《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】【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（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）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 各位同学，我们上到最后一章——**第 14 章 软件测试展望**。
>
> 前面十三章一直在回答"**怎么做**"：怎么定概念、选方法、走流程、写单元与集成测试、测功能与专项、管缺陷、算覆盖率、做项目实践。这一章换一个问题：**这条路往下走，往哪儿走**；以及更要紧的第二问——**你从这门课里到底带走了什么**。
>
> 这一章有三个任务。第一，看技术前沿：大数据怎么测、AI 系统怎么测、AI 怎么反过来帮我们测、测试工具往哪个方向变、什么叫彻底的持续测试。第二，练判断力：面对"AI 会不会取代测试"这种问题，你得能给出一段有论据的回答，而不是跟着热搜走。第三，**把十四章收成一张图**，让你在毕业设计和面试里能一口气讲明白。
>
> 请先记住这一章的总纲，也是全书的总纲：**工具会换、框架会换、平台会换，但"怎么定义质量、怎么设计验证、怎么用证据说话"这套判断力不会过期。**

> **板书：**中间写"第 14 章 软件测试展望"；三行分列："看趋势（大数据・AI 系统・AI 助力・工具・持续测

> 试）→ 练判断（AI 能做什么不能做什么）→ 做收口（14 章一张图・毕设・求职）"。底部一行："工具会换，判断力不会过期。"

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"——那我们把十三章压缩成十三章工具名好了，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。测试的分析思路、用例设计方法、缺陷判断标准，换了工具照样用。
- 答"有用，因为面试还考"——这个回答很实在，但只说对一半：面试考的是判断力，不是背诵。
- 收口：这一章就是回答这个问题的——把"会过期的东西"和"不会过期的东西"分开，你把后者带走。

易错点 / 考点：① 这是全书收尾章，考试容易出"综合论述题"，比如"结合本课程谈谈你对软件测试发展趋势的理解"，答题必须**先给课件六节的结构，再落到自己的判断**；② 别把这一章当成"名词罗列"，课件里的每个新技术都要能说清"它解决了老办法解决不了的什么问题"；③ 记忆抓手：这一章的路线是"**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：先把这一章的六节结构摆出来，你心里有个时间预算。

【14.2】14.1 大数据的测试（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是本章的六节目录，我按顺序点一遍，你记结构就行。

14.1 大数据的测试——数据量从 GB 涨到 TB、PB 之后，测试的对象从"页面和接口"变成了"数据管道"。

14.2 AI 系统的测试——被测对象本身变聪明了，它的输出不再是确定的，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：图像识别做 UI、OCR 认元素、NLP 转用例、算法分析失败日志。

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——测试不再是阶段，而是流水线里的常态。

14.6 软件测试发展趋势——最后收成六条趋势，也是本课程对行业走向的总结。

这六节有两条内在逻辑，你听的时候抓住它：**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"**。被测对象、测试手段、交付方式——这三样一变，测试人员的活儿就跟着变。

板书：竖排六行：14.1 大数据测试（数据管道）→ 14.2 AI 系统测试（对象不确定）→ 14.3 AI 助力测试（手段智能化）→ 14.4 工具未来（低代码・平台化）→ 14.5 持续测试（测试常态化）→ 14.6 六大趋势。右侧竖写三组："对象变了｜手段变了｜交付变了"。

提问：这六节里，哪一节讲的是"测试手段"的变化，哪一节讲的是"被测对象"的变化？

预设回答：

- 答"14.1 和 14.2 是对象，14.3 和 14.4 是手段"——准确。14.5、14.6 是交付方式与总体趋势。

- 答"全都是趋势"——太笼统，答题时说不出层次，论述题就拿不到结构分。

- 答"14.3 是对象变化"——容易混：14.3 的主语是"测试"，AI 是工具；14.2 的主语是"AI 系统"，它才是被测对象。**看清主语，是这一章最容易拿分也最容易丢分的地方**。

- 收口：以后读任何一篇测试新技术文章，先问一句"它变的是对象、手段，还是交付方式"。

易错点 / 考点：① 六节名称与顺序要能默写；② **14.2 与 14.3 的方向千万别弄反**：14.2 是 Test **for** AI（测 AI），14.3 是 AI **for** Test（AI 帮测试）；③ 常考简答："大数据测试与传统测试的根本区别"——传统测试比的是"输入输出对不对"，大数据测试比的是"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，测试到底在测什么？

【14.3】14.1 大数据测试（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是 14.1 节的地图，四小节，我用一分钟画出来。

14.1.1 大数据的特性与挑战——先弄清大数据测试和普通测试差在哪，为什么老办法不够用。

14.1.2 大数据的测试方法——落在 ETL 这条主线上：抽取、转换、加载，每一段测什么、怎么验。

14.1.3 大数据的测试实践——真实项目里测试的主要工作、完整流程，以及自动化怎么做。

14.1.4 大数据的测试工具——工具层面的支撑。

一句提醒：这一节的概念不少，但**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**。你把这句记住，后面所有细节点都能挂上去。

为什么把大数据放在本章第一节？不是因为最难，而是因为**它是 AI 的原料**。下一节要讲 AI 系统的测试，可如果喂给模型的数据本身就是脏的，模型再好也白搭。**数据质量是模型质量的地基**，这个道理，这一节先讲透。

板书：一条横线串四段：特性与挑战 → 测试方法（ETL：抽取・转换・加载）→ 测试实践（工作・流程・自动化）→ 测试工具。横线下方一行大字："数据不丢、不算错、不变形"。

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，这就是大数据测试的核心：**界面只是表皮，数据才是本体**。

- 答"得跟业务对数"——非常专业，行业里叫"多方报表对比验证"，后面流程页会专门讲。

- 收口：大数据测试的第一句话是——**看得见的地方往往没问题，问题都在看不见的数据流里**。

易错点 / 考点：① 四小节名称与逻辑顺序；② 常考判断："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 记住这一节的服务对象：它为 14.2 的 AI 系统测试提供"数据质量"这一半地基。

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）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页回答最基础的一问：**大数据测试具体验什么**。课件给了几组落点，我按"单个节点 → 节点之间 → 整体结果"三层给你排好。

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，每个节点都在做一件事：过滤、清洗、汇总、关联。节点里的业务逻辑对不对，必须先单独验——这一步相当于我们第 5 章讲的**单元测试**，

只不过被测的"单元"不是函数，而是一个计算任务。

第二层，节点之间的一致性。 课件讲了两条：**数据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。这三件事的共性是什么？**它们都发生在数据"搬家"的时候**。搬家最容易出错：该合的不合，该分的不分，键配错了值。

第三层，整体结果落在哪儿。 课件有一句特别关键：**验证正确的数据被提取并加载到正确的 HDFS 位置**。注意这里是两个"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，而且这种错最难查，因为文件看起来好好的。同时还要**检查转换规则、检查数据完整性、一致性**。

说白了，大数据测试就是一句话：**该来的都来了（完整性）、来的都对（正确性）、放的地方对（位置）、彼此不打架（一致性）**。这四句就是这一页的抓手。

板书：三层：① 单节点（业务逻辑＝任务的"单元测试"）→ ② 节点间（聚合规则・隔离规则・键值对）→ ③ 整体（数据正确＋HDFS 位置正确；转换规则；完整性与一致性）。底部四句："该来的都来・来的都对・放得对・不打架"。

提问：如果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 0，业务方第二天才发现。

- 答"看有没有监控告警"——补得好，这正是后面"数据监控"存在的理由：**这类缺陷靠人肉发现成本极高，必须靠监控兜住**。

- 收口：所以验收大数据系统，不能只看"数算得对不对"，还要看"数落得对不对、被不被发现"。

易错点 / 考点：① 三层落点要能复述，尤其"数据正确＋位置正确"这个双正确；② 常考：聚合规则与隔离规则的区别——聚合是"该合的合起来"，隔离是"不该串的不能串"（多租户、多账套场景必考）；③ 键值对验证属于第二层，别答成第一层；④ 记忆口诀："**单点验逻辑、跨点验规则、整体验落位**"。

过渡：三层里最核心的是中间那层——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）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页给大数据处理的主干流程起了名字：**ETL**——数据**抽取（Extraction）、转换（Transformation）、加载（Loading）**。抽取是从业务系统把数据拿出来，转换是按规则加工，加载是写进数据仓库。三个字母，就是数据的一生。

课件把它拆成了三个中间文件，这三个文件是这一页的精华，你一定要分清：

第一个，EXF，Extracted Format，抽取格式。 它由数据源 Extract 产生，**文件结构与源相似**，但**经过过滤，部分字段被忽略**。注意"结构与源相似"这句话——它意味着这一步还没做业务加工，只是把用不上的字段丢掉、把不该进来的数据挡住。

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"四个字的用意是**解耦**：上游各种异构数据源，下游统一成一种格式，后面所有处理只认这一种。

第三个，PLF，Pre-Load Format，预加载格式。 它经过数据转换，**用于直接加载到数据仓库**，而且**其数据结构与数据仓库中的表定义一致**。这句话是说：到了 PLF，格式已经和数据仓库的表严丝合缝。

为什么要分三段文件？三个理由：**可回溯**——出错能定位到哪一段；**可分工**——抽取、转换、加载可以由不同的人甚至不同的系统做；**可复跑**——某一段失败，从中间文件重跑，不用回源头重来。这就是工程设计里"**把长流程切成有契约的短段**"的思路。

> **板书：**三段流水：数据源 →【EXF 抽取格式：结构似源、字段被过滤】→【CIF 公共接口格式：C/S/S 中间文件、统一格式解耦】→【PLF 预加载格式：与数仓表定义一致】→ 数据仓库。旁注："可回溯・可分工・可复跑"；写全 ETL＝Extraction・Transformation・Loading。

