《软件测试课程设计》讲稿（第 14 章）

第 16 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 14 章　软件测试展望

第 16 周 · 2 学时（90 分钟）

《软件测试课程设计》第 14 章 讲稿（第 16 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 14 章 软件测试展望 —— 技术会变，对质量的判断力不会变
>
> 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（全书收尾）
>
> 配套课件：逐页还原版 **69 页**（`lecture14/`，云端 http://111.231.0.226:2345/lecture14/index.html）
>
> 教案对应：第 16 次课「软件测试展望」
>
> 本讲重点：测试技术的演进趋势；智能化测试与 AI 辅助的能与不能；质量工程与 DevOps／平台化；云原生对测试的影响；测试人员的能力模型与职业发展；**全书 14 章收口**
>
> 本讲难点：面对"AI 会不会取代测试"这类问题给出有判断力的回答；把 14 章内容串成一张可复习的总图
>
> **本讲稿为可照读完整版**：每页含【口播】【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 能说出测试技术演进的三个方向；② 能列举 AI 在测试中的四类应用及其局限；③ 能说明 DevOps 与质量工程对测试角色的改变；④ 能画出全书 14 章的知识总图；⑤ 能说出测试岗位的三条常见发展路径
能力	① 能就"AI 会取代测试吗"写出 500 字有论据的判断；② 能给自己列一份"毕业前要补的三项能力"清单；③ 能把这门课的方法用在自己的毕业设计里
素质（思政）	① AI 是工具，判断在人；② 人人对质量负责的工程文化；③ 终身学习的能力观；④ 踏实积累不追热点；⑤ 把技术用于正道

本讲要建立的一个总印象：这一章不是"听个热闹"，而是**把整门课收拢成你自己的判断力**——工具会换、框架会换，但"怎么定义质量、怎么设计验证、怎么用证据说话"这套东西不会过期。

二、时间分配总表

> **容量提示**：本章课件 **69 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，**最后务必留足 15 分钟做全书收口**。

时段	页码	内容	分钟	备注
第 1 节	14.1–14.20	技术演进趋势、云原生与微服务对测试的影响	24	趋势三条线必讲
第 1 节	14.21–14.40	智能化测试与 AI 辅助（能做什么、不能做什么）	26	**AI 的边界必讲**
——	——	**课间休息 10 分钟**	——	——
第 2	14.41–	质量工程、DevOps 与平台化、测试组织演进	20	——

节	14.56			
第 2 节	14.57–14.69	能力模型与职业发展、**全书 14 章总图**、结课寄语	20	**收口必讲**

> **压缩优先级**：先压缩厂商与产品示例页 → 再压缩单一技术点的展开 → 最后才压缩 AI 边界、质量工程、能力模型、全书总图。

三、逐页讲稿

> 每页第一段是可直接朗读的口播；其后是提问与预设回答、易错点/考点、过渡语。
>
> 页码锚点与课件页一一对应（14.1–14.69）：http://111.231.0.226:2345/lecture14/index.html
>
> **收口要求**：本章末尾要把 14 章串成"观念（1–2）→ 方法（3）→ 流程（4）→ 技术（5–9）→ 项目实践（10–13）→ 展望（14）"一条主线，并给出毕业设计与求职的具体建议。

【14.1】软件测试方法和技术（1 分钟）

> 四年之后你入职，第一天用的工具，今天课上可能一个都没讲过。

>

> 那这十四章你带走了什么？这个问题先放在这儿，别急着答。

>

> 前面十三章一直在回答怎么做。概念怎么定，方法怎么选，流程怎么排。单元和集成怎么测，功能和专项怎么测。缺陷怎么管，覆盖率怎么算，项目怎么落地。到这一章，问法换了。换成两句：这条路往下往哪走，你手里还剩下什么。

>

> 工具的生命周期比你想的短。一个平台今年用得顺手，明年可能被并进另一套流水线。脚本要重写，界面全换，连报告长什么样都不一样。你要是只学了点按钮，那一轮更新就白学。面试也考这一层。面试官给一个你没做过的场景，考的不是你背过几个工具名，是你的判断走不走得通。

>

> 这一章有三个任务。一是看前沿。大数据怎么测。AI 系统怎么测，AI 又怎么帮我们测。测试工具往哪个形态变。什么叫彻底的持续测试。二是练判断。碰见"AI 会不会取代测试"这种问题，你得给得出一段有论据的话，不能跟着热搜跑。三是做收口。把十四章收成一张图，答辩和面试能一口气讲完。

>

> 为什么这三件事排在最后？前十三章给你的是零件。零件散在桌上，拿不出手。得有个东西把它们串起来，你才说得清自己学过什么、能干什么。

>

> 总纲一句：工具会换，框架会换，平台会换。怎么定义质量、怎么设计验证、怎么用证据说话，这套判断力不过期。

提问：如果四年后你入职时，今天课上讲的某个工具已经没人用了，这门课对你还有什么用？

预设回答：

- 答"没用了，得重新学"——那十三章干脆压缩成十三章工具名算了，显然不对。工具是载体，方法才是内容。
- 答"有用，方法还在"——方向对。分析思路、用例设计方法、缺陷判断标准，换个工具照样用。
- 答"有用，因为面试还考"——实在，但只说对一半：面试考的是判断力，不是背工具名。
- 收口：这一章干的就是这件事，把会过期的和不会过期的分开，你把后者带走。

易错点 / 考点：① 这是全书收尾章，综合论述题高频，比如"结合本课程谈谈你对软件测试发展趋势的理解"，答题先给课件六节结构，再落到自己的判断；② 别把它当成名词罗列，每个新技术都要说清它解决了老办法解决不了的什么问题；③ 记忆抓手是一条链：数据变大 → 被测对象变聪明 → 测试手段变聪明 → 工具形态变轻 → 交付节奏变快 → 趋势六条。

过渡：先把六节结构摆出来，你心里好有个时间预算。

【14.2】14.1 大数据的测试（1 分钟）

先别急着背那六行标题。六节背后有三个开关，抓住它们，标题自己会站住。

>

> 14.1 大数据的测试。数据量从 GB 涨到 TB、PB，测的东西从页面和接口，变成数据管道。

>

> 14.2 AI 系统的测试。被测对象自己变聪明了，输出不再是确定的，怎么判对错成了新问题。

>

> 14.3 AI 助力软件测试。反过来，把 AI 用到测试上。图像识别看界面，OCR 认元素，NLP 把需求转成用例，算法去分析失败日志。

>

> 14.4 软件测试工具的未来。工具从写脚本，走向低代码、零代码、平台化。

>

> 14.5 彻底实现持续测试。测试不再是一个阶段，而是流水线里的常态。

>

> 14.6 软件测试发展趋势。收成六条趋势，也是这门课对行业走向的总结。

>

> 三个开关是什么？前两节讲被测对象变了。中间两节讲测试手段变了。后两节讲交付方式和整体趋势变了。对象、手段、交付，这三样一动，测试人员的活儿就跟着动。顺序也不能倒着走。对象先变，手段才跟着变，交付方式最后才变。你把 14.5 挪到 14.2 前面，就成了一种怪谈：对象还没弄清，先谈流水线。

>

> 为什么要这么分？因为六节按顺序读一遍，六个名词转眼就忘。按三组切开，你拿到的才是三条变化线。考场上问你哪一节讲什么，你从线上找，比从名词表里翻快得多。名词会记混，线不会。

>

> 还有一处必须掰开。14.2 的主语是 AI 系统，它是被测的。14.3 的主语是测试，AI 只是工具。看着只差一个字，方向整个反了。方向一反，后面所有论述都跟着错。

提问：这六节里，哪几节动的是被测对象，哪几节动的是测试手段？

预设回答：

- 答"14.1 和 14.2 是对象，14.3 和 14.4 是手段"——准确。剩下两节是交付方式和整体趋势。

- 答"全都是趋势"——太笼统。答题时你说不出层次，论述题的结构分就没了。

- 答"14.3 讲的是对象变了"——最容易混的一处。14.3 里 AI 是工具，不是被测的东西。

- 收口：看主语。主语是谁，就知道这一节动的是哪一头。

> **易错点 / 考点：** ① 六节名称和顺序要能默写；② 14.2 与 14.3 的方向别弄反，14.2 是测 AI，14.3 是 AI 帮测试；③ 简答常考"大数据测试与传统测试的根本区别"：传统比输入输出对不对，大数据比数据在整条管道里有没有走错、算错、丢错。

> **过渡：** 从 14.1 开始。先问一个最朴素的问题：数据大到一定程度，到底在测什么。

【14.3】14.1 大数据测试（1 分钟）

> 为什么大数据排在最前面？不是因为它最难。

>

> 四个小节，先把地图画出来。14.1.1 特性与挑战，弄清大数据测试和普通测试差在哪，老办法为什么不够用。14.1.2 测试方法，落在 ETL 主线上，抽取、转换、加载，每一段验什么。14.1.3 测试实践，真实项目里干什么活、走什么流程、自动化怎么做。14.1.4 测试工具，工具这一层拿什么支撑。

>

> 这一节概念不少，主线只有一条：数据从源头到报表，中间不能走丢、不能算错、不能变形。所有细节都挂在这句话上。读的时候把每个概念拿去对一次：它对得上走丢、算错、变形里的哪一条。对不上，多半是你没弄清它在管哪一段。

>

> 四小节里最难的是第一小节。特性与挑战讲不清，后面的方法就变成背流程。你不懂数据量大带来的麻烦，抄一遍 ETL 也解释不了为什么要分段。

>

> 排在首位的原因在下一节。AI 的原料是数据。喂给模型的数据本身是脏的，模型再好也白搭。数据质量是模型质量的前提，这个道理得先讲透，后面讲 AI 才讲得下去。

>

> 顺手拆一个误区。很多人一听大数据测试，想到的是拿大数据量压性能。数据量大只是表象。它要验的是加工过程对不对。这一段要合的合上没有，那一段要滤的滤干净没有，最后落的位置对不对。

>

> 位置摆清楚：界面是表皮，数据才是本体。表皮出问题，用户当场就喊；本体出问题，只有对着数的人才看得见。

提问：一家银行的报表系统，页面打开正常、按钮都能点，你就能判定它没问题吗？

> **预设回答：**

- 答"能，功能都正常"——那报表上的数字比上个月少了三千万，你看得出来吗？页面不会告诉你。

- 答"不能，得看数据对不对"——对。界面只是表皮，数据才是本体。

- 答"得跟业务对数"——很专业，行业里叫多方报表对比验证，后面流程那一页专门讲。

- 收口：看得见的地方往往没问题，问题都藏在看不见的数据流里。

易错点 / 考点：① 四小节的名称和顺序；② 判断题"大数据测试就是拿大数据量做性能测试"——错，数据量大只是表象，它测的是数据加工过程的正确性；③ 这一节的服务对象要记准，它为 14.2 的 AI 系统测试提供数据质量这一半前提。

> **过渡：**先把测什么说清楚，大数据测试到底验哪几件事。

【14.4】大数据的测试（3 分钟）

一个任务算得一分不差，报表出来照样是错的。这种事在数据项目里天天发生。

>

> 课件给的落点不少，我按三层排好：单个节点、节点之间、整体结果。

>

> 第一层，单个节点上的业务逻辑验证。大数据系统是一串处理节点连起来的，每个节点干一件事：过滤、清洗、汇总、关联。节点里的逻辑对不对，必须先单独验。这一步相当于第 5 章讲的单元测试，只是被测的单

> 元不是函数，是一个计算任务。

>

> 为什么要单独验？因为一个任务里几步连着走，出了错你分不清是哪一步。拆开单验，错就落在某一个节点上，范围立刻缩小。

>

> 第二层，节点之间的一致性。课件讲了两条：聚合或隔离规则验证，还有生成的键值对验证。聚合是多条并一条，隔离是一条不能串到另一条，键值对是分组键和度量值配得对不对。三件事有个共性，都发生在数据搬家的时候。搬家最容易出事：要合的没合，要分的没分，键配错了值。

>

> 第三层，整体结果落在哪儿。课件那句话特别关键：验证正确的数据被提取并加载到正确的 HDFS 位置。两个正确，数据正确，位置也正确。数据对但落错目录，下游照样全错，这种错最难查，因为文件看起来好好的。这一层还要检查转换规则，检查数据完整性和一致性。

>

> 四句收住：要来的都来了，来的都对，放的地方对，彼此不打架。完整、正确、位置、一致。

提问：一个任务算得完全正确，但输出写到了错误的 HDFS 目录，这算不算缺陷？下游会怎样？

> **预设回答：**

- 答"不算，数据是对的"——典型错答。下游任务读的是路径，不是数据的正确性。路径错了，等于数据没来。

- 答"算，下游会读到空数据或者旧数据"——准确。更麻烦的是它不报错，下游读到空表照样跑完，报表出来是零，业务方第二天才发现。

- 答"看有没有监控告警"——补得好。这正是数据监控存在的理由，这类缺陷靠人眼发现，成本极高。

- 收口：验收大数据系统，不能只看数算得对不对，还要看数落得对不对、错了会不会被发现。

易错点 / 考点：① 三层落点要能复述，特别是数据正确加位置正确这个双正确；② 聚合与隔离的区别常考：聚合是要合的合起来，隔离是不能串的绝不串，多租户、多套数据场景必考；③ 键值对验证属于第二层，别答成第一层；④ 口诀：单点验逻辑、跨点验规则、整体验落位。

> **过渡：**三层里最关键的是中间那层，数据在节点之间怎么搬。搬家的标准流程有个名字，叫 ETL。

【14.5】ETL 过程示例（2 分钟）

数据从业务库出来，到报表上，中间要过三道门。这道流水线有名字，ETL。

>

> 三个字母：抽取 Extraction，转换 Transformation，加载 Loading。抽取是把业务系统里的数据拿出来，转换是按规则加工，加载是写进数据仓库。

>

> 课件把这条线切成三个中间文件。三个名字容易混，一个一个看。

>

> 第一个，EXF，Extracted Format，抽取格式。它由数据源 Extract 产生。文件结构和源库相似，但经过过滤，部分字段被忽略。为什么这一步就丢字段？用不上的字段搬过来只占地方，还容易在下游被误用。

>

> 第二个，CIF，Common Interface Format，公共接口格式。它是 ETL 经过 C/S/S 过程产生的中间数据文件。公共接口四个字干什么用？解耦。上游十几个异构数据源，下游只认一种格式，后面所有处理都省事。不做这一层，十个源对五个目标，就是五十套适配。

>

> 第三个，PLF，Pre-Load Format，预加载格式。它经过数据转换，直接加载进数据仓库，结构跟仓库里的表定义一致。到这一步，格式和表严丝合缝，加载不必再改。三个文件摆在一起，你还能看出流水线在哪儿换了格式：源库一种，中间一种，仓库一种。

>

> 为什么不一口气跑完，非要切成三段？三个理由。可回溯，出错能定位到是哪一段。可分工，抽取、转换、加载可以三个人做，也可以三套系统做。可复跑，某一段挂了，从中间文件重跑，不必回源头再抽一遍。

>

> 一句话：把一条长流程，切成几段有契约的短流程。切的时候每段都要能自己说清输入什么、输出什么，否

> 则切了也白切。

> **提问：**加载完成后发现报表数字不对，你先查哪一段文件？为什么？

> **预设回答：**

- 答"查 PLF，它最接近仓库"——起点对，方向得反着想：PLF 对，问题在前两段；PLF 错，问题在转换。

- 答"从 EXF 往后逐段对数"——标准做法，行话叫分段对数。相邻两段的量和对得上，断点就找到了。

- 答"直接查源库"——源库数据本来就对，你要确认的是抽出来了没有。

- 收口：三个中间文件的价值，就是让你把一次排查切成三次比对。

> **易错点 / 考点：**① 三个缩写的全称和中文名都要会：EXF 抽取格式、CIF 公共接口格式、PLF 预加载格式；② 三者最容易混的是结构像谁：EXF 像源、PLF 像数仓表、CIF 谁都不像，它是统一的中间格式；③ 简答常考"为什么要设 CIF 这一层"：解耦异构数据源，免掉 N 个源对 M 个目标的两两适配；④ ETL 三个英文全称是送分题，Transformation 别拼错。

> **过渡：**ETL 的第一段是抽取。抽取看着最不需要动脑，脏数据恰恰都从这儿进来。

【14.6】数据抽取与清洗（2 分钟）

> 抽取这一步，看着最不需要动脑。脏数据全从这儿进来。

>

> 抽取两种方式。全量，每次把源表整份搬过来，简单，代价高。增量，只搬变化的那部分，快，难在怎么准确知道哪些变了。靠时间戳、靠日志、靠触发器，各有各的坑。

>

> 清洗要干四件事：清洗异常数据和不符合规范的数据；空值处理；规范化数据格式；按数据规则过滤。

>

> 空值最见功夫。年龄字段空了，补零、补均值、还是整条丢掉？三种做法业务含义完全不同，报表上的结论也不同。这不是技术选择，是业务决策，得跟数据 owner 定。测试的活是验证这条规则有没有被照做。

>

> 抽取有两个硬要求。准确性：能把业务系统里的变化数据，按一定频率准确捕获到。性能：不能给业务系统太大压力，影响现有业务。

>

> 这两条是一对矛盾。要准就多查几次源库，多查就压业务。所以增量抽取的调度频率，真实项目里是业务方和 DBA 一起定的，不是测试拍脑袋。

>

> 测试怎么验？五个手段。抽样验证，抽一批记录人工核。对比处理前后的数据量，进去多少、出来多少、差在哪。验证数据合法性，格式、范围、类型。字段逻辑验证，字段之间的关系成不成立，比如结束时间不能早于开始时间。最后一条，枚举字段看分布：性别、年龄、地域、学历，还有活跃小时段。

>

> 分布这一条最实用。一个地域字段，昨天九成是广东，今天剩一成，代码一行没改，你也能立刻看出不对。

> **提问：**一张订单表昨天一百万条，今天增量抽取只进来三条，你先查什么？

> **预设回答：**

- 答"查源库昨天是不是真没订单"——第一步正是这个，先分清是源没数据，还是抽没抽到。

- 答"查增量字段有没有被业务改过"——很专业。增量抽取最常见的失效原因，就是时间戳或自增 ID 被上游改了语义。

- 答"三条也是数据，说明抽取正常"——错答。量的突变本身就是缺陷信号，抽成功不等于抽对。

- 收口：所以课件把对比数据量单列成一个手段。数量级的变化，永远比内容的变化更早暴露问题。

> **易错点 / 考点：**① 两个硬要求是简答高频点，性能那条最容易漏；② 空值处理属于业务决策，测试的职责是验证规则被执行，不是自己拍一个默认值；③ 五个验证手段要能列举，枚举分布最容易被忽略，也常拿来出案例；④ 判断题陷阱"全量抽取一定比增量可靠"：可靠但代价高，选哪种看数据量和业务窗口，不是绝对的。

> **过渡：**数据抽进来了，接下来按规则改造它。这一段是缺陷最密集的地方。

【14.7】数据转换及其测试（2 分钟）

> 报表上的金额整体小了一百倍，代码没报一个错。这种事故多半出在转换这一段。

>

> 课件列了八类动作，每类我只说它会怎么错。

>

> 数据替换，把 A 换成 B，错在映射表不全。字段映射，源字段对到目标字段，错在张冠李戴。把下单金额对到实付金额，数字看着都合理，业务全错。数据过滤，按条件筛，错在多筛或者少筛。数据加解密，敏感字段脱敏，错在漏字段，身份证没脱敏就落仓。类型统一转换，字符串转数值、日期统一格式，错在精度丢失和时区错位。拆分数据，一个字段拆成几个，错在分隔符异常时拆歪。按主题划分和合并，错在归类规则。数据排序，错在排序键选错，下游取最新一条就取错了。

>

> 八类里最要盯住的是字段映射和类型转换。它们的错不报异常，只是让数字悄悄变形。报错会把你叫醒，变形不会。不报错的错，最难查，也最难解释给业务听。

>

> 怎么测？两条路，正好是第 3 章那对老搭档。

>

> 白盒转换测试，看三样：转换逻辑和映射设计对不对；代码实现符不符合设计；转换后的数据跟文档里的数据对不对得上。

>

> 黑盒转换测试，不看代码。按文档列出的各种转换场景造数据，输进去看输出。

>

> 一句话分开：白盒问你是不是照设计做的，黑盒问你做出来的对不对。两条都要走。因为设计本身也可能是错的，只做白盒，你会把设计里的错一起验成合格。

> **提问：**把金额从字符串转成数值型，哪种错误最危险：报错、四舍五入、还是单位不一致？

预设回答：

- 答"报错最危险"——反过来，报错最容易发现，流水线当场就红。

- 答"四舍五入"——危险，但偏差有界，量级还在。

- 答"单位不一致，一个按元一个按分"——最危险的一种。数量级差一百倍，数值类型完全合法，任何校验都不报错，报表直接错两个数量级。

- 收口：转换环节的重点不是能不能跑通，是跑通以后数还对不对。

易错点 / 考点： ① 八类动作能列举，每类能举一个典型错误；② 白盒与黑盒转换测试的判据差别要说清，一个对设计，一个对文档场景；③ 案例题高频"报表金额整体偏小一百倍，定位到哪一环"：先查类型转换和单位，再查字段映射；④ 判断题陷阱"转换测试只要比对最终结果"——错，中间结果不保留，出了问题没法定位。

过渡： 转换做完，还要验一件事，源和目标之间的对应关系是不是真的通。

【14.8】数据映射验证（2 分钟）

前面几页问的是每一步算得对不对。现在换个问法：整条链路接得通不通。

>

四类验证，按接头、公式、零件、通水的顺序看。

>

连接验证：必需端口都连上，每条连接都是通的、能用的。源库连不上，Kafka 端口不通，HDFS 没权限，后面全是空谈。这一层最便宜，一条连接测试几秒钟，能挡掉后面几小时的瞎查。

