里多，两张对比才能说明改进有没有效。上一版集中在那两个模块，这一版还是那两个，说明措施没落地；换了模块，说明那两个修好了，问题在别处。

>

> 还有个常见误用：别用分布图评价个人。> 三个动作有先后。轴定含义，形状定结论，对比定改进。少一步，这张图就只是一张图。

>

> 还有个常见误用：别用分布图评价个人。按提交人切分缺陷数，看着像绩效排名，可缺陷数受任务难度、测试投入、模块复杂度影响太大。这种图一用来考核，数据立刻失真，大家会少提缺陷，或者把缺陷提给别人。

> **提问：**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，也可能是订单这次改动大。

- 答"要结合变更量看"——专业。缺陷密度是缺陷数除以规模，不看规模只看数量会误判。

- 答"可能测试重点转过去了"——也要考虑，这一版可能重点测了订单。

- 收口：分布变化要跟变更量、测试投入一起读，不然容易误伤。

> **易错点 / 考点：**三个观察动作；三种形状对应的结论；判断"缺陷数多的模块质量就差"——错，要看缺陷密度；"用缺陷数据考核个人会导致数据失真"是重要考点，为 13.4 埋了伏笔。

> **过渡：**趋势和分布都还在看现象，再往里一层是挖原因。

【13.37】缺陷分析：从 Bug 中学习（3 分钟）

> 修完一个 Bug，你带走的是什么？

>

> 标题叫从 Bug 中学习，思想性在 13.3 里最强。它给了三个词、三个方法、三个问句。

>

> 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，是人的能力或态度？是流程缺了环，比如没有评审、没有回归？还是技术限制，比如框架缺陷、工具不够？多数团队习惯把问题归到人头上，可真正能改的往往是过程和技术。

>

> 第一个方法，根因分析。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"规范里没强制边界检查，评审清单也没这一项"。改直接原因只修一个 Bug，改根因能修一类。

>

> 第二个方法，案例分析与学习。把典型缺陷整理成案例在团队里讲，讲的不是谁写的，是这类问题长什么样、怎么早发现。这是组织级的经验积累。

>

> 第三个方法，缺陷预防：代码规则、设计 Checklist。这是根因分析的产出物。分析完不落成规则，下次还犯；落成规则，编码和评审阶段就能拦住。

>

> 三个问句：问题是什么，根因是什么，对策是什么。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。

>

> 配一条判据。一个对策，如果没法验证有没有执行、也没法验证有没有效，它就不是对策，是口号。"加强代码评审"是口号；"评审清单里加一条，所有外部输入必须判空，每次评审记录里勾选"，才是对策。

> **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，没证据"——对，而且粗心不可改、不可验证。

- 答"没分析流程原因"——关键。为什么一个粗心能造成事故？是不是缺评审、缺自动检查、缺回归？这些才可改。

- 答"归因到人会影响氛围"——也是实际后果，以后没人愿意暴露问题。

- 收口：好的根因分析，结论一定指向能改的东西，流程、规则、工具，不是性格。

易错点 / 考点：三个层面和三个方法；三问的顺序不能乱；判断"根因分析就是找出出错的那行代码"——错，那是直接原因；简答"怎么从缺陷反推过程改进"——根因分析、落成规则和清单、前移到评审与自动化。

【思政落点 4】：这一块讲的是复盘与担当。一个团队敢不敢做根因分析，看它把缺陷当什么。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。落到价值观上，就是出了问题先问我们的流程哪里漏了，少问谁该背这个锅。一个人愿意把自己漏掉的缺陷写进复盘，成长会比只会说"这不是我的问题"的人快得多。

过渡：分析结果怎么天天摆在团队眼前？靠看板。

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

墙上挂一块看板，数据很好看。然后呢？

>

Bug dashboard，缺陷看板。看这类看板要抓三件事：给谁看、看什么、看完做什么。

>

先说给谁看。三类读者。> 为什么强调按角色分区？同一块看板，测试要的是今天该验哪几条，项目经理要的是这周还欠多少。两类信息混在一屏，两边都得自己筛一遍。

>

先说给谁看。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和风险。所以好的看板不是把数据全堆上去，是按角色分区。

>

> 再看常放的几块。缺陷总量和未解决量，回答还欠多少。按严重性、优先级分组，回答欠的是不是要命的。按状态分布，回答卡在哪个环节，比如大量停在"已修正待验证"，说明验证是瓶颈。按模块分布，回答哪里最痛。新增和修复的日速、周速，回答追不追得上。缺陷年龄，回答有没有放了很久没动的单子。

>

> 看完做什么？这是看板最关键的地方。看板不是展示，是驱动动作。看到大量停在待验证，就加验证人力。看到某模块年龄普遍偏长，就开专项会诊。看到未解决数连续三天上升，就去找原因。

>

> 有个反模式叫装饰性看板。数据好看，没人照它做决定。判断它有没有用，只问一句：上周你看板上哪个数字，让你做了什么决定？答不上来，它就是墙纸。

>

> 还有一条，看板要实时或准实时。13.3.1 说实时监控，13.3.2 说实时趋势分析，两次都强调实时。周报式的数据，等你看出来已经晚了。

提问：看板上大量缺陷停在"已修正待验证"，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，问题在验证没跟上。

- 答"验证是瓶颈，加验证人力或简化验证流程"——对。

- 答"说明提得多、修得快"——只对一半。卡在验证，发布还是被拖住。

- 收口：状态分布图最大的价值，就是找出流程里最堵的那一段。

易错点 / 考点：看板的核心内容和三类读者；判断"看板只要数据全就行"——错，要能驱动动作、要按角色分区；"已修正待验证积压等于验证瓶颈"是典型判读题；"实时"在课件里出现两次，属于考点。

> **过渡：**看板上那些图，拆开就是四类缺陷报告。

【13.39】示例（2 分钟）

> 读看板这件事，我建议练成肌肉记忆，四步，顺序固定。
>
> 第一步，找总量和趋势。> 为什么四步要固定顺序？风险判断有依赖。不知道总量，就看不出某一列是不是异常；不知道瓶颈，就分不清某个模块缺陷多是真的差，还是卡在没人验。
>
> 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，走向告诉你变好还是变坏。这两个数字定调，别的都是细节。
>
> 第二步，找瓶颈。看状态分布，哪一列柱子最高，瓶颈就在哪。停得最久的那一堆，就是流程里最堵的地方。常见三个：待分配，没人认领；待验证，验证人力不够；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。这个模块接下来重点测、重点评审，这就是分布分析的落地。
>
> 第四步，找异常。看严重性分布和缺陷年龄。有没有致命、严重的挂了很多天？有没有一批年龄超过两周？这些是风险的集中体现，必须当场上报。
>
> 四步走完，看板才叫读完，而且读出了动作。> 四步练熟，一张看板从头看到尾不超过三分钟。三分钟换来当天该找谁、该催哪条，这个时间花得值。
>
> 四步走完，看板才叫读完，而且读出了动作。
>
> 还有一句看板之外的话。看板上的数字是给团队用的，不是给上级检查的。一旦它变成检查工具，团队就开始管理看板，而不是管理缺陷。把单子改得好看一点，把不好看的关掉。这个现象在行业里很普遍，你们以

> 后带团队要留意。

> **提问：**一张看板上所有指标都很漂亮，你会信吗？

> **预设回答：**

- 答"会，说明质量好"——要谨慎。全好通常两种解释：质量真行，或者数据被管理过。

- 答"抽几条原始单看看"——专业。抽五条看步骤全不全、状态合不合理，就能判断数据可不可信。

- 答"看跟上一版比有没有突变"——好思路，突然全面转好往往不正常。

- 收口：读图是看板的第一层，判断数据可不可信是第二层。

> **易错点 / 考点：**读板四步的顺序；三个常见瓶颈是待分配、待验证、待修复；判断"看板指标越好质量越好"——要先看数据质量；"抽查原始记录验证可信度"能写进答题。

> **过渡：**看板上那些图，其实是四类报告的组合，逐类看定义。

【13.40】缺陷报告（3 分钟）

> 四类报告，名字长得像，各管一件事。定义都写得很短，所以特别好考。

>

> 第一类，缺陷分布报告。原文是这么说的：允许把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，比如测试需求和缺陷状态、严重性的分布情况。关键词是函数和属性，横轴属性，纵轴数量。用途是找热点、看结构。

>

> 第二类，缺陷趋势报告。按各种状态把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。累计和非累计的区分很重要。累计只增不减，看长期走向；非累计是每天新增多少、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，看一个缺陷在某种状态里待了多久，就知道处理进度卡在哪。这一类最容易被忽视，也最实用，因为它直接暴露卡住的地方。一个团队缺陷总数不多，可平均