提问：如果加载完成后发现报表数字不对，你会先去查哪一段文件？为什么？

预设回答：

- 答"查 PLF，因为它最接近仓库"——顺序对，但方向反了：PLF 对说明问题在前两段，PLF 错说明问题在转换。

- 答"从 EXF 往后逐段对数"——这就是标准做法，行话叫"**分段对数**"：相邻两段的量与关键字段能对上，断点就找到了。

- 答"直接查源库"——源库数据本来就对，你是去确认"该抽的抽出来了没有"。

- 收口：这三个文件的存在价值，就是让你**能把一次排查切成三次比对**。

> **易错点 / 考点：** ① 三个缩写的**全称与中文名**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 常考简答："为什么要设置 CIF 这一层"——解耦异构数据源，避免 N 个源对 M 个目标的两两适配；④ ETL 三个字母的英文全称是送分题，别拼错 Transformation。

> **过渡：** ETL 的第一段是抽取。抽取看着简单，其实最容易把脏数据带进来。

【14.6】数据抽取与清洗（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页讲 ETL 的第一段，两件事：**抽取**和**清洗**。

抽取方式有两种：**全量抽取**和**增量抽取**。全量的意思是每次都把源表整份搬过来，简单可靠但代价高；增量是只搬变化的部分，效率高但难在"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。

清洗要干的活儿，课件列了四件：**清洗异常数据、不符合规范的数据**；**空值处理**；**规范化数据格式**；**实现数据规则过滤**。空值处理最见功夫：一个"年龄"字段空了，补 0、补均值、还是整条丢弃？三种选择业务含义完全不同，**这不是技术选择，是业务决策**，要和数据 owner 确认，测试的职责是把这条规则验出来。

然后是抽取环节的**两个硬要求**，这两条一定要背：**第一，准确性**——**能将业务系统中的变化数据按一定的频率准确地捕获到**；**第二，性能**——**不能对业务系统造成太大的压力，影响现有业务**。这两条是一对矛盾：要准确就多查几次源库，可多查就压业务系统。所以真实项目里增量抽取的调度频率，往往是业务方和 DBA 一起定的。

最后看**测试怎么做**，课件给了五种手段：**抽样测试验证**——抽一批记录人工核；**对比处理前后的数据量变化**——进去多少条、出来多少条，差在哪；**验证数据合法性**——格式、范围、类型对不对；**字段逻辑验证**——字段之间的关系成不成立，比如"结束时间不早于开始时间"；还有**枚举类型的字段查看枚举分布**，比如性别、年龄、地域、学历的分布，以及活跃小时段分布。最后这条特别实用：**分布是最好的体检表**，一个地域字段昨天 90% 是广东、今天变成 10%，代码一行没改也能立刻看出问题。

> **板书：**抽取（全量／增量）→ 清洗（异常数据・空值处理・格式规范化・规则过滤）→ 两硬要求（准确性：按频率准确捕获变化；性能：不压垮业务系统）。测试五手段：抽样验证｜量对比｜合法性｜字段逻辑｜枚举分布。旁注："分布是体检表"。

提问：一张订单表昨天 100 万条，今天增量抽取只进来 3 条，你先查什么？

预设回答：

- 答"查源库昨天是不是真没订单"——第一步就该这样：先确认是"源没数据"还是"抽没抽到"。

- 答"查增量字段（时间戳／自增 ID）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**增量标识被上游改了语义**。

- 答"3 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**，抽取"成功"不等于"抽对"。

- 收口：所以课件把"对比处理前后的数据量变化"列为独立手段——**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 抽取的"两个硬要求"是简答高频点，尤其"性能不能影响现有业务"这条常被漏；② 空值处理属于**业务决策**，测试的职责是验证规则被正确执行，而不是自己拍一个默认值；③ 五种验证手段要能列举，其中"枚举分布"最容易被忽略但最常考案例；④ 判断题陷阱："全量抽取一定比增量抽取可靠"——可靠但代价高，选哪种取决于数据量与业务窗口，不是绝对的。

过渡：数据抽进来了，接下来要按规则改造它——这一段是 bug 最密集的地方。

【14.7】数据转换及其测试（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲 ETL 的第二段：**转换**。课件列了八类转换动作，我一个一个过，你重点记"每一类会怎么错"。

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**——按条件筛掉不要的，错在多筛或少筛；**数据加解密**——敏感字段脱敏，错在漏字段，身份证没脱敏就直接落仓；**数据类型统一转换**——字符串转数值、日期格式统一，错在精度丢失和时区错位；**拆分数据**——一个字段拆成多个，错在分隔符异常时拆歪；**数据按主题划分与合并**——按业务主题归并，错在归类规则；**数据排序**——错在排序键选错，导致下游取"最新一

条"取错。

这八类里最值得警惕的是**字段映射**和**类型转换**：它们的错误**不会报异常**，只会悄悄地让数字变形。**不报错的错，才是最贵的错。**

怎么测？课件给了两条路，正好对应我们在第 3 章学的白盒黑盒：**白盒转换测试**——检查转换逻辑和映射设计、检查代码实现逻辑、把转换后的数据与文档中的数据做比较。注意它验的是"**设计对不对、实现符不符合设计、结果符不符合文档**"三层。**黑盒转换测试**——不看代码，只**准备测试数据来反映文档中列出的不同转换场景**，输入进去看输出对不对。

一句话对比：**白盒问"你是不是照设计的做"，黑盒问"你做的事对不对"**。两条都要做，因为设计本身也可能错。

> **板书：**八类转换竖排：替换・映射・过滤・加解密・类型统一・拆分・按主题划分合并・排序。下方两栏：白盒（设计→代码逻辑→与文档比对）｜黑盒（按文档场景造数据→看输出）。旁注："不报错的错，才是最贵的错"。

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"——恰恰相反，报错最容易发现，流水线直接红。

- 答"四舍五入"——危险，但至少偏差有界。

- 答"单位不一致，比如一个按元、一个按分"——这才是最危险的：**数量级差 100 倍，数值类型完全合法，任何校验都不报错**，报表出来直接错两个数量级。

- 收口：所以转换环节的测试重点不是"能不能跑通"，而是"**跑通了以后数还对不对**"。

易错点 / 考点：① 八类转换动作能列举，能各举一个典型错误；② 白盒与黑盒转换测试的**判据差别**要能说清（对设计／对文档场景）；③ 案例题高频："报表金额整体偏小 100 倍，定位到哪一环"——先查类型转换与单位，再查字段映射；④ 判断题陷阱："转换测试只要比对最终结果就行"——错，中间结果不保留，出了问题无法定位。

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。这叫数据映射验证。

【14.8】数据映射验证（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲**数据映射验证**。前面讲的是"每一步算得对不对"，这一页讲的是"**整条链路接得通不通**"。课件给了四类验证，我按"接头—公式—零件—通水"的顺序讲，你就好记了。

第一，连接验证：必需端口均已连接，并且所有连接都有效。这是最底层的一层——源库连不上、Kafka 端口不通、HDFS 没权限，后面全是空谈。它的特点是**便宜且必须先做**：一条连接测试几秒钟，能挡掉后面几小时的排查。

第二，表达式验证：所有表达式都有效。转换规则是用表达式写的——求和、拼接、条件判断、日期函数。表达式写错有两种：语法错（会报错，好办）和语义错（能跑，但算错）。所以这里验的是"**表达式在给定字段上是否成立、结果是否符合预期**"。

第三，对象验证：独立对象定义与映射中的实例匹配。翻成大白话：设计文档里定义了一张"客户宽表"有 30 个字段，实际映射里是不是真有这 30 个、类型对不对、必填的有没有。**这是"设计"与"实现"的对账**。

第四，数据流验证：数据必须能够从源流到目标，而不会在阻止转换处挂起。这句话画面感很强——**通水试压**。把一小股数据放进去，看它能不能一路走到终点；走不到，就卡在哪个"阻止转换处"。

四类的顺序是有讲究的：**从静态到动态，从局部到整体**。连接和对象是静态对账，表达式是半静态，数据流是端到端实跑。做映射验证就按这个顺序，前面不通就别急着往后跑。

> **板书：**四行：① 连接验证（端口通・连接有效）→ ② 表达式验证（表达式有效、语义对）→ ③ 对象验证（定义 vs 映射实例：字段・类型・必填）→ ④ 数据流验证（端到端不挂起＝通水试压）。旁注："静态对账 → 半静态 → 端到端"。

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，但**挂起往往有确定原因**，重启只是把原因藏起来，下次同一时间还会挂。
- 收口：这也是为什么"数据流验证"必须做——**它专治那种不报错、只是不动的问题**。

易错点 / 考点：① 四类验证的名称与验什么，尤其"对象验证"最容易被理解错（它不是验数据对象，是验**定义与映射的一致性**）；② 顺序原则：静态在前、动态在后；③ 常考案例："任务长时间无进展、无报错"——指向数据流验证与上游依赖；④ 记忆口诀："**通不通（连接）→ 算不算（表达式）→ 齐不齐（对象）→ 走不走（数据流）**"。

过渡：到这里，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工作。

【14.9】大数据测试主要工作（3 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是 14.1 的骨架页，信息量最大，我分三层讲：**先看要干哪几件事，再看分几个阶段，最后看每件事的要点清单**。

第一层，四件主要工作：**测试数据准备**、**ETL 测试**、**OLAP 和 Cube 测试**、**报告测试**。这四个顺序就是数据的一生：准备 → 加工 → 多维分析 → 呈现。课件还给了术语，顺手记住：**ETL 用于完成 DB 到 DW（数据仓库）的数据转存**；**OLAP 是在线分析系统**；**Cube 是面向多维数据库的立方体**——Cube 可以理解为"提前算好各种切面"的立方体，切片、切块、钻取都很快。