>

表达式验证：所有表达式都成立。转换规则是用表达式写的，求和、拼接、条件判断、日期函数。表达式错有两种。语法错会报错，好办。语义错能跑，算出来是错的。所以这里验的是，这个表达式在给定字段上成不成立，结果对不对。

>

对象验证：独立对象定义和映射里的实例匹配。翻成大白话：文档里定义客户宽表有三十个字段。实际映射

> 里是不是真有这三十个，类型对不对，必填的有没有。这是设计和实现的核对，两边字段数目都不同，后面算得再准也没用。

>

> 数据流验证：数据能从源流到目标，不会卡在阻止转换的地方。像通水试压。放一小股数据进去，看它能不能走到终点。走不到，看卡在哪个转换。四类里它最像验收，一股水从头放到尾。

>

> 四类的顺序有讲究，从静态到动态，从局部到整体。连接和对象是静态对表，表达式算半静态，数据流是端到端实跑。前面不通，别急着往后跑。为什么？数据流验证最贵，跑一次要等。前一关没过就跑它，等于花一小时等一个注定失败的作业。

> **提问：**一条数据流在中间某个转换处挂起了，不报错也不前进，你从哪儿入手？

> **预设回答：**

- 答"看日志最后一条停在哪"——标准第一步，挂起点通常就是最后一条日志的位置。

- 答"查这个转换的前置条件，比如依赖的分区还没到"——很专业。大数据任务最常见的挂起，就是等上游分区。

- 答"重启任务"——能缓解，可挂起往往有确定原因。重启只是把原因藏起来，下次同一时间照样挂。

- 收口：这就是数据流验证必须做的理由，它专治那种不报错、只是不动的问题。

> **易错点 / 考点：**① 四类验证的名称和各自验什么，对象验证最容易被理解错，它验的是定义和映射的一致性，不是数据对象本身；② 顺序原则，静态在前、动态在后；③ 案例常考"任务长时间无进展、无报错"，指向数据流验证和上游依赖；④ 口诀：通不通、算不算、齐不齐、走不走。

> **过渡：**ETL 三段各自的验法都清楚了。现在把它们放回项目里，看真实的大数据测试有哪几块活。

【14.9】大数据测试主要工作（3 分钟）

> 数据准备、ETL、OLAP、报告，四件事排下来，正好是数据的一生。

>

> 四件主要工作：测试数据准备，ETL 测试，OLAP 和 Cube 测试，报告测试。准备，加工，多维分析，呈现。

> 四件事的产物也不一样。准备阶段交的是数据集，ETL 交的是加工后的明细。Cube 交的是聚合结果，报告交的是人看得懂的数。

>

> 术语顺手带走。ETL 是 Extraction-Transformation-Loading。它完成 DB 到 DW 的数据转存。DW 是 Data Warehouse，数据仓库。OLAP 是在线分析系统。Cube 是面向多维数据库的立方体。Cube 不是报表，它把各种切面提前算好，切片、切块、钻取都快。

>

> 四个阶段：数据采集清洗测试，数据转换测试，专项测试，加上前面的准备环节。这里的专项测试指性能、数据质量、可恢复性这类非功能项。跟第 7 章说的是同一件事，只是对象换成了数据任务。

>

> 要点清单，课件三列，我合成五组。

>

> 一组，源头与规则：验证所要求的数据和数据源、数据分析、数据转换规则、验收标准。

>

> 二组，元数据与模型：看数据字典，也就是元数据；验证源与目标映射；ETL 和 DW 的验证体系结构；数据模型的验证，包括建模维度和规范化治理；索引、分区。

>

> 三组，质量与性能：数据质量和性能指标。

>

> 四组，运维韧性：档案和清洗策略；错误日志、异常处理、可恢复性；并行执行和优先级；ETL 的 pull 逻辑；全量和增量策略。

>

> 五组，职责：谁验规则、谁验模型、谁验调度，测试方案里写清楚，别到出问题才找人。

>

> 清单看着碎。它就是大数据测试的日常。难点不在技术，在面广。面广这件事只能靠清单管，靠脑子记不住。

> **提问：**大数据项目里，专项测试这一块为什么可恢复性和错误日志的权重比普通项目高？

> **预设回答：**

- 答"因为数据量大，跑一次很久"——抓到要害。跑一次几小时，失败重跑的代价极高，能不能从断点恢复直接决定效率。

- 答"因为一错就是全量错"——对。批处理一批几千万条，错一条规则就是全批重来。

- 答"专项测试就是性能测试"——不完整。可恢复性、异常处理、优先级调度都算专项。

- 收口：小系统怕出错，大系统怕出错之后不知道从哪里恢复。

> **易错点 / 考点：**① 四件主要工作和四阶段的名称要能默写；② 术语解释是送分题：ETL、DW、OLAP、Cube 各一句；③ 简答高频"大数据测试的主要工作包括哪些"，按数据准备、ETL、OLAP 与 Cube、报告四块答，再补质量、性能、可恢复性；④ 别把 Cube 说成报表，它是预计算的多维结构。

> **过渡：**工作清单有了，接下来看它们在项目里按什么顺序推进。

【14.10】大数据测试流程（清洗、分析）（3 分钟）

> 六段流程，骨架跟第 4 章那张图几乎是同一张。换了对象，骨头没换。

>

> 需求分析验四样：数据来源，这批数据从哪来、几个源、什么协议；数据关系，表和表怎么关联；计算逻辑，指标怎么算出来；任务配置，调度周期、依赖、并发。

>

> 设计评审评两样：代码评审，测试框架评审。评的是测试框架本身。大数据项目里测试也是一套要设计的系统。工具、脚本、比对逻辑都得有人看，不然你验别人的东西，自己那套没人验。这一步最容易被跳过，跳过之后错的比对逻辑会一路绿灯。

>

> 测试框架编写，三个工具方向：服务日志构造工具，报表数据构造工具，数据 diff 工具。为什么偏偏是这三

> 个？对三个痛点：造不出数据，比不出差异，复现不了现场。

>

> 数据准备：维度字段的映射转换，指标字段的计算逻辑，过滤条件，异常场景。还要准备配置文件、数据通道、产物路径。异常场景得专门造，空分区、脏数据、超时、乱序，一个都不能少。

>

> 测试执行：多方报表对比验证，接口协议测试，服务主流程，服务日志。再加上主流程各任务调度、定时任务调度、服务告警触发。到这一步，验的不只是数，还有调度和告警。

>

> 现网验证：测试数据验证功能，现网数据回归，现网数据的新老版本对比。最后回到总量上验证和多方报表对比。这一步跑在业务真流量上，发现问题还来得及补。

>

> 这条流程的脾气一句话：不是测一次，是从需求一路盯到现网，用总量和多方对比兜底。

提问：为什么最后一步一定要做新老版本数据对比，而不是只跑一遍新版本的用例？

> **预设回答：**

- 答"因为需求本来就没写清"——很诚实，也很真实。大数据项目的口径常常就一句"和以前一样"，而这句话只能用对比来证明。

- 答"因为要保证平滑迁移"——对。切换版本时业务最怕数变了却没人知道。

- 答"因为用例覆盖不全"——也对。对比是不依赖用例的兜底：用例只能验你想得到的，对比能验你没想到的。

- 收口：总量验证加多方报表对比的分量就在这儿，它们补的是用例覆盖率的洞。

易错点 / 考点：① 六段顺序要能复述，考试常出排序或流程填空；② 三个工具方向是简答高频，日志构造、报表构造、diff；③ 执行阶段不只有对数，还有调度和告警，这一点最容易漏；④ 概念辨析：现网验证不是上线后再说，它是上线前拿现网数据做的一次预演。

过渡：流程里每个环节都要人做，人做就一定慢。于是问题变成：哪些环节能交给脚本。

【14.11】大数据测试自动化（2 分钟）

> 人做一定慢。慢，就一定有人想省。

>

> 自动化的目标写得很短：缩短测试过程，提高测试效率和效果。后半截别丢。效果的意思是，能发现手工发现不了的东西。人能抽一百条，脚本能比一百万条。省时间是第一层，看得更宽是第二层。两层都做到，才叫提效。

>

> 三个方向：数据准备自动化，数据验证自动化，回归测试自动化。正好对上三个耗时大户，造数据、对数、重复跑。三件事有个共同点，都是重复。重复的活交给机器，人腾出手来做判断。

>

> 三项措施：开发服务日志和报表数据的构造工具，开发数据 diff 工具，引入接口自动化平台。三项措施不用一次全上。先挑最疼的那一件，跑顺了再加第二件。

>

> 适用那一列最值得琢磨。按数据的生命铺开，清洗阶段、分析阶段、数据服务阶段。再按报表类型分，离线报表、实时报表、数据服务报表。

>

> 为什么这一列重要？它告诉你自动化不是一把梭。清洗阶段重点是造脏数据、验清洗规则。分析阶段重点是 diff 指标口径。数据服务阶段重点在接口自动化，因为对外是 API，别人调的是你的接口。

>

> 给个判断标准。一件事每次都要做、规则稳定、结果可比对，就交给脚本。每次判断都不一样、要靠业务经验，就留给人。造数和 diff 属于前者。判断这个数业务上合不合理，属于后者。

>

> 这个标准比工具清单耐用。工具年年换，这条标准不换。

> **提问：**数据准备、数据验证、回归测试，只能先自动化一个，你选哪个？依据是什么？

> **预设回答：**

- 答"数据验证，它最耗时"——有依据，后面那页会给耗时缩短 90% 的说法，验证自动化的收益最直观。

- 答"数据准备，没数据什么都干不了"——也对，而且它能解锁另外两个。造数自动化了，回归才跑得起来。

- 答"三个一起上"——资源有限时这是空话。工程上必须排优先级。

- 收口：这不是选择题，是投入产出排序题。给出排序依据，比给出排序结果更值钱。

> **易错点 / 考点：** ① 三个方向和三项措施要对得上号：准备对构造工具，验证对 diff 工具，回归对接口自动化平台；② 目标里效果两个字别丢；③ 判断题"自动化能替代人工探索"——错，探索式判断和业务合理性判断仍然是人干；④ 三阶段乘三类报表这张适用矩阵能显著提升案例题得分。

> **过渡：** 三个方向里最底层、也最容易被忽略的是第一个。没有数据，后面全停。

【14.12】数据准备自动化（2 分钟）

> 没有数据，后面全停。所以自动化第一个要动的，是最不起眼的那一步。

>

> 两个场景：服务日志构造，报表数据构造。两个场景瞄的是同一样东西，把输入和输出都变成可复现的。

>

> 服务日志构造四步：拉取最新日志，解析现网日志，测试数据构造，数据格式转换。路径是先从真实里来，再造成我们要的样子。

>

> 为什么从现网日志来？日志里藏着最真实的字段组合、最真实的异常形态。你自己凭空造的数据太干净，跑得通，压不出问题。

>

> 报表数据构造，四类来源，这一块最有价值。

>

> 特殊数据用造数平台。边界值、异常值、极端组合，只能自己造。

>

> 真实数据用脚本同步。从现网同步一份真实样本回来。

>

> 配置表全量同步。配置表小，变化少，可它影响全局，所以整份搬。

>

> 数据表取最新天数据。只取最近一天，新鲜度和体量兼顾。

>

> 四类的分法照着数据的变化特性：特殊要造，真实要抄，配置要全，明细要新。

>

> 效果三条：解决输入输出不明确的问题；数据可复用，维护成本低；准备时间明显缩减。

>

> 第一条最要紧。大数据测试最难受的地方，就是不知道输入是什么、期望输出是什么。数据准备标准化了，等于把输入和输出两头都钉住。

>

> 再补一句工程上的实话：真正的价值是可重复。手工准备的数据，三小时后你别想再复现一遍。脚本准备的数据，一条命令就重建。能复现，才谈得上定位。

> **提问：**为什么配置表要全量同步，数据表却只取最新一天？

> **预设回答：**

- 答"配置表小，数据表大"——说出了表层。深层原因是影响面不同：配置表错一行，全链路结果都错，所以一条都不能漏。

- 答"因为配置表不常变"——正因为不常变，一变就是大事。全量同步才能保证它一变就被发现。

- 答"随便，能跑就行"——错答。准备策略选错，会出现本地全绿、现网全红。

- 收口：分数据的标准不是大小，是变化特性和影响面。

易错点 / 考点：① 服务日志构造四步、报表数据四类来源都要能列举；② 解决输入输出不明确的问题是简答

得分点，很多人只答省时间；③ 造数平台管特殊数据，脚本同步管真实数据，别反；④ 判断题陷阱"测试数据越接近真实越好"：接近真实对，但必须夹着刻意构造的异常数据，否则压不出问题。

过渡：数据准备好了，接下来是最耗人的活，验证。下面那组对比会让你看清自动化到底省在哪。

【14.13】数据验证自动化（2 分钟）

手工对数对到第三十遍，人就开始飘了。这一组对比最能说明自动化省在哪。

>

维度字段。手工是直接取，或者联表查询，一条条写 SQL。自动化是程序直接输出不一致的结果。差别在哪？它不给你全量数据，只给你对不上的那几十行。人不必在几万行里找。少看几十行，人才愿意真看；全量摆上去，人只会往下滑。

>

指标字段。手工要面对简单聚合、复杂场景计算、多层计算、分账逻辑，越复杂越容易算错。自动化保留中间计算结果，留着排查问题，还能输出每种场景下的实际结果和期望结果。手工算完就丢了过程，出问题重算一遍。自动化把过程留下来，等于自带证据。证据在手，你不必重跑一遍才能解释。

>

多表。手工一张张对。自动化可复用，一次运行，所有表的结果自动输出。表一多，手工的漏检几乎必然发生。

>

手工的弊端三条：容易疲劳、容易出错；每个指标都要算，比较耗时；类似表的处理逻辑没法复用。

>

自动化的优势三条：复用性高；计算速度快，测试耗时缩短了 90%，这个数按课件给的记，别自己再加码；定位问题方便。

>

第三条最容易被低估。很多人以为自动化值钱在快。快只是顺带的。手工发现少了三条，你还得再花两小时找是哪三条。脚本直接告诉你哪三条、差多少、在哪张表。

>

收一句：自动化不是把人替掉，是让人从算里出来，去做判。

提问：自动化验证只输出不一致的结果，这样做有没有风险？

预设回答：

- 答"没有，效率高"——有风险。比对逻辑本身写错，它会告诉你全部一致，然后你就放心上线了。

- 答"有风险，比对脚本自己也得验"——非常关键。行话叫验证器的验证：拿一份已知有差异的数据，确认脚本报得出来。

- 答"风险是漏掉格式差异"——真实情况。1 和 1.0 数值相等，字符串比对会报不一致，得先归一化。

- 收口：比对器上线前先做一次故意造错的自检。能抓错的比对器，才配叫比对器。

易错点 / 考点：① 手工三条弊端、自动三条优势要能列举；② 耗时缩短 90% 按课件口径引用，问数据时写清据课件；③ 核心考点是验证器的验证，比对的正确性要先被证明；④ 判断题陷阱"自动化验证通过就等于数据一定正确"——错，前提是比对逻辑和期望值都对。

【思政落点 1】：比对脚本说全部一致，你就敢上线吗？脚本也是人写的，它的错会以合格的样子出现。所以先用一份已知有差异的数据，确认它报得出来。这一步很小，态度很大：对自己造的工具保持怀疑，对结论保持诚实。数据岗位上，一次看起来都对，比一次明确的报错危险得多。

过渡：14.1 讲完了。数据是原料，接下来看拿原料做出来的东西，AI 系统本身怎么测。

【14.14】14.2 AI 系统的测试（1 分钟）

标题下面有一行英文：Test for AI。四个字定了立场，这里的主语是 AI 系统，它是被测对象，不是工具。

>

四小节。14.2.1 不确定性和不可解释性，先说清为什么难测。14.2.2 白盒测试，钻进模型内部看结构和覆盖。14.2.3 算法验证，用量化指标判一个模型好不好。14.2.4 示例：针对智能语音的设计与执行，落到一个具体场景。

>

> 这四节是递进的。先承认它难，因为这个对象本身不确定。再看能不能像测代码那样钻进去，这是白盒那一条路。钻不进去的一层，用量化的指标来判，这是算法验证。最后拿智能语音走一遍，你能看见这四节怎么落在同一件事上。顺序也别记反。先承认不确定，再想办法钻进去看。看不进去的部分改用量化指标，最后拿一个真实场景走通。

>

> 立个靶子：传统软件的缺陷是写错了，AI 系统的缺陷是学偏了。写错了，你能定位到某一行代码。学偏了，你只看到一个概率分布不对。所以测 AI，方法得换一套。换的不是工具，是判据：从对不对，换成够不够好、稳不稳。

>

> 这一节会出现一堆数学名词，覆盖率、召回率、AUC、KS、PSI。别被吓住。它们干的都是同一件事，把好不好翻译成数字。跟第 7 章用响应时间、吞吐量衡量性能是一个路数。你要抓的是每个指标在回答什么问题，不是它的公式怎么推。

>

> 分界线一句话：AI 测试不是能不能测的问题，是用什么标准测、测到什么程度算过的问题。

提问：有人说"AI 系统没法测，只能上线看效果"。你怎么回应？

预设回答：

- 答"确实没法测"——这是放弃。行业里已经有成体系的评价指标和验证方法，这一节就讲这些。

- 答"能测，但测的不是对错，是够不够好"——说到位了。这正是 AI 测试和功能测试的分水岭。

- 答"能测，看准确率就行"——不够。准确率单独看会被数据分布骗，后面还要召回、F1、AUC 和分布稳定性。

- 收口：问题从来不是能不能测，是用什么标准测、测到什么程度算过。

易错点 / 考点：① 四小节名称；② Test for AI 和 AI for Test 的方向必须分清，这是本章第一号易错点；③ 判断题"AI 系统测试可以完全复用传统功能测试方法"——错，传统方法只能覆盖工程部分，数据管道、接口、性能，模型效果要专门方法。

过渡：要讲清 AI 为什么难测，得回到七十多年前那个著名的思想实验。

【14.15】Test for AI ~ 从图灵测试说起（2 分钟）

> 1950 年，图灵提出那个著名的测试。隔着一道墙对话，人分不清对面是人还是机器，就认为机器有智能。

>

> 这个设计妙在哪？它不比内部，只比外部行为。不打开机器的脑袋看它怎么想，只看它表现得像不像人。

>

> 课件点了四个维度，四类像人的能力。计算能力，算得快算得准。产生随机数，输出不重复、不呆板。常识，知道水是湿的、鸟会飞。推理力，能从前提推出结论。四个维度今天看依然成立。大模型强在前三项，弱在第四项的一致性。一致性差在哪？同一件事问两遍，答案可能不一样，这在专业场景里很要命。

>

> 可图灵测试有个致命局限，也是今天必须超越它的理由。它判的是像不像人，不是对不对、稳不稳、公不公平。一个模型可以把话说得极像人，同时给你错的事实。可以在测试集上滴水不漏，换个场景就崩。

>

> 所以现代 AI 测试换了判据：从行为像人，转向行为可信。可信包含准确、稳定、可解释、无偏见、抗攻击。

>

> 图灵给了一个思路，用行为判智能。现代测试在这个思路上加了一层，用指标判质量。两层不冲突。行为观察告诉你它像不像人，指标告诉你它在什么条件下可靠、在什么条件下崩。

>

> 举个例子。一个模型通过了图灵测试，能拿去读 CT 片吗？像人只能说明它说话像医生，说明不了它看得准。医疗场景要的是准确率、可解释、能追责。这三样它一样都给不出来。

> **提问：**一个模型能通过图灵测试，能不能说明它可以放心用在医疗诊断上？

> **预设回答：**

- 答"能，说明它够聪明"——最危险的错答。像人不等于正确，医生要的是准确率和可追责。

- 答"不能，还得看准确率和可解释性"——对。医疗对错误的容忍度极低，指标和解释都得摆出来。

- 答"不能，还得看它错了会不会说不知道"——很好的补充，这叫拒识能力，专业场景里比答得像更重要。

- 收口：图灵测试是智能的入门考，不是质量的合格证。

> **易错点 / 考点：** ① 图灵测试的核心思路是行为等价，不看内部；② 四个维度要能列举；③ 简答高频"图灵测试为什么不足以评价今天的 AI 系统"，答三条：只判像不像人不判对不对，不涉及稳定性与泛化，不涉及公平、可解释、安全；④ 别把图灵测试和中文屋混起来，后者是塞尔对它的反驳。

> **过渡：** 既然不能只靠像人来判，那 AI 软件到底怎么测？下一页给答案。

【14.16】如何测试 AI 软件？（2 分钟）

> 课件用一个长句回答了 AI 软件怎么测。这个句子结构密，值得一个字一个字拆。

>

> 大意是这样：测 AI 软件，检验它有没有学习能力，落到根上就是算法验证。验的是元搜索、启发式搜索、深度强化学习这些算法，还有它们的组合。一般靠实验做，借大量数据做普适性验证。

>

> 拆四个关键词。

>

> 一，算法验证。AI 软件跟普通软件最大的结构差别在这儿。普通软件的家底是代码逻辑，AI 软件的家底是算法加数据训出来的模型。所以验的重点从代码分支走没走到，变成算法在数据上表现如何。

>

> 二，算法族。元搜索、启发式搜索、强化学习，加上组合。这句话在提醒你，AI 不是一种算法，是一族。搜索类、学习类各有各的验法，别拿一套标准套到底。

>

> 三，靠实验。这是方法上的关键。AI 测试更像科学实验，不像用例执行。设假设、控变量、跑多组、看统计。实验就得讲可重复：同一份数据、同一个随机种子，结果要能复现。