> 年龄 30 天，这比数量多但三天清完要糟得多。

>

> 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里的执行结果以及测试周期。它跨版本，回答的是这几个版本测执行得怎么样，介于测试报告和缺陷报告之间。

>

> 四类的分工一句话。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。

> **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。趋势看总量，会盖住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，按状态看时长，能定位卡在验证还是卡在修复。

- 答"跟趋势差不多"——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，年龄报告照的就是结构。

> **易错点 / 考点：**四类定义要对上号，尤其分布等于属性函数、趋势等于时间函数；趋势报告分累计和非累计是高频细节；缺陷年龄报告的定义常考；判断"测试结果进度报告就是测试报告"——不是，它只是数据来源之一。

> **过渡：**除了日常跟踪，行业里还有一种集中清理缺陷的活动，下面接着看。

第 13 章讲稿（13.41–13.60，v3）

> 锚点与课件一一对应，不增删页。每页五个标记，口播按四拍：起 → 推 → 收。

【13.41】“Bug bash” 活动（2 分钟）

> 发布会前一周，全公司被叫去点自己的产品。产品、开发、测试、运营、市场，坐一屋，半天不干别的，只找毛病。找到就当场提。这就叫 Bug bash，中文是缺陷大扫除。

>

> 换个角色，为什么就能找出新东西？测试天天走熟路，路径依赖，总走那几条。产品按业务场景走，开发按实现细节走，市场按真实用户那种乱点走。用法不一样，撞出来的缺陷就不一样。

>

> 第二，成本低、见效快。半天，几十个人，扫出来的多是显而易见的东西。错别字、文案不通、按钮写的和功能不符、界面明显错位。单条不重要，堆起来就是"这产品不精致"。

>

> 它还有一层作用：大家一起看见产品现在什么样，对能不能发，判断会更一致。

>

> 局限得说清。它扫不出深层缺陷，替不了系统测试、性能测试、安全测试。所以一般放在主要测试做完、准备发布那一段，当补充。

>

> 组织上还有一条容易忘：活动前得有人写清"今天找什么"，比如专攻新用户从注册到首单那条路。方向不写，几十个人会同时挤在首页。

>

> 组织上两条。一是统一提交、统一标签，比如都打上 bugbash-2026xx，不然事后没法统计。二是当场去重和分诊，几十个人同时提，重复率极高，得有人实时合并。做完出小结：提了多少、去重后多少、修了多少、哪些延期。

>

> 收一句：它补的是打磨，不是兜底。

> **提问：**Bug bash 排项目哪个阶段最合适？为什么？

> **预设回答：**

- 答"一开始就搞"——效果差，功能还不完整，扫出来全是"没做完"。

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

- 答"随时都可以"——形式上成立，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 中文含义是缺陷大扫除，组织形态是集中式、跨角色；② 价值和局限各记两条；③ 判断题"Bug bash 能替代系统测试"——错；④ 组织要点（统一标签、当场去重、事后小结）是实操题得分点。

> **过渡：** 把这一节的实践收成清单，五条全叫"必须"。

【13.42】必须做的…（2 分钟）

> 五条全写成"必须"，为什么条条都得这么硬？一条一条看。

>

> 第一条，尽早定义和推广 Bug 管理流程。关键词是"尽早"和"推广"。流程不是出事之后再补，项目第一天就得有。推广是说所有人都知道、都会用。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的衡量标准。这条在 13.2.2 和 13.3.1 都露过面，第三次出现，说明它真要紧。标准要具体到"什么情况算致命"，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，也就是 13.2.4 那些必填字段。设成必填，是从系统这一层保证数据质量。靠自觉靠不住。

>

> 第四条，周期性地会诊 Bug，就是三国会议。周期意味着固定节奏，比如每周一次，不是想起来才开。

>

> 第五条，周期性地检查 Bug 列表。它跟第四条不是一回事：会诊是集体决策，检查是责任人自查。看自己名下有没有到点没处理的、信息不全的、长期挂着的。

>

> 五条串起来是一条链：流程、标准、字段、决策、巡检。齐了，缺陷管理才自己转，不靠某个人天天催。

>

> 给你一个自检法。拿这五条去问一个团队，缺两条，它就是人治。核心成员一走，就散。

>

> 五条里最容易被当成形式的，是第一条。流程写出来贴在墙上、没人按它走，比没有流程更糟，因为它让人以为已经有了一套。

> **提问：**"周期性会诊"和"周期性检查列表"，为什么两个都得做？

> **预设回答：**

- 答"一个开会一个看表，重复了"——不重复。会诊管有分歧、要取舍的；检查管被漏掉、没人管的。

- 答"会诊管集体，检查管个人"——说得准。

- 答"只做会诊就够了"——会诊一周一次，这一周里新挂的单子没人看。检查可以每天做。

- 收口：决策要集体，巡检要高频，节奏不同，互相替不了。

> **易错点 / 考点：**① 五条要能列举；② "推广"两字最容易被漏；③ 会诊（集体决策）与检查（个人自查）的辨析；④ 简答"如何建立可持续的缺陷管理机制"，按这五条答。

> **过渡：**知道要做哪些，还得知道哪些不能做，课件也列了四条。

【13.43】不要做的…（2 分钟）

> 有些做法看着省事，只是把麻烦挪到了以后。

>

> 第一条，把 I18N Bug 和 L10N Bug 混为一谈。I18N 是国际化，产品本身要支持多语言、多地区，属于架构和代码的事。文字硬编码、日期格式写死、货币符号不支持，都算这一类。L10N 是本地化，针对某个具体地区做适配，属于内容和配置的事。日期习惯、法规文案、计量单位，差异都在这一层。混在一起有两个后果。架构问题被当成翻译问题，一直修不好；或者内容问题被拔高成架构问题，白花力气。

>

> 第二条，试图隐藏你的 Bug。动机多半来自考核。缺陷数跟绩效挂钩，隐藏就成了理性选择。治这个现象，

> 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。13.3.1 讲过，只有测试人员有关闭权限。让开发自己关，等于把验证环节取消。再补一句：测试也别去关自己发现、自己修的问题，原则是谁改谁不判。

>

> 第四条，未经诊断就发布一个补丁，这条最要命。没搞清原因就上线一个改动，生产事故的紧急处理里最常见。后果是老的没解决，新的进来；情况急、没有回归，新问题的破坏力更大。正确顺序是：先止血（回滚、限流、降级），再定位，再修，再验证，再发布。

>

> 四条有个共同点：都在省一步。省掉区分、省掉暴露、省掉验证、省掉诊断。省下的每一步，后面都要还。

> **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"——最常见的止血，代价最小、最可预测。

- 答"限流、降级"——也对，回滚成本高的时候用它。

- 答"赶紧改代码修"——正好踩上"未经诊断发补丁"：改动本身是新的风险源，而这时你还没搞清原因。

- 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（国际化能力）与 L10N（本地化适配）的区别，本节最可能出题；② 判断题"开发可以关闭自己修的缺陷"——错；③ "先止血再定位"的顺序是案例题得分点；④ 隐藏 Bug 的根因常在考核口径。

> **【思政落点 5】**：隐藏缺陷往前追一步，就不只是流程问题。缺陷不会因为你没报就消失，它只在用户手里出现。手上的技术可以练，肯不肯把坏消息报上去，是当场就能选的一件事。把问题说出口的人短期看着吃亏，长期是团队敢把系统交给的那个人。

> **过渡：**这些做和不做，最终都要落在一个系统上。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）

> 三个人的团队，缺陷记在哪儿？很多组是微信群。

>

> 微信群能存图能存话，不能筛、不能算、不能追。人一过十个就乱。课件给了六项作用，归成三类看就清楚了。

>

> 沟通这类两条。一，推动团队内部沟通，而且是真推得动的那种。系统是一份共享的事实。有争议，不用互相回忆当时谁说的。六，报告任何一个 Bug 的当前状态，任何人任何时刻都能查到这事走到哪一步，不用去问人。

>

> 管控这类三条。三，按优先级排列重要 Bug，能排序能筛，先看要命的。四，跟踪整体生命周期，从新建到关闭，每一步都在。五，记录任何跟 Bug 有关的操作，谁改的状态、什么时候改的、改之前的备注是什么，全留痕。

>

> 分析这条一条。二，提供报表和分析。数据聚成趋势、分布、年龄，撑着 13.3.2、13.3.3 和 13.4 的分析。

>

> 六项里最容易被低估的是第五项。很多团队只看能不能提缺陷，不看有没有审计日志。一旦有人问"这条谁关的、为什么关"，没日志就说不清；要做过程改进，没日志就复现不了当时的决策。