第二层，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，构成从数据进来到数据出去的完整链条。这里的"专项测试"指的是**性能、数据质量、可恢复性**这类非功能项，跟我们第 7 章的专项测试是一回事，只是对象换成了数据任务。

第三层，要点清单，我把课件三列合成五组给你：

一组是**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。

二组是**元数据与模型**：查看数据字典（元数据）、验证源与目标映射、ETL／DW 的验证体系结构、数据模型验证（建模维度、规范化治理）、索引与分区。

三组是**质量与性能**：数据质量与性能指标。

四组是**运维韧性**：档案与清洗策略、错误日志、异常处理、可恢复性、并行执行与优先级、ETL 的 pull 逻辑、全量与增量策略。

五组是**职责分配**：谁验规则、谁验模型、谁验调度，要在测试方案里写清。

这张清单看着琐碎，它就是大数据测试的日常：**难度不在技术，在"面广"**。

> **板书：**四件事：测试数据准备 → ETL 测试 → OLAP／Cube 测试 → 报告测试。四阶段：采集清洗 → 数据转换 → 专项。要点五组：源头与规则｜元数据与模型｜质量与性能｜运维韧性｜职责分配。术语栏：ETL（DB→DW）・DW 数据仓库・OLAP 在线分析・Cube 多维立方体。

提问：为什么大数据项目里"专项测试"这一块，可恢复性和错误日志的权重比普通项目更高？

预设回答：

- 答"因为数据量大，跑一次要很久"——抓住了要害：**跑一次几小时，失败重跑的代价极高**，所以"能不能从断点恢复"直接决定项目效率。

- 答"因为一错就是全量错"——对，大数据任务往往是批处理，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：记住一句话：**小系统怕出错，大系统怕"出错之后不知道怎么恢复"**。

易错点 / 考点：① 四件主要工作与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/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——完整流程。

【14.10】大数据测试流程（清洗、分析）（3 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页给的是**完整流程**，六段：需求分析 → 设计评审 → 测试框架编写 → 数据准备 → 测试执行 → 现网验证。我按段讲清"这一段验什么"。它和第 4 章的流程高度同构——**换了对象，骨架没变**。

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**（表与表怎么关联）、**计算逻辑**（指标怎么算出来的）、**大数据任务配置**（调度周期、依赖、并发）。

设计评审：评审两样——**代码评审**和**测试框架评审**。注意评审的是"测试框架"——**大数据项目里测试本身也是一套要设计的系统**，工具、脚本、比对逻辑都要评审。

测试框架编写：课件点名三个工具方向——**服务日志构造工具、报表数据构造工具、数据 diff 工具**。为什么是这三个？因为三大痛点是：造不出数据、比不出差异、复现不了现场。

数据准备：验**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**。这里"异常场景"要专门造：空分区、脏数据、超时、乱序。

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**。到了执行阶段，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、现网数据回归测试、现网数据新老版本数据对比**，最后回到**总量上验证**与**多方报表对比验证**。

一句总结这条流程的精神：**大数据测试不是"测一次"，而是"从需求一路盯到现网，用总量和多方对比来兜底"**。

板书：六段横排：需求分析（来源・关系・计算逻辑・任务配置）→ 设计评审（代码・测试框架）→ 框架编写（日志构造・报表构造・diff）→ 数据准备（维度映射・指标逻辑・过滤・异常・配置通道产物路径）→ 测试执行（多方报表对比・接口协议・主流程・调度・告警・日志）→ 现网验证（回归・新老对比・总量验证）。

提问：为什么大数据项目的最后一步一定要做"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是"和以前一样"，**"和以前一样"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"数变了但没人知道"。

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

- 收口：这就是"**总量验证＋多方报表对比**"的分量——**它们是对用例覆盖率的补偿**。

易错点 / 考点：① 六段流程顺序要能复述，考试常以"排序题"或"流程填空题"出现；② 三个工具方向（日志构造、报表构造、diff）是简答高频；③ **执行阶段不只有"对数"，还有调度与告警**，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**上线前用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。于是问题来了：哪些环节能自动化？

【14.11】大数据测试自动化（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲自动化，结构很清楚：**一个目标、三个方向、若干措施、各自适用在哪儿**。

目标：缩短测试过程，提高测试效率和效果。注意后半句——不只是"快"，还要"**效果**"，也就是**能发现手工发现不了的问题**。这一点很重要：自动化的价值不只是省时间，更是**扩大观察面**。人能抽 100 条，脚本能比 100 万条。

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**。这三个方向正好对应测试工作的三个耗时大户：造数据、对数、重复跑。

措施：**服务日志和报表数据的构造工具开发、数据 diff 工具开发、引入接口自动化平台**。

适用：课件把它按数据的三段生命铺开——**清洗阶段、分析阶段、数据服务阶段**，以及三类报表——**离线报表、实时报表、数据服务报表**。这个"适用"列特别值得琢磨：它告诉你自动化不是一把梭，**不同阶段自动化的抓手不一样**：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；数据服务阶段重点是接口自动化，因为对外是 API。

给你一个判断标准：**一件事如果"每次都要做、规则稳定、结果可比对"，就该自动化；如果"每次判断都不同、需要业务经验"，就留给人。** 造数据和 diff 属于前者，判断"这个数业务上合不合理"属于后者。

> **板书：**目标（缩短过程・提高效率与效果）→ 三方向（数据准备・数据验证・回归）→ 措施（日志与报表构造工具・diff 工具・接口自动化平台）→ 适用（清洗／分析／数据服务 三阶段 × 离线／实时／数据服务 三类报表）。右侧：判断标准"重复・稳定・可比对 → 自动化；需经验判断 → 留给人"。

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

预设回答：

- 答"数据验证，因为它最耗时"——有理有据，课件后面那页会给出"耗时缩短 90%"的量化证据，说明验证自动化的收益最直观。

- 答"数据准备，因为没有数据什么都干不了"——也对，且它能**解锁**后面两个：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。排序依据是**"单位投入节省的时间"**。

- 收口：这不是选择题，是**投入产出排序题**；答题时给出排序依据，比给出排序结果更加分。

> **易错点 / 考点：** ① 三个方向与三项措施要能对上号（准备→构造工具；验证→diff 工具；回归→接口自动化平台）；② 自动化的目标里"**效果**"二字别丢；③ 常考判断："自动化能替代人工探索"——错，探索式判断与业务合理性判断仍是人的活；④ 记住"三阶段×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——数据准备。没有数据，后面全停。

【14.12】数据准备自动化（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 这一页把数据准备自动化拆成两个场景：**服务日志构造**和**报表数据构造**，每个场景又给了四个步骤和四类数据源。我按顺序讲。
>
> **服务日志构造**四步：**01 拉取最新日志、02 解析现网日志、03 测试数据构造、04 数据格式转换**。这四步的逻辑是"**从真实里来，再造成我们需要的样子**"。为什么从现网日志来？因为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，往往"太干净"，跑得通但压不出问题。
>
> **报表数据构造**四类数据源，这是这一页最有价值的部分：**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；**配置表全量同步**——配置表小、变化少但影响全局，所以整份同步；**数据表取最新天数据**——只取最近一天，兼顾新鲜度和体量。
>
> 这四类的划分标准是**"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**，课件给了三条：**解决输入输出不明确问题**——注意这条很关键，大数据测试最痛苦的就是"不知道输入是什么、期望输出是什么"，把数据准备标准化，等于把输入输出钉死了；**数据可复用、维护成本低**；**准备时间有效缩减**。
>
> 我补一句工程经验：数据准备自动化的真正价值是**可重复**。手工准备一次数据，三小时后你复现不了；脚本准备的数据，任何时候一条命令就能重建。**可复现，是一切缺陷定位的前提。**

> **板书：** 两场景。服务日志构造：拉取最新日志 → 解析现网日志 → 测试数据构造 → 数据格式转换。报表数据构造：特殊数据（造数平台）｜真实数据（脚本同步）｜配置表（全量同步）｜数据表（最新天）。效果三行：输入输出明确・可复用低成本・准备时间缩减。旁注："可复现才可定位"。

提问：为什么"配置表"要全量同步，而"数据表"只取最新一天？

预设回答：

- 答"配置表小，数据表大"——说出了表层原因。深层原因是**影响面不同**：配置表一行错，全链路结果都错，所以不能漏；明细表取最近一天已足够复现大多数逻辑。

- 答"因为配置表不常变"——正因为不常变，**变了就是大事**，全量同步才能保证"变了立刻被发现"。

- 答"随便，能跑就行"——错答。数据准备策略选错，会出现"本地全绿、现网全红"。

- 收口：所以数据准备的分类标准不是"大小"，而是"**变化特性与影响面**"。

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问题"是简答得分点，很多人只答"省时间"；③ 概念辨析：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"——接近真实是对的，但**必须包含刻意构造的异常数据**，否则压不出问题。

过渡：数据准备好了，接下来就是最耗人的活——验证。这一页的对比会让你明白自动化到底省在哪。

【14.13】数据验证自动化（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页是本节最有说服力的一页，因为它把**手工验证和自动化验证**摆在一起比。我按"维度字段、指标字段、多表"三块对比，最后看利弊。

维度字段测试：手工是"直接获取或联表查询得到"，靠人写 SQL 一条条查；自动化是"程序直接输出不一致的结果"——注意，**自动化输出的不是全部数据，而是"不一致的部分"**。这是个很聪明的设计：人只需要看不一致的那几十行，而不是几万行全看。