>

> 四，大量数据做普适性验证。普适性三个字最要紧，它管的是边界之外。小样本上表现好很容易。大量、多

> 样、带噪声的数据上还能稳住，才算真行。这也是 AI 测试绕不开数据集建设的原因：训练集、验证集、测试集，还有专门造的对抗样本集。

>

> 一句对比收住：传统测试问这个输入下输出对不对。AI 测试问这一大批输入上，它稳不稳、够不够好。四个词各对一个坑，漏一个，答题就少一分。

提问：为什么 AI 测试特别强调可复现？随机性不是很正常吗？

预设回答：

- 答"不可复现就没法定位缺陷"——核心原因。今天效果掉了，你连是不是随机种子变了都排除不了，别谈定位。

- 答"固定随机种子、固定数据版本、固定代码版本"——专业，这三样就是 AI 实验的三固定。

- 答"AI 本来就有随机性，复现不了很正常"——错答。承认输出有随机性，不等于承认过程不可控。

- 收口：AI 项目的测试报告里必须写清数据版本和训练配置，没有这些，数字没有意义。

易错点 / 考点：① 这句定义是本节第一名简答，要能背下来；② 四个关键词能展开；③ 普适性要能解释成大数据量、多样化、含噪声的条件下仍然稳定；④ 判断题陷阱"AI 测试就是跑一遍测试集算准确率"——错，还有交叉验证、对抗验证、A/B 测试、稳定性与公平性验证。

过渡：方法定了，接下来看它在工程上怎么组织成闭环。

【14.17】测试方法（2 分钟）

> 质量需求、验证与测试、评估报告、发布，四个框用箭头连成一个圈。失败就回头，重新调，再测一遍。圈上的箭头是双向的，回头的路必须有。

>

> 起点是质量需求。这一步决定后面所有指标怎么选。先回答一个问题：这个模型最怕哪种错？是漏判，本来要抓的没抓到，还是误判，冤枉了好人？风控怕漏判，推荐怕误判，医疗筛查宁可多报不可漏报。需求不同，指标就不同。

>

> 中间是验证与测试，课件点了三样。模型解释，它为什么给这个结论，能不能给出人看得懂的理由。模型检验，形式化或者系统性地检查模型性质。测试标准评估，按既定标准打分。输出是评估报告。

>

> 有两个技术点必须讲。Neuron coverage，神经元覆盖。Neuron boundary coverage，神经元边界覆盖。

>

> 这是第 8 章覆盖率思想搬到 AI 上。神经元覆盖相当于语句覆盖，看网络里每个神经元有没有被激活过。神经元边界覆盖相当于边界值分析，看神经元有没有被推到接近激活阈值的临界区。

>

> 这么一类比就通了：覆盖率量的都是同一件事，我到底摸到了模型的多少种状态。状态从代码行变成了神经元激活，问法没变。

>

> 最后两条路线。结构化方法看内部，神经元覆盖属于这一类。黑盒方法只看输入输出，测试集评估属于这一类。两条都要走，道理跟第 3 章一样。

>

> 还得说清一处：覆盖率是测了多少的度量，不是测得对的证明。这两句被混着用，答题就丢分。

提问：神经元覆盖率达到 100%，能不能说明这个模型被测透了？

> **预设回答：**

- 答"能，全覆盖了"——错答，覆盖率最经典的陷阱。覆盖到不等于验对了，第 8 章讲过同一句话。

- 答"不能，激活了输出也可能还是错的"——准确。神经元被激活只说明这条路走过，不说明走得对。

- 答"还要看激活的强度、组合和边界"——很专业。这正是边界覆盖和组合覆盖存在的理由。

- 收口：覆盖率是测了多少的度量，不是测得对的证明。传统测试和 AI 测试里都成立。

易错点 / 考点：① 闭环四环节加反馈回路；② Neuron coverage 与 Neuron boundary coverage 的类比是本节最常被问的辨析；③ 结构化与黑盒两条路线要能对上具体方法；④ 判断题"覆盖率越高模型越可靠"——错，覆盖率量的是测得够不够，不是测得对不对。

> **过渡：**要看懂 AI 测试插在哪儿，得先看清机器学习怎么开发。

【14.18】机器学习开发流程&架构（1 分钟）

图上没有一个字，全靠你自己读出来。读出三件事，你就知道测试往哪儿插。

>

> 第一，分层。数据从下往上走：数据层管采集、存储、清洗；特征层把原始数据加工成模型能吃的特征；模型层做训练和评估；服务层把模型包成接口对外给预测；再往上还有一层监控与反馈，把线上的真实表现收回来。以后看任何一张 AI 系统架构图，先找这五层在不在。缺监控的那张，一定没上过线。没上过线的系统，也谈不上测试。

>

> 第二，层与层之间有契约。数据层交给特征层的是格式和口径。特征层交给模型层的是哪些特征、什么取值范围。模型层交给服务层的是输入什么、输出什么、耗时多少。契约就是测试的抓手。契约写得清，每两层之间就能设检查点。

>

> 第三，测试的插入点不止一个。很多人以为 AI 测试就是最后测一下模型准不准。这张图上，每个层间接口都是测试点。数据层出来验数据质量，特征层出来验特征一致性。模型层出来验指标，服务层出来验接口和性能。监控回路验告警准不准。

>

> 这就是第 4 章那句测试贯穿全生命周期，在 AI 系统上长成的样子。流程图上没有测试框，可每一个箭头都是一次检查机会。

>

> 图怎么用？有人说我们的 AI 效果不好，你顺着五层往下问：数据脏了，特征偏了，模型弱了，还是服务慢了。这一问，比任何一份模型报告都省时间。

> **提问：**线上模型效果突然下降，你按这张图从哪一层开始查？

> **预设回答：**

- 答"先查数据，看输入分布变了没有"——最佳起点。行业里叫数据漂移，是线上效果下降的头号原因。

- 答"先查模型，是不是被人改了"——也可能，但模型版本变更通常有记录，查起来快，可以顺手排除。

- 答"先重训一版"——危险动作。原因没查清就重训，等于把证据毁掉，还可能把问题带进新版本。

- 收口：先看数据漂移，再看服务与依赖，最后才怀疑模型本身。

> **易错点 / 考点：** ① 五层结构要能默写；② 层间契约就是测试检查点，这个观点能明显抬高论述题的层次；③ 效果下降的排查顺序：数据漂移、服务与依赖、模型版本；④ 别把这张图只当开发流程，它同时是一张测试布点图。

> **过渡：** 把五层压成一条时间线，就是下一页那个简化模型。

【14.19】机器学习研发生命周期：简化模型（2 分钟）

> 把上一张架构图横过来看，就是一条五个台阶的路。

>

> 定义问题，数据收集，特征工程，模型训练，部署。五个台阶下面挂着三个质量：数据质量、特征质量、模型质量。

>

> 怎么挂？数据收集对应数据质量，进来的数据对不对、全不全。特征工程对应特征质量，加工出来的特征有没有区分度、稳不稳。模型训练对应模型质量，学出来的模型准不准、泛不泛化。

>

> 部署之后还有一件事，回归。数据变了，特征变了，模型质量得重新验一遍，不能拿上次的报告顶。

>

> 三层质量有先后，跟缺陷越早发现越便宜是一个道理，在 AI 项目里更极端。数据错了，特征再精也救不回来。特征错了，模型再强也学不到真规律。前提塌了，后面全塌。行业里有句话，AI 项目八成的功夫在数据和特征上。这不是贬低算法，是说质量的重心在上游。

>

> 还有第一步，定义问题。很多团队跳过它，直接开训。可它决定后面一切：这是分类还是回归？判错的代价两边对称吗？要多快出结果？问题定义偏了，你连什么叫好模型都不知道，指标也就无从选起。指标一错，后面所有努力都在错的方向上，越努力越远。

>

> 这一步跟整门课的关系点一下：这就是测试左移在 AI 项目里的样子。测试不是等模型训完才进场，是从定义问题就参与。帮业务把验收标准说清，帮数据团队把质量规则定下来。进场越早，后面返工越少。

提问：一个团队模型指标一直上不去，你建议先调模型，还是先回去看数据？

预设回答：

- 答"先调模型，换更强的网络"——最常见的反应。换模型能提几个点，可数据错了，天花板就摆在那儿。

- 答"先看数据和特征"——对。先做数据质量分析和特征分析，往往能挖出标签错了、特征穿越了这类致命问题。

- 答"两个一起做"——可以，但得先做一次数据体检，否则你分不清模型调优的效果是真是假。

- 收口：定义问题、数据、特征这三步，是 AI 项目里最需要测试盯住的三步。

易错点 / 考点：① 五步顺序和三层质量的对应关系；② 简答常考"机器学习项目的三个质量层次"，答数据质量、特征质量、模型质量；③ 判断高频"模型质量是 AI 项目最重要的质量"——错，数据和特征质量是前提；④ 结合测试左移答论述题，是这一页最好用的加分点。

过渡：道理都懂，可实际做起来为什么这么难。

【14.20】机器学习测试难点（3 分钟）

机器学习测试最难的地方，不在算法，在会议室里。

>

> 四组难点，每一组我补一句测试怎么应对。

>

> 一，人的门槛。技能和知识要求高，抽象、因素多、数学深；数据获取和问题分析也难。传统功能测试，你懂业务就能上手。AI 测试，你得同时懂业务、懂数据、懂一点统计。应对：团队里得有人专职盯评价指标和

> 数据集，别指望一个人全包。

>

> 二，强数据相关。数据质量要求高，脏数据直接把模型带歪；特征维度多，几十上百维，人眼看不过来，得靠分析工具。应对：把数据质量做成常规检查项，像做回归一样做数据体检。

>

> 三，节奏。模型优化慢，上线才刚开始，迭代慢。三句连起来读：训一次几小时甚至几天，上线后效果还要靠线上数据慢慢验。迭代周期天然比普通软件长。应对：能自动的部分先自动，数据检查、指标回归、A/B 分流分析，把每次迭代的验证时间压下来。

>

> 四，标准。验收标准不确定，上线效果不确定。传统项目验收是用例全过。AI 项目验收是准确率到 95%。可 95% 是谁定的？样本怎么抽的？换个场景还算不算？

>

> 应对就一条：项目一开始就写清三件事，指标是什么，在哪个数据集上算，达到多少算过。三样缺一样，这张验收单就签不了。

>

> 四组收一句：AI 测试难，不难在会不会用工具，难在标准不清的时候，你敢不敢把标准定清楚。这恰恰是测试岗最不好被替代的地方。

提问：业务方说"这个模型要准一点"，你怎么把它变成可验收的标准？

> **预设回答：**

- 答"那就把准确率提到 95%"——太粗。在什么数据上算？换个分布可能只剩七成，类别不均衡时准确率高而漏判严重。

- 答"先问清最怕哪种错，再选指标"——专业。怕漏判看召回率，怕误判看精准率，两头都要看 F1 或 AUC。

- 答"让业务方自己定"——责任推回去了，可你作为测试得提供选项和代价：召回提多少会带来多少误报，误报一次成本多少。

- 收口：把准一点翻译成在某数据集上召回率不低于多少、误报率不高于多少，这才叫验收标准。

易错点 / 考点： ① 四组难点要能列举；② 验收标准三要素，指标、数据集、阈值，是简答高频；③ 判断题"AI 项目上线就意味着项目结束"——错，课件明说上线才刚开始，线上监控和迭代才是主战场；④ 结合不确定的验收标准来答测试人员的价值，是很稳的论述题素材。

【思政落点 2】：业务方说准一点，你把 95% 写进验收单，签字那一刻，责任就落到你身上。风控误判一次，是一个人的贷款被拒；医疗漏筛一次，是一个人错过治疗窗口。指标是冷的，后果是热的。把标准定清楚、把代价说清楚、把不确定的地方标出来，这是测试岗的职业底线。

过渡：难点清楚了，再看机器学习测试和传统测试比，特点到底在哪。

【14.21】机器学习测试特点（1 分钟）

先看一组对照。传统软件的一个函数，三个参数、每个几档取值，组合数得完。换成一张图片，进来就是几百万个像素，你永远枚举不完。

>

输入这一头先变了。所以 AI 测试靠不了穷举，只能靠采样和分布。你测的是"这一类输入"，不是"这一个输入"。还想着把所有情况列一遍的人，路一开始就堵死了。

>

输出这一头也变了。传统软件输出 3 就是 3，模型输出的是"0.87 的概率是猫"。判据跟着换：不问等不等于，问置信度够不够、阈值定得合不合理。

>

第三处，预期结果没有唯一答案。同一张图，两个模型给出不同答案，可能都对；同一个模型跑两次，结果也可能差一点。断言这条路走不通，得换成指标加容差。

>

第四处，缺陷不必然复现。传统缺陷照着步骤必现，AI 的缺陷可能只在某类边缘样本上偶发。缺陷单只写步骤没用，样本得带上。

>

第五处，视角要两个。工程那一面照旧：接口、性能、异常处理。效果那一面像做实验：指标、分布、稳定

> 性。少一条腿都站不住。

>

> 为什么非要双视角？只验工程，模型效果一塌糊涂也能全绿；只验效果，接口一崩什么都读不出来。两条腿一条短，走着走着就摔。

>

> 课设里把这两条写进测试计划：一条验接口和异常，一条验指标和分布。两边都留证据，答辩才讲得清。

>

> 五条收成一句：判定对错升级成评估好坏，用例升级成数据集，复现步骤升级成复现样本。

> **提问：**模型"偶发判错"的缺陷单，你要求提交者带上什么？

> **预设回答：**

- 答"带上出错的样本"——最基本也最要紧。没有样本，别人复现不了，也判断不了是不是数据的问题。
- 答"带上当时的输入、模型版本、阈值配置"——更完整，这三样缺一个，结论都可能不同。
- 答"带上复现步骤就行"——不够。AI 场景里步骤往往就是"喂这个样本"，关键在样本本身。
- 收口：AI 缺陷单的模板得改：样本、版本、配置、期望与实际输出，四样都要。

> **易错点 / 考点：**五组特点要能对比着说；"输出概率化"和"没有唯一预期"是分量最重的两条；简答高频"机器学习测试与传统测试的区别"，按输入、输出、判据、缺陷复现、验证视角五点答；别把"偶发"当成"不是缺陷"，偶发在 AI 里往往意味着边缘样本覆盖不足。

> **过渡：**特点定下来，接下来看力气该往哪儿使。

【14.22】机器学习测试的重点（2 分钟）

> 力气往哪儿使？这个问题在 AI 项目里最容易答错。把四层摊开看就清楚了。

>

> 最底下一层是数据。数据测试，加上 Data Monitoring。数据放最下面，因为它一歪，上面全歪。数据测试验的是质量那六个维度，下一节专讲。Data Monitoring 是上线以后盯着输入分布有没有漂移。

>

> 往上一层是特征。特征是从数据加工出来、直接喂给模型的东西。特征错有两种典型：一种是算错，逻辑bug；一种是穿越，用了未来信息。穿越最难查，离线指标漂亮得不像话，一上线就崩。

>

> 再往上是模型。课件这层标的是 Model／Training，对象里有模型，也有代码。模型测试，旁边还挂着 ML Infrastructure Tests、Unit Tests、Integration Tests。这一层不只验模型指标，还验撑着模型的那套东西：训练管线、特征服务、推理服务。这些都是普通软件，单测和集成测试一样省不掉。

>

> 最上面是系统运行。Running／System，两块监控。Prediction Monitoring 看输出分布有没有变。System Monitoring 看服务还活着吗、延迟多少。

>

> 看这张图上的面积。数据和特征占的面积，远大于模型。它在说一件反直觉的事：AI 项目的测试工作量，大头不在测模型，在测数据和特征。很多团队把八成精力砸在调模型上，回头反复在数据和特征上翻车。

>

> 四层顺序：数据、特征、模型、系统运行。自下而上，一层盖一层。

>

> 顺带说个边界。这四层不是让你每层都做满。项目周期短，先把数据那两层的检查做实，模型层的指标自然好谈；反过来先调模型、后补数据，返工的概率很高。

>

提问：为什么"特征穿越"这类缺陷在离线阶段很难被发现？

预设回答：

- 答"因为离线指标反而更好看"——正是。用了未来信息，模型等于作弊，离线 AUC 高得不正常。指标好得离谱，本身就是可疑信号。

- 答"因为离线数据是全量的，天然带着未来数据"——准确。训练时把整段时间的数据混在一起，最容易穿越。

- 答"因为没人看特征代码"——现实原因。特征往往是 SQL 或者脚本，缺评审。

- 收口：特征测试要把"时间切分对不对"列成必查项。

> **易错点 / 考点：**四层结构与各层的测试项；核心结论是工作量重心在数据和特征；特征穿越（数据泄漏）是高频案例题，识别信号是"离线指标异常好"；判断题陷阱"模型测试就是 AI 测试的全部"——错。

> **过渡：**重心落在数据上，那就先把数据这一层讲透。

【14.23】数据质量（2 分钟）

> 数据质量这六个词，考试爱考，工作里更爱出事。六个维度一个个过，每个配一个反例，听完就分得清。

>

> 正确性，Accuracy。数据是否正确体现真实的业务，或者有没有可证实的来源。反例：系统里用户年龄 200 岁，格式合法，范围荒唐。

>

> 完整性，Integrity。基于业务分析，数据在字段、维度、参数上齐不齐、够不够完备。反例：风控模型缺了近半年逾期次数这个关键字段，模型再好也算不准。

>

> 一致性，Consistency。数据有没有被一致地定义、一致地理解。反例：同一个"活跃用户"，市场部按登录算，财务部按付费算，两份报表永远对不上。这一条讲的常常不是数据错了，是口径没统一。

>

> 有效性，Validity。数据是否落在业务定义的可接受范围之内。反例：利率字段冒出 -3%，或者枚举值出现系统里根本没定义过的状态。

>

> 时效性，Timeliness。数据是不是已经过时。反例：特征表 T+3 才更新，业务要的是实时风控。数据是对的，只是来晚了，等于没有。

>

> 可获取性，Accessibility。数据是否容易拿到、容易用。反例：数据躺在离职同事的本地库里，或者权限申

> 请要两周。这条最容易被当成管理问题忽略，可它是实打实的质量维度。

>

> 给一个抓手，六问：对不对、全不全、同不同、在范围内、新不新、拿不拿得到。以后做数据体检，按这六问过一遍。

>

> 六条里，前两条查事实，中间两条查口径，后两条查用不用得上。分组不严，答题时最容易把时效和可获取性记混。

>

> 落到项目上，数据体检按周跑，问题按维度归口：谁定义的口径，谁去改定义。

提问："两份报表数字不一样，但各自都对"，这属于哪个维度的问题？

预设回答：

- 答"正确性问题"——不一定。两边各自的计算都符合自己的口径，数据本身就是对的。

- 答"一致性问题，口径没统一"——准确。这类问题在大数据项目里占比很高，解决靠指标定义管理，不靠改代码。

- 答"完整性问题"——偏了，完整性讲的是字段齐不齐。

- 收口：遇到"两个数对不上"，第一步先问两边的口径定义是不是同一份，别急着查 bug。

易错点 / 考点：六个维度的英文名和中文名都要会，考试常给英文让翻译；一致性与正确性的区别是最高频辨析，正确性看和事实符不符，一致性看彼此的理解统不统一；时效性容易被忽略，案例常见（实时风控却用 T+1 数据）；口诀"对、全、同、规、新、得"。

【思政落点 3】：数据质量说到底是在问人肯不肯较真。年龄 200 岁这种记录，改一行就能遮过去，也可以顺着往上游查为什么没校验。挑掉难看的数据、只报好看的口径，模型指标会好看，可结论是假的。不修饰样本、不将就口径，就是实事求是落在数据这一行上的样子。

过渡：六个维度管的是查什么，接着看怎么查。

【14.24】数据质量分析方法（2 分钟）

报表上的数突然翻倍，你先查哪儿？这种时刻最考人，方法不对，就在几百张表里瞎翻。

>

第一问，某模块涉及哪些表，表之间怎么关联。对应的动作是输出血缘关系图和血缘基数分析。血缘图回答这个指标从哪些表、经过哪些加工来。血缘基数回答一条源记录对应几条目标记录。一对多还是多对一，直接决定聚合会不会重复算。没有这张图，你只能凭记忆猜。

>

第二问，数据在时序上有没有问题。对应波动分析。看指标随时间的曲线：突然跳变、周期性尖峰、断崖下跌。波动分析最擅长抓两件事。漏数，某天数据没进来，曲线掉零。重复，重跑没去重，数据翻倍。

>

第三问，每个字段有没有问题。对应值域分析和数据分布分析，也就是异常分析。值域看边界，年龄出现 200、金额出现负数，都属于越界。分布看形状，某个枚举值占比从 30% 掉到 3%，一定有事情发生。

>

第四问，数据是否一致。对应有效字段一致率。同一个业务实体的关键字段，在不同表里要对得上。一致率掉下来，说明某一环处理有问题。

>

四问连起来是一条固定路径：先看结构，血缘；再看时间，波动；再看单个字段，值域和分布；最后看跨表，一致率。做数据体检按这个顺序走，不会漏。

>

四问不必一次问完。出事先用哪一问，看现象决定。数字跟别人对不上，先看结构；数字自己跳变，先看时间；单个字段冒怪值，先看值域。

>

四把工具用顺手，排查就从瞎翻变成按图走。

提问：某天报表的订单量突然翻倍，别的指标正常，你先用哪种分析？

> **预设回答：**

- 答"波动分析，先看曲线"——对。翻倍是典型的时间序列突变，一眼能定位到具体日期。

- 答"查是不是任务重跑了"——很准。翻倍最常见两个原因：重跑没去重，关联把行数放大了。

- 答"查值域"——方向偏了。值域看单字段取值范围，翻倍是量的问题。

- 收口：量的问题看波动和血缘基数，值的问题看值域和分布。先分清是量错还是值错。

> **易错点 / 考点：**四问与四法的对应关系是送分题；血缘基数分析能解释"为什么关联之后行数变多"；高频案例是数据翻倍对重跑未去重、一对多关联，数据掉零对上游分区缺失；判断题陷阱"分布分析只看单个字段"——错，分布也包括多字段的组合分布。