>

> 这六项还排成一个阶梯：先能存，再能看，再能推，再能学。往上一步都难。能看，要报表口径统一；能推，要有人盯提醒；能学，要有人肯认自己的问题。多数团队停在第二步。

>

> 所以价值不在"存"，在"推"：推沟通、推处理、推决策。

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

> **预设回答：**

- 答"方便追责"——能追，可主要价值不是追责，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：没日志的系统只是电子表格，有日志的才叫管理系统。

> **易错点 / 考点：**① 六项作用要能列举；② 判断题"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 三类归法便于记忆；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：**光说作用太虚，看一套真界面。

【13.45】示例（2 分钟）

> 打开一套缺陷系统，别从头看菜单。专门找五样东西，它们对应刚才那六项作用。

>

> 第一样，列表和筛选条件。按项目、状态、严重性、优先级、指派人、时间筛。筛选器就是管控的入口：能不能一键筛出"我名下所有未处理的严重缺陷"，决定它好不好用。

>

> 第二样，一条缺陷的字段区。对照 13.2.4 那张表挨个点。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往备注里塞信息，到头来统计不出来。

>

> 第三样，状态流转按钮。看它从当前状态能走到哪些状态。按钮就是流程。界面上摆着"开发直接关闭"，那条只有测试能关闭的制度就是废的。制度得靠系统强制，靠自觉撑不过一个赶版本的深夜。

>

> 第四样，操作历史。看它记不记谁在什么时候把状态从 A 改成 B、备注是什么。这就是审计。

>

> 第五样，关联关系。缺陷能不能挂到用例、需求、构建包、别的缺陷上（重复或相关）。第 11 章讲的追溯在这里落地，能回答"哪些需求被缺陷堵住了"。

>

> 补一句读法。这五样不必都翻到底，先看两处就够判断一套系统值不值得用：筛不筛得动，关不关得住。筛不动，工单表就是一本翻不完的册子；关不住，制度只剩文档。

>

> 五样看完，评价就有依据。字段决定数据质量，按钮决定流程有多硬。日志决定可信度，关联决定能分析多深。

提问：如果系统里允许开发直接把缺陷改成"已关闭"，你怎么处理？

> **预设回答：**

- 答"跟大家说一声别这么干"——靠自觉，撑不过赶版本的那几个深夜。

- 答"改权限配置，去掉开发的关闭权限"——对，制度要靠系统强制。

- 答"无所谓，测试能看出来"——看不出来。关掉的缺陷不进待验证列表，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 五样要素与六项能力的对应；② 判断题"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 "字段残缺导致数据统计不出来"是简答可用的因果链。

> **过渡：**具体到开源工具，从 MantisBT 看起，它也是实验 9 的主角。

【13.46】MantisBT 主要功能（2 分钟）

装 MantisBT，多数人卡在第一天。不是装不上，是装完不知道配什么。那就按你会用到什么来讲。

>

> 一，提交与字段配置。管理员能自定义哪些字段必填、哪些显示、下拉里有哪些选项。这就是"清晰设置提交必须的信息"的落点：把环境和版本设成必填，数据质量在系统这层就守住了。

>

> 二，状态与工作流。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。谁能从哪个状态改到哪个状态，管理员可配。这就是"只有测试能关闭"的落点。

>

> 三，过滤器与视图。常用筛选能存下来，比如"我负责的未解决缺陷"，还能订阅，缺陷一变就发邮件。实时监控靠它，不用天天手动刷。

>

> 四，报表与图表。按状态、严重性、模块、时间出统计，也有汇总报表。

>

> 五，权限与角色。报告者、开发者、管理员权限不同，配着工作流做职责分离。

>

> 六，通知与集成。邮件通知先放下，要紧的是跟 Git、SVN 这类版本控制关联。提交代码时引用缺陷编号，哪个缺陷由哪次提交修，就有据可查。这一条补上的正是 13.2.4 里构建包跟踪那一环。

>

> 实验 9 的验收线这样走。配好字段。配好工作流，关闭权限那一项别放过。建好过滤器。看一次报表。走一遍完整生命周期。五步走完才算会用。

>

> 这类老工具今天多半不是主流了，可状态机、权限、通知、报表这套模型，被后来所有工具继承下来了。学工具学它的模型，别只学按钮。

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

> **预设回答：**

- 答"方便灵活"——是，不过更要紧的是各团队职责划分不同，工具得能适配流程，不能让流程迁就工具。

- 答"默认配置一般已经限制了开发关闭"——对，但管理员得懂这条配置的意义，否则很容易被改松。

- 答"配置太麻烦，用默认就行"——默认往往偏宽松，一松就等于没制度。

- 收口：工具的默认值常常决定团队的实际流程，选型和配置时最要紧的就是这一点。

> **易错点 / 考点：** ① 核心能力：字段配置、工作流、过滤器、报表、权限；② 判断题"缺陷跟踪系统的工作流是固定的"——多数可配；③ 实验 9 的验收要能说出"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、构建包三者串起来）是加分点。

> **过渡：** 同类工具不止一个，课件列了六个，过一遍名字和定位。

【13.47】开源缺陷跟踪系统（2 分钟）

> 六个名字摆在一起，你多半只认识第一个。

>

> Mantis，地址是 mantisbt.sourceforge.net。它就是刚讲的 MantisBT。PHP 写的，部署简单、概念清楚。教学和小团队最常见，实验 9 用它。

>

> Bugzilla，地址是 mozilla.org/projects/bugzilla/。Mozilla 出品，历史很久。查询和统计能力强，大型开源项目用得多，界面和配置偏传统。顺带说一句，bug 这个说法能普及，跟它有很大关系。

>

> Bugzero，bugzero.findmysoft.com。商业产品加开源版本，支持多项目和邮件集成，配置较轻。
> Scarab，scarab.tigris.org，从 Eclipse 社区体系里出来的。强调可定制，适合想深改的团队。
> TrackIT，trackit.sourceforge.net。轻量、功能聚焦，只要基本跟踪就够用。Itracker，itracker.org，基于 J2EE 的老牌项目，企业里曾用得不少。

>

> 名字不必都背。要带走的是选型的六个维度。技术栈对不对，PHP 还是 Java。部署与运维成本。工作流能改到什么程度。报表能力。是否还在维护。社区活跃度。这几条里"是否还在维护"最实际。不少老项目多年不更新，装上去就是安全隐患，而缺陷系统里存的是产品和客户的敏感信息。

>

> 课件底部注明更多工具见一篇公众号文章，说明这份名单不是穷举，考的是选型方法。

>

> 今天很多团队不自建，用 SaaS 或综合平台。开源工具一半价值在功能，另一半在你能看懂它的流程。装一套，把状态机、权限、过滤器摸一遍，比看十页文档管用。

提问：选型时，为什么"是否还在维护"这么关键？

预设回答：

- 答"不维护就没新功能"——这是小事。要紧的是安全漏洞没人修，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，自建团队得自己扛。

- 答"能用就行，反正是内部系统"——内部系统被攻破常常更麻烦，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 六个工具的名称与定位，Mantis、Bugzilla 必记；② 选型维度能说出四条以上；③ 判断题"开源工具一定比商业工具差"——看场景；④ 名单不是穷举，考点在选型方法。

过渡：到这里 13.3 把怎么管讲完了。13.4 换个更硬的问题：拿这些数据，怎么算质量。

【13.48】13.4 产品质量评估与度量（1 分钟）

> 质量好不好，凭感觉能说得清吗？说不清，就得算。这一节是本章最"数学"的一块，骨架却只有一句话：质量不能靠感觉，得靠算。

>

> 四个小节按这个顺序排。13.4.1 基于缺陷的质量度量，用缺陷数据算。缺陷密度、缺陷清除率、缺陷逃逸率，加趋势和分布两个判据。13.4.2 经典的种子公式，往程序里种已知缺陷，看能捞出多少，反推还剩多少。13.4.3 基于缺陷清除率的估算方法。拿功能点、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。

>

> 这条线是递进的：先算单指标，再估总量，再估整体质量，末了做综合评价。顺序有它的道理。单指标先会算，才谈得上估总量；总量估出来，整体质量才有分母；整体质量有了，才谈得上加权重合成一个分。跳着来会出事。不会算密度和清除率，直接套种子公式，代值时符号都能填反。

>

> 提前打个预防针。公式看着多，重点不在推导，在每个指标回答什么问题、什么时候会骗人。密度低可能是质量好，也可能是测得浅。清除率高可能是修得好，也可能是发现得少。单个指标都会骗人，指标得成组看。

>

> 度量的目的不是打分，是发现问题、支撑决策。

> **提问：**一个项目缺陷密度很低，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"要看覆盖率和逃逸率"——对，成组看才有意义。