指标字段测试：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；自动化则"保留中间计算结果以备排查问题"、"输出每种场景下实际与期望结果"。这两条正是手工的软肋：**手工算完就忘了过程，出问题重算一遍；自动化把中间结果留下来，等于自带证据链。**

多表测试：手工要一张张表对；自动化"可复用、一次运行所有表结果自动输出"。

弊端（手工的）：**容易疲劳测试、易出错；每个指标都需要计算，比较耗时；类似表的处理逻辑，手工计算**

无法复用。这三条你一听就懂——人做重复劳动，前三遍准，第三十遍就飘。

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

我要强调"定位问题方便"这一条：很多人以为自动化的价值是"快"，其实更值钱的是**"准确定位"**。手工对数发现"少了 3 条"，你得再花两小时找是哪 3 条；脚本直接告诉你哪 3 条、差多少、在哪张表。

一句话收口：**自动化不是把人替掉，是把人从"算"里解放出来，去做"判"。**

板书：三行对比表（左手工｜右自动）：维度字段（联表查询 ↔ 只输出不一致结果）｜指标字段（手工算、无中间结果 ↔ 保留中间结果、输出现实与期望）｜多表（逐表对 ↔ 一次跑全部）。手工弊端：疲劳易错・耗时长・不可复用。自动优势：复用高・快（据课件耗时缩短 90%）・定位方便。底部："把人从算里解放出来，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：**如果比对逻辑本身写错了，它会告诉你"全部一致"**，然后你就放心地上线了。

- 答"有风险，比对脚本本身要验"——非常关键。行业里管这叫**"验证器的验证"**：拿一份已知有差异的数据，确认脚本能报出来。

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

- 收口：所以自动化验证上线前，必须做一次**"故意造错"**的自检——**能抓错的比对器，才叫比对器**。

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，考试若问数据要写上"据课件"；③ 核心考点：**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＝数据一定正确"——错，前提是比对逻辑与期望值正确。

过渡：14.1 大数据讲完。数据是原料，接下来看拿到原料之后做出来的东西——AI 系统，它本身怎么测?

【14.14】14.2 AI 系统的测试（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是 14.2 节的地图，标题下面有一行英文：**Test for AI——测 AI**。这四个字是整节的立场：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 AI 系统的不确定性和不可解释性**——先讲清楚为什么 AI 难测；**14.2.2 AI 系统的白盒测试**——钻进模型内部看结构和覆盖；**14.2.3 AI 系统的算法验证**——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。

我用一句话给你立个靶子：**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，你只能看到一个概率分布不对。所以测 AI，方法要换一套。

还有一句提醒：这一节会出现不少数学名词——覆盖率、召回率、AUC、KS、PSI。别被吓住，**它们本质上都是"把'好不好'翻译成数字"的工具**，跟我们第 7 章用响应时间、吞吐量衡量性能是一回事。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。

板书：写"14.2 Test for AI（测 AI，AI 是被测对象）"；四小节竖排：不确定性与不可解释性 → 白盒测试 → 算法验证 → 示例（智能语音）。右侧一行："传统缺陷＝写错了；AI 缺陷＝学偏了"；下方："指标＝把'好不好'翻译成数字"。

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，这一节就要讲。

- 答"能测，但测的不是'对错'，是'够不够好'"——说得很到位，这正是 AI 测试与功能测试的分水岭。

- 答"能测，用准确率就行"——不够：准确率单独看会被数据分布骗，后面会讲为什么要看召回、F1、AUC 和分布稳定性。

- 收口：AI 测试不是"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 与 AI for Test 的方向**必须分清，这是本章第一号易错点；③ 常考判断："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），模型效果部分需要专门方法。

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——图灵测试。

【14.15】Test for AI ~ 从图灵测试说起（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

测 AI 这件事，起点是图灵。1950 年图灵提出那个著名的测试：**隔着一道墙对话，如果人分不清对面是人还是机器，就认为机器有智能**。这个设计的巧妙之处在于——**它不比内部，只比外部行为**。你不打开机器的脑袋看它怎么想，只看它表现像不像人。

课件在这一页点了四个维度，其实是四类"像人"的能力：**计算能力**——算得快算得准；**产生随机数**——能给出不重复、不呆板的输出；**常识**——知道水是湿的、鸟会飞；**推理力**——能从前提推出结论。这四个维度今天看依然有效，你会发现大模型强在前三项，弱在第四项的一致性。

但图灵测试有一个致命局限，也是我们今天必须超越它的原因：**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其像人，同时给出错误的事实；可以在测试集上表现得天衣无缝，换个场景就崩。所以现代 AI 测试从"**行为像人**"转向"**行为可信**"——可信包括准确、稳定、可解释、无偏见、抗攻击。

这个转变你要记住，因为它是整个 14.2 的逻辑起点：**图灵测试给了我们"用行为判智能"的思路，现代 AI 测试在这个思路上加了"用指标判质量"**。

板书：写"图灵测试：隔墙对话，分不清人机 → 有智能"；四维度：计算能力｜产生随机数｜常识｜推理力。下方箭头："判'像不像人' → 判'可不可信'（准确・稳定・可解释・公平・抗攻击）"。

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"——错答，而且是最危险的错答。像人不等于正确，医生要的是**准确率与可追责**。
- 答"不能，还得看准确率和可解释性"——对，医疗场景对错误的容忍度极低，必须看具体指标和可解释性。
- 答"不能，还得看它错了会不会说'我不知道'"——非常好的补充，这叫**拒识能力**，专业场景里比"答得像"更重要。
- 收口：所以图灵测试是**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（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

及公平性、可解释性、安全性；④ 别把图灵测试与"中文屋"混为一谈，后者是塞尔对它的反驳。

过渡：既然不能只靠"像人"来判，那到底怎么测 AI 软件？下一页给答案。

【14.16】如何测试 AI 软件？（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页回答上一页留下的问题，课件给了一句话的定义式回答，我把这句拆开讲，因为它是本节的核心考点。

原话是：**测试 AI 软件，检验它有没有学习能力，本质上就是算法的验证**，即对**元／启发式搜索算法、（深度）强化学习等算法或其组合**进行验证，**一般通过实验进行，借助大量数据进行普适性验证**。

这句话里有四个关键词：

第一，**算法的验证**。AI 软件和普通软件最大的结构差别是：普通软件的核心资产是**代码逻辑**，AI 软件的核心资产是**算法加数据训练出来的模型**。所以你验的重点从"代码分支走没走到"变成"算法在数据上表现如何"。

第二，**元／启发式搜索、强化学习等算法或其组合**。这句话在提醒你：AI 不是一种算法，是一族算法。搜索类、学习类、还有它们的组合，各有各的验证要点。

第三，**通过实验进行**。这是方法论上的关键——AI 测试更像**科学实验**而不是**用例执行**：设假设、控变量、跑多组、看统计。实验就要讲可重复：同一份数据、同一个随机种子，结果要能复现。

第四，**借助大量数据进行普适性验证**。"普适性"三个字最要紧：**在小样本上表现好是容易的，在大量、多样、含噪声的数据上稳定地好，才是真的行**。这就是为什么 AI 测试离不开数据集的建设——训练集、验证集、测试集，还有专门构造的对抗样本集。

一句话总结：**传统测试问"这个输入下输出对不对"，AI 测试问"在这一大批输入上，它的表现稳不稳、够不够好"。**

板书：核心一句："测 AI ＝ 算法验证（元/启发式搜索・强化学习及其组合），通过实验、借大量数据做普适性验证"。四个关键词竖排：算法验证｜算法族｜实验法（可复现）｜普适性（大数据集）。底部对比："传统：单输入对不对｜AI：一批输入稳不稳、够不够好"。

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，就别谈定位。

- 答"要固定随机种子、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：所以 AI 项目的测试报告里，必须写清数据版本与训练配置——**没有这些，数字没有意义**。

易错点 / 考点： ① 这句定义要能背下来，是本节第一名简答题；② 四个关键词能展开；③ "普适性验证"要能解释为"大数据量、多样化、含噪声条件下仍稳定"；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一个闭环——从质量需求到发布。

【14.17】测试方法（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页把 AI 测试组织成一个**闭环流程**，我沿着箭头走一遍：**质量需求 → 验证与测试 → 评估报告 → 发布**，发布结果如果是 Passed 就上线，如果是 Failed 就**反馈**回去，重新调整再测。

起点是质量需求。这一步决定了后面所有指标的选法。你要先回答：这个模型最怕什么错？是漏判（该抓的没抓到）还是误判（冤枉了好人）？风控模型怕漏判，推荐系统怕误判，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，课件点了三样东西：**模型解释**——它为什么给出这个结论，能不能给出人看得懂的理由；**模型检验**——形式化或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**。

这一页还有两个技术点必须讲：**Neuron coverage（神经元覆盖）** 和 **Neuron boundary coverage（神经元边界覆盖）**。这是我们第 8 章覆盖率思想在 AI 上的移植：**神经元覆盖**相当于"语句覆盖"——网络的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"——神经元有没有被推到接近激活阈值的临界区。你这么一类比就明白了：**覆盖率的本质是"我到底测到了模型的多少种状态"，只不过状态从代码行变成了神经元激活**。

最后，课件标了"**结构化／黑盒方法**"两条路线：结构化方法看内部（神经元覆盖属于此类），黑盒方法只看输入输出（测试集评估属于此类）。**两条路线都要走**，跟你第 3 章学的道理一模一样。

板书：闭环箭头：质量需求 → 验证与测试（模型解释・模型检验・测试标准评估）→ 评估报告 → 发布（Passed 上线／Failed 反馈重来）。技术点：Neuron coverage ＝ 神经元的"语句覆盖"；Neuron boundary coverage ＝ 神经元的"边界值"。两路线：结构化（看内部）｜黑盒（看输入输出）。