> **过渡：**数据层的四把工具到手了，往上一层是特征。

【14.25】特征测试的关键（2 分钟）

> 特征测试有五个关键点，我按一句一个问题讲：准不准、有没有用、方向对不对、来不来得及、守不守规矩。

>

> 第一个，特征阈值。特征要离散成区间才能用，阈值定在哪儿直接决定模型行为。要验阈值附近的表现，还要验阈值改了以后影响面有多大。

>

> 第二个，相关性和重要性。这个特征对结果有没有贡献，贡献多大。没贡献的特征是噪声，拖累模型，还可能带出过拟合。测的时候要能答一句：去掉它，指标掉多少。

>

> 第三个，特征与结果变量的关系。这比相关性再往前一步，问的是方向对不对。收入越高、违约率越低，这是业务常识。模型要是学出反方向，说明数据或者标签有问题。这类检查行话叫业务合理性校验，人工经验最值钱的地方就在这儿。

>

> 第四个，适合性和计算成本。特征好不好算、要多久算出来、拿不拿得到实时值。一个离线要跑两小时的特

> 征塞进实时风控，就是灾难。特征本身没错，是地方用错了。

>

> 第五个，合规性。技术同学最容易漏这条，可它能整条业务下架。用了不允许用的个人信息，或者把性别、地域这类字段直接喂进模型，都会出事。合规不是法务一个部门的事，设计阶段就得问。

>

> 再加两条工程手段。特征工程的单元测试，每个特征的计算逻辑都要有单测，固定输入样本、断言输出值。特征工程的静态检查，扫硬编码、扫除零风险、扫时间函数误用。

>

> 最后补一个词，特征一致性。离线算出来的和线上算出来的必须是同一个值，这是后面那套 diff 架构要解决的事。

提问：一个特征离线效果很好，但计算要跑两小时，能用在实时风控里吗？

> **预设回答：**

- 答"能，效果最重要"——错答。实时风控要求毫秒级响应，两小时的特征根本取不到值。

- 答"不行，得看可不可以改成实时计算"——对。要么重构算法，要么降级换成近似特征。

- 答"可以先离线算好缓存起来"——可行，但时效性要盯住。缓存可能是昨天的，风控要的是此刻。

- 收口：特征测试不只问准不准，还要问来不来得及。这就是适合性和计算成本的意义。

易错点 / 考点：五个关键点要能列举，合规性最容易被漏；"特征与结果变量的关系"要能举例说明方向性检查；简答高频"特征测试包括哪些内容"；判断题陷阱"相关性越高的特征越好"——错，还要看稳定性、成本、合规性，以及会不会造成过拟合。

> **过渡：**五个点放进一个真实场景，看它变成什么样。

【14.26】特征测试要点示例：风控模型场景（2 分钟）

> 上一页那五点在风控里长什么样？课件把它摊成一大片，按四组归好，你就不用死背几十个词。

>

> 第一组，特征离线。验计算逻辑对不对，HQL 写得对不对，应用逻辑准不准。验特征回溯跑不跑得出历史值，回溯数据处理有没有漏。验分区修改有没有生效，性能也就是计算时长在窗口里跑不跑得完。离线那一侧的 Map 环节，输入输出也要对得上。还有四元组顺序，主体、时间、特征、取值，这组标识要能唯一定位一条记录。最后是 Shuffle 乱序之后的特征产出顺序稳不稳。这条最阴：分布式计算乱序，代码里要是隐含"按到达顺序取第一条"，结果就随机漂移，数据量一大才现形。

>

> 第二组，特征在线。接口响应时间、在线取值对不对、覆盖度够不够、下游超时和字段缺失的时候怎么降级。

>

> 第三组，一致性，这是重心。前后一致，同一主体不同时间取到的同一特征不许乱变。离线在线一致，同一个样本两边各算一遍，值要相同。覆盖率一致，离线九成，线上只剩六成就说不过去。特征值一致，还有特征 KS 验证，看区分能力在线上还在不在。离线 AUC 0.8、上线跌到 0.6，八成栽在这里。

>

> 第四组，工程规范。命名规范、编码规范、空字符处理、异常处理，这四项落在代码上。跑批日志、特征调度、监控效果、loan_dt 校验，落在运行上。响应时长和跑批时间也算在内。业务日期取错一天，全盘错。

>

> 四组里最容易省的是第四组。可它出的错最不像测试能发现的错。日期取错、日志没打，事后只能靠人回忆。谁都说不清那天跑的是哪一版。

>

> 收一句：风控特征测试，一半在算得对，一半在对得上。

提问：模型离线 AUC 很高，上线后效果明显变差，最可能是什么问题？

> **预设回答：**

- 答"离线在线特征不一致"——头号嫌疑。训练用的特征和线上取的特征，不是同一份口径。

- 答"线上数据分布跟训练集不一样"——第二大嫌疑，也就是数据漂移。

- 答"模型过拟合了"——可能，但离线验证集通常看得出来。上线才崩，往往指向工程链路。

- 收口：排查顺序是先验一致性，再验分布，最后才怀疑模型。

易错点 / 考点：一致性五项（前后、离线在线、覆盖率、特征值、KS）是案例题核心；"Shuffle 之后特征产出顺序"这类分布式陷阱要能解释成因；业务日期口径取错是真实高频事故；判断题陷阱"离线指标好就代表线上一定好"——错，中间隔着特征一致性这道关。

过渡：特征好不好，最后还是要落到指标上。

【14.27】特征效果分析指标（1 分钟）

有人拿一张特征评估表来问你：AUC 0.75、PSI 0.35、覆盖率 0.6，这特征用不用得上？三个数各说一件事，混着看就会被骗。指标分三类。

>

第一类，区分度，代表是 KS 和 AUC。它们回答一个问题：好坏样本分不分得开。KS 看好坏两个分布之间的最大差距，差距越大区分力越强。AUC 看的是随机取一个好样本和一个坏样本，好样本得分更高的概率。风控和评分卡最常用这两个。

>

第二类，稳定性，代表是 PSI。它回答：这个特征在不同时间、不同人群上，分布变了没有。PSI 小说明稳，PSI 大说明人群结构变了，模型可能失效。区分度和稳定性是两件事。一个特征可以区分度很高、又极不稳定，上线三个月就废。

>

第三类，覆盖率和缺失类指标。它回答：这个特征有多少样本能取到值。覆盖率低，等于模型在大量样本上盲猜，指标再好也不可信。

>

三类指标不是并列的，有先后。先看覆盖率，取不到值的特征，后面两个都不用算。再看稳定性，漂移得厉害的特征，区分度只是历史成绩。最后才看区分度，这时的 KS 和 AUC 才可信。顺序反了，你会拿着一堆好看的数，上线一个已经废掉的特征。

>

三句话收住：KS 和 AUC 看分不分得开，PSI 看会不会变，覆盖率看有没有值。选特征三个都得看。只高不

> 稳的特征，上线即报废；覆盖率高但区分度低的特征，占着位置不干活。

>

> 评估表上常还能看到 IV 一类的数，考纲没要求，看到不慌就行。

> **提问：**一个特征 AUC 0.75、PSI 0.35，你要不要用它？

> **预设回答：**

- 答"用，AUC 挺高"——只看了一半。PSI 0.35 说明分布已经明显漂移，通常 0.25 以上就要警告，上线后表现会掉。

- 答"不用，PSI 太高"——也不绝对。要看漂移是短期波动还是趋势性变化，以及可不可以重算分箱。

- 答"先查漂移原因再决定"——最专业的说法。分布漂移往往意味着客群变了，或者上游口径变了，先定性再定量。

- 收口：指标要成组看，单看一个数一定被骗。

> **易错点 / 考点：**三类指标的用途分工要清；KS 和 AUC 都在衡量区分度，别把 KS 说成稳定性指标；PSI 的经验阈值（0.1 以下稳定、0.25 以上要关注）答题时注明"经验值"；判断题陷阱"AUC 高就是好特征"——错，还要看稳定性和覆盖率。

> **过渡：**一致性要验成事实，就得靠一套架构。

【14.28】特征稳定性测试架构（2 分钟）

> 一致性光嘴上说，验不出来。要把它验成事实，得有架构。核心想法一句话：同一批样本，走两条路取特征，比差异。

>

> 先看三条比对路线。离线 diff，两个离线环境各算一遍，比如新老两版逻辑。在线 diff，同一个在线接口的不同环境比，或者同一环境不同时间比。在线／离线 diff 是最关键的一条：离线算一遍、线上取一遍，比同一个样本的特征值。

>

> 再看数据怎么流。先样本获取。一路获取离线特征。另一路通过 MapReduce 请求打到在线的不同环境，或者直接调在线接口拿在线特征。两路都送进 Diff 处理模块，由统一 diff 脚本产出 Diff 处理结果。

>

> 结果要看四样。各环境下的整体覆盖率，先确认两边都取到了值，不然比出来的差异没意义。每维特征的 diff 率，哪个特征差异大，一眼定位。相同覆盖下的 diff 样本及对应的特征，两边都有值但值不同，这些才是真问题。覆盖率不同的样本，一边有一边没有，属于取值链路的问题，跟数值不一致是两类缺陷，得分开报。

>

> 再有两个衍生指标：相同覆盖样本下的整体 diff 率，覆盖率不同的样本占比。前者高就是算错了，后者高就是取不到。

>

> 工程上还有一句：diff 要固定样本集和时间窗口。每次都换一批样本，diff 率就没有可比性了。

>

> 为什么要两条路？因为离线环境和线上环境跑的压根不是同一套东西。一个跑批，一个走服务，写的人可能都不是同一拨。不比对，你只能选择相信它们一致。

提问：在线离线 diff 率 15%，你判断是严重问题还是可接受？依据是什么？

预设回答：

- 答"15% 太高了，肯定有问题"——方向对，缺依据。得先看这 15% 集中在哪几维特征。集中在两三个特征上，多半是逻辑差异；均匀分布，可能是时间窗口不一致。

- 答"要看业务容忍度"——专业。特征重要性不同，核心特征 1% 的 diff 也扛不住。

- 答"看是不是浮点精度问题"——很好的补充。浮点和精度截断会造出一堆假差异，比对前要归一化。

- 收口：diff 报告别只报一个总数，要按特征维度拆开报。

易错点 / 考点：三条 diff 路线，在线／离线这条分量最重；结果四项要能列举，"覆盖率不同的样本"和"相同覆盖下的 diff 样本"是两类不同缺陷；比对前要做归一化，精度、空值表示、时区都要管；判断题陷阱"diff 率越低越好"——别为了降 diff 率去放宽比对标准，那是自欺。

> **过渡：**特征验完，轮到模型本身。

【14.29】算法验证（2 分钟）

算法怎么验？很多人第一反应是跑个指标看看。指标只是六种手段里的一种，前面还有五道。

>

第一，评审。看算法设计合不合理、假设成不成立、有没有明显的方法性错误。这一步最便宜，也最容易被跳过。评审能挡掉的问题，别留到跑实验。

>

第二，验证和测试数据集。要有独立的验证集、测试集，还得跟训练集在时间上、人群上合理切开。切错了，比如把时间序列随机切，指标立刻虚高。

>

第三，具体验证指标。准确率、召回、F1、AUC、MSE 这一套。指标要先定再跑，别跑完挑一个好看的。

>

第四，交叉验证。数据切成 K 份，轮流当验证集。它的用处是压掉一次切分带来的偶然性，小样本场景更需要。

>

第五，对抗验证，这一招很巧。训练一个分类器去分"训练集样本"和"测试集样本"。它要是轻松分得开，说明两份数据分布差得远，你的离线指标不可信。这是分布一致性的探测器。

>

第六，A/B 测试。前面五种都在离线，只有它能回答上线以后到底有没有用：切一部分用户用新模型，跟对照组比业务指标。

>

六种手段落在三个层次：设计层的评审，离线层的数据集、指标、交叉验证、对抗验证，线上层的 A/B。三层都过了，才敢说这个算法验过。

>

> 项目里最容易被跳过的是评审和对抗验证。评审费人，对抗验证要多训一个模型。可这两样恰恰最便宜地暴露大问题，剩下的靠指标和 A/B 兜住。

提问：离线指标很好，但你不确定测试集和训练集分布是否一致，用什么手段检查？

预设回答：

- 答"对抗验证"——标准答案。让分类器去分辨两份数据，分得开就说明分布不同。

- 答"画分布图对比"——可行，可高维特征人眼看不过来，对抗验证更系统。

- 答"重新随机切分一次"——解决不了根本问题。时间序列上随机切，反而制造泄漏。

- 收口：记住这三件套：切分要合理、CV 压偶然、对抗验分布。

易错点 / 考点：六种手段的名称与用途；对抗验证的原理是简答高频，要说清"分类器分得开就是分布不一致"；A/B 是唯一的线上验证手段，看清它跟离线指标的分工；判断题陷阱"交叉验证能解决数据泄漏"——不行。泄漏只能靠切分方式解决。

过渡：六种手段里，"具体验证指标"得单独讲，它决定你会不会读一张评估表。

【14.30】算法评估指标（2 分钟）

> 指标选错，后面的活全白费。评估指标先按问题类型分两大类：回归问题和分类问题。先分清类型再选指标，顺序别反。

>

> 回归预测的是连续数值，比如房价、违约损失金额。课件给了四个。均方误差 MSE，误差平方的平均，对大误差敏感。平均绝对误差 MAE，误差绝对值的平均，单位跟原数据一致，好解释。可释方差分数，看模型解释了多少方差。R2 分数，也就是决定系数，越接近 1 拟合越好，等于 0 相当于只用均值预测。MSE 和 MAE 的分工要讲清：MSE 罚大错，MAE 反映平均偏差。怕大错用 MSE，要业务口径用 MAE。

>

> 分类预测的是类别，比如是不是猫、会不会违约。课件给了准确率、精准率、召回率、F1 分数，还有 ROC 曲线和 AUC。

>

> ROC 这条要记准。横轴是 False positive rate，FPR，特异度那一侧。纵轴是 Recall，也就是 true positive rate，TPR，灵敏度。AUC 是 ROC 曲线下的面积。直观含义是随机取一正一负两个样本，正样本得分高于负样本的概率。它最大的好处是不受阈值和类别比例影响，类别不均衡的场景特别合适。

>

> 再补一句方法。指标要跟业务代价挂钩。漏判一个欺诈客户损失十万，误判一个正常客户只是多一次人工复核。两种错代价差得远，指标就得偏向召回。选指标，其实是在选我们更怕哪一种错。

>

> 两类指标别互相借。有人拿准确率去讲一个预测金额的模型，听众只能糊里糊涂点头。

>

> 还有一件事：同一个模型换一批阈值，所有分类指标都会变。报告里不写阈值，这些数就没法比。

> **提问：**一个反欺诈模型，你更看重精准率还是召回率？为什么？

> **预设回答：**

- 答"召回率，别漏掉坏人"——主流答案。漏判的代价通常就是直接的资金损失。

- 答"精准率，别打扰好人"——在某些场景也成立，比如直接把卡停掉。误判会让正常用户投诉、流失。

- 答"看业务能承受多少人工复核"——最专业的回答。召回提上去，多出来的疑似要靠人核，人力就是约束条件。

- 收口：正确做法是把召回和精准的权衡曲线交出去，让业务在代价扛得住的点上做决定。

> **易错点 / 考点：**回归四指标和分类指标别混；ROC 的横纵轴是送分题也是最易错题，横 FPR、纵 TPR；AUC 的含义和"不受阈值影响"这个优点必考；判断题陷阱"准确率高就说明模型好"——类别不均衡时准确率会骗人，99% 负样本，全判负也有 99%。

> **过渡：**分类指标里还有两组进阶概念在等着。

【14.31】分类算法评价指标（2 分钟）

> 多分类模型报出一堆数，Micro 和 Macro 打架，你信哪个？先别急，把三件事拆开看。

>

> 先看 P-R 曲线。课件给了两句判断：精确率越高，模型对负样本的区分能力越强；召回率越高，模型对正样本的区分能力越强。为什么这么讲？精确率高，意思是判成正的里面假阳性少，没把负样本误认成正的。召回率高，意思是实际的正样本基本都抓到了，正样本没漏。这两头天生互相拉扯，调阈值就是在这条曲线上挪。

>

> 再看 Micro 和 Macro。多分类要把各类指标汇总成一个总指标，两种汇总法。Micro 把所有类别的 TP、FP、FN 合起来算一次，大样本类别的影响更大。Macro 先算每个类别的指标再取平均，每个类别权重相同，小样本类别的影响更大。

>

> 选择规则课件写得很清楚。类别比较均衡，Micro 或者 Macro 都行。认为大样本类别要占更重要的位置，用 Micro。认为小样本类别要占重要位置，用 Macro。

>

> 最值钱的是这两条判断，考试特别爱考。Micro 明显低于 Macro，意味着大样本类别里出现了严重的分类错误。Macro 明显低于 Micro，意味着小样本类别里出现了严重的分类错误。道理不难：Micro 被大类别主导，它低就是大类别差；Macro 被小类别拉平权，它低就是小类别差。

>

> 还有一点常被问：Macro 低不等于模型没用。它可能只是把几个稀有类别牺牲掉了。要不要管，取决于那几个类别在业务里值多少钱。

>

> 两个指标摆在一起，差异本身就是诊断结论。所以看多分类报告，先问一句小类别是什么。

> **提问：**一个多分类模型 Macro F1 只有 0.55，Micro F1 有 0.88，问题出在哪？

> **预设回答：**

- 答"小样本类别分类效果差"——正确。Macro 低而 Micro 高，典型的小类别被牺牲。

- 答"整体效果不错，可以上线"——危险。小类别要是恰好是高风险类别，比如罕见欺诈类型，上线就是把风险

放过去。

- 答"加大整体数据量就行"——不一定。小类别的问题往往是样本太少，或者特征没有区分度，加数据不加小类样本没用。

- 收口：看多分类报告，两个指标一起看，差异就是诊断线索。

> **易错点 / 考点：**Micro 与 Macro 的计算差别、各自被谁主导；两条差异判断的方向极易记反，记"谁低谁背锅"，Micro 低是大类别错，Macro 低是小类别错；P-R 曲线的两句判断要能解释；判断题陷阱"Macro 一定比 Micro 更公平"——它只是给了小类别更高权重，公不公平看业务目标。

> **过渡：**概念摆完了，来算一遍真数。

【14.32】示例：分类问题指标计算（2 分钟）

> 十只猫，七个判对，这句话背后是一道必考题。先把混淆矩阵四个格子钉死。

>

> TP，True Positive，真阳性：实际是猫，模型也说是猫，判对了的正例。FN，False Negative，假阴性：实际是猫，模型说不是，漏掉的。TN，True Negative，真阴性：实际不是猫，模型也说不是，判对了的负例。FP，False Positive，假阳性：实际不是猫，模型说是猫，冤枉的。

>

> 三个公式跟上。准确率等于 (TP+TN)/(TP+TN+FP+FN)，所有样本里判对的比例。精准率等于 TP/(TP+FP)。我说是猫的里面真有猫的占多少。分母是我判成正的，所以它衡量我的判断有多可信。召回率等于 TP/(TP+FN)，真的猫里我抓到多少，分母是实际为正的，所以它衡量我漏没漏。

>

> 套课件的数据：准确率 7/10，70%；精准率 3/4，75%；召回率 3/6，60%。

>

> 这三个数才是这道题的价值。准确率 70% 看着一般。精准率 75% 说明我说是猫的时候，四次里对三次。召回率 60% 说明六只真猫我漏了两只。业务要是抓猫，漏掉四成扛不住，这时候就得把阈值调低，把召回提上来，代价是精准率往下掉。

>

> 口诀：精准率看判得准不准，召回率看漏得多不多，准确率看整体对不对。准确率最容易被不均衡骗：十个里只有一只猫，全判不是猫也有九成。

>

> FP 和 FN 的名字最容易绕。F 是 False，判错了；后面那个 P 和 N 说的是模型给出的那一票。FP 是判成正的、其实为负，FN 是判成负的、其实为正。念两遍就顺了。

>

> 混淆矩阵一画，三个公式都能从格子里推出来，不用硬背。考试常换一组新数，公式记牢比记结果管用，画个格子数一遍就出来了。

提问：如果业务最怕"漏掉真猫"，你调阈值往哪边调？三个指标会怎么变？

预设回答：

- 答"调低阈值，让更多样本被判成猫"——方向对。召回上升，精准下降，准确率可能升也可能降。

- 答"调高阈值更严格"——反了。那会让召回更低、精准更高。

- 答"三个指标会一起上升"——不可能。精准和召回此消彼长，这正是 P-R 曲线存在的理由。

- 收口：面对业务需求，测试交出去的不是一个更好的模型，是不同阈值下的指标组合，让业务来选。

易错点 / 考点：混淆矩阵四格的定义，FP 和 FN 最容易颠倒，F 后面的 P/N 说的是模型判成什么，不是实际是什么；三个公式必会，代入数要算得对；高频陷阱"准确率 90% 的模型好不好"，得先问类别分布；记住精准率分母是判为正的，召回率分母是实际为正的，这一条能救不少分。

过渡：分类算完，回归那一侧怎么验？

【14.33】示例：回归类算法验证（1 分钟）

> 回归这件事，最容易被简化成一个误差数。算一个 MSE 交上去，看着挺完整。回归验证其实是六步，每一步都在防一种错。

>

> 第一步，明确预测目标和业务口径。预测的是金额还是时长，含不含税，按哪个时间点算。口径不清，后面所有误差都没有意义。

>

> 第二步，划分数据集。训练、验证、测试三份。切分方式要合理，时序数据按时间切，别随机切。

>

> 第三步，选评价指标。MSE、MAE、可释方差、R2 里挑，挑的标准是业务最怕什么样的误差。怕大偏差用 MSE，要业务听得懂用 MAE。指标定完就别再动，动了后面的结论都没法比。