- 答"要看跟历史基线比"——也好，后面专门有一页讲缺陷度量-基线。

- 收口：分母不动、分子小，未必是好事。要看这个数是在多强的测试下得来的。

> **易错点 / 考点：**① 四小节的名称与递进关系；② 判断题"缺陷密度越低质量越好"——不成立，得配覆盖率、逃逸率；③ "指标要成组看"这条方法论简答常考。

> **过渡：**先看度量的整体框架：分几类，按什么步骤走。

【13.49】软件度量的分类与过程（2 分钟）

> 先量什么，比怎么量更要紧。这块先分类，再看过程。

>

> 三类：软件过程度量、项目度量、产品质量度量。区别一句话说得清。过程度量看活干得怎么样，评审有效性、缺陷注入率、测试效率都算。项目度量看进展如何，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。13.4 主要讲第三类，数据会用到前两类。

>

> 过程五步。

>

> 一，识别目标。分析出度量的工作目标和列表，由管理者审核确认。审核确认是说目标得有人认下来，不然数据做出来没人用。这一步要问的是：我们为什么量这个，量出来准备做什么决定。答不上来就别量。

>

> 二，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。给具体的收集活动、分析、反馈活动提供指导。IT 设备和工具怎么开发，也照这份定义走。说白了，把怎么收、怎么算、怎么反馈写成规范。不写，每个人算法不一样，数据没法比。

>

> 三，搜集数据。用 IT 工具收集，按指定方式审查和存储。关键词是审查。数据进来要校验，脏数据进去，结论一定错。

>

> 四，数据分析与反馈。按已定义的方法分析，出规定格式的图表，然后反馈。反馈是个动作，不是发个邮件，得有人看、有人讨论、有人拿它做决定。

>

> 五，过程改进。管理者基于度量数据做决策。这是整个循环的目的。度量不带来改进动作，五步全白走。

>

> 闭环是目标、定义、收集、分析反馈、改进，改完再重定目标。跟 13.3 的跟踪是一套思路，一个管缺陷，一个管度量。

提问：领导说"把这些指标都量起来吧"，你先问什么？

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"——对，这就是第一步识别目标。

- 答"有没有工具"——那是后面的问题，工具服务于目标。

- 收口：没有决策场景的度量，迟早都会变成填表负担。

易错点 / 考点：① 三类度量的名称与区别；② 五步按顺序说得出来，识别目标在最前；③ 判断题"度量就是收集数据"——不完整，是五步闭环，终点是改进；④ 第四步里的"反馈"是个动作，最易漏。

> **过渡：**框架有了，进第一个指标组——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）

> 一个模块 3 千行出 30 个缺陷，另一个 3 万行出 60 个。哪一个更让人担心？先把五个指标摆开。

>

> 一，代码的缺陷密度：每千行代码的缺陷数，写成 Bug#/KLOC。用来横向比模块、纵向比版本。风险在语言和风格：同样功能，Java 和 Python 的行数能差一倍，密度就没法直接比。要比，就在同语言、同团队、同口径下比。

>

> 二，缺陷清除率：各阶段的清除率，加总的清除率。含义是某个阶段之前清掉了多少比例。分阶段看很有用。需求阶段的清除率低，说明需求评审形同虚设，问题都漏到后面去了。

>

> 三，缺陷逃逸率。课件写得很清楚：没在研发阶段发现的缺陷就是逃逸出去的，也就是线上、生产环境发现的。公式是线上发现的缺陷数，除以线上加研发环境下发现的缺陷数之和。这个指标最能反映测试有效性，它不看你测了多少，只看你漏了多少。分母把两侧都算进去，比单看线上数量公平，大项目线上多，可能只是因为它大。

>

> 四，缺陷趋势是否良好、是否收敛。把 13.3.2 的趋势当判据，收敛说明过程稳。

>

> 五，缺陷分布是否符合正态，是否过于集中在 1-2 个模块。课件用的是问句，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。

>

> 五句收口：密度看浓不浓，清除率看拦不拦得住，逃逸率看漏多少，趋势看稳不稳，分布看偏不偏。前三个是数字，后两个是形状。

> **提问：**逃逸率的分母为什么要含研发侧发现的缺陷，只算线上数量不行吗？

预设回答：

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

- 答"要算比例才有可比性"——对。分母把发现的总量算进来，得到的是漏出去的比例。

- 答"分母大说明测试强，逃逸率就低"——逻辑对，但分母大也可能是提得多、都是小问题，所以还得配严重性看。

- 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 逃逸率公式必须默写：线上除以线上加研发，是本章最高频的计算考点之一；③ 清除率越高越好、逃逸率越低越好，方向相反，容易混；④ 判断题"缺陷密度可以直接跨语言比较"——错。

过渡： 有了数字，还得有个参照系才知道好坏，那就是基线。

【13.51】缺陷度量-基线（2 分钟）

缺陷密度 2 个/KLOC，算好还是坏？这个问题单独答不了。

>

因为数字本身没有好坏。你的历史基线是 1.5，2 就是退步；基线是 4，2 就是大幅改善。没有基线，数字只能看不能判。基线就是那个被认可的参照值。

>

它一般从三条路来。一，历史数据。把过去几个版本同口径的指标算出来，取个区间，最常用。二，行业基准。参考同类产品的公开数据或报告，要谨慎，口径差异大。三，项目目标。项目组和管理层事先约定，比如逃逸率控制在 5% 以内。这条最管用，因为它直接连着决策。

>

怎么用？关键在同口径。上个版本按提交数算缺陷，这个版本按去重后算，两个数字没法比。建议把口径文档化：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。

>

> 还有一条容易被忽略：基线要随成熟度调。刚起步的团队密度高是正常的，评审、自动化、规范立起来以后，基线要逐步收紧。基线一成不变，就等于没有目标。

>

> 还有个用法容易被忘：基线不只比一次。每次发布留一组数，三个月后回看，你会看见一条斜线。单点只回答这次好不好，斜线能回答过程到底在往前走，还是在原地打转。

>

> 参照值不准，后面量出来全歪。

> **提问：**两个版本缺陷密度 3.2 和 2.1，能直接说质量提升了 34% 吗？

> **预设回答：**

- 答"能，降了就是提升"——得小心。口径一样吗，代码量怎么统计，测试强度一样吗。

- 答"要先确认口径一致"——对，这是前提。

- 答"还要看测试深度，测得浅密度自然低"——关键补充。密度降可能是测得少，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 判断题"指标可以直接跨版本、跨项目比较"——要同口径；③ 口径文档化是能写进答案的实践建议；④ 基线随成熟度收紧是加分点。

> **过渡：**基线只能判定单指标。可连总共有多少缺陷都不知道时，还能估吗？课件给了个巧办法。

【13.52】13.4.2 经典的种子公式（3 分钟）

> 测试做到什么时候算够？凭直觉答不了，可这个问题必须答，发布决策就卡在上面。

>

> 种子公式给一条路。往程序里人为植入一批已知缺陷，叫种子。然后照常测。测完数两个数：捞出多少种子（s），捞出多少非种子缺陷（n）。种子总数（S）你自己知道。

>

> 逻辑很直。测试能捞出 s/S 比例的种子，它应当也能捞出同比例的真实缺陷。于是 N = S × n / s。N 是程序

> 里全部非种子缺陷的估计总数。

>

> 课件给了判据：n = N，说明所有 Bug 都找出来了，测试做得够。

>

> 还有信心指数 C，一个百分比，值越大对质量越有信心，最大值为 1。n > N 时 C = 1；n ≤ 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。第二个是课件给的形式，代值时 S 和 N 的量纲要一致，别把总数和测出数混着用。

>

> 优点是只用三个你能数出来的数，不用猜总数，也能估出量级。局限三条也得说清。种子要像真的，全是显眼的低级错误，捞出的比例虚高，N 就被低估；植入要动代码，得严格管理，测完清干净；它还假设测试对种子和真实缺陷的发现能力相同，现实里只是近似。

>

> 用已知的漏，推未知的漏。所有估算方法走的都是这条路。

> **提问：**植入 100 个种子，测出 60 个种子、90 个真实缺陷。真实缺陷总数估多少？

> **预设回答：**

- 答"按公式 100 × 90 ÷ 60 = 150"——对，直接代。

- 答"还剩 60 个没找到"——也对，150 减 90，这是估剩余量的用法，很实用。

- 答"就是 90，测出多少算多少"——常见误解，把已发现当全部，正是这个公式要纠正的。

- 收口：这公式最大的价值，是告诉你没找到的那部分大概多大。发布决策最需要的正是这个数。

> **易错点 / 考点：**① N = S・n/s 必须默写并会代值；② 四个符号不能混：s 测出的种子、n 测出的非种子、S 全部种子、N 全部非种子；③ 信心指数分段（n>N 取 1，n≤N 取 S/(S−N+1)）；④ 判断题"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。