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。这正是覆盖率最经典的陷阱：**覆盖到不等于验对了**，第 8 章讲过一模一样的话。
- 答"不能，激活了但输出可能仍错"——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，这正是边界覆盖与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**——这句话在传统测试和 AI 测试里都成立。

易错点 / 考点：① 闭环四环节＋反馈回路；② **Neuron coverage 与 Neuron boundary coverage 的类比**是本节最容易被问的辨析题；③ 结构化与黑盒两条路线要能对应到具体方法；④ 常考判断："覆盖率越高模型越可靠"——错，覆盖率衡量充分性，不衡量正确性。

过渡：要看懂 AI 测试插在哪儿，得先看清机器学习的开发流程。下一页是一张流程图。

【14.18】机器学习开发流程&架构（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是一张架构图，课件没有配文字，我带你看出**三件事**，看完你就知道测试该往哪儿插。

第一件事，这张图有明确的分层。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提供预测；再往上还有一层**监控与反馈**，把线上的真实表现收回来。你以后看任何一个 AI 系统的架构图，先找这五层在不在——**缺监控的那张图，一定是个没上过线的系统**。

第二件事，每一层之间都有"契约"。数据层交给特征层的是"什么格式、什么口径的数据"；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——契约清楚，就能在每一层之间设检查点。

第三件事，测试的插入点不止一个。很多同学以为"AI 测试就是最后测一下模型准不准"，其实在这张图上，**每个层间接口都是一个测试点**：数据层出来要验数据质量，特征层出来要验特征一致性，模型层出来要验指标，服务层出来要验接口与性能，监控回路要验告警是否准。这就是我们在第 4 章讲的"**测试贯穿全生命周期**"，在 AI 系统上的具体样子。

记住这张图的用法：**当有人跟你说"我们的 AI 效果不好"，你就顺着这五层往下问——是数据脏了、特征偏了、模型弱了，还是服务慢**。

板书：五层自下而上：数据层 → 特征层 → 模型层 → 服务层 → 监控与反馈（回路）。左侧竖写"层间契约＝测试检查点"；右侧五个测试点：数据质量｜特征一致性｜模型指标｜接口与性能｜告警准确性。

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，可以先排除。
- 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，而且可能把问题带进新版本。
- 收口：正确顺序是"**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＝测试检查点"这个观点能显著提升论述题层次；③ 线上效果下降的排查顺序：数据漂移 → 服务／依赖 → 模型版本；④ 别把这张图只当"开发流程"，它同时是**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——五步，三个质量。

【14.19】机器学习研发生命周期：简化模型（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页把上一张架构图压成一条五步时间线：**定义问题 → 数据收集 → 特征工程 → 模型训练 → 部署**。五步之下挂着三个质量，这是本页真正的重点：**数据质量、特征质量、模型质量**。

先说三个质量怎么挂：**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、稳不稳定；**模型训练对应模型质量**——学出来的模型准不准、泛不泛化。部署之后呢？部署之后是**回归**：一旦数据或特征变了，模型质量要重新验。

这三层质量的排序有讲究，它和我们讲"缺陷越早发现越便宜"是同一个道理，而且在这里更极端：**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**。所以行业里有句话：**AI 项目八成的功夫在数据和特征上**。这句话不是贬低算法，而是在说**质量的重心在上游**。

再说第一步"**定义问题**"，很多团队跳过了它，直接开训。但定义问题决定了后面一切：这是分类还是回归？判错的代价对称吗？要多快出结果？**问题定义错了，你连"什么是好模型"都不知道**，评估指标也就无从选起。

这一页和整门课的关系，我给你点一下：**这就是"测试左移"在 AI 项目里的样子**——测试不是在模型训练完之后才进场，而是从"定义问题"就开始参与：帮业务把验收标准说清楚，帮数据团队把质量规则定下来。**你越早进场，越能省下后面几周的返工。**

板书：五步横排：定义问题 → 数据收集 → 特征工程 → 模型训练 → 部署（→ 回归）。三质量分别挂在下三步：数据质量｜特征质量｜模型质量。右侧竖写："质量重心在上游：数据 > 特征 > 模型"；底部："测试左移＝从定义问题就进场"。

提问：一个团队模型指标一直上不去，你会建议他们先调模型，还是先回去看数据？

预设回答：

- 答"先调模型，换更强的网络"——最常见的错误反应：换模型能带来几个点的提升，但**数据错了，天花板就在那儿**。

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

- 答"两个一起做"——可以，但要**先做一次数据体检**，否则你不知道模型调优的效果是真是假。

- 收口：所以"定义问题—数据—特征"这三步，是 AI 项目里**最该被测试盯着**的三步。

易错点 / 考点：① 五步顺序与三层质量的对应关系；② 常考简答："机器学习项目的三个质量层次是什么"——数据质量、特征质量、模型质量；③ 高频判断："模型质量是 AI 项目最重要的质量"——错，数据与特征质量是前提；④ 结合"测试左移"答论述题，是本页最好用的加分点。

过渡：道理都懂，可为什么实际做起来这么难？下一页把难点摊开。

【14.20】机器学习测试难点（3 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

> 这一页列了机器学习测试的难点，我按"人—数据—节奏—标准"四组讲，每组我都补一句"所以测试该怎么应对"。
>
> **第一组，人的门槛：技能、知识要求高——抽象、因素多、数学深；数据获取、问题分析难。**这一条最实在：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、懂一点统计学。**应对：**团队里要有人专门负责评价指标与数据集，别指望一个人什么都干。
>
> **第二组，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，必须靠分析工具。**应对：**把数据质量做成常规检查项，像我们做回归一样做"数据体检"。
>
> **第三组，节奏问题：模型优化慢、上线才刚开始、迭代慢。**这三句要连起来读：训练一次几小时甚至几天，上线后效果还要靠线上数据慢慢验证，所以**迭代周期天然比普通软件长。应对：**把可自动化的部分（数据检查、指标回归、A/B 分流分析）自动化，缩短每次迭代的验证时间。
>
> **第四组，也是最要命的一组：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：**验收标准必须在项目开始就写清三件事——**指标是什么、在哪个数据集上算、达到多少算过。**
>
> 四组难点用一句话收口：**AI 测试难，不难在"会不会用工具"，难在"敢不敢在标准不清的时候把标准定清楚"。**这恰恰是测试人员最不可替代的价值。

> **板书：**四组：人（技能门槛・数学深・取数难）｜数据（质量要求高・维度多）｜节奏（优化慢・上线才开始・迭代慢）｜标准（验收不确定・效果不确定）。每组右侧一句"应对"。底部一行："难不在工具，在把标准定清楚"。

> **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。**准确率在什么数据上算？** 换个分布可能只剩 70%。而且类别不均衡时，准确率高但漏判严重。

- 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"——把责任推回去了，但作为测试你要**提供选项和代价**：召回提高会带来多少误报、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 常考判断："AI 项目上线就意味着项目结束"——错，课件明确说"上线才刚开始"，线上监控与迭代才是主战场；④ 结合"不确定的验收标准"答"测试人员价值在哪"，是很稳的论述题素材。

> **【思政落点 1】**：AI 测试最容易被误解的一句话是"让工具去判"。工具能算准确率、能跑覆盖率，却不会替你决定"这个模型能不能上线"。这个决定要人签字，签了就要担责任——风控、医疗场景里，一个指标背后是一个人的机会。所以学 AI 测试，第一课不是算法，是**边界感**：清楚工具给了什么证据，也清楚判断必须由人负责。

> **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）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是"特点"，课件没有铺文字，我用五组对比把它讲实——**每一组都拿传统软件测试做参照**，你就知道新在哪儿。

第一，输入不确定。传统软件的输入是有限的、可枚举的：三个参数、每个几档取值，组合得完。AI 的输入是高维的：一张图片就是几百万个像素，你永远枚举不完。所以 AI 测试**不能靠穷举，只能靠采样与分布**——你测的是"这一类输入"，不是"这一个输入"。

第二，输出是概率化的。传统软件输出 3 就是 3；模型输出"0.87 的概率是猫"。于是判据变了：不是"等不等于"，而是"**置信度够不够、阈值定得合不合理**"。

第三，没有唯一预期结果。同一张图，两个模型给出不同答案，可能都对；甚至同一模型两次推理结果略有不同。这就逼着我们从"断言"转向"**指标＋容差**"。

第四，缺陷不必然复现。传统缺陷"照步骤必现"，AI 的缺陷可能是"在某类边缘样本上偶发"。所以缺陷报告

要带**样本**，不能只带步骤。

第五，要双视角。既要像普通软件那样验工程（接口、性能、异常处理），又要像科学家那样验效果（指标、分布、稳定性）。**这两条腿缺一条都站不住。**

合起来一句话：**机器学习测试，是把"判定对错"升级成"评估好坏"，把"用例"升级成"数据集"，把"复现步骤"升级成"复现样本"。**

板书：五组对比（传统 ↔ ML）：输入可枚举 ↔ 高维只能采样；输出确定 ↔ 概率化；有唯一预期 ↔ 无唯一预期；缺陷必现 ↔ 偶发需样本；只验工程 ↔ 工程＋效果双视角。底部："判对错 → 评好坏；用例 → 数据集；步骤 → 样本"。

提问：模型"偶发判错"的缺陷单，你要求提交者必须带什么？

预设回答：

- 答"带上出错的样本"——最基本也最重要：没有样本，别人无法复现，也无法判断是不是数据问题。

- 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 场景下"步骤"往往就是"喂这个样本"，关键在样本本身。