>

> 第四步，训练和调参。调参只用验证集。测试集只能开一次，开多了等于拿测试集训练。

>

> 第五步，用测试集评估，再做残差分析。残差分析是回归特有的动作：看误差是不是随机分布。误差要是系统性偏移，比如大额样本总被低估，说明模型结构有问题。光看一个 MSE 数字，看不出来。

>

> 第六步，跟基线比，再决定上不上线。基线可以是均值预测，也可以是上一版模型。R2 等于 0.3 到底算好算坏，取决于基线在哪儿。

>

> 这六步里，前两步定的是测什么，后四步定的是怎么判。顺序颠倒，后面全是白算。

>

> 项目里还有一个动作常被省：把误差按样本分组看。大额总被低估、某个地区全偏高，这类规律只有分组才看得见。

>

> 六步收一句：回归验证不是算一个误差数，是证明误差小得有意义、错得没规律。

提问： R2 等于 0.3 的回归模型，算好还是不好？

> **预设回答：**

- 答"不好，0.3 太低"——不一定，要看任务难度和基线。

- 答"要看基线，基线是 0.1 就还行"——思路对，指标得相对基线看。

- 答"要看业务用不用得上"——也很关键。有些场景 0.3 的解释力，拿来做排序已经够。

- 收口：回归报告里必须有"本版对基线"这一栏，没有对比就没有结论。

> **易错点 / 考点：**六步的顺序和每步防的错；残差分析是回归特有、也最容易被漏的一步；"测试集只能开一次"是纪律性考点；判断题陷阱"R2 越高模型一定越好"——可能过拟合，不同任务之间也横向比不了。

> **过渡：**指标只管效果，AI 系统还要过几道专项关。

【14.34】AI 算法的专项测试（2 分钟）

AI 系统上线前，除了功能，还得过六道关。这六项正好对应第 7 章讲过的非功能专项，只是对象换成了 AI。

>

> 数据隐私。训练数据里有没有不允许出现的个人信息，模型会不会把训练数据背出来。测法是数据脱敏检查、成员推断测试、输出内容审计。

>

> 预测效率。一次推理多久、吞吐多少、显存占多少。压测、批量与单条对比、长尾输入的耗时分布都要看。AI 的耗时对输入长度极敏感，测的时候要把极端长度覆盖到。

>

> 可靠性。输入被扰动，输出还稳不稳。下一页那张熊猫图就是典型。测法是对抗样本、噪声注入、重复推理一致性。

>

> 公平性。模型对不同群体，比如性别、地域、年龄，表现有没有系统性差异。分组比指标，看各组召回率、误判率差得远不远。这一项最容易被忽略，也最容易引发舆情。

>

> 可解释性。模型给不给得出人听得懂的理由。看特征重要性和业务常识是否一致，看局部解释合不合理，比

> 如"因为 A 特征高所以拒贷"讲不讲得通。

>

> 安全性。会不会被恶意输入打穿：对抗样本、提示注入、数据投毒。红队测试、异常输入轰炸都用得上。

>

> 六项有一个共同点。它们问的都不是功能对不对，而是这个 AI 值不值得信任。

>

> 六项里，数据隐私和安全性最容易在验收前才被想起来。它们出问题不是指标掉，是事故上报。

>

> 公平性和可解释性更像慢性病。指标看着都在，报告里就是少一段，出事以后补不回来。

> **提问：**六个专项里，哪一个在项目里最容易被跳过？后果是什么？

> **预设回答：**

- 答"公平性"——很常见。很多团队觉得"数据是客观的"。后果是上线后被指出对某群体系统性不利，整改成本极高。

- 答"可解释性"——也常被跳过，在效果优先的团队更常见。出了问题没法向监管和用户解释，金融、医疗场景直接过不了合规。

- 答"安全性"——被跳过的后果是对抗样本一击即溃，甚至被提示注入诱导做坏事。

- 收口：被跳过的专项往往不是不重要，是不好测。这恰好是测试人员补位的地方。

> **易错点 / 考点：**六项名称要能默写；每项的典型测法至少记一个；简答高频"AI 系统的专项测试包括哪些方面"；判断题陷阱"公平性就是准确率在各组都相同"——不是简单相等，是不存在系统性偏差。

> **【思政落点 4】**：公平性和可解释性最容易被省，因为不好测，也不好写进汇报。可一个模型对着某类人群系统性判错，受损的是一个个具体的人。技术人员的责任，是把这个差异测出来、写进报告、说给业务听。会测是本事，肯把不利于自己的结果摆上桌，是职业底线。

> **过渡：**六项专项靠什么手段验，下一层就是方法。

【14.35】常用的测试方法（2 分钟）

> 有一类测试，它不知道正确答案，照样能判对错。这就是蜕变测试，AI 测试里最重要的方法之一。

>

> 思路很直接：我不知道预期输出是什么，但我知道输入按某种方式变过之后，输出要跟着怎么变。课件给的例子都在这。类别标签乱序，把标签顺序打乱，预测结果不变。属性乱序，把特征列顺序调换，结果不变。增加无信息属性，塞一列随机噪声进去，结果不变。一致重复预测，同样输入多跑几次，结果应当一样。这四条都叫根据公式特点，特定变换数据，结果比较。

>

> 它的价值在于绕开了那个死结：没有预期结果。有 oracle 的时候你比数值，没有 oracle 的时候你比关系。

>

> 剩下四类各有分工。效果测试，用专业评测指标，KS、PSI、精确率、召回率、F1-Score、AUC，对象是模型，判据是指标达不达标。数据测试，对象是数据，看 ETL 质量、特征质量，用 Fuzz 构造异常数据，看分布、diff、性能时长。常规逻辑，就是学了一学期的业务逻辑和性能，接口照测，性能照压。小样本试验和 A/B，样本验证评估，小流量先试，看真实业务指标。

>

> 蜕变测试还有一层好处：不依赖标注。标注贵、标注慢，标注还会错。蜕变关系是人想出来的，成本低得多。

>

> 代价是关系得想对。想歪了，测出来的结论也是歪的，而且比不测更让人放心。

>

> 五类收一句。蜕变测试治没有预期结果，效果测试治好不好。数据测试治原料行不行，常规逻辑治工程对不对。A/B 治上线有没有用。

> **提问：**为什么"把特征列顺序打乱、预测结果不变"能算一种测试？

> **预设回答：**

- 答"因为模型不依赖列的顺序"——正是。顺序一打乱结果就变，说明实现里有隐蔽的位置依赖，这是真 bug。

- 答"这不算测试，没有预期结果"——恰恰相反，这就是蜕变测试的精髓：不必知道预期值，只需知道关系。

- 答"随机打乱多次更可靠"——好补充。单次打乱可能碰巧不变，多次才暴露问题。

- 收口：蜕变测试的公式是输入变换加输出关系断言，不需要 oracle。

> **易错点 / 考点：**蜕变测试的四类变换与原理，是本节最高频考点；五类方法的分工要能对上号；"没有 oracle"是 AI 测试的根本困难，蜕变测试就是对它的解法；判断题陷阱"有了 AI 方法就不用做常规逻辑测试"——错。

> **过渡：**方法讲完，看一个让人印象深的例子。

【14.36】示例：可靠性测试（2 分钟）

> 一张熊猫的照片，加一层人眼看不见的噪点，模型说这是长臂猿。课件把三步摆得很清楚：左边一张干净的熊猫图，被模型正确分类为熊猫；中间是扰动；右边扰动后的图，被识别成长臂猿。

>

> 三句话，信息量很大。第一，模型原本是对的，它认识熊猫，说明不是没学好。第二，扰动是看不见的，人眼看过去还是熊猫，顶多觉得有噪点。第三，结论完全翻转，从熊猫变成长臂猿，置信度往往还很高。

>

> 这说明模型的看和人的看，用的不是同一套特征。人看轮廓、纹理、语义。模型看像素的数值组合。只要在梯度方向上挪一点点精心设计的偏移，高维决策边界就翻了。FGSM 的做法就是沿着损失函数梯度方向加扰动，一步到位。

>

> 对测试的启示三条。第一，准确率高不等于可靠。测试集上 99% 的模型，可能被一张加噪的图骗过去。第二，对抗测试要做成常规项，把对抗样本放进常测集，别等出了事故才想起来。第三，对抗训练是防御手段之一，把对抗样本混进训练数据能提鲁棒性，但要持续更新，攻击方法也在变。

>

> 再延伸一句。这个例子不只对图像成立。语音加一点人耳听不见的噪声，文本改几个不影响语义的字，都可能让模型判错。凡是模型，就有它的对抗样本。

>

> 顺带说清一个常见误会：对抗样本不是随机噪声。随机加噪，模型多半还认得出熊猫。扰动是朝着让模型出错的方向算出来的。

> **提问：**既然人眼看不出扰动，测试怎么发现这类问题？

> **预设回答：**

- 答"用算法自动生成对抗样本"——对。FGSM、PGD 这类方法就是干这个的，能批量生成。

- 答"做人眼对比"——做不到。扰动本来就是照人眼不可见来设计的。

- 答"随机加噪声试试"——效果有限。随机噪声很难命中梯度方向，对抗样本是优化出来的，不是碰出来的。

- 收口：对抗测试必须工具化、自动化，靠人工抽查发现不了。

> **易错点 / 考点：**这个例子的三步要能复述，干净图判对、加扰动、判成另一种；FGSM 的定位是基于梯度的对抗样本生成方法；简答高频"对抗样本说明了什么"，答模型依赖的特征跟人不同，准确率不等于鲁棒性；别把对抗样本跟普通数据噪声混成一件事，前者是恶意构造的。

> **过渡：**被测对象讲完了，接下来把方向掉过来。

【14.37】14.3 AI 助力软件测试（1 分钟）

> 同一个 AI 技术，换个位置，身份就变了。这一节标题六个字：AI 助力软件测试。上一节是 Test for AI，AI 是被测对象。这一节是 AI for Test，AI 是工具。方向一反过来，立场全变。

>

> 五个小节。14.3.1 基于图像识别技术的 UI 测试，让机器看界面。14.3.2 基于 AI 的、全自动化的 API 测试，让机器自己生成和调度接口测试。14.3.3 AI 助力代码深度分析，让机器读代码找问题。14.3.4 AI 驱动测试，从用例生成到缺陷分析的整条链路。14.3.5 AI 测试工具，具体产品。

>

> 这一节的主线一句话：把测试里最机械的部分交给机器，最需要判断的部分留给人。

>

> 听的时候留心一件事。每一种 AI 应用，解决的都是传统自动化测试的一个具体痛点。UI 元素定位太脆，接口用例写不完，日志太多看不过来。痛点在前，AI 在后，这个顺序别弄反。先有工具再找场景，做出来的东西没人用。

>

> 判断一个工具值不值得上，先问它替你省了哪一段重复劳动，再问省下来的时间你打算干什么。第二问答不上来，工具只是换个更贵的方式加班。

>

> 五个小节不是并列的五件事，是一条链。识别让机器看得见。API 自动化让机器跑得动，代码分析让机器读得懂。AI 驱动测试把前几样串成流程，工具是这条链的成品。

>

> 学这一节不用记产品，记链条。先有能看、能跑、能读的能力，才有流程，才有工具。

>

> 课设里这五样最多碰到一两样，知道它站在链条哪一段就够了。这一节的考题多是名词对号，别把方向也背反。

> **提问：**同一个"AI 识别界面元素"的技术，放在 14.2 和 14.3 里，身份有什么不同?

> **预设回答：**

- 答"14.2 里它是被测对象，14.3 里它是工具"——准确，这正是两节的分界。

- 答"没区别，都是 AI"——错答。考试里方向一混，分直接丢。

- 答"14.3 里它帮我们找 bug，14.2 里我们要找它的 bug"——这个说法精炼，可以当口诀。

- 收口：读论文、看工具，先判断 AI 在这篇里是主语还是宾语。

> **易错点 / 考点：**两节方向必须分清，这是本章第一号易错点；五小节名称要能列举；常考简答"AI 在软件测试中的典型应用有哪些"，按 UI、API、代码分析、测试驱动、工具五个方面答；口诀是 Test for AI 找它的 bug，AI for Test 用它找 bug。

> **过渡：**先看全景，智能化的野心铺得比想象中宽。

【14.38】测试的智能化（2 分钟）

> Test smarter, not harder，这句话印在很多工具的首页上。意思不复杂：更聪明地测，不是更拼命地测。

>

> 课件给了一张智能化清单，按测试流程的顺序重排，它的意图就很清楚。智能的单元测试，自动生成单测、自动补 mock、自动找边界。智能的 UI 和 API 测试，用图像识别定位元素，用接口描述自动生成用例。AI 驱动的脚本生成，把自然语言描述或者操作录制转成可执行脚本。优化或补充测试用例，分析已有用例的覆盖和冗余，推荐还缺哪些。质量风险评估的自我调整，按历史缺陷分布和代码改动热度，动态挪测试重点。缺陷诊断机器人 DDB，自动分析失败原因。测试环境的智能运维，自动拉起环境、自动修依赖。

>

> 看这份清单能看出来，智能化的目标不是替掉某一个环节，是把整条链路每个环节都加一层智能。

>

> 课件还列了几个工具名：Applitools、Appvance IQ、Eggplant AI、Test.AI、Functionize。名字不用背，方向要清楚。Applitools 主打视觉比对，Appvance IQ 主打无脚本自动化。Eggplant AI 主打基于模型的测试，Test.AI 主打从用户行为里学。Functionize 主打端到端的 AI 测试。

>

> 为什么这些环节能交给机器？因为它们都有明确判据。单测的判据是断言，定位的判据是图，脚本生成的判据是能跑通。

>

> 判据说不清的环节，工具一上就露馅：它给你的东西你还得从头判一遍，省下的时间又还回去了。

>

> 还有一句得说在前面。这些工具解决的是降低门槛、提高覆盖，不解决测什么这个问题。选题、定标准、判风险，还是人的活。

> **提问：**如果工具能自动生成用例、自动定位缺陷，测试人员还剩下什么工作？

> **预设回答：**

- 答"什么都不剩了"——最需要纠正的判断。工具不知道这个版本的业务重点在哪儿，也不知道哪个风险冒不得。

- 答"剩判断：测什么、测到什么程度、上不上得了线"——准确。这就是测试设计和质量决策两块。

- 答"剩跟业务沟通"——也对。需求澄清、验收标准谈判，工具替代不了。

- 收口：能力模型要往懂业务、会设计、能判断上走，不是往会点工具上走。

> **易错点 / 考点：**七类智能化应用要能列举；口号 Test smarter, not harder 和它的中文意思要记住；常考判断"AI 能自动生成用例，所以测试设计不重要了"——错，生成的前提是有人定义目标和判据；工具名不必全记，方向要清楚。

> **过渡：**逐个看案例，第一个是最朴素的一类。

【14.39】AI 应案例：SikuliX（2 分钟）

> 界面上的按钮，除了 DOM，还能怎么找？靠眼睛看。SikuliX 走的就是这条路，很多人第一次接触"AI 帮测试"就是从它开始。

>

> 先说它治什么病。传统 UI 自动化靠 DOM 结构、XPath、控件 ID 定位元素。这套办法有三个死穴。一是脆，开发改个 class 名、包了一层 div，脚本全废。二是覆盖不到，有些界面根本没有可用的 DOM。Canvas 绘制的界面、远程桌面、老式桌面客户端、游戏画面，都在这一类。三是跨技术栈难，Web、桌面、移动三套定位方式各不相同。

>

> SikuliX 把问题反过来。人怎么用界面？用眼睛看。那就让脚本也用眼睛看。你把界面截图喂给它，它在一屏里找到这张图的坐标，然后点下去。定位依据从代码结构换成视觉效果，上面三个死穴一次解决：不怕改 class，不怕没 DOM，跨技术栈也能用。

>

> 课件还提了两件事。项目挂在 GitHub 上，仓库主是 RaiMan。它的 API 完全实现为由目标机器上运行的服务支撑的 REST-API。意思是做成了服务化形态，测试机和被测机可以分开。后面讲测试服务化时，这条会接上。还有 SikuliX1 演进到 SikuliNG，说明这条路线一直在维护。

>

> 局限也在这儿。图像匹配怕分辨率变化、怕主题换色、怕字体渲染差异，所以工程上要控住运行环境的一致性。

>

> 还有个实际问题：截图从哪来。分辨率变了、系统主题换了，图库就得重做。所以用它之前先定一件事：跑脚本的机器把分辨率、缩放比例、字体全锁死。

>

> 锁不住，图像方案会变成每天早上的第一件麻烦事。这跟后面讲界面基线是同一个道理。

> **提问：**图像识别定位看起来这么万能，为什么没有完全取代 DOM 定位？

> **预设回答：**

- 答"因为识别会失败，DOM 更精确"——核心原因。DOM 定位是确定性的，图像匹配是概率性的，要设阈值，可能误匹配。

- 答"因为慢"——也是实情。每次截图匹配都比查 DOM 慢。

- 答"因为环境差异，分辨率、缩放、主题"——很到位，这是图像方案落地时最大的坎。

- 收口：工程上是互补。能用 DOM 用 DOM，DOM 不可靠时用图像，选型看场景不看时髦。

> **易错点 / 考点：**SikuliX 的定位原理是截图匹配加坐标点击；它最合适的场景是无 DOM 可依，Canvas、桌面客户端、游戏、远程桌面；"REST-API 服务化"这个点可以拿去答"测试工具形态演进"；判断题陷阱"图像识别定位一定比 DOM 定位好"——错，概率性方法有误匹配风险。

> **过渡：**认出一张图之外，机器还能认出一类东西。

【14.40】AI 应用案例：目标检测技术（1 分钟）

> 图像匹配找的是一张特定的图，目标检测找的是一类东西。两者的差别，就这一句。

>

> 图像匹配，你给它一张按钮截图，它去找这一张。目标检测，你训练它认识按钮、输入框、图标这些类别。它就能在任何界面上把所有按钮框出来，还给位置和置信度。

>

> 这个能力对测试有三点用处。第一，脚本不再依赖那张截图。按钮换了颜色、换了文案、换了尺寸，只要它还是按钮，目标检测照样认得出，这叫泛化，图像匹配做不到。第二，可以做界面结构分析。一屏元素都框出来，就能自动判断这页有几个可点区域、布局有没有错位、有没有元素被遮挡。人做很费眼，机器做很快。第三，可以做视觉回归。版本迭代后自动比对两版界面的元素分布，找出多出来的、少掉的、挪了位置的。

>

> 它还能给出元素的位置和大小，位置本身就能当断言。按钮跑到屏幕外、两个元素叠在一起，这类错在 DOM 层看着都正常。所以它补的不是识别能力，是一类原来没法自动判的缺陷。

>

> 代价也得说清。目标检测要训练数据，得先标注大量界面截图，投入不小。识别还带置信度，低置信度的结果要人复核。所以工程上通常这样用：高置信度自动执行，低置信度转人工。

>

> 这也是人机协同的典型形态，模型给概率，人给底线。阈值定在哪儿不是模型说了算，是这一步操作的错误代价说了算。

提问：目标检测识别出一个按钮，置信度只有 0.4，脚本要不要点？

预设回答：

- 答"要点，能识别出来就点"——风险高。误点可能触发不可逆操作，比如提交订单、删除数据。

- 答"不点，低于阈值就报错转人工"——标准做法。设阈值，低置信度不自动执行。

- 答"看这个按钮是干什么的"——很专业。操作的危险等级决定阈值高低，删除类操作阈值要设得极高。

- 收口：AI 辅助测试落不落地，关键往往不是模型多准，是阈值和兜底策略设计得好不好。

易错点 / 考点：图像匹配与目标检测的区别是特定对类别；三个典型用途要能列举；"置信度阈值加人工兜底"是人机协同的标准答案；判断题陷阱"识别准确率 95% 就能全自动跑"——还要看错误代价和兜底机制。

过渡：把这类识别能力和游戏 AI 打包，就有现成的开源工具包。

【14.41】AI 驱动的智能测试（2 分钟）

> 手机游戏让你点一百下，脚本能撑住几次？先别答。这类系统的麻烦不在点，在"点哪里"。

>

> 游戏界面没有 DOM。原生 App 能读到控件 ID，网页能读到节点，游戏只有一帧一帧的像素。选择器找不到任何东西。腾讯的 GameAISDK 就是从这个坑里长出来的开源工具包。课件给的定义是：基于游戏图像来开发游戏 AI 的开源工具包。包括 UI 检测、UI 元素识别、AI 算法等功能。

>

> 它能分成三层。第一层，UI 检测与元素识别，从画面里找出按钮、血条、对话框、道具。没有这一层，后面全做不了，因为机器连"那里有个东西"都不知道。上一节的目标检测讲的就是这件事。

>

> 第二层，AI 算法。工具包里带着训练和推理的能力，你可以拿自己的截图训一个识别模型。工具给你结果，工具包给你造结果的能力，差别就在这儿。

>

> 第三层，面向游戏的测试与评估。游戏测试有三难。状态空间巨大，一局的可能走法近乎无穷。判定标准主观，好不好玩写不成断言。复现困难，同一串操作两次结果可能不一样。三难压在一起，人写用例是写不完的，只能让 bot 自己去玩，玩出人想不到的路径，玩到崩溃和卡死。

>

> 这里接住一个常见的错觉：AI 测出来的问题，多半不是功能错了，是某个极端操作序列下崩了。这种缺陷传统用例永远碰不到，只能靠大规模随机探索加异常检测。

>

> 所以这一层的结论可以带走。界面不可编程，状态空间又巨大，这时候只剩一条路：让 AI 去操作，让 AI 去探索。移动端里的游戏化页面、车载中控、IoT 面板，都长得像这一类。