【13.53】13.4.3 基于缺陷清除率的估算方法（3 分钟）

> 发布那一天，缺陷的意义就变了。前一天记的是我们的问题，后一天记的是用户替我们发现的问题。

>

> 符号先钉死。F 是描述规模用的功能点；D1 是开发过程中发现的所有缺陷数；D2 是发布后发现的缺陷数；D 是发现的总缺陷数，D = D1 + D2。分界线就是"发布"这个时间点，所有研发侧与线上侧的对比都靠它。

>

> 三个指标。

>

> 质量 = D2 / F，每个功能点摊到多少发布后缺陷，越小越好。好处是把规模消掉了：大项目发布后缺陷多很正常，除以功能点才可比。

>

> 缺陷注入率 = D / F，每功能点注入了多少缺陷。它衡量开发过程有多容易产生缺陷，反映过程能力，不是测试能力。注入率高，说明需求理解、设计、编码环节有问题，光靠测救不回来。

>

> 整体缺陷清除率 = D1 / D，被发现的缺陷里有多少在发布前清掉了。它直接衡量测试与评审的效果，越大越好。

>

> 三个关系记牢。注入率看产生多少，清除率看拦住多少，质量看漏到用户手里多少。注入率高、清除率也高，是错得多但拦得住，能发，得治本。注入率低、清除率也低，是错得不多但漏得多，问题在测试环节。两边都糟，就是边错边漏。

>

> 补一条。清除率是 D1/(D1+D2)，逃逸率是 D2/(D1+D2)。口径一致时，两者相加等于 1。算一个，另一个就出来了。

> **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%"——对，还验证了互补。

- 答"质量 = 20/200 = 0.1"——对，每功能点 0.1 个发布后缺陷。

- 答"注入率 = 200/200 = 1"——对，每功能点 1 个缺陷。

- 追问：这项目好不好？答"要跟基线比"——对，0.1 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，D1 与 D2 以"发布"为界；② 三个公式要能默写、能代值；③ 方向易混：质量越小越好、清除率越大越好、注入率反映过程能力；④ 清除率与逃逸率相加为 1，是高分点。

过渡：缺陷视角的度量到这儿。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）

给一个系统打总分，最省事，也最容易吵起来。

>

先看四类度量：软件可靠性度量、复杂度度量、缺陷度量、规模度量。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；缺陷度量就是前面那一组；规模度量看多大，代码行、功能点。规模本身不是质量，可它是所有比率指标的分母。

>

合成公式是 Mi = c1×f1 + c2×f2 + … + cn×fn。Mi 是一个软件质量因素，比如 SQRC 层各项待计算值；fn 是影响质量因素的度量值，比如 SQDC 层各项估计值；cn 是加权因子。

>

形式上就是加权求和。里头三件事得想清楚。

>

一，选哪些 f。选错，合成出来的是伪质量。只选缺陷密度和代码行数，这个分基本在奖励代码写得少。

>

> 二，权重 c 谁定。权重代表你认为什么更重要，是价值判断，不是技术判断。得由项目干系人共同确认，不能让测试一家定。

>

> 三，量纲要统一。f 有的是个数、有的是百分比、有的是秒。直接相加没有意义，得先归一化，比如都映射到 0–100。课件没展开这一层，你真用的时候一定遇上。

>

> SQRC、SQDC 是课件举的层级例子，一个是质量因素层，一个是度量值层。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。

>

> 加权求和容易，选对因素和权重才是本事。

> **提问：**为什么加权之前必须先归一化?

> **预设回答：**

- 答"因为单位不一样"——对。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起来。

- 答"归一化就是除以最大值"——那是一种常见做法（极值归一），也可以映射区间、做 Z-score，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 判断题"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，常作简答加分点。

> **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）

> 缺陷统计表交上来，第一行写着"需求与沟通占 40%"，后面就没话了。这 40% 说明什么，得看分类。

>

> 课件给了十二类。从上游往下游看，它是一条缺陷产生链。

>

> 需求与沟通两类。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。这两类都是没搞清要做什么，一个出在文档上，一个出在沟通上。它们最贵，一旦错了，后面全错。

>

> 第二层，故意偏离：IDS，故意与说明偏离。"故意"是说明知规格这么写，实现时按别的做，图省事、赶工期。性质最特殊，不只是技术问题。

>

> 第三层，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，设计与逻辑：EDL，设计逻辑有错。最硬的一类，往往是算法、流程、状态机错了，改起来影响面大。

>

> 第五层，测试与文档：IET，不完整或错误的测试；IID，不准确或不完整的文档。IET 值得多说一句，测试也会产生缺陷，一个错的用例给你一个错的结论，还会掩盖真问题。

>

> 第六层，实现转换：PLT，把设计翻译成程序设计语言时的错误。设计对，写代码写错。

>

> 兜底两类：HCI，不清晰或不一致的人机界面；MIS，杂项。

>

> 十二类的用途只有一个：统计各类占比，找主要矛盾。IES 和 MCC 占了 40%，要抓的是需求评审和用户沟通，不是单元测试；IMI 和 EDR 占大头，接口契约和数据规范要立起来。分类的价值不在分类本身，在它能指向改进动作。

> **提问：**如果 IES 和 MCC 合计占 40%，你建议团队加强什么？

预设回答：

- 答"加强单元测试"——方向错了。这两类在需求侧，测得再细也补不回来。

- 答"加强需求评审和用户沟通"——对。

- 答"加强编码规范"——也不对，编码规范针对的是 VPS、PLT 那几类。

- 收口：缺陷分类的第一用途是指路，告诉你资源往哪一环投。

易错点 / 考点： ① 十二个缩写的含义，至少记 IES、MCC、EDL、IET、PLT、IMI 六个；② 判断题"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（文档不完整或错误 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： 十二类在教材里排成一张表，看点不在名单，在怎么读。

【13.56】质量度量的统计方法 – 续（2 分钟）

一张表，横着读一遍，竖着读一遍，得到的东西完全不同。

>

做法五步。

>

一，建二维表。行是缺陷分类，IES、MCC、IDS 一直到 HCI、MIS；列是阶段，需求、设计、编码、测试、发布。格子里填这个阶段发现了多少这一类缺陷。

>

二，看行。某一类总数高，说明它是主要矛盾。EDL 最高，设计评审必须加强。

>

三，看列。某阶段总数高，说明这一阶段漏进来的多。某阶段低到零，要怀疑它根本没在查。需求阶段发现 0 个缺陷，通常不是需求完美，是评审没做，或者做了不算数。

>

四，看对角线。理想情况，缺陷在引入它那个阶段、或者紧邻的下一阶段被发现。EDR 大量在发布后才被

> 发现，说明数据相关的测试和评审缺了。对角线两侧也有话说：偏左上，发现得早；偏右下，一路漏到用户手里。

>

> 五，算清除率。每阶段"本阶段发现"除以"本阶段引入"，就是分阶段清除率。它正好接上 13.4.1 和 13.4.3。

>

> 这么一拆，缺陷从一个笼统的数量，变成哪一类、在哪个阶段、被谁发现的三维结构。有结构，改进才有靶子。

>

> 实操上有个难点：分类要互斥且穷尽。同一个缺陷既算 IES 又算 MCC，统计就重复计数。团队得有一份分类判定说明，模糊的先讨论、后固化。这跟 13.2 那些字段的口径是一回事，口径不清，数据全废。

> **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。更可能是评审没做，或者发现了没记录。

- 答"不信，要查评审记录"——对。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看。

- 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 判断题"某阶段缺陷数为零说明那个阶段质量高"——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：**有了分类表，课件给两个计算公式：一个算阶段，一个算总体。

【13.57】质量度量计算（3 分钟）

> 两个公式，看着就头大。拆开看，结构很直。

>

> 先看阶段错误度量。PIi 等于 Ws 乘 Si/Ei，加 Wm 乘 Mi/Ei，加 Wi 乘 Ti/Ei。课件给的权重是 Ws、Wm、Wi 等于 0.6、0.3、0.1。三个比值共用一个分母 Ei，那就按"分母是什么、分子分三类"来解。

>

> Ei 是这个阶段按计划要发现的错误总数，是个期望值。三个分子按严重程度分档：Si 严重，Mi 中等，Ti 轻微。权重递减就是这个道理：同样漏一个，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，得给严重性加权。一个阶段发现了 10 个轻微问题，漏了 1 个致命问题。按个数算，它发现了 90%。按这个公式算，会明显低于 90%。

>

> 再看总体质量度量：Ep = Σ (i × PIi) / Ps。课件的 i 取 1、2、3、4、5，分别代表需求分析、设计、编程、测试、发布；括号里又注明 i 等于 1、2、5、10、100。两组数不矛盾：前一组是阶段编号，后一组是权重。越靠后的阶段，同样错误率的代价系数越大，发布是 100，需求是 1。Ps 是分母，按教材口径取用。

>

> 它的教学价值不在算数，在于把一句工程常识变成了数字：同一个错误，晚发现一百倍代价。第 4 章讲尽早测试，13.4.3 讲清除率，13.3.1 讲评审前置。说的是一件事的三种说法。

>

> 用时两条：权重是课件给的示例参数，实际项目按自己的代价模型校准；量纲要一致，都是个数或都是比率。

提问：为什么发布阶段的权重给到 100，需求分析只有 1?