- 收口：所以 AI 缺陷单的模板要改：**样本 + 版本 + 配置 + 期望与实际输出**。

易错点 / 考点：① 五组特点能对比着说；② "输出概率化"与"无唯一预期"是最本质的两条；③ 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；④ 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：特点决定了测试的重心。下一页给出一张金字塔，告诉你力气该往哪儿使。

【14.22】机器学习测试的重点（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是一张分层图，回答"力气往哪儿使"。我按从下往上的顺序讲，你会发现它和我们第 5 章讲的金字塔是同一个思路，只是把"单元"换成了"数据"。

最底层是数据与数据监控：数据测试、Data Monitoring。为什么数据在最底下？因为它是地基，**地基歪了，**

上面全歪。数据测试验的是质量六性（下一页专门讲）；数据监控是上线后盯着输入分布有没有漂移。

第二层是特征：**特征测试**。特征是从数据加工出来的、真正喂给模型的东西。特征错了有两种典型：算错（逻辑 bug）和穿越（用了未来信息）——**特征穿越是 AI 项目最隐蔽的缺陷之一**，离线指标漂亮得不像话，一上线就崩。

第三层是模型：**模型测试**，配合 **ML Infrastructure Tests**、**Unit Tests**、**Integration Tests**。这一层既包括模型本身的指标验证，也包括支撑模型的基础设施测试——训练管线、特征服务、推理服务，这些都是普通软件，该做单元和集成测试就得做。

第四层是系统运行：**Running／System**，包括 **Prediction Monitoring** 和 **System Monitoring**。预测监控看的是"模型输出的分布有没有变"，系统监控看的是"服务还活着吗、延迟多少"。

这张图最重要的信息是**面积比例**：**数据和特征占的面积远大于模型**。这在告诉你一件反直觉的事——**AI 项目的测试工作量，大部分不在"测模型"，而在"测数据和特征"**。很多团队把 80% 精力放在调模型上，结果反复在数据和特征上翻车。

板书：四层自下而上：① 数据（数据测试・Data Monitoring）→ ② 特征（特征测试）→ ③ 模型（模型测试・ML Infra Tests・Unit・Integration）→ ④ 系统运行（Prediction Monitoring・System Monitoring）。右侧写"面积＝工作量：数据+特征 > 模型"。

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"——正是如此：用了未来信息，模型"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，天然有未来数据"——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"——也是现实原因，特征往往是 SQL 或脚本，缺评审。

- 收口：所以特征测试要把"**时间切分是否正确**"列为必查项。

易错点 / 考点：① 四层结构与各层测试项；② 核心结论：**工作量重心在数据与特征**；③ 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"；④ 判断题陷阱："模型测试就是 AI 测试的全部"——错。

过渡：既然重心在地基，我们就把地基讲透——先看数据质量的六个维度。

【14.23】数据质量（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页给数据质量下了六个维度，这六个词是 AI 测试里最基础也最常考的一组，我一个个讲，每个配一个反例，你听完就能记住。

正确性（Accuracy）：数据是否正确体现真实的业务或可证实的来源。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（字段／维度／参数等）是否充分、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）：数据是否被一致的定义或理解。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——**注意，一致性讲的常常不是数据错，而是口径不统一**。

有效性（Validity）：数据是否在业务定义的、可接受的范围之内。反例：利率字段出现 -3%，或者枚举值出现了系统里根本没定义过的状态。

时效性（Timeliness）：数据是否已过时。反例：特征表 T+3 才更新，可业务要的是实时风控——**数据是对的，只是来晚了，等于没用**。

可获取性（Accessibility）：数据是否易于获取与使用。反例：数据躺在某个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，但它是实打实的质量维度**。

六个维度给你一个记忆抓手：**对不对（正确）、全不全（完整）、同不同（一致）、在范围内（有效）、新不新（时效）、拿不拿得到（可获取）**。以后做数据体检，就按这六问过一遍。

> **板书：**六行：正确性（对不对）｜完整性（全不全）｜一致性（同不同，含口径统一）｜有效性（在不在范围内）｜时效性（新不新）｜可获取性（拿不拿得到）。底部："数据体检六问"。

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。如果两边各自的计算都符合自己的口径，那数据本身是对的。

- 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极高，**解决靠的是指标定义管理，不是改代码**。

- 答"完整性问题"——不太对，完整性讲的是字段是否齐备。

- 收口：所以遇到"两个数对不上"，第一步先问"**两边的口径定义一样吗**"，别急着查 bug。

> **易错点 / 考点：** ① 六个维度的**英文名与中文名**都要会，考试常给英文让翻译；② 一致性与正确性的区别是最高频辨析：**正确性是"和事实符不符"，一致性是"彼此的理解统不统一"**；③ 时效性容易被忽略但案例常见（实时风控用 T+1 数据）；④ 记忆口诀："对、全、同、规、新、得"。

> **过渡：** 六个维度是"查什么"，下一页讲"怎么查"——四类分析方法。

【14.24】数据质量分析方法（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页给四种分析方法，每一种都对应一句"我们要回答的问题"，你把这四句问句记住，方法自然就记起来了。

第一问：某模块涉及哪些表？表的关联关系是什么？ 对应的分析动作是**输出血缘关系图、做血缘基数分析**。血缘图回答"这个指标是从哪些表、经过哪些加工来的"，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，直接决定了聚合会不会重复计算。**血缘是排查的导航图**，没有它，你只能凭记忆猜。

第二问：数据在时序上是否存在问题？ 对应**波动分析**。看指标随时间的变化曲线：突然跳变、周期性尖峰、断崖下跌。波动分析最擅长抓两类问题——**漏数**（某天数据没进来，曲线掉零）和**重复**（重跑导致数据翻倍）。

第三问：每个字段是否有问题？ 对应**值域分析和数据分布分析（异常分析）**。值域看边界：年龄不该是 200、金额不该为负；分布看形状：某个枚举值占比从 30% 变成 3%，一定有事发生。

第四问：数据是否一致？ 对应**有效字段一致率**。同一业务实体的关键字段在不同表里应该对得上，一致率掉下来就说明某一环处理有问题。

四问连起来是一条排查路径：**先看结构（血缘）→ 再看时间（波动）→ 再看单字段（值域分布）→ 最后看跨表（一致率）**。这也是我给你的一套固定套路，做数据体检按这个顺序走，不会漏。

> **板书：** 四问四法：涉及哪些表、怎么关联 → 血缘关系图・血缘基数｜时序有无问题 → 波动分析｜每个字段有无问题 → 值域分析・分布分析｜数据是否一致 → 有效字段一致率。底部路径："结构 → 时间 → 字段 →

跨表"。

提问：某天报表的订单量突然翻倍，其他指标正常，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，一眼就能定位到具体日期。

- 答"查是不是任务重跑了"——非常准，翻倍最常见的两个原因就是**重跑未去重**和**关联导致行数放大**。

- 答"查值域"——方向偏了，值域看的是单字段取值范围，翻倍是量的问题。

- 收口：量的问题看波动与血缘基数，值的问题看值域与分布——**先分清楚是"量错"还是"值错"**。

易错点 / 考点：① 四问与四法的对应关系是送分题；② 血缘基数分析能解释"为什么关联后行数变多"；③ 高频案例：数据翻倍 → 重跑未去重／一对多关联；数据掉零 → 上游分区缺失；④ 判断题陷阱："分布分析只看单个字段"——错，分布也包括多字段组合分布。

过渡：数据层的分析方法有了，往上一层是特征——特征测试有五个关键点。

【14.25】特征测试的关键（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲特征测试的五个关键点，加两条工程手段。五个关键点我按"准不准—有没有用—守不守规—稳不稳"来讲。

第一，特征阈值。特征要离散化成区间才能用，阈值定在哪儿直接决定模型行为。阈值测试要验：边界值附近的表现、阈值变更后的影响面。

第二，特征相关性／重要性。这个特征对结果有没有贡献？贡献多大？没有贡献的特征是噪声，会拖累模型还可能带来过拟合。测试要能回答"去掉它，指标掉多少"。

第三，特征与结果变量的关系。这一条比相关性更进一步：**关系方向对不对**。收入越高违约率越低，这是业务常识；如果模型学出反方向，说明数据或标签有问题。这类检查行话叫"**业务合理性校验**"，是人工经验最值钱的地方。

第四，特征适合性与计算成本。特征好不好算、要多久算出来、能不能实时拿到。一个离线要跑两小时的特

征，放进实时风控里就是灾难——**特征是好的，但用错了地方**。

第五，特征合规性。这条最容易被技术同学忽略，却可能让整个项目下架：用了不该用的个人信息、用了带歧视性的字段（比如性别、地域直接进模型）。**合规不是法务的事，是设计阶段就该问的事**。

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，输入固定样本、断言输出值；**特征工程的静态检查**——扫代码里的硬编码、除零风险、时间函数误用。再补一个词：**特征一致性**，也就是离线算出来的和线上算出来的必须一致，这是下一页的主题。

板书：五点：阈值｜相关性/重要性｜与结果变量的关系（方向对不对）｜适合性与计算成本｜合规性。两手段：单元测试（定样本断言输出）｜静态检查（硬编码・除零・时间函数）。右侧："特征一致性＝离线＝在线"。

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"不能，得看能不能改成实时计算"——对，要么重构计算方式，要么降级用一个近似特征。

- 答"可以先离线算好缓存起来"——可行，但要注意**时效性**：缓存的特征可能是昨天的，风控要的是"此刻"。

- 收口：所以特征测试不只问"准不准"，还要问"**来不来得及**"——这就是适合性与计算成本的意义。

易错点 / 考点：① 五个关键点要能列举，尤其"合规性"最容易被漏；② "特征与结果变量的关系"要能举例说明方向性检查；③ 简答高频："特征测试包括哪些内容"；④ 判断题陷阱："相关性越高的特征越好"——错，还要看稳定性、成本、合规性和是否造成过拟合。