> **提问：**为什么游戏测试特别依赖机器人自己去玩，而不是录脚本回放？

> **预设回答：**

- 答"游戏状态太多，脚本覆盖不全"——核心原因。录脚本只能走一条路，游戏的可能路径是爆炸的。

- 答"脚本一改版本就废"——实情。游戏更新勤，录制的脚本很脆。

- 答"要模拟真人的操作节奏"——也对。bot 能换各种节奏去点，时序类的缺陷才露得出来。

- 收口：这类系统的打法是随机探索加异常检测，覆盖靠随机，判定靠监控。

> **易错点 / 考点：**三层能力要能列举；游戏测试三难（状态空间、标准主观、复现困难）是简答素材；"界面不可编程就让 AI 去探索"能迁到 App、IoT、车载；陷阱题"AI 探索能替代人工试玩"——体验类判断仍归人。

> **过渡：**游戏画面里还有一样更常见的东西，文字。

【14.42】AI 应用案例：OCR（2 分钟）

> 活动页、支付页、客服页，一个 App 里一半的界面是嵌进去的网页。你的脚本偏在"确认支付"这四个字上卡住，报元素找不到。

>

> 这不是你选择器写错。App 里嵌的 H5 跑在 webview 里，自动化工具拿不到里面元素的属性。原生控件的 ID、层级、位置都能读，webview 的内容对工具来说是一整块黑盒。你看得见，它读不到。课件那句话就是这么来的。OCR 文字识别能力可以很好地帮助解决 webview 页面元素无法识别的问题。

>

> 解法很直接。读不到属性，就去读像素上的字。把 webview 那块截下来做文字识别，拿到"这里写着确认支付"，再按文字的位置去点。定位的依据从控件属性换成了屏幕上的字。这和 SikuliX 是同一个思路，只是把图像匹配换成文字识别，稳一些，通用一些。

>

> 顺手还带来三件事。跨技术栈，原生、webview、小程序、Flutter 画出来的界面，只要有字就能认。内容断言，不只是点，还能验页面上有没有出现"支付成功"。多语言适配，同一套脚本换个语言包照样跑。

>

> 局限也要说清。识别率受字体、字号、抗锯齿、背景干扰影响。手写体和艺术字容易错。纯图标按钮上没有字，这条路直接断。

>

> 所以工程上常见的做法是三层兜底。能读属性就读属性，读不到上 OCR，OCR 也不行再退回图像匹配或目标检测。别指望一层打通所有界面。

> **提问：**一个纯图标按钮，只有图形没有文字，OCR 能定位吗？

> **预设回答：**

- 答"不能，没有字可认"——判断对。

- 答"退回图像匹配，或者用目标检测认这个图标"——标准兜底，就是刚才那三层。

- 答"让开发补个无障碍标签"——很专业。推动被测系统变得可测，本来就是测试人员的活。

- 收口：好的方案不只挑技术，还会反过来要求产品留出可测的口子。

> **易错点 / 考点：**核心问题就是那句原文，webview 元素无法识别，常直接考；三种定位手段的优先次序，属性、OCR、图像；可测试性设计是加分概念；陷阱"OCR 只用来认验证码"——界面定位、内容断言都在用。

> **过渡：**文字之后是语言，人话怎么变成用例。

【14.43】AI 应用案例：NLP（2 分钟）

> 同一件事，五个人能写出五种说法。"额度超限""超过授信上限""额度不够了"，人知道是一回事，机器不认。

>

> NLP 转用例这件事，卡点从头到尾都不在模型。课件给了四个动作，顺序不能换。

>

> 第一步，约定测试用例描述规范。前置条件、操作步骤、预期结果，用相对固定的句式写。为什么放在第一步？因为输入乱，抽取就不稳。同一句话三种写法，模型每次抽出来的字段都不一样，后面全要人工校正。行业里的做法是先定模板，再上模型。

>

> 第二步，建立专业知识图谱。用例里全是业务名词，授信额度、逾期天数、T+1 核对。模型不认识这些词，就读不懂意图，只能按字面猜。知识图谱把概念、关系、同义词都摆出来。机器才知道"额度超限"和"超过授信上限"是一件事。

>

> 第三步，与用例管理系统打通。生成的东西不能躺在某个脚本目录里。它要回到团队天天在用的系统，Jira、TestLink、PingCode 都行。人不进那个系统，工具等于没做。

>

> 第四步，一键批量转化。选中一批需求文档或文字用例，一次转成结构化用例或可执行脚本。这一步管的是体验，前面三步管的是准不准。

>

> 四个动作连起来是规范、知识、集成、转化。给你一个判断：准确率八成取决于前三步，跟你选多大的模型关系不大。很多团队搞反了顺序，才觉得 AI 不好用。

> **提问：**为什么约定用例描述规范要放在上 NLP 之前，而不是之后?

> **预设回答：**

- 答"机器需要稳定的输入"——对。输入的规范性决定抽取的稳定性，数据先决定上限。

- 答"先有规范才能批量"——也对。没规范，每条都要人工校正，批量就没意义。

- 答"规范太死会限制表达"——真实顾虑。折中是关键字段结构化，描述部分放开。

- 收口：这件事的实质是流程改造，不是模型选型。

> **易错点 / 考点：**四个动作按顺序能列举；先规范、后模型是核心观点，简答高频；知识图谱解决的是业务术语理解；陷阱"上了 NLP 就能自动生成全部用例"——生成的是草稿，评审和补充仍是人的活。

> **过渡：**再换一种 AI，这回不是识别，是进化。

【14.44】AI 应用案例：神经元演化算法（2 分钟）

> 三个算法名字摆在这儿，没一个是测试行业自己起的。它们凑在一起，是要造一个会自己玩的机器人。

>

> 课件原文写的是三样算法：Monte Carlo 树搜索、自动启发式构建、增强拓扑的神经元演化。用它们来训练 AI 测试工具（bots），模拟人类交互能力，完成对游戏程序的测试或评估。署名 Alexander

> Andelkovic，King／Midasplayer AB。就是做 Candy Crush 的那家。

>

> 三个名字翻成人话。Monte Carlo 树搜索，边试边算。不把可能的路走完，随机采样一批，再按结果把力气挪到有希望的分支上，围棋 AI 就这么下。自动启发式构建，自己总结经验规则。人写规则是"我觉得先点这里"，它让算法从数据里长出这类偏好。NEAT，增强拓扑的神经元演化，不光调参数还改结构。普通网络的层和节点是人定的，它让网络自己增删节点和连接，边进化边变复杂。

>

> 六步是这样的。获得训练数据，预测成绩。创建随机的机器人。竞争测试，算成绩。创建新的机器人。按 ANN 选出最好的机器人。再按 ANN 选择移动，继续操作游戏。

>

> 顺着看，这是一个进化循环。随机生一批，让它们互相打，按成绩留下好的，变异出新的。再用神经网络把好 bot 的打法学下来，让它接着玩。玩得越久，数据越多，模型越准。

>

> 最后从多个 bot 里挑出最合适的那一个。为什么要有多个？不同关卡、不同游戏类型，最优策略不是同一个。

>

> 这套东西能用的前提得说清：正确的操作序列人定义不出来，但结果能打分。有分数，才谈得上进化。

提问：这套进化出 bot 的办法，能直接搬到 Web 功能测试上吗？

预设回答：

- 答"能，让 AI 自己点页面"——有限可行。Web 功能测试有明确的业务预期，随机探索效率很低，还会点出一堆无意义操作。

- 答"不能，Web 有明确判据"——说得对。进化搜索适合没有明确操作路径、只有结果评分的场景。

- 答"可以用在探索式测试里"——迁移得专业。把覆盖率、崩溃数当成绩，让 bot 去撞。

- 收口：迁移的关键不在技术，先问这个场景有没有可算的成绩。

易错点 / 考点：三个算法名称与各自作用；六步顺序（竞争测试、选优、学习、继续操作）；适用前提是能算出成绩，路径又定义不出来；陷阱"进化算法可以替代测试设计"——它替代的是探索。

> **过渡：**跑完一堆用例，最烦的是红了三百条，从哪儿看起。

【14.45】AI 应用案例：失败用例分析（2 分钟）

> 一晚上三千条用例，红了三百条。你打开报告从第一条往下看，看到第二十条，已经忘了自己在看什么。

>

> 三百条红，真实原因可能只有七个。九条用例挂在同一个空指针上，报错文本一模一样，可你得一条条点开才知道。时间全花在检索上，不是花在判断上。

>

> 课件给了两条线：离线分析和在线分析。

>

> 离线先做粗清理。拿用例全量执行日志和报错日志，把失败相同的用例聚成一类，三百条变成七堆。再做精细清理，对每一堆用智能化算法辅助分析，产出日志聚类信息。

>

> 在线是另一条，靠失败用例分析服务和 JavaAgent，做到单用例级。它干的活是建立用例级多源数据画像。执行环境、用例执行日志、业务日志、用例脚本，全收进来。用例覆盖代码、调用链日志、业务配置、流水线部署参数，也收进来。还要分清是服务自身的问题，还是它依赖的系统出的问题。

>

> 画像建好，给的是失败原因推荐，可疑点一到四，带排序。它给的是可疑点，不是结论。

>

> 这个设计挺诚实。失败的原因常常是环境、数据、代码搅在一起，机器给一个确定答案，八成是错的，还会把你带偏。所以分工是：机器缩小范围，人下结论。人点一次"是这个原因"，反馈回灌，下次更准。

>

> 价值一句话：把三百条红变成七个原因，每条带可疑点排序。测试工程师的时间要花在判断这是不是真 bug 上，不是花在这条为什么红上。

提问：为什么系统给的是可疑点一到四，而不是直接给根因？

预设回答：

- 答"因为不准，怕误导"——实在。原因常常是环境、数据、代码搅在一起，给唯一答案反而危险。

- 答"因为最后要人来判"——正是。机器缩小范围，人下结论，这就是人机协同的边界。

- 答"模型够强就能给根因"——过于乐观。失败原因的分布拖着很长的尾，模型覆盖不住。

- 收口：这类工具的设计是给排序、不给判决。

易错点 / 考点：离线做聚类、在线做画像与推荐，两条线的分工；用例级多源数据画像有哪些数据源，简答高频；粗清理与精细清理的区别；陷阱"AI 能直接定位根因"——给的是可疑点排序，结论人定。

【思政落点 5】：〔机器只给可疑点、不给判决，这个分寸值得停一下。它牵着 14.3 里人机协同的边界：模型缩小范围，人下结论，最后一下判断压在测试工程师身上。落到价值观上，是实事求是——不把工具的排序当成定论，也不为了交差把四条可疑点原样抄进缺陷单。敢说这条红是环境脏了，也敢说这是真缺陷，这份判断力才是岗位的分量。〕

过渡：最后一个 AI 案例，落在接口的精准调度上。

【14.46】基于形式化语言的 API 智能测试（2 分钟）

接口测试最费时间的不是写用例，是选用例。几百个接口，改了三行代码，全量跑一遍四十分钟。

>

课件这张图是一条数据流，从左到右：输入、加工、两个库、执行，再回灌。

>

输入有五样：问题反馈、执行日志、在线数据、交付场景、测试数据。里面最值钱的是前两样。问题反馈和执行日志是真实世界给你的信息，比任何人坐在会议室里设计出来的东西都贴近实际。

>

加工走两条线。一条是特征数据清洗，用特征提取规则库和数据清洗模型库清洗，再去量纲化。不同量纲的指标放不到一起比，响应时间按毫秒、成功率按百分比，不归一就没法排序。另一条是模型参数训练，拿数

> 据样本集做真实数据学习，加上强化学习和加速学习，然后周期性同步。周期性同步这四个字别漏，模型不是训一次就完，线上在变，它得跟着更新。

>

> 产出两个库。测试脚本库里有四项。测试覆盖度组合、API 参数关联分析、变更-测试脚本关联关系、测试相似度聚类。另一个是测试模型库。四项里"变更-测试脚本关联关系"最要紧，它回答的是代码改了哪一行，要跑哪些脚本。

>

> 执行侧接上四步。测试脚本筛选和测试模式选择，精准测试任务调度和执行，测试管理平台，测试数据采集与分析。采集分析的结果再回灌到输入，闭环就合上了。

>

> 收一句：精准测试是建立变更和用例之间的映射，AI 只是把这个映射做得更自动、更准。

> **提问：**变更-测试脚本关联关系，为什么是精准测试的核心?

> **预设回答：**

- 答"能只跑相关的用例，省时间"——直接价值。

- 答"全量回归跑不完"——现实约束。微服务下改一处，牵动的服务可能几十个。

- 答"怕漏测"——担心也对。映射不全就会漏，所以要定期用全量回归校验映射的完整性。

- 收口：精准测试一直有一对矛盾，省时间和怕漏测，靠映射质量加定期全量兜底来平衡。

> **易错点 / 考点：**数据流主线要能复述，输入、清洗与训练、两库、执行、回灌；变更-脚本关联是精准测试核心，必考；去量纲化的作用要能解释；陷阱"精准测试可以取消全量回归"——要用定期全量兜底校验映射完整性。

> **过渡：**AI 帮测试的案例到这儿，接下来换角度看工具本身。

【14.47】低代码测试平台（1 分钟）

> 自动化项目第一年很漂亮，第二年没人管，第三年全废。见过不止一次。

>

> 不是技术选错。写一套 UI 自动化，要会一门语言、一个框架、一套定位方式、一套断言库，还要会搭环境、会造数据。门槛摆在这儿，结果是能做的人少，能长期维护的人更少。维护成本把人熬走，脚本就烂在仓库里。招人也难，简历上写会自动化的不少，能接手别人脚本的不多。

>

> 低代码要解决的就是这个。它低的是写代码的量，不是能力。常用操作封装成可视化的组件：打开页面、点击、输入、断言、循环、条件。定位方式做成配置，不写成脚本。数据驱动做成表格，不写循环。

>

> 三个变化跟着来。参与面变宽，业务测试人员自己就能搭，不用排队等开发支援。维护成本降下来，改一个步骤改配置，不用改代码再回归一遍。标准统一，大家用同一套组件，脚本长得像，交接时不必重读别人的风格。

>

> 代价也得说。平台能力有边界。复杂的前置数据构造、非标准协议、特殊加密，绕不过去还是得写代码。平台本身还成了新的依赖，它升级、它有 bug、它把你锁住，都是新风险。

>

> 所以边界很清楚：大量、标准、重复的场景交给它；少量、特殊、复杂的场景，别指望它。组里人少的时候，它省下的往往不是写脚本的时间，是交接的时间。

提问：低代码让不会编程的人也能做自动化，是好事还是有隐患？

预设回答：

- 答"好事，参与的人多了"——是好事，但要配套组件规范和用例规范，否则会造出一堆没人维护的脚本。

- 答"有隐患，脚本质量参差"——真实风险。断言写得随意，会出现跑绿了但什么都没验到。

- 答"看平台有没有强制规范"——到位。好平台会强制写断言、强制命名，把规范做进工具里。

- 收口：选低代码平台，先看断言能力，再看复用和分层，最后才看拖拽顺不顺手。

易错点 / 考点：低代码低的是编码量，不是能力；三类收益与两类风险能列举；简答高频"低代码平台适合什么场景"；陷阱"低代码能完全替代编码自动化"——复杂场景仍要代码。

> **过渡：**低代码最典型的一手，是把用例画成一张图。

【14.48】低代码测试平台- 续（2 分钟）

> 需求文档里那句"用户可以修改订单"，你能直接写成用例吗？多半写不下去。

>

> 课件给了低代码平台的两个能力，两句话都很实。

>

> 第一句：为测试用例创建一个模型，直观地展示测试执行的路径，适合设计复杂的测试场景。这是用例建模。不是一条条写步骤，是画一张图：节点是操作，边是流转，条件分支画成岔路。为什么复杂场景特别需要图？因为难点从来不在执行，在想清楚。有几种分支，异常路径怎么走，前置状态有几种。写成文字，这些关系藏在段落里；画成图，漏掉的那条岔路自己会露出来。这和第 2 章的因果图、第 3 章的判定表是同一路做法，用图形管住组合。

>

> 第二句：在需求分析阶段就创建测试步骤，有助于团队内部沟通澄清需求。这句的分量比第一句还重，它把测试的介入点挪到了需求阶段。你想，把"修改订单"拆成步骤，当场就会冒出问题。改到什么时候就不能改了？改了要不要重新计价？已经支付的呢？这些问题在需求评审上提，成本是一句话；等测到一半才发现，成本是一次返工。

>

> 所以低代码平台的定位，不是省事的脚本工具，是让需求变得可以讨论的建模工具。

>

> 再补一句工程上的好处：模型能复用。同一张模型换一套数据，能跑多种场景。这也给下一节的 MBT 留了个话头。

> **提问：**在需求阶段就把测试步骤画出来，为什么能减少返工？

> **预设回答：**

- 答"问题发现得早"——对。同一个问题，越早发现越便宜。

- 答"能逼需求写清楚"——更准。画步骤的过程，就是把模糊需求逼成明确规则的过程。

- 答"开发能照着步骤实现"——也对。步骤图常常变成验收依据，双方理解对齐。

- 收口：测试在需求阶段最值钱的产出不是用例，是问出来的那些问题。

> **易错点 / 考点：**两个能力要能列举；测试左移在这里的具体体现，论述题好素材；用例建模与因果图、判定表同源，可做知识串联；陷阱"低代码只给不会编程的人用"——建模能力对复杂场景价值更大。

> **过渡：**再往极简走一步，连低代码都不用了，叫零代码。

【14.49】零代码智能测试工具 APIAuto（1 分钟）

> 一份 Swagger 文档丢进去，脚本自己出来了。第一次看到这个场面的人，多半会愣一下。

>

> 零代码和低代码差在哪？低代码是少量配置加少量代码，零代码是不写代码，只填信息。APIAuto 这类工具的路子是这样。给它接口文档，Swagger、OpenAPI 都行，一份写得规范的接口说明也行。它自动生成请求，自动推断参数取值，自动校验响应结构，然后批量跑。

>

> 为什么这条路在接口上走得通，在界面上走不通？因为接口有契约。参数类型、必填项、取值范围、响应结构，文档里白纸黑字写着。有契约，就能推导。界面没有契约，只有像素和坐标，推不出来。

>

> 这句可以往外推：测试自动化的容易程度，取决于被测对象的形式化程度。形式化得越清楚，机器能替你做的越多。

>

> 能力边界也要摆明。结构性问题它能自动发现：响应字段缺了、类型不对、状态码异常、必填参数不传会怎样。业务性问题它发现不了：这笔金额算得对不对、这个状态流转合不合理。前者靠契约，后者靠人。

>

> 还有一个顺带的价值：契约校验。接口实现和文档不一致，是微服务里最常见的坑之一。这类工具天天跑，等于天天在比对文档和实现。

提问：零代码工具为什么在接口测试上比在 UI 测试上更容易落地？

> **预设回答：**

- 答"接口有文档、有契约"——核心答案。形式化程度决定自动化程度。

- 答"UI 变化太快"——也成立。界面改版频繁，脚本维护成本高。

- 答"接口测试更重要"——不能这么讲。两者都重要，只是自动化难度不同。

- 收口：判断一个场景能不能自动化，先问它的契约写得清不清楚。

> **易错点 / 考点：**零代码与低代码的区别；"形式化程度决定自动化程度"这句分量重，论述题能用；能力边界，结构问题管、业务问题不管；陷阱"零代码生成的用例不用人审"——结构对了不等于业务对了。

> **过渡：**工具形态之外还有更根本的一变，把测试的依据从用例换成模型。

【14.50】MBT+AI 合力相助测试（1 分钟）

> 人想得出二十条路径，工具能把两百条都走一遍。同一个系统，两种覆盖，差在哪里？

>

> 差在依据。传统测试的依据是用例，人想出一条条场景，写成步骤。MBT 的依据是模型，全称 Model-Based Testing。把系统行为画成状态机、流程图或者更形式化的规约，再由工具从模型里生成测试路径。

>

> 你在上一节见过它的雏形。那句"为测试用例创建一个模型"，就是 MBT 的起手式。

>

> 换依据换来三件事。生成能力强，你想到二十条，工具能把模型里所有合法路径走一遍，包括你没留意的组合。可维护，需求变了改模型，路径重新生成，不用手改一堆用例。可验证，模型本身能被检查：有没有走不到的状态，有没有死锁，有没有永远出不去的循环。

>

> 那 AI 进来干什么？解决 MBT 最疼的地方，建模太贵。手画状态机，几百个状态能画到崩溃。AI 能做三件事。从需求文档、日志、真实操作记录里自动抽取模型。补全缺失的迁移。按风险给路径排序，先跑最可能出问题的那几条。

>

> 分工就清楚了：MBT 给严谨，AI 给建模的效率。

>

> 换个说法收：MBT 把测试从经验驱动改成模型驱动，AI 让这个转变变得付得起。

> **提问：** MBT 听起来处处都好，为什么现实中用得不多？

> **预设回答：**

- 答"建模成本太高"——头号原因，也正是 AI 要解决的问题。

- 答"模型和代码容易脱节"——真实痛点。模型改不及时，生成的路径就过期了。

- 答"团队不会画模型"——也对。状态机、形式化方法有门槛，门槛在人身上。

- 收口：所以 MBT 落地的前提是模型能自动或半自动维护。

> **易错点 / 考点：** MBT 全称与核心思想，依据从用例换成模型；三个好处与 AI 解决的三件事要对得上；简答高频"MBT 的优缺点"；陷阱"MBT 生成的用例一定比人工好"——模型不全，路径也不全。