预设回答：

- 答"发布后修最贵"——对。发布后要定位、修复、回归、重新发、跟用户解释，还可能赔钱、丢口碑。

- 答"因为发布阶段缺陷少"——不是理由。权重反映代价，不反映数量。

- 答"数字太大不合理"——它是相对权重，只用来比阶段之间的代价差异，绝对值没有意义。

- 收口：权重是把"越晚越贵"这句常识变成可算的量。

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 判断题"缺陷数相同则质量相同"——错，要按严重性加权；④ "晚发现代价呈数量级增长"是本章的价值落

点。

> **过渡：**算法讲完了，可度量本身有个更靠前的问题：它会不会把团队带偏。

【13.58】度量及其指标（3 分钟）

每天必须提五个 bug，这条规定你听过没有？它听着像在抓质量，实际上在教人凑数。

>

> 课件列了四个指标。Designed cases，设计的用例数，看测试设计的工作量。它能被刷：一条拆成五条，数字上去了，覆盖一点没多。Executed cases，执行的用例数，看执行进度。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，每天发现的缺陷数，看发现效率。这个指标问题最大。一旦当 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。难啃的反而没人碰。它还天然对测试早期有利、对后期不利，后期本来就发现得少。Missed bugs，漏掉的缺陷数，看漏测。它最有价值也最难量，因为你不知道漏了什么。实践里常用逃逸到线上的缺陷来近似，就是 13.4.1 的逃逸率。

>

> 课件两句判断是这一页的骨头：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"这个词用得准。它会这么出现。只提容易提的。把复杂问题拆成小问题凑数。用例只写不跑。跑只跑简单的。把缺陷推到下个版本，美化本期数据。把阻塞改成通过。共同原因只有一个，指标被当成了评价人的工具，不是改进流程的工具。

>

> 铁律一条：度量流程，不度量个人。用指标找问题，不用指标找责任人。违反这条，你拿到的不是数据，是数据表演。

提问：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，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"难查的那些没人碰了"——对。难查的耗时间、产出少，按件计酬下最不划算。

- 收口：你考核什么，就得到什么，包括你不想要的那部分。

> **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 判断题"缺陷发现数越多测试就越好"——片面，还可能诱导凑数；④ 铁律"度量流程不度量个人"要能展开说理由。

> **【思政落点 6】**：度量手里握着权力，谁定指标，谁就在定义什么叫好。指标一旦用来追责，人会挑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定，吃亏的是产品和用户。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。

> **过渡：** 13.4 讲完怎么算，末了一块是怎么下结论：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）

> 前面一直在干活，收尾这一块是交代。很多人干得不错，栽在交代上。

>

> 三个小节。13.5.1 测试过程的评估，回头看我们是怎么测的：计划里说的事做到没有、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，对内对外交付。

>

> 关系一句话。过程评估看做得对不对。13.5.2 那个看做得够不够。测试报告，是把这两件事说清楚。

>

> 这块重要，因为它是很多团队的短板。13.1 到 13.4 都是做事，13.5 是交代。不交代有三个后果。经验沉不下来，这个项目踩过的麻烦，下个项目重踩；决策没有依据，老板问能不能发，你只能说感觉差不多；责任说不清，出了事讲不明当时测了什么、没测什么、风险披露没有。

>

> 测试报告的本质不是汇报工作，是风险披露。一份合格的报告，要让读的人看清五件事。测了什么，没测什么。发现了什么。还剩什么风险。结论和建议是什么。

>

> 再收一层：这三小节其实是三个问句。我们按计划测了吗？测得够不够？结论敢不敢签字？三个都答得出来，测试才有资格叫结束。

>

> 报告不是写给测试自己看的，是写给要做发布决策的人看的。

提问：测试做得很到位，报告写得很糟，会损失什么？

> **预设回答：**

- 答"白干了"——有点绝对，但确实大打折扣。

- 答"决策者看不懂，可能做错决定"——对，这是最直接的损失。

- 答"风险没披露，出事要担责"——也很现实。测了却没写未覆盖范围，等于没披露。

- 收口：测试的价值要靠报告兑现。没报告，成果换不成决策依据。

易错点 / 考点：① 三个小节的名称与分工；② 判断题"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，它其实就是拿一串问题对着计划做体检。

【13.60】13.5.1 测试过程的评估（2 分钟）

答辩现场，老师问"性能测了吗"，你说"测了"。再问"怎么测的、结果怎么样"，就卡住了。

>

> 课件对这件事的说法是，测试过程评审结合测试计划来评。相当于把计划的活动和实际执行的活动比一比，看执行得怎么样、效果如何。比的基准是计划，这一条是定义性考点。

>

> 六个问题，每个查一件事。

>

> 一，是否完成了测试计划所要求的各项测试内容？查范围。计划说要测的，测了没有。

>

> 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。这条最常被跳过。用例本身没评审，就可能漏关键场景，后面跑得再认真也白搭。

>

> 三，系统测试是否包含性能、兼容性、安全性等各项测试？查专项覆盖。这是第 7 章在这里的回响：功能测完，不等于测完。

>

> 四，如果执行了，是怎么执行的、结果如何？查执行质量。这一问最狠，不问你做没做，问你怎么做、结果怎么样。"我们做了性能测试"，跟"500 并发压 2 小时，TP99 是 1.8 秒"，是两回事。

>

> 五，需要执行的测试用例是否百分之百完成了？查执行率。要看清"需要执行的"这个限定。不是所有用例都必须跑，有些因需求变更作废；但计划要跑的必须完成，没完成的要有说明。

>

> 六，所有严重的 Bug 都修正了？查遗留风险。问的是严重的，不是所有的。微小缺陷可以带着发布，严重的必须清掉，或者有明确处置。

>

> 六问覆盖五个维度：范围、用例质量、专项覆盖、执行质量、遗留风险。以后写测试总结，按这六问组织，一个不漏。

提问：第四问"如果执行了，是怎么做的、结果如何"，为什么最容易把人问住？

预设回答：

- 答"很多人只做了没记录"——对。没记录就答不上怎么做的。

- 答"大家习惯回答'做了'"——形式化汇报的通病。做和做到位是两件事。

- 答"结果可能不好看"——也有，但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），定义性考点；③ 判断题"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：六问查的是过程。可有些风险不在计划里，它们藏在过程本身，13.5.2 接着讲。

第 13 章讲稿（13.61–13.68，v3）

锚点与课件一一对应，不增删页。取消板书。口播按四拍：起 → 推 → 收。

【13.61】发现测试过程的风险（2 分钟）

跑了一周，进度表上全是绿的。这时候你会不会觉得，这次挺顺？

>

顺有两种。一种是真测到了，一种是没测到的东西根本不会报错。

>

风险先按来源分成五个方面。

>

范围风险，要测的没测。需求覆盖不全，新功能没进测试范围，需求改了用例没跟着改。这一类最隐蔽，因为没测不会亮红灯。

>

环境与数据风险。环境不稳，数据不真，依赖全是 mock。表现很具体：测试环境好的，生产坏的；数据量小的时候没事，上量就崩。

>

执行风险。用例跑了，但跑得浅；回归范围不够；缺陷修完不验证。这一条对着前面那套操作规范，也就是测了不等于测到位。

>

> 进度风险。时间被压缩，人被抽走，阻塞一直不解除。

>

> 协作风险。缺陷被藏起来，状态乱标，争议挂着不解决。

>

> 五个方面摆完了，怎么发现？三个手段。

>

> 一是评审。拿过程记录去对计划，就是前面那六问；再加一步同行评审，让别人看你的用例和缺陷单。

>

> 二是度量。别盯单个数字，找异常的组合。执行率很高、缺陷发现率几乎是零，最可能的解释是在重复跑已经通过的用例。清除率漂亮、逃逸率也高，那要么口径有问题，要么数据不实。