过渡：把五个关键点放进一个真实场景，看它变成什么样——风控模型。

【14.26】特征测试要点示例：风控模型场景（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是上一页的落地版。课件把风控特征测试要点摊成一大片，我按"离线—在线—一致性—规范"四组归好，你就不用死记几十个词。

第一组，特征离线。要验：**计算逻辑**是否正确（HQL 写得对不对）、**特征回溯**能不能跑出历史值、**分区修改**是否生效、**性能（计算时长）**能不能在窗口内跑完、**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特征记录）、**Shuffle（乱序）**之后**特征产出顺序**是否仍然稳定。最后这条的成因是：分布式计算乱序，若代码隐含"按到达顺序取第一条"，结果就随机漂移，**这类 bug 只在数据量大时出现**。

第二组，特征在线。要验：**接口响应时间**、**特征在线**取值是否正确、**覆盖度**是否足够（该有特征的样本不能大面积为空）、**异常测试**（下游超时、字段缺失时的降级行为）。

第三组，一致性，这是本页的核心：**前后一致**（同一主体不同时间取到的同一特征不乱变）、**离线在线一致**（同一个样本，离线算一遍、线上取一遍，值必须相同）、**覆盖率一致**（离线覆盖 90%，线上不该只有 60%）、**特征值一致**、**特征 KS 验证**（特征的区分能力在线上是否还在）。**离线在线不一致是风控最致命的问题**：离线模型 AUC 0.8，上线跌到 0.6，八成就是这里出了问题。

第四组，工程规范：**命名规范**、**编码规范**、**空字符处理**、**异常处理**、**跑批日志**、**特征调度**、**monitoring 效果**、**loan_dt 校验**（业务日期口径要统一，取错一天全盘错）、**响应时长与跑批时间**。

一句话总结：**风控特征测试，一半在算得对，一半在对得上。**

> **板书：**四组：离线（计算逻辑・回溯・分区・计算时长・四元组顺序・Shuffle 后产出顺序）｜在线（响应时间・取值・覆盖度・异常降级）｜一致性（前后・离线在线・覆盖率・特征值・KS）｜规范（命名・编码・空值・异常・日志・调度・日期口径・耗时）。底部："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"线上数据分布和训练集不一样"——第二大嫌疑，也就是数据漂移。
- 答"模型过拟合了"——可能，但通常离线验证集就能看出来；上线才崩往往指向工程链路。
- 收口：所以排查顺序是"**先验一致性，再验分布，最后才怀疑模型**"。

易错点 / 考点：① 一致性五项（前后・离线在线・覆盖率・特征值・KS）是案例题核心；② "Shuffle 后特征产出顺序"这类分布式陷阱要能解释成因；③ 业务日期口径（loan_dt 一类）取错是真实高频事故；④ 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，最终要落到指标上。下一页是特征效果的一整套指标。

【14.27】特征效果分析指标（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

> 这一页是特征效果指标的汇总，课件没有铺文字，我讲清"三类指标各回答什么问题"，你以后看到任何一张特征评估表都能读懂。
>
> **第一类，区分度指标**，代表是 **KS** 和 **AUC**。它们回答的问题是：**这个特征能不能把好坏样本分开**。KS 看的是"好坏两个分布的最大差距"，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"。这类指标最常用于风控、评分卡。
>
> **第二类，稳定性指标**，代表是 **PSI**。它回答的问题是：**这个特征在不同时间、不同人群上的分布变了没有**。PSI 小说明稳定，PSI 大说明人群结构变了，模型可能失效。**注意区分度与稳定性是两件事**：一个特征可以区分度很高但极不稳定，上线三个月就废掉。
>
> **第三类，覆盖率与缺失类指标**。它回答：**这个特征有多少样本能取到值**。覆盖率低意味着模型在大量样本上其实是在"盲猜"，指标再好也不可信。
>
> 三句话记住这张表：**KS／AUC 看能不能分得开，PSI 看会不会变，覆盖率看有没有值**。选特征的时候三个都要看，缺一个就可能踩坑——只高不稳定的特征，是"上线即报废"；覆盖率高但区分度低的特征，是"占着位置不干活"。

> **板书：**三类：区分度（KS・AUC）｜稳定性（PSI）｜覆盖与缺失（覆盖率・缺失率）。右侧三句："分得开吗・会变吗・有值吗"。

> **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，以及能否重算分箱。

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

- 收口：所以指标要**成组看**，单看一个数一定会被骗。

易错点 / 考点：① 三类指标的用途分工要清；② **KS 与 AUC 都在衡量区分度**，别把 KS 说成稳定性指标；③ PSI 的经验阈值（0.1 以下稳定、0.25 以上需关注）可作参考，答题时注明"经验值"；④ 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：要把"离线在线一致"验实，需要一套架构。下一页就是它。

【14.28】特征稳定性测试架构（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲怎么把"一致性"验成事实。架构的核心思想一句话：**同一批样本，两条路径取特征，比差异**。

先看三条比对路线：离线 diff——两个离线环境（比如新老两版特征逻辑）各算一遍，比差异；**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**——这是最关键的一条：离线算一遍、线上取一遍，比同一个样本的特征值。

再看数据流：先**样本获取**，然后分别**获取离线特征**、通过 **MapReduce 请求**在线的不同环境、调**在线接口**拿在线特征，全部送进 **Diff 处理模块**，由**统一 diff 脚本**产出 **Diff 处理结果**。

结果要看四样，这是本页的精华：**各环境下的整体覆盖率**——先确认两边都取到了值，否则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**——把"两边都有值但值不同"的样本挑出来，这些才是真问题；**覆盖率不同的样本**——一边有一边没有，属于取值链路问题，跟数值不一致是两类缺陷，要分开报。

最后两个衍生指标：**相同覆盖样本下的整体 diff 率**和**覆盖率不同的样本占比**。这两个数放在一起，就是特征稳定性的体检报告：**前者高＝算错了，后者高＝取不到**。

工程上再提醒一句：diff 一定要**固定样本集与时间窗口**，否则每次跑出来的样本不同，diff 率没有可比性。

板书：路线：离线 diff｜在线 diff｜在线/离线 diff。流程：样本获取 → 取离线特征＋在线接口／MapReduce 各环境 → Diff 处理模块 → 统一 diff 脚本 → 结果。结果四项：整体覆盖率｜每维特征 diff 率｜相同覆盖下 diff 样本｜覆盖率不同样本。底部："前者高＝算错，后者高＝取不到"。

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"——结论方向对，但缺依据：**要先看这 15% 集中在哪几维特征**，如果集中在两三个特征上，多半是逻辑差异；如果均匀分布，可能是时间窗口不一致。

- 答"要看业务容忍度"——专业：不同特征的重要性不同，核心特征 1% 的 diff 也不能接受。

- 答"看是不是浮点精度问题"——很好的补充，浮点与精度截断会制造大量"假差异"，比对前要归一化。

- 收口：所以 diff 报告不能只报一个总数，必须**按特征维度拆开报**。

> **易错点 / 考点：** ① 三条 diff 路线与在线／离线这条的分量；② 结果四项要能列举，尤其"覆盖率不同的样本"与"相同覆盖下的 diff 样本"是**两类不同缺陷**；③ 比对前要做归一化（精度、空值表示、时区）；④ 判断题陷阱："diff 率越低越好"——不能为降 diff 率而放宽比对标准，那是自欺。

> **过渡：** 特征验完，最后是模型本身。模型怎么验？先看方法论。

【14.29】算法验证（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页给算法验证的六种手段，我按"人看—数据看—指标看—换着看—攻着看—上线看"的次序讲。

第一，Review（评审）。看算法设计是否合理、假设是否成立、有没有明显的方法性错误。这是最便宜也最容易被跳过的一步——**评审能挡掉的问题，别留到跑实验**。

第二，验证／测试数据集。要有独立的验证集和测试集，而且要和训练集**在时间上、人群上合理切分**。切分错了（比如随机切分时间序列数据），指标就会虚高。

第三，具体验证指标。也就是下一页要讲的准确率、召回、F1、AUC、MSE 这一套。**指标要先定，再跑**，不能跑完挑一个好看的。

第四，交叉验证（CV，cross validation）。把数据分成 K 份，轮流做验证集。它的价值是**降低"一次切分带来的偶然性"**，让指标更可信。小样本场景尤其必要。

第五，对抗验证。这一招很巧：训练一个分类器去区分"训练集样本"和"测试集样本"，如果它能轻松区分开，说明**两份数据分布差异大**，你的离线指标不可信。**对抗验证是分布一致性的探测器**。

第六，A/B 测试。前面五种都是离线，A/B 是唯一能回答"上线后到底有没有用"的方法：分流一部分用户用新模型，和对照组比业务指标。

六种手段其实覆盖了三个层次：**设计层（评审）、离线层（数据集·指标·CV·对抗验证）、线上层（A/B）**。三层都过，才敢说这个算法是"验过的"。

板书：六手段竖排：Review｜验证/测试数据集｜具体验证指标｜交叉验证 CV｜对抗验证｜A/B 测试。右侧三层：设计层｜离线层｜线上层。旁注："对抗验证＝分布一致性探测器"。

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，但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，随机切分在时间序列上反而会制造泄漏。

- 收口：记住这三件套：**切分要合理、CV 降偶然、对抗验分布**。

易错点 / 考点：① 六种手段名称与用途；② **对抗验证的原理**是简答高频，要能说清"分类器分得开＝分布不一致"；③ A/B 测试是唯一线上验证手段，注意它与离线指标的分工；④ 判断题陷阱："交叉验证能解决数据泄漏"——不能，泄漏要靠切分方式解决。