> **过渡：** MBT 加 AI 到底能干什么，下一页给一个具体例子。

【14.51】MBT+AI 合力相助测试 – 续（2 分钟）

> 登录、搜索、加购、下单、支付。一条 E2E 用例，五个步骤，四个依赖。

>

> 没登录就不能下单，没加购就不能支付。这些关系人梳理得出来，但一条条写清楚很费劲，而且一定会漏。课件给的做法分四步。

>

> 第一步，用 NLP 做 E2E Web 测试的依赖项检测。从测试脚本、页面文本、接口描述里自动抽出谁依赖谁。

> 脚本里点了什么、页面上出现了什么字样、接口返回了什么，都是线索。

>

> 第二步，验证这些依赖关系。抽出来的不一定真，可能是巧合，也可能随版本变掉。这一步是去伪。

>

> 第三步，用一个恢复算法，确保最终的测试依赖关系图里没有漏掉一条真实的依赖。这一步最见功夫。恢复算法的做法是：从已知的依赖出发，反推可能被漏掉的，再回头确认。为什么绕这一圈？因为漏一条依赖的后果不是报错，是看起来跑了、实际没验到。

>

> 举个具体的。支付用例依赖登录态，脚本里没写。测试环境里登录态还留着，用例天天绿。某天环境清干净了它就红了；更糟的是它带着别人的会话跑通，你以为验了支付，结果什么都没验。

>

> 第四步，落到提高回归测试覆盖率。依赖图清楚，就知道改 A 会牵动 B、C、D，要回归哪几条也就定了。这和前面讲的变更与脚本关联是同一个目标。

> **提问：** E2E 用例漏了依赖，为什么会出现假通过这种最坏的结果？

> **预设回答：**

- 答"前置条件碰巧满足了"——正是。测试环境里登录态还留着，用例就过了，换个干净环境立刻红。

- 答"它没真的验到业务"——对。跳过了要验的步骤，等于拿一条空用例充数。

- 答"环境是共享的"——重要诱因。共享环境会盖住依赖问题，所以要做环境隔离和前置清理。

- 收口：E2E 用例的第一条纪律，是每条都能在干净环境里独立跑通。

> **易错点 / 考点：** 四个动作按顺序列举；漏依赖导致假通过，是案例题的核心；恢复算法的作用，反推补全依赖图；陷阱"E2E 依赖越多越好"——依赖越多越脆，要显式声明并隔离。

> **过渡：** 14.3 讲 AI 怎么帮测试，14.4 讲工具往哪儿变。

【14.52】14.4 软件测试工具的未来（1 分钟）

过去二十年，测试工具一直在做同一件事：把脚本写得更强。录制更聪明，断言更灵活，协议支持更多。

>

> 这条路走到头了。脚本越强，会用的人越少；会用的人越少，维护成本越集中在那几个人身上。大量团队不是不想自动化，是养不起。

>

> 所以方向变了，从让人写更强的脚本，转成让人少写甚至不写，但覆盖更多。Codeless 测试自动化就是这条线上的产物。这个词容易误解。无代码不等于没有代码，代码被挪进平台内部，用的人看不见。看不见不等于不存在，平台升级、组件失效、定位失效，还是有人要处理。

>

> 这一节两小节，软件测试工具的发展趋势、Codeless 测试自动化。你要带走三件事：趋势有哪几条主线，Codeless 到底怎么做，这些变化对测试人员意味着什么。

>

> 第三件最要紧。工具越傻瓜，对人的设计要求越高。它只负责执行，测什么、断言什么、怎么判，还是人定。录制工具能帮你把点击录下来，它不知道这一步要验金额还是要验状态。

>

> 这也回到这门课的主线：方法比工具活得久。工具三年换一茬，判断力不会过期。听这一节的时候，把新工具三个字换成新要求，会清楚得多。

提问：工具已经能录制即用例，为什么还要学用例设计方法？

> **预设回答：**

- 答"录出来的用例又长又脆"——实情。录一条下单流程八十步，七十步是无关操作。

- 答"录不出异常场景"——关键。录制只能录走过的路，异常、边界、并发它录不到。

- 答"不知道要验什么"——最要紧的一条。工具能执行，判据要人给。

- 收口：工具越强，设计能力越值钱，这也是这门课存在的理由。

> **易错点 / 考点：**两小节名称；演进的转折点，从增强脚本到降低门槛；简答高频"录制回放式自动化为何难以长期维护"；陷阱"工具智能化后测试设计方法过时"——恰恰相反。

> **过渡：**先把这条演进线看清，再谈智能。

【14.53】软件测试工具的发展趋势（1 分钟）

> 你们实验室现在用的工具，是第几代？这个问题不好答，先看这条线。

>

> 第一代，手工和半自动。人点，表格记。痛点很直白，重复劳动。

>

> 第二代，脚本化。Selenium、QTP 这一代，用代码驱动浏览器和控件。解决的是可重复执行，代价是要会编程。

>

> 第三代，关键字驱动和数据驱动。操作抽成关键字，数据挪进表格。解决的是脚本复用，一套流程换数据能跑多个场景。

>

> 第四代，平台化和低代码。可视化编排、统一调度、统一报告。解决的是协作和维护，让不写代码的人也能进来。

>

> 第五代，智能化。AI 生成用例、AI 定位元素、AI 分析失败。冲的是覆盖率和效率的天花板。

>

> 五代不是替代关系，是叠加。今天你在一个平台上，很可能同时跑着脚本、关键字和 AI 生成的用例。

>

> 这条线上有个规律：每一代都在降门槛，同时抬覆盖。但有个反直觉的地方，门槛降了，工具能做的判断没有变多。它永远不知道这个业务这个版本最怕什么。

>

> 所以看趋势要分两层。表层是形态，从脚本到关键字到平台到智能。底层是不变的两件事，判据谁定，风险谁评。答案还是人。这两句分开记，答趋势类的论述题都用得上。

提问：五代演进里，哪一代对测试人员的技能结构改变最大？

预设回答：

- 答"第二代，因为要会编程"——确实。它把测试分成会写代码和不会写代码两类人。

- 答"第四代，因为改了协作方式"——到位。平台化把测试从个人技艺变成团队资产。

- 答"第五代，因为要懂 AI"——对一半。更准的是要懂怎么用 AI 的产出做判断。

- 收口：技能的变化方向是从会操作到会设计，从写脚本到定标准。

易错点 / 考点：五代的大致阶段与每代解决的痛点；底层不变的两件事，判据和风险，论述题核心；简答高频"发展趋势对测试人员的影响"；别死记年份和产品名，课件没给具体时间点，答题也不要编。

过渡：趋势里最具体的一支是平台，再看它在研发体系里的位置。

【14.54】低代码测试平台（2 分钟）

一个低代码平台，摆在研发体系的哪个位置？这问题比它会什么功能重要。

>

> 先说平台化。低代码平台不是孤立工具，它是测试能力的集散地。用例管理、执行调度、环境管理、报告分析、质量看板，都收在一处。企业为什么肯投钱做平台？因为个人脚本沉淀不成组织能力。人一走脚本就废，平台把能力沉下来，人换，能力不换。

>

> 再说 DevOps。DevOps 的主干是流水线：提交、构建、测试、部署、监控，一串自动跑。低代码平台要嵌进这条流水线，就得提供接口、触发器、质量门禁。测试不过，流水线拦住。所以平台化不是为了好看，是为了被调用。那句判断就是这么来的：接不上 CI 的测试工具，在企业里活不长。

>

> 最后说云原生。它带来两个变化。环境是弹性的，用时拉起，用完销毁。服务是拆开的，微服务和容器。对测试的影响很直接。环境准备从申请机器变成声明式拉起；测试对象从一个系统变成一张服务调用网。平台的责任跟着扩，要能对接 K8s，要能按需造环境，要能做服务之间的契约测试。

>

> 三句收：平台化解决能力沉淀，DevOps 解决嵌进流程，云原生解决环境和服务形态。低代码是这三件事在测试侧的落点。

提问：为什么说接不上 CI 的测试工具，在企业里活不长?

> **预设回答：**

- 答"没人会手动跑"——对。流水线是主干道，脱离主干道的工具会被忘掉。

- 答"质量门禁需要它"——更要紧。测试的价值在于拦住不合格的版本，拦不住就没有存在感。

- 答"它沉淀不出数据"——也对。不进流水线，就没有稳定的质量趋势。

- 收口：选型先问一句，它能不能被流水线调用。这比功能列表重要。

易错点 / 考点：三个概念与测试的关系；"个人技艺到组织资产"这个说法适合答平台化；云原生对测试的两个影响，环境弹性、服务拆分带来契约测试；陷阱"平台化就是买个商业工具"——核心是能力沉淀和流程嵌入。

> **过渡：**平台走到云原生这一层，接口测试就成了主战场。

【14.55】零代码智能测试工具 APIAuto（2 分钟）

> 微服务一拆，接口从三十个变成三百个。UI 自动化还在跑，可它覆盖的那点界面，跟系统真实的风险已经不成比例。

>

> 决定系统对不对的，是服务之间的契约。我这个接口的入参、出参、错误码，调用方能不能正确理解。所以微服务时代，接口测试是主战场，UI 测试是补充。

>

> 在这个背景下，零代码接口工具的价值被放大了。它能做三件事。

>

> 第一，契约即测试。有 OpenAPI 文档就能自动生成用例、自动校验响应结构。文档和实现不一致，一跑就露。这个能力天天跑，等于天天给文档做体检。

>

> 第二，批量覆盖。三百个接口，人工写用例不现实，工具分钟级铺开。

>

> 第三，嵌进流水线做门禁。每次服务发布前跑一遍，保证我改的东西没破坏别人的调用。这条最值钱，它把契约测试从有空再跑，变成每次发布必跑。

>

> 边界一定要说清：它验的是形式，不是语义。返回 200、字段齐全、类型正确，这些它管。这个金额算得对不对、这个状态能不能从已退款跳回待发货，它不管。

>

> 所以成熟的体系是三层叠着来。契约层用零代码工具自动化，业务层用人设计的用例，数据层用比对工具。三层各管一段，谁也替不了谁。

提问：接口全部返回 200、字段齐全，能说明系统没 bug 吗?

> **预设回答：**

- 答"能，接口都通了"——错答。下单成功但金额算错，完全可以返回 200。

- 答"不能，还要验业务语义"——准确。形式正确和业务正确是两件事。

- 答"要看有没有数据比对"——对。这就是数据层比对存在的理由。

- 收口：完整链条是通不通、结构对不对、语义对不对、数据对不对，四层都过才算过。

易错点 / 考点：零代码接口工具的三个价值；"验形式不验语义"是核心判断，案例题常用；三层叠加的体系结构，论述题好框架；陷阱"接口测试就是测 HTTP 状态码"。

> **过渡：**工具讲完，接着是更大的变化，测试不再是阶段，而是常态。

【14.56】14.5 彻底实现持续测试（1 分钟）

> 前面十三章的测试，藏着一个共同的假设：测试有起点，也有终点。需求阶段评审，开发阶段单测，提测后系统测试，上线前验收。走完就算结束。回头看这门课的结构，就是按这个假设排下来的。

>

> 持续测试把这个假设拿掉了。测试没有终点，它跟着流水线一直跑。提交就跑单测，合并前跑接口，上预发跑 E2E，上线后跑巡检。测试从一道关卡，变成一条一直在响的反馈。

>

> 这个转变带来三个连锁反应。

>

> 一是节奏。测试得快到能在流水线里跑完，否则它不是质量保障，是阻塞。一道要等四十分钟的门禁，没人会等。

>

> 二是选择。跑不完所有用例，就得精准选。选不准，要么漏，要么慢。选得准，才敢把全量回归让位给一条主路径。

>

> 三是组织。测试不再是测试组一个部门的事。开发要写单测，运维要接监控，谁都不能站在旁边看。

>

> 这三条不是并列的，互相咬着。节奏逼你选，选择逼你把测试分层，分层逼开发和运维一起进来。

>

> 这一节四小节：重新理解持续测试、实现框架、持续测试成熟度模型、彻底的持续测试。四条走完，回答的是同一个问题，怎么让测试跟得上交付速度。

>

> 这是全书最后一组工程问题，也是你们进公司第一年就会撞上的问题。

> **提问：**持续测试是不是就是多跑几次自动化？

> **预设回答：**

- 答"是，跑得越勤越好"——错答。跑得勤但跑得慢，直接把流水线堵死。

- 答"不是，还要快、准，还要和流程融合"——准确。

- 答"还要看跑的东西对不对"——补得好。没有精准选择的多跑，只是把噪声放大。

- 收口：持续测试是快、准、融合三件事，缺一件就退化成频繁地跑无效用例。

> **易错点 / 考点：**四小节名称；持续测试与阶段式测试的根本差别，有无终点；三个连锁反应是后面内容的纲；陷阱"持续测试就是持续集成里加个测试步骤"——远不止，涉及流程、组织、成熟度。

> **过渡：**持续测试这个词，得先把三条定义立住。

【14.57】持续测试（2 分钟）

> "等提测"这三个字，在持续测试里是没有的。什么时候测？任何一次代码变更之后。

>

> 课件给了三条定义。

>

> 第一条：测试可以随时开展起来，且具有连续性。随时开展的意思是任何时刻都能把测试拉起来，背后是环境随时可用、用例随时可跑、数据随时可造。连续性说的是结果能连起来看，今天的通过率和昨天能不能比，趋势往上还是往下。做不到这一条，每次测试都是孤立的一次性事件。

>

> 第二条：测试和开发、运维能很好地融合、匹配和同步起来。这条讲组织。传统模式下测试在下游，开发交什么就测什么。持续测试要求测试往前走，参与需求和设计；也往后走，参与线上监控。融合的意思是测试不再是一个独立环节，它是研发流程里的一层能力。

>

> 第三条最有分量：以最少的测试、最快的速度覆盖交付所面临的业务风险。措辞很讲究，不是覆盖最多的代码，不是跑最多的用例，是覆盖业务风险。这是一个立场。测试的目标不是测得多，是挡住风险。改动大的

> 地方多测，没风险的地方少测，几年没出过事的老代码只做冒烟。这一条也把测试从证明我做完了，改成证明风险被挡住了。

>

> 三条合起来：随时能测，与研发运维同步，用最小代价挡最大风险。

> **提问：**以最少的测试覆盖业务风险，这和测试要尽量充分矛盾吗？

> **预设回答：**

- 答"矛盾，得尽量多测"——这是把充分理解成数量。充分是相对风险说的，不是相对用例条数。

- 答"不矛盾，充分指风险覆盖充分"——准确。这是基于风险的测试的核心。

- 答"资源有限，必须取舍"——现实约束，也是这条定义站得住的原因。

- 收口：判断测试做得好不好，标准不是跑了多少条，是挡没挡住风险。

> **易错点 / 考点：**三条定义能复述；"覆盖业务风险"而不是"覆盖代码"，简答和论述都高频；与第 7 章基于风险的测试呼应，可做串联；陷阱"持续测试要求百分百回归"——恰恰相反，它要求精准选择。

> **过渡：**三条定义落到工程上，就是四个特点。

【14.58】持续测试的特点（3 分钟）

> 足够快、准确且有效、平滑有序、与 CI/CD 集成。四个特点，四句课件原话，每句后面都压着一条工程约束。

>

> 足够快。课件写的是：整个测试过程要快，一方面高度自动化测试，另一方面业务端到端的探索式测试。这里有两条腿。一条是高度自动化，机器能跑的都让它跑掉；一条是端到端探索式，人要看的那部分还得看。为什么探索式不能省？自动化只能验你想得到的东西，想不到的问题只能靠人去碰。所以持续测试不是全自动，是自动化兜底，人做探索。

>

> 准确且有效。课件解释得干脆：被测系统往往很复杂，不可能做全回归测试，而是要推行精准测试。这句是技术核心。前提是变更和用例之间有映射，目标是改哪儿测哪儿，而且不漏。

>

> 平滑有序。打通整个测试过程：测试左移到测试右移、单元测试到系统测试、静态测试到动态测试。三个"到"分量很重。时间上从左到右，需求阶段一直到线上。层次上从下到上，单元一直到系统。手段上从静到动，评审一直到执行。打通的意思是这些环节不再是各管一段的孤岛，是一条链。需求里的验收标准，要能一路传到线上的监控指标。

>

> 与 CI/CD 集成。和研发的持续构建、持续集成、持续部署、运维环境接起来。测试得能被流水线调用，能给出明确的通过或失败信号，能当门禁拦住不合格的版本。

>

> 四个词收：快、准、通、接。

> **提问：**为什么精准测试在持续测试里是必需的，不是可选的优化？

> **预设回答：**

- 答"因为跑不完"——现实原因。系统一大，全回归几小时甚至几天，流水线等不起。

- 答"等不起就等于没测"——更要紧。跑得太慢的测试会被团队想办法绕过，那就彻底失效。

- 答"门禁要分钟级出结论"——也对。门禁等不了半天。

- 收口：精准测试不是偷懒，是让测试能在真实节奏里活下去的前提。

> **易错点 / 考点：**四个特点能默写；三个"打通"是简答高频；精准测试的必要性要从节奏角度论证；陷阱"持续测试等于全自动化"——必须保留探索式测试。

> **【思政落点 6】**：〔课件一边要高度自动化，一边坚持留着端到端的探索式测试。为什么留着人？因为机器只验人写下的判据，事故常出在人没想到的地方。这就是质量工程的含义，它不是某个部门的活，是每个人对自己那一段负责。开发对代码负责，运维对稳定性负责，测试对判断负责。落到价值观上，是敬业与担当：防线是一环一环站出来的，不是一个人扛出来的。〕

> **过渡：**四个特点说的是要什么，接下来看怎么搭。

【14.59】持续测试实施框架（1 分钟）

一套自动化体系，你怎么判断它能不能真落地？别数脚本多少条，问五个问题。

>

第一问，测什么。也就是测试范围和风险选择。哪些用例进流水线，哪些放夜间，哪些只在发布前跑。这一层决定精准，也是整个框架的入口。

>

第二问，在哪儿测。环境和数据。环境要能按需拉起，数据要能按需构造。这一环最容易被低估。大量"自动化跑不起来"的抱怨，根子不在脚本，在环境和数据。

>

第三问，谁来跑、什么时候跑。调度和触发。提交触发、定时触发、发布触发，还有各层测试的时间预算。单测几分钟，接口十几分钟，E2E 半小时往上。超出预算的那一层，就得拆。

>

第四问，跑完怎么判。结果判定和门禁。通过标准是什么，失败怎么分级，门禁卡在哪一环。这一层决定测试说话算不算数。

>

第五问，判完怎么改。反馈闭环。失败原因能不能快速定位，回到前面讲的失败用例分析。缺陷能不能回流到用例库。质量趋势能不能被看见。

>

五问连起来是五个字：范围、环境、调度、门禁、反馈。缺一环，体系就在那儿断。拿它去对任何一家公司的实践，能立刻看出它走到第几步。

>

这五环里最难的不是技术，是门禁。门禁意味着测试说不许发，就真的不发，那要组织给权限。

提问：脚本写了几千条，但流水线从不因测试失败而阻断发布，这套体系算成功吗？

预设回答：

- 答"算，脚本多说明投入大"——错答。不能阻断的测试只是装饰。

- 答"不算，缺门禁这一环"——准确。第四问空着，整套体系说话不算数。

- 答"得看失败率高不高"——补得好。失败率高又从不阻断，团队会迅速学会忽略红灯，这比没有测试更糟。

- 收口：判断成熟度先看一条，红灯能不能拦住发布。

> **易错点 / 考点：**五环能列举；"环境与数据被低估"这个判断很实用；门禁要组织授权，是论述题的高级观点；陷阱"自动化用例数量是成熟度核心指标"——门禁和反馈才关键。

> **过渡：**最后一节，把镜头拉到整个行业。

【14.60】14.6 软件测试发展趋势（1 分钟）

> 六条趋势，你现在能背出几条？先别急着背，最后这一节要的是理解，不是默写。

>

> 14.6 两小节，MBT 的应用前景，软件测试六大趋势。

>

> 为什么把 MBT 单独拎出来讲前景？六大趋势里，模型化最不显眼，但可能最深刻。它改的不是工具，是测试的依据：从人想出来的用例，变成从模型推出来的路径。依据一变，系统性就上一个台阶。你不再靠谁的灵感多，而是靠模型盖得全不全。

>

> 六大趋势是全章的收口：敏捷化、高度自动化或者说超级自动化、云化、服务化、模型化、智能化。六条要能背，还要能各举一个例子。这是本章最标准的论述题，例子平时就得备好。六条之间也不是并列的，敏捷化是前提，另外五条是手段。云化和服务化改的是被测对象的形态，智能化改的是产出方式。

>

> 提前交个底，这一节讲完，我会把整门课收起来。所以现在带着一个问题听：六条里，哪一条对我们课上的内容冲击最大？

>

> 我的答案是敏捷化。它把测试从阶段变成常态，逼着前面所有方法提速。基于风险的测试要提速，精准选择要提速，用例设计也要提速，因为你没时间把用例写全。

>

> 模型化改的是依据，敏捷化改的是节奏。这两句分开记，六条就串起来了。

> **提问：**六条趋势里，哪一条最可能改变测试用例这种产物的形态？

> **预设回答：**

- 答"模型化，用例从手写变成生成"——很有洞见，MBT 正走这条路。

- 答"智能化，AI 直接生成"——也成立。AI 生成加模型约束，是最可能的组合。

- 答"敏捷化，用例会变少"——对一半。敏捷要的是够用的用例，不是全的用例。

- 收口：未来的用例可能是模型、数据、断言三件套，不是一份步骤清单。

> **易错点 / 考点：**两小节名称；六大趋势是本章第一名必背；"模型化改依据、敏捷化改节奏"便于记；论述题要求每条举一例，平时就备好。

> **过渡：**趋势讲完，这一章就到这里；后面把整门课的线收一收。

【14.61】一个新时代（2 分钟）

> 你们大一那年装的 App，现在还剩几个？剩下的那几个，功能越堆越多，什么都往里塞。这就是今天的底色。

>

> 课件用四个词形容脚下这个时代：移动互联、智能、Cloud、数字化。四个词落到测试上是四件事，一件一件说。

>

> 移动互联，终端从一台电脑变成几亿部手机。系统版本、屏幕尺寸、网络状况、厂商定制，全不一样。你没法用"在我这儿是好的"交差。测一个环境，变成测一片环境。兼容性专项就是这么被逼出来的。