>

> 三是抽查。随机抽几条缺陷单、几条用例，看质量。抽的时候别挑，挑过的样本没有意义。这个办法成本最低，命中率最高。

>

> 还有一条要落到纸面上：风险得写下来。写进报告的叫已披露风险，出事的时候是尽职的证据；没写的，出事的时候只能叫没想到。

提问：执行率 98%，可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”——有可能，但概率低。

- 答“测得浅，或者根本没在测新东西”——方向对。执行率高、发现率为零，最常见的原因是在重复跑已经通过的用例。

- 答“这周在回归，发现率本来就低”——说得通，但那要确认一件事：是不是只跑了回归，新功能压根没碰。

- 收口：单个指标的绝对值意义有限，组合起来异常才说明问题。

易错点 / 考点：① 五类过程风险要能点名：范围、环境与数据、执行、进度、协作。② 三个发现手段：评审、

度量、抽查。③ 判断题“没写进报告的风险不算风险”——反了，没写的那部分最危险。④ 案例分析常给“高执行率加零发现”，答指标组合异常。

过渡：过程查完了，还剩一个更硬的问题：到底测够了没有？

【13.62】13.5.2 测试充分性的评估（2 分钟）

测到什么时候能停，这个问题最后要落到一个数上。

>

课件第一句是：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，也是能不能收工的依据。所以它接得上第 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。

>

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，往往就是问题藏着的地方。

>

第三句最关键：测试覆盖率可以看作由业务需求、系统特性功能和非功能特性、代码等多个层次构成。它说明覆盖率不是一个数，是一个立体结构。你在需求层算一遍，再在代码行层算一遍，回答的是两件不同的事。

>

有一处容易误会。行覆盖 100%，只说明每一行都被执行过。分支怎么组合、边界取什么值、异常路径走没走、并发时序对不对，它一条都没保证。

>

> 所以两个方向的结论不对称。覆盖率高，推不出测得好；覆盖率低，几乎一定说明测得不够。它的用处就在这儿：证明不了你测得够，能证明你测得不够。把这句话当成一条规矩，讨论“能不能停”的时候就不会空对空。

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过。

- 答“会，覆盖只说明跑过，不说明测对”——对。

- 答“看是哪种覆盖率”——很专业。判定覆盖、条件组合覆盖、路径覆盖，强度差得远。

- 收口：覆盖率是必要条件，不是充分条件，跟第 5 章的逻辑覆盖准则一个道理。

易错点 / 考点：① 覆盖率两大用途：判断充分性、判断能否结束。② 多层次是高频点：业务需求、系统特性、代码。③ 判断题“覆盖率高就是测试充分”错，“覆盖率低就是不充分”对。④ 综合题常跟第 4 章退出准则、第 5 章逻辑覆盖连着考。

过渡：既然它分层次，那层次怎么划、往哪个方向看？下一页就是这张图。

【13.63】测试覆盖率分析（2 分钟）

覆盖率报告拉出来，你第一眼看哪儿？

>

> 先分三个层次，从外往里。

>

> 最外层是业务需求层。需求覆盖了多少，主流程走了几条，异常流程走了几条。这一层最贴用户，也最容易空着。很多团队从用例直接跳到代码，中间的需求覆盖没人算。

>

> 中间是系统特性层，分功能和非功能。功能特性问的是这个功能的各种场景测了没有；非功能特性是性能、

> 兼容、安全、可靠性，第 7 章那一整套。这一层回答的是，除了功能对不对，别的指标达不达标。

>

> 最里层是代码层。语句覆盖、判定覆盖、条件覆盖、路径覆盖。这一层最精确，也最容易骗人。你可能把一段本来用不上的代码测到满。

>

> 再看两种视角。

>

> 正向的视角问，我测了多少。汇报进度、写周报，用的都是它。

>

> 反向的视角问，还剩什么没测。未覆盖的是哪几块，为什么没覆盖，风险有多大。这个视角更值钱，因为风险都在没覆盖的地方。做过几年的人翻覆盖率报告，先找空白。

>

> 三个层次配上两种视角，就是一个 3×2 的自检表。三层我各算了吗？是不是只看了正向，没看反向？这张表的好处是能当场问出来，不必等报告写完才发现漏了一层。

>

> 顺带说一句，层次之间不互相替代。需求层覆盖率写得漂漂亮亮，也盖不住代码层一堆分支没走过。

>

> 最后把缺陷分布叠上去。缺陷密集的模块，覆盖率必须高；覆盖率低、缺陷又多的模块，就是最需要动手的地方。两张图叠起来，风险在哪儿就出来了。

提问：为什么说看覆盖率要先看没覆盖的部分？

预设回答：

- 答“没覆盖的地方可能有缺陷”——对，风险集中在没测到的地方。

- 答“覆盖过的至少能说明执行过”——对，虽然不保证测透。

- 答“覆盖率高显得工作做得好”——这是汇报视角，容易自欺，质量视角是反着看。

- 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，层次顺序是需求、特性、代码。② 判断题“代码覆盖率高说明测试充分”——错，需求层可能大片空白。③ 覆盖率乘缺陷分布等于风险地图，这条能当方法论写进答题。④ 两种视角里，反向的“还剩什么没测”是重点。

> **过渡：** 评估的东西最后都要落到一份文件上。国标对这份文件是有硬要求的。

【13.64】13.5.2 测试报告（2 分钟）

> 月末要交报告，很多人的第一步是打开上一版，改个日期。

>

> 国标 GB/T 17544－1998 的附录 E 对测试报告有具体要求。对测试记录、结果如实汇总分析，再出报告。这句话里最要紧的是“如实”两个字。报告的第一要求是真，不是好看。

>

> 七项内容，一项一项说它为什么必须在。

>

> 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。

>

> 用于测试的计算机系统。这条决定结论的适用范围。在 8 核 16G 上测出来的性能，不能拿去承诺 2 核 4G 的部署。

>

> 使用的文档，还有文档的标识。依据了哪些需求文档、设计文档、用例文档。这里是期望结果的来源，也是可追溯的凭据。

>

> 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。用户文档被单独列出来，说明文档测试也是测试内容。

>

> 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：哪些地方不满足要求，明明白白写出来。

>

> 针对建议的要求不符的清单，产品未作符合性测试的说明。这一条最容易被漏，也最见专业度。测了没通过要写，没测也要写。

>

> 测试结束日期。一个时间戳，界定结论的有效期：这份报告说的是截至这一天的情况。

>

> 七项收成三块就好记了。对象与环境是第一项、第二项；依据与结果是第三项、第四项；风险与时效是后三项。写报告按这三块搭骨架，漏不了。真到了答辩现场，你被问“报告里为什么没写这个”，答得出的就是这三块。

> **提问：**第六项要求写“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，但想到这一层就浅了。

- 答“让读的人知道风险边界”——对。不写，读的人会默认报告没提就是没问题。

- 答“写了显得报告不完整”——反过来，写了才叫完整。

- 收口：报告的可信度，来自它敢写自己的边界。

> **易错点 / 考点：**① 国标编号和附录记准：GB/T 17544－1998 附录 E。② 七项要能列，第六项“未作符合性测试的说明”是高频陷阱。③ 判断题“测试报告只写测试结果”——错，还要写不符项和未测项。④ 小节编号在课件标题上写成 13.5.2，按目录它属于 13.5.3，答题写 13.5.3。

> **【思政落点 7】**：报告要“如实”，最硬的考验在第六项——把没测的部分写出来。承认自己没覆盖到哪儿，当下要挨质疑，可它换来的是别人能做对决策。把没测说成没问题，是把风险悄悄转给用户和后面接手的人。实事求是落在工程上，就是这份肯把自己的边界写在纸上的自觉。

> **过渡：**这一章讲完了，把线收一下。

【13.65】本章小结（2 分钟）

小结最容易写得像公文，就在最后那一页。这一章的五句不一样，五句就是五个抓手。

>

第一句，怎么有效地执行测试，包括测试进度控制。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 曲线看。时间被压缩的时候调范围，别直接砍用例。

>

第二句，怎么正确、规范地描述、分离、分类、记录和跟踪缺陷。五个动词各管一件事。描述，写清步骤、期望、实际。分离，一条记录一个缺陷。分类，类型、来源、原因、严重性、优先级。记录，字段完整。跟踪，生命周期和状态流转。五个少一个，缺陷就修不快。