>

> 智能，软件从按规则执行变成自己判断。输出不再是确定的，"对不对"这个判据就不够用，得换成"够不够

> 好"。这一块 14.2 认真讲过。

>

> Cloud，基础设施从买机器变成租资源。环境成了弹性的、可编程的。好处在哪？测试环境按需拉起、用完销毁。持续测试靠的就是这一条。

>

> 数字化，业务从线下为主转成线上为主。软件一停，生意就停。一次宕机不是"系统慢了点"，是"今天不卖了"。质量在生意里的分量被抬上去。

>

> 四个词收成一句：软件定义一切。再往前半步，软件定义一切，质量就跟着定义一切。所以测试这个岗位不是变轻，是要求变了。不再是点点点，是在高速变化里守住底线。

>

> 给一条能带走的建议：给自己搭三条腿。技术深度，会写代码、会看架构、会用工具。业务理解，懂这行的规则和风险。工程判断，知道哪些要测、哪些能放过。缺一条，走不远。

提问：三个能力方向里，在校期间最值得先补哪一个？为什么？

预设回答：

- 答"技术深度，先会写代码"——合理，技术是入场券。只补技术会变成工具人。

- 答"业务理解，学校没教"——很清醒。这块多半要在实习里补，越早碰真实业务越好。

- 答"工程判断，最难"——最难也最值钱，只能靠做过、错过、复盘过攒出来。

- 收口：技术打底，业务补课，判断靠实践。三条腿交替长，别指望一次长齐。

易错点 / 考点：① 四个时代特征及各自对测试的影响：碎片化、输出不确定、环境弹性、故障等于业务中断；② "软件定义一切"的含义与推论；③ 能力模型三条腿是简答与面试高频；④ 陷阱题"测试岗位会因自动化消失"——岗位形态会变，判断责任不消失。

过渡：时代讲完，回到技术。MBT 凭什么被看好。

【14.62】MBT 的应用前景（2 分钟）

画完的流程图，大多死在文档目录里。项目一改，那张图早就对不上了。

>

课件给 MBT 三条前景，每一条都在治这类浪费。

>

第一条，MBT 工具让建模过程更规范、更可视化，模型的可读性和重用性都上来了。手画的图躺在文档里没人维护，工具里的模型是活的：能版本管理、能复用、能被工具读出来生成用例。可读性对人，新人看一眼就懂系统怎么走。重用性对机器，同一个模型换一种覆盖策略，又是一批用例。

>

第二条，敏捷里用 MBT 能促进持续交付，把敏捷测试的风险压住。为什么？敏捷的节奏是快速迭代，快速迭代最怕改一处、崩一片。模型在这个位置上的角色，是变更的影响分析器。需求变了，改模型，工具立刻报出哪些路径受影响、要回归哪些。精准测试就从人工猜，变成按结构推。

>

第三条，AI 和 MBT 合起来更强。用 AI 创建、训练测试模型，再用算法按覆盖率和风险推荐测试路径。分工很清楚：AI 管建模和排序，MBT 管生成和验证。时间不够的时候先跑高风险路径，这一条最实用。

>

三条收成一句：MBT 的前景不在多一个工具，在把测试依据从经验换成模型。模型成了一等公民，建模、变更分析、路径生成、风险排序才能一起自动化。

>

现实也要说清。落地有两个前提：团队愿意画模型，模型能跟着代码走。这两条做不到，它就只是 PPT 上的概念。

提问：为什么说"模型是变更的影响分析器"？

预设回答：

- 答"改了模型就能看出哪些路径变了"——对。节点与边的关系让影响面变得可计算。

- 答"因为能自动生成回归集"——这是直接产出，也是精准测试的走法。

- 答"因为模型能查出走不到的状态"——好补充。模型本身能被静态检查，死状态、死锁、不可达路径都能挑出来。

- 收口：MBT 干的事，是把"改了什么会影响什么"从经验判断变成结构推导。

> **易错点 / 考点：** ① 三条前景要能复述；② "模型等于变更影响分析器"这个提法适合答题；③ AI 与 MBT 的分工：AI 建模与排序，MBT 生成与验证；④ 陷阱题"MBT 只适用于大型嵌入式系统"——Web、API、业务流程同样适用。

> **过渡：** 模型说完了，模型怎么被检测？有一批成熟工具。

【14.63】示例：模型检测工具（2 分钟）

> 死锁、竞态这类毛病，跑一万遍用例也未必撞上一次。撞不上，不代表它不在。

>

> 模型检测走另一条路：先把系统建成状态机，再把所有路径走一遍。课件列了六个经典工具，参数不用背，要抓住每个工具代表哪种思路。

>

> SPIN，全名 Simple Promela Interpreter，LTL 模型检测工具。先用 Promela 给并发系统建模。再检测这个有限状态系统满不满足 PLTL 或 CTL 性质。它看家的是并发：死锁、竞态、活锁，人读代码很难看出来，它可以系统性地穷举。

>

> SMV，CMU 做的，CTL 模型检测。它用二叉图表示状态转换关系，靠计算不动点去查状态的可达性，以及这些状态满足什么性质。二叉图的作用是压缩状态空间。状态爆炸是这个方法的天敌，压下去才算得动。

>

> CWB，爱丁堡大学和美国北卡大学做的，mu 演算工具。查系统间的等价关系、PRE-ORDER 关系，也查系统满不满足 mu 演算公式。特长是比较两个系统：重构之后行为还一致不一致，用它。

>

> 后面三个是演进。NuSMV 是符号模型检测器，基于 BDD 和 SAT，引了 SAT 能啃的状态空间更大。NuXmv 也是符号模型检测，多了同步的有限状态和无限状态系统分析。取值无限的真实系统，这样才进得来。Murphi 出自斯坦福和犹他大学。许多模型检测工具都拿它当基础，CMurphi、Preach 都从它长出来。

>

> 六个工具的意义在方法论上：验证从跑用例升到数学证明。用例只能说这条路上没问题，模型检测能说所有可能路径上都没问题，前提是模型本身没写错。这也是 MBT 的理论根基。

> **提问：**模型检测能证明"所有路径都没问题"，是不是比跑用例强得多？

> **预设回答：**

- 答"是，数学证明更可靠"——理论上强，但压着一个致命前提：模型必须准确。

- 答"不是，模型错了证明就没意义"——正确判断。这是模型检测最大的局限，也是 MBT 的软肋。

- 答"状态爆炸时算不完"——第二大局限，BDD、SAT 就是为它准备的。

- 收口：两者互补。模型检测保证逻辑上无懈可击，测试保证实现与模型一致。

> **易错点 / 考点：**① 六个工具与各自技术路线要能对上：LTL、CTL、mu 演算、SMC、BDD、SAT；② 两大局限（模型准确性、状态爆炸）是简答高频；③ "验证与测试的关系"可作论述素材；④ 陷阱题"模型检测可以替代测试"——不能，二者互补。

> **过渡：**工具和模型都过完了，最后收成六条趋势。

【14.64】软件测试六大趋势（3 分钟）

> "测试会不会被 AI 取代"，这句你们听过不止一遍。听完容易慌，慌完什么也没做。

>

> 课件把方向压成六条，一条一条过。

>

> 敏捷化：测试左移到位，ATDD 让测试在写代码之前就把验收标准定下来；测试右移也明显加强，灰度、监控、线上验证都往后铺。边界从提测到上线，扩成需求到运维。对你意味着什么？得会和产品、开发一起写

> 验收标准。

>

> 高度自动化、超级自动化：让自动化测试成为流水线。抓"成为流水线"这几个字。自动化不是跑脚本，它得有触发、有门禁、有报告，是流水线上的一环。

>

> 云化：基础设施支持自动化运行。环境、数据、执行都能放到云上，按需扩容、按需销毁。它治的是排队等环境这个老毛病。

>

> 服务化：一切测试皆 API，把测试能力包成服务。造数、比对、环境、执行，谁要用谁来调，不必每家重写一遍。

>

> 模型化：MBT 让自动化测试更彻底。前面讲过，它动的是测试依据。

>

> 智能化：靠大数据和 AI，让自动化更聪明。用例生成、元素定位、失败分析、风险预测，都归这一条。

>

> 六条给个记法：两化管节奏，敏捷化和自动化；两化管形态，云化和服务化；两化管内核，模型化和智能化。答题按这个骨架铺，条理自己就出来了。

提问：六条里，哪一条对"测试人员的数量需求"影响最大？

> **预设回答：**

- 答"智能化，AI 替代人"——短期有限。AI 顶掉的是重复执行，判断和设计仍然缺人。

- 答"服务化，能力集中就不用那么多人"——有可能。重复建设的岗位减少，平台型和专家型岗位增加。

- 答"敏捷化，测试融进开发"——行业里真在发生。很多团队改叫质量工程师。

- 收口：趋势带来的不是人变少，是人的结构变了。执行型收缩，设计型和平台型扩张。

易错点 / 考点：① 六条必须能默写，本章第一考点；② 每条配一个例子：ATDD、流水线门禁、云测平台、

造数服务、MBT、AI 失败分析；③ 记忆结构"节奏—形态—内核"好用；④ 陷阱题"六大趋势是先后关系"——它们并行发生，不同企业进度不同。

【思政落点 7】：六条趋势摆出来，最省事的反应是追热点：听说智能化火就扎进 AI，明年换个平台又从头来。可这六条没有一条能靠学一个新工具拿下。敏捷化要会和人对齐标准，服务化要懂架构，模型化要会抽象，都只能一年一年攒。挑一条主线扎下去，扎到能解决真实问题为止。

过渡：六条里，云化最容易被想成"把测试放上云"。看一个更深的例子。

【14.65】示例：云化（1 分钟）

环境申请单递上去，等排期的日子你还有印象吗？等到手，需求已经改了两轮。

>

云化动的是这一层。课件在"云化"这一条下面没配正文，内容得说透，它是三层变化。

>

第一层变化，环境从资产变成能力。以前环境是稀缺资产，谁先占到谁用。云上可以声明式拉起，写一份配置，几分钟一套隔离环境，用完销毁。这解开了持续测试最大的物理死结：并行。十个分支同时跑，各用各的，谁也不用等。环境一便宜，测的频次就上来了，人也就敢改代码了。

>

第二层变化，被测对象的样子变了。云原生把单体拆成微服务、装进容器、跑在编排平台上。于是三个新问题冒出来。服务发现，我调的那个服务现在落在哪个实例上。配置外置，同一份镜像在不同环境行为不一样。有状态和无状态混在一起，数据库、缓存、消息队列怎么造数据。

>

第三层变化，故障变成常态。实例会被调度、被驱逐，网络会抖。测试重点就从功能对不对，扩到出事能不能自己恢复。混沌工程干的就是这个：主动注入故障，看系统扛不扛得住。

>

三句收口：云化让环境变便宜，让架构变复杂，让故障变常态。对应的能力是环境即代码、契约测试、混沌实验。

> **提问：**微服务下，为什么端到端测试要少做、契约测试要多做？

> **预设回答：**

- 答"服务太多，端到端组合爆炸"——核心原因。N 个服务的组合数远超可测范围。

- 答"端到端慢且脆"——实情。一条用例拉起一整套环境，挂了还说不清是谁的问题。

- 答"契约测试盯住调用方不被破坏"——准确。它把服务间一致性单独拎出来验，成本低、定位准。

- 收口：策略是大量契约测试加少量关键端到端，和测试金字塔一个道理。

> **易错点 / 考点：**① 云化对测试的三层变化；② 混沌工程的定义：主动注入故障、验证韧性；③ 微服务测试策略（契约多、端到端少）是高频简答；④ 陷阱题"云化就是把本地脚本放到云主机上跑"。

> **过渡：**云化之外还有一条更虚的趋势，服务化。它改的是测试能力的组织方式。

【14.66】测试服务化（2 分钟）

> 同一个造数脚本，三个组写三份。改一次，三处都要动。

>

> 课件给测试服务化列了一串关键词，按四层归拢就好记。

>

> 第一层，基础设施，基础设施即代码，IaC。环境不靠人搭，用代码描述、用工具创建。环境都拉不起来，后面的服务化无从谈起。

>

> 第二层，能力，基于 API 和服务的云平台。造数、比对、执行、报告，都做成 API。一句话，一切测试皆 API。价值在复用：一个团队做好的造数服务，十个团队直接调。

>

> 第三层，使用体验，支持一键部署，自动监控、高度可视化，在线实时收集数据。这一层管的是门槛。能力再强，用起来费劲就没人用。一键部署降启动成本，可视化降理解成本，实时数据降判断成本。

>

> 第四层，数据，大数据处理能力。执行日志、覆盖率、性能指标、失败聚类，量都很大。存得下、算得出、看得见。缺这一层，平台就只是个跑脚本的地方。

>

> 四层合起来是测试中台的雏形：把能力从每个项目各自造，变成一次建设、多次复用。

>

> 副作用得说一句。服务化会让测试离被测系统更远。你调一个造数服务，未必知道数据怎么造的，出了事，服务就成了新的黑盒。所以服务本身要有可观测性，还要有降级方案。这两条不写进方案，别急着推。

提问：测试服务化最大的收益和最大的风险，各是什么？

> **预设回答：**

- 答"收益是复用，风险是依赖"——简明。依赖一旦断，全组停摆。

- 答"收益是效率，风险是黑盒"——更贴。服务成了新的不可见层。

- 答"风险是服务挂了全组停摆"——真实存在。关键服务要有 SLA 和降级。

- 收口：配套是两件事，可观测、可降级。不写进方案就别推。

易错点 / 考点：① 服务化四层内容要能分层说；② "一切测试皆 API"的含义；③ 收益与风险要成对答，只答收益丢分；④ 陷阱题"服务化只对大公司有意义"——小团队用开源组件也能做轻量服务化。

> **过渡：**最后一个案例来自真实业务，用高速摄像头量一个 App 的加载时长。

【14.67】示例：手淘 RXT 测试机器人（2 分钟）

点开消息 Tab，屏幕上一块白。内容出来之前，先看见的是空白。

>

> 这一格要量的就是那块空白持续多久。传统做法是打点：代码里埋时间戳，算差值。它有三个毛病。侵入，得改代码。不准，只覆盖代码层，渲染、合成、上屏它看不见。跟用户感受对不上，用户认的加载完，是屏幕上真出现了内容，不是接口返回了。

>

> 手淘 RXT 的做法是换测量工具，用眼睛，还是高速的眼睛。课件给的参数很硬：240fps 高速摄像头，误差小于 5ms；无需参照目标；自动识别准确率不低于 95%。方法叫基于视频特征分析的加载时长自动化。

>

> 三个参数拆开看。240fps 是每四毫秒一帧，毫秒级的画面变化都留得住。误差压在 5ms 以内，够判断达不达标。无需参照目标，意思是界面上不用做标记、代码里不用埋点，对被测应用零侵入。零侵入才敢测真实版本。准确率 95% 以上，说明内容出没出来是视觉算法在判，不是人盯屏幕数帧。

>

> 案例值钱的是方法。测量工具可以跳出软件本身，摄像头、示波器、功耗仪都算数，谁说测试只能用软件测软件。体验指标要用用户的眼睛定终点，用户看屏幕，就测屏幕。零侵入是测量真实的前提。

提问：为什么"接口返回了"不等于"用户看到了"?

> **预设回答：**

- 答"还要渲染、布局、绘制、合成"——准确。数据到像素中间有好几道工序。

- 答"可能还有图片、字体要加载"——也对。资源加载常是体验瓶颈。

- 答"接口快不代表页面快"——所以体验指标必须单独测。

- 收口：体验类指标的终点用用户视角定，就是内容出现在屏幕上的那一刻。

易错点 / 考点：① RXT 三个参数与"零侵入"的意义；② 打点法的三个局限：侵入、不准、与感受不一致；③ "测量工具可以跳出软件"这条能抬高论述题层次；④ 陷阱题"性能测试只能用软件工具做"——硬件测量是重要补充。

> **过渡：**全章讲完了，最后一页把十四章收起来。

【14.68】小结（3 分钟）

最后一节课，我不加新东西。

>

> 先收本章四句。大数据的测试，贯彻数据清洗、转换、存储、加载全过程。靠抽样测试验证、数据对比分析、

> 数据映射验证兜底。智能应用的测试，是为了保住数据质量、特征质量、模型质量，现在已经有一套成熟的评价指标体系。图像识别、NLP、OCR 这些 AI 手段已经用开了。单元测试、系统的 API 测试、UI 测试、日志分析，都在用它们。跟 MBT 合起来，效果更好。最后一句，六大趋势：敏捷化、高度自动化、云化、服务化、模型化、智能化。

>

> 再收整本书。十四章是一条线，六个台阶。观念在第一二章，给你立场：测试不是为了证明对，是为了发现错。方法在第三章，把无穷的输入变成有限的用例。流程在第四章，告诉你什么时候做什么。技术在五到九章，单元集成、系统功能、专项、缺陷与度量。项目实践在十到十三章，把方法用进一个有 deadline 的真项目。展望就是今天这一章：对象在变，手段在变，交付在变，判断力不变。

>

> 两条建议。毕业设计里选一个小而完整的系统，需求、设计用例、执行、缺陷、报告走一整圈，三个模块也够。答辩看的是结论有没有证据撑着。求职那边，面试八成问三件事：用例怎么设计，能不能上线怎么判，发现过什么有价值的缺陷。前两问考方法，第三问考你有没有真做过。

提问：只允许一句话向别人介绍这门课学了什么，你怎么说？

预设回答：

- 答"学了怎么找 bug"——太窄。找 bug 只是手段。

- 答"学了怎么用证据判断一个软件能不能交付"——到位。方法、流程、判断都在里面。

- 答"学了黑盒白盒、等价类边界值"——这是知识点，不是能力概括。面试这样答会被追问到崩。

- 收口：我学会在有限时间里，用可复现的证据判断一个软件的风险在哪、能不能上线。

易错点 / 考点：① 本章四句要能复述；② 六台阶主线（观念—方法—流程—技术—项目实践—展望）是综合论述题的标准框架；③ 六大趋势与本章四句常合并出题；④ 答题纪律：趋势只讲课件观点，不编市场数据和时间点。

【思政落点 8】：十四周走下来，我最想留下的不是某个方法的名字，是一种习惯：说话要有证据。说"没问题"得拿出用例和日志，说"能上线"得说清挡掉了哪些风险。往小说是职业素养，往大说是把技术用在正道上。同样能发现真问题，也能把问题藏起来。技术决定你能做什么，肯不肯做干净是你自己定的。

过渡：最后一页，我把话说完。

【14.69】感 谢 聆 听（1 分钟）

> 十四个星期，讲到这里，可以收了。

>

> 这门课教的东西，很多会在几年内过时。工具会换，框架会换，平台会换，连"测试"这个岗位的名字都可能变。有一样不过时：面对一个软件，你能问出两句——它可能在哪儿出错，我凭什么说它没错。这两句才是这门课想给你的东西。不是知识点，是一种看系统的方式。

>

> 拆开看，两句分工不同。第一句是眼力，知道风险一般藏在什么位置；第二句是规矩，结论必须有证据托着。只有眼力没有规矩，判断站不住；只有规矩没有眼力，测一堆不相干的东西。

>

> 还有一句，别把质量当成别人的事。写代码时多想一步边界，评审时多问一句异常怎么处理，赶工期时把要测的测掉。动作都很小，加起来就是用户愿不愿意信我们做的软件。

>

> 谢谢这十几周坐在下面的你们。有同学问过很好的问题，有同学在实验里跟我较过真。较真在工程里是最值钱的品质，别弄丢。以后你会发现，肯为一个细节跟你争到底的同事，最好合作。

>

> 课件最后一行是落款：广东金融学院，周宇文。名字写在这儿，课就讲完了。

>

> 祝你在毕业设计里做出一个自己愿意拿出去讲的作品，祝你在工作里遇到愿意带你的人。江湖再见。

提问（收尾式，不要求回答）：在笔记上写一句，这门课你打算真正带走的是哪一样东西？

> **预设回答：**

- 答"用例设计方法"——实用。写任何一份测试方案都用得上。

- 答"用证据说话的习惯"——最有价值的答案。它超出测试岗位本身。

- 答"好像都记住了，又好像都忘了"——正常。知识会在用到时回来，留下的是你面对问题时的第一反应。

- 收口：写哪一句都行，请在毕业设计里用它一次。用过一次，它才真归你。

> **易错点 / 考点：** ① 课程结束页，考试不涉及；② "两个问题"（可能在哪儿出错、凭什么说没错）可作结课论文的点题句；③ 教材第 3 篇的项目案例仍可作毕业设计的参考框架。

> **过渡：** 课结束了，书留着。遇到问题时回来翻，比任何笔记都管用。

四、思政落点清单（共 8 处）

序号	所在页	落点主题
1	14.13	比对脚本说全部一致，你就敢上线吗？脚本也是人写的，它的错会以合格的样子出现。所以
2	14.20	业务方说准一点，你把 95%写进验收单，签字那一刻，责任就落到你身上。风控误判一次
3	14.23	数据质量说到底是在问人肯不肯较真。年龄 200 岁这种记录，改一行就能遮过去，也可以
4	14.34	公平性和可解释性最容易被省，因为不好测，也不好写进汇报。可一个模型对着某类人群系
5	14.45	〔机器只给可疑点、不给判决，这个分寸值得停一下。它牵着 14.3 里人机协同的边界：
6	14.58	〔课件一边要高度自动化，一边坚持留着端到端的探索式测试。为什么留着人？因为机器只
7	14.64	六条趋势摆出来，最省事的反应是追热点：听说智能化火就扎进 AI，明年换个平台又从头
8	14.68	十四周走下来，我最想留下的不是某个方法的名字，是一种习惯：说话要有证据。说"没问