>

第三句，建立缺陷跟踪数据库。存储、搜索、分析，生成图表，看趋势，控进度，做根因分析，预防缺陷。动词链的终点是预防，不是统计。

>

第四句，怎么评估产品质量和度量质量。这里有两类办法，一类基于缺陷，一类是经典的种子公式。指标记密度、清除率、逃逸率、趋势、分布；方法记种子公式 N 等于 S 乘 n 除以 s、清除率估算、加权综合。不良的度量会让流程变形，这句别忘了。

>

第五句，测试过程评审、测试结果分析、测试报告。对着计划评过程，用覆盖率评充分性，按国标写报告。

>

五句连起来是一条链：执行、描述、跟踪、度量、报告。这条链记住，第 13 章就装进脑子了。

>

留个问题给自己：这五步，你实训那个小组哪一步最弱？最弱的那一步，就是你实习最需要补的课，也是答辩最容易被问住的地方。

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，算起来麻烦，容易变成填表。

- 答“报告”——也常见，复制上一版改个日期。

- 答“跟踪”——如果只在版本末期集中关单一波，跟踪也是形式。

- 收口：任何一步只剩形式，整条链就断了，下一环吃进去的输入是假的。

> **易错点 / 考点：**① 五句话对应本章五块，是论述题的现成框架。② 描述缺陷的五个动作是简答高分点。③ 跟踪的终点是预防，这点常被漏。④ 主线执行、描述、跟踪、度量、报告要能默写。

> **过渡：**理论讲完，两次实验把它落到手上。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）

> 装了就能用，这句话在工具课上最容易骗人。

>

> 实验 9 用 MantisBT，地址是 mantisbt.org，安装本身不复杂。它要你交的不是一个能登录的系统，是你通过配置看懂缺陷管理系统怎么把制度固定下来。四步。

>

> 第一步，装起来跑通。MantisBT 是 PHP 加数据库的经典组合。装的时候盯三件事。一是 Web 服务器，Apache 或者 Nginx；二是 PHP 和它的扩展；三是数据库，MySQL 或者 PostgreSQL。装完先做管理员初始化，改掉默认管理员密码，建项目，建用户。

>

> 第二步，配字段和工作流。这一步才是重点。把环境、版本、重现步骤、期望结果、实际结果设成必填。调状态机，关闭权限只给测试角色。配严重性和优先级的选项集，每个级别写清判定标准。配完这套，前面讲的规范就长在系统里了。

>

> 第三步，走一个完整生命周期。至少三条缺陷，走不同的路。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。一条被判成不是 Bug，或者延期。三条都跑完，生命周期那张图才算真看懂。

>

> 第四步，出报表。用内置统计出两张图：按状态和严重性的分布，按时间的趋势。然后在实验报告里写一句话结论，从这张图里你看到什么、会做什么动作。这句话是本实验最值钱的部分。

>

> 提交别只堆界面截图。字段配置和工作流配置的关键截图。三条缺陷的完整生命周期，含操作历史。两张报表。再加两百字以上的分析。要交的是你的配置决策和理由，光有界面说明不了你为什么这么配。

> **提问：**为什么要求至少有一条缺陷被判成“不是 Bug”或者延期，而不是三条都走顺利路径？

> **预设回答：**

- 答“凑数量”——不是。顺利路径只验得了主干，验不了分支。

- 答“真实项目里一定会有分歧”——对。这两个分支最需要流程支撑：判不是 Bug 要有依据，判延期要挂版本号。

- 答“三条都顺利不更好吗”——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，跟测代码要测异常路径是一回事。

> **易错点 / 考点：**① 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”是核心验收点。③ 三条缺陷要覆盖分支路径。④ 报告要写配置决策与理由，不是界面截图堆砌。

> **【思政落点 8】**：关闭权限只给测试角色，这条配置在讲一件很简单的事：谁都不能自己给自己签字。开发改完，得由别人验证才能关。岗位上有边界，权力才有约束。以后你在任何流程里配权限，都别图方便把关闭权放给改动方——守住这条，等于守住了团队里最基本的公信。

> **过渡：**实验 9 练的是缺陷管理一条线，实验 10 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）

> 一个学期学的东西，最后要在一套工具里串一遍才算数。

>

> 实验 10 用 MeterSphere，五个任务。

>

> 第一，在线或者离线安装，然后创建项目、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。别用一个管理员角色从头走到尾，那样你体会不到权限设计在管什么。

>

> 第二，完成一个完整的接口测试：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。三段是递进的。先调试通，发得出请求、看得懂响应；再设计用例，正常、异常、边界、参数组合；最后做场景自动化，把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：场景配置、压力配置、高级配置、结果分析。这一段接第 7 章。场景是谁在用什么方式访问；压力是人数、时长、节奏；高级配置是集合点、思考时间、断言这些；结果分析看响应时间分布、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。这一段就是本章主线，一环不少。

>

> 课件还说明，课程资源提供了详细的指导视频，本实验可以作为大作业或者实践考试。这句话的分量在评分上：考的是完整走通，不是某一步做得多漂亮。

>

> 给个建议。每一步都留证据，截图、导出文件、报告。最后按任务一到任务四，加一段总结。总结至少答三个问题：这条链里哪一步最花时间，哪一步最容易出错，重做一次你会怎么改。

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，那是本章的中心。

- 答“生成报告”——也对，13.5 的落点在这儿。

- 答“用例评审”——补得好，对应 13.5.1 的第二问。

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

> **易错点 / 考点：** ① 五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 性能测试四段跟第 7 章的衔接。④ 本实验可作大作业或实践考试，看完整性，不看单点漂亮。

> **过渡：** 第 13 章到这里全部讲完。

【13.68】感 谢 聆 听（1 分钟）

> 第 13 章到这儿收尾。

>

> 这一章讲到最后一页，回头看，它讲的事比前面任何一章都琐碎。琐碎的东西最容易讲完就忘。

>

> 把走过的这条线拉一遍。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、严重性和优先级。缺陷怎么追到底：生命周期、状态流转、争议处理、趋势和分布。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。

>

> 这条线的顺序不是排出来的。前一步不做完，后一步就没有输入。用例没执行，缺陷就没处来；缺陷没描述清楚，跟踪就追不下去；跟踪出来的数据不干净，度量算的全是假的；度量不实，报告只能写漂亮话。

>

> 所以这五块看着是五件事，实际是一条流水线。你在哪一段偷懒，代价都会堆到后面那一段。

>

> 这一章几乎不讲测试技术，讲的全是测试工作。可就是这些，决定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 章学得最好的那个，缺陷写得没人看得懂，价值就兑现不出来。

>

> 课程设计里请留一份东西下来。从你项目里挑三条真缺陷，按 13.2 的字段填全，走完一次关闭。再截一张

> 覆盖率报告，把没覆盖的那几块圈出来，写一句为什么没覆盖。这份材料比任何总结都好用。

>

> 三句话带走。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”。度量流程，不度量个人；数据一旦用来伤人，就再没人跟你说真话。测试报告本质上是风险披露，敢写“我没测什么”，才配说“我测了什么”。

>

> 谢谢大家。

> **提问：**这一章里，你觉得最难做到的是哪一条？

> **预设回答：**

- 答“如实写缺陷，不夸大不隐瞒”——难在长期坚持，压力一直在。

- 答“不用度量去追责”——也难，它逆着管理直觉。

- 答“敢在报告里写没测什么”——最难，要承担当下的质疑。

- 收口：技术可以学，这几条靠职业习惯，而习惯从第一份缺陷报告就开始养。

> **易错点 / 考点：**① 全章主线要能默写。② 三句总结性的话可以当论述题素材。③ 跟第 14 章的衔接是从单点能力到体系。④ 复习按 13.1 到 13.5 五块整理笔记最省时间。

> **过渡：**第 14 章把测试从单点能力推到体系，怎么组织、怎么持续。

四、思政落点清单（共 8 处）

序号	所在页	落点主题
1	13.16	同一个缺陷，开发觉得影响不大，用户觉得天塌了，测试夹在中间最容易被情绪带走。把判
2	13.20	写清版本、附上日志、留下录屏，这些动作没人检查，偷懒也看不出来。可一条缺陷能不能
3	13.23	这份报告最难得的地方，是它写了"换成别的字体格式显示正常""Solaris 和 Ma
4	13.37	这一块讲的是复盘与担当。一个团队敢不敢做根因分析，看它把缺陷当什么。当成某人的罪

隐藏缺陷往前追一步，就不只是流程问题。缺陷不会因为你没报就消失，它只在用户手里出

度量手里握着权力，谁定指标，谁就在定义什么叫好。指标一旦用来追责，人会挑最省力的

报告要“如实”，最硬的考验在第六项——把没测的部分写出来。承认自己没覆盖到哪儿，

关闭权限只给测试角色，这条配置在讲一件很简单的事：谁都不能自己给自己签字。开发改