过渡：六种手段里，"具体验证指标"要专门讲一页——因为它决定你能不能读懂一张模型评估表。

【14.30】算法评估指标（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页把评估指标分成两大类：**回归问题和分类问题**。你先分清问题类型，再选指标，顺序不能反。

回归问题——预测的是一个连续数值，比如预测房价、预测违约损失金额。课件给了四个指标：**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，单位与原数据一致，好解释；**可释方差分数**，看模型解释了多少方差；**R2 分数（决定系数）**，取值越接近 1 说明拟合越好，等于 0 相当于"只用均值预测"。**MSE 与 MAE 的分工**要讲清：MSE 惩罚大错，MAE 反映平均偏差——怕大错就用 MSE，要业务口径就用 MAE。

分类问题——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 曲线与 AUC**。

ROC 曲线这一页给了定义，必须记准：横轴是 **False positive rate（FPR，特异度那一侧）**，纵轴是 **Recall（true positive rate，TPR，灵敏度）**；**AUC 是 ROC 曲线下的面积**。AUC 的直观含义：随机取一正一负样本，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**，所以特别适合类别不均衡的场景。

再补一句方法论：**指标必须和业务代价挂钩**。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——这两种错的代价差得远，指标就该偏向召回。**选指标，本质是在选"我们更怕哪种错"。**

板书：两类。回归：MSE（罚大错）｜MAE（可解释）｜可释方差｜R2（越近 1 越好）。分类：准确率・精准率・召回率・F1｜ROC（横轴 FPR、纵轴 TPR）・AUC（曲线下面积，不受阈值与类别比例影响）。底部："选指标＝选更怕哪种错"。

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，漏判的代价通常是直接资金损失。

- 答"精准率，别打扰好人"——在某些场景（比如直接冻结账户）也成立：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：召回提高后多出来的"疑似"要靠人工核，**人力是约束条件**。

- 收口：所以正确的做法是给出"**召回—精准**"的权衡曲线，让业务在代价可承受的点上做决定。

易错点 / 考点：① 回归四指标与分类四指标不能混；② **ROC 的横纵轴**是送分题也是最易错题（横 FPR、纵 TPR）；③ AUC 的含义与"不受阈值影响"这一优点必考；④ 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：分类指标里还有一组进阶概念——Micro 与 Macro，以及 P-R 曲线。

【14.31】分类算法评价指标（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲三个进阶点：**P-R 曲线**、**Micro 与 Macro 平均**、以及怎么用这两者的差异定位问题。

先看 P-R 曲线。课件给了两句判断：**精确率越高，则模型对负样本的区分能力越强**；**召回率越高，则模型对正样本的区分能力越强**。为什么这么讲？因为精确率高意味着"我判成正的里面假阳性少"，也就是**没把负样**

本误认成正的；召回率高意味着"真正的正样本我基本都抓到了"，也就是**正样本没漏**。这两者天生互相拉扯，调阈值就是在这条曲线上移动。

再看 Micro 与 Macro。多分类问题里，要把各个类别的指标汇总成一个总指标，有两种汇总法：**Micro** 是把所有类别的 TP、FP、FN 合起来算一次，**大样本类别影响更大**；**Macro** 是先算每个类别的指标再取平均，**每个类别权重相同，小样本类别影响更大**。

选择规则课件给得很清楚：**类别比较均衡时，用 Micro 或 Macro 都行；认为大样本类别应该占更重要位置，用 Micro；认为小样本类别应该占重要位置，用 Macro**。

最有价值的是这两条判断，考试非常爱考：**如果 Micro 明显低于 Macro，意味着在大样本类别中出现了严重的分类错误；如果 Macro 明显低于 Micro，意味着在小样本类别中出现了严重的分类错误**。为什么？因为 Micro 被大样本类别主导，它低说明大类别差；Macro 被小类别拉平权，它低说明小类别差。**记住这个方向，案例题就是白送的分。**

板书：P-R：精确率高 → 对负样本区分强；召回率高 → 对正样本区分强。Micro（合并算，大样本主导）｜Macro（分类算再平均，小样本主导）。判断：Micro ≪ Macro → 大样本类别严重错；Macro ≪ Micro → 小样本类别严重错。

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：如果小类别恰好是高风险类别（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**样本太少或特征不具区分度**，加数据不加小类样本没用。

- 收口：所以看多分类报告，**两个指标一起看**，差异就是诊断线索。

易错点 / 考点：① Micro 与 Macro 的计算差别与主导类别；② 两条差异判断的方向**极易记反**，建议记"**谁低谁背锅**"——Micro 低是大类别错，Macro 低是小类别错；③ P-R 曲线的两句判断要能解释；④ 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，是否"公平"取决于业务目标。

过渡：概念讲完，来算一遍。下一页是一个十只猫的具体例子。

【14.32】示例：分类问题指标计算（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是必考的计算题，我带你算一遍，顺便把混淆矩阵四个格子彻底讲清。

先立场景：一共 10 个样本，模型判断"是不是猫"。四个格子的含义：

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，模型说不是猫，**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：

准确率（Accuracy）＝(TP+TN)/(TP+TN+FP+FN)——所有样本里判对的比例。

精准率（Precision）＝TP/(TP+FP)——**我说是猫的里面，真的是猫的占多少**。注意分母是"我判成正的"，所以精准率衡量"我的判断有多可信"。

召回率（Recall）＝TP/(TP+FN)——**真的猫里面，我抓到了多少**。分母是"实际为正的"，所以召回率衡量"我漏没漏"。

套课件给的数据：**准确率＝7/10＝70%，精准率＝3/4＝75%，召回率＝3/6＝60%**。

现在解读这三个数，这才是这道题的价值：准确率 70% 看着一般；精准率 75% 说明"我说是猫时四次里对三次"；召回率 60% 说明"**六只真猫我漏了两只**"。如果业务是"抓猫"，漏掉 40% 不能接受——**这时该调低阈值把召回提上来，代价是精准率下降**。

一句口诀：**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"**。但准确率最容易被不均衡骗：10 个里只有 1 只猫，全判"不是猫"也有 90%。

板书：混淆矩阵四格：TP（是猫判是）｜FN（是猫判否，漏）｜FP（非猫判是，冤）｜TN（非猫判否）。三公式：Accuracy＝(TP+TN)/全部；Precision＝TP/(TP+FP)；Recall＝TP/(TP+FN)。代入：准确率 7/10＝70%、精准率 3/4＝75%、召回率 3/6＝60%。口诀："精准看判得准，召回看漏多少，准确看整体"。

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"——正确方向：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。精准与召回是**此消彼长**的关系，这正是 P-R 曲线存在的理由。

- 收口：所以面对业务需求，测试要给的不是"一个更好的模型"，而是"**不同阈值下的指标组合，让业务选**"。

易错点 / 考点：① 混淆矩阵四格的定义，**FP 与 FN 最容易颠倒**：F 后面的 P/N 指的是"模型判成什么"，不是"实际是什么"；② 三个公式必会，代入数要能算对；③ 高频陷阱题："准确率 90% 的模型好不好"——必须问类别分布；④ 记住精准率分母是"判为正的"，召回率分母是"实际为正的"，这一条能救很多分。

过渡：分类算完了，回归类怎么验？下一页给一个六步流程。

【14.33】示例：回归类算法验证（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是回归类算法的验证示例，课件用六个步骤把流程串起来，我按顺序讲，你重点记"每一步在防什么错"。

第一步，明确预测目标与业务口径。回归预测的是一个数值，先要定清它是什么：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——时序数据必须按时间切，不能随机切。

第三步，选定评价指标。MSE、MAE、可释方差、R2 里选，选的标准是"**业务最怕什么样的误差**"：怕大偏差用 MSE，要业务可解释用 MAE。

第四步，训练与调参。注意调参只能用验证集，**测试集只能用一次**，用多了就是拿测试集训练。

第五步，用测试集评估并做残差分析。残差分析是回归特有的关键动作：看误差是不是随机分布，如果误差呈现系统性偏移（比如总低估大额样本），说明模型结构有问题，光看一个 MSE 数字看不出来。

第六步，与基线对比并决定是否上线。基线可以是"用均值预测"或"用上一版模型"。**没有基线的指标是没有**

说服力的——R2 等于 0.3 到底好不好，取决于基线是多少。

六步的共性，我总结成一句：**回归验证不是算一个误差数，而是证明"误差小得有意义、错得没规律"**。

板书：六步：定目标与口径 → 划分数据集（时序按时间切）→ 选指标（MSE 罚大错／MAE 可解释）→ 训练调参（测试集只用一次）→ 测试集评估＋残差分析 → 与基线对比。底部："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，如果基线是 0.1 就还行"——正确思路：**指标必须相对基线看**。

- 答"要看业务能不能用"——也很关键，有些场景 0.3 的解释力已足够排序使用。

- 收口：所以回归报告里必须有"**本版 vs 基线**"这一栏，没有对比就没有结论。

易错点 / 考点：① 六步顺序与每步防的错；② 残差分析是回归特有的、易漏的步骤；③ "测试集只能用一次"是纪律性考点；④ 判断题陷阱："R2 越高模型一定越好"——可能过拟合，且不同任务不可横向比较。

过渡：模型指标只是"效果"，AI 系统还要过六道专项关卡。

【14.34】AI 算法的专项测试（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页列了 AI 算法的六个专项，正好对应我们第 7 章讲的"非功能专项"，只是对象换成了 AI。我一个个讲，每个都给你一句"怎么测"。

数据隐私：训练数据里有没有不该有的个人信息？模型会不会把训练数据"背"出来？测法：数据脱敏检查、成员推断测试（判断某个样本是否在训练集里）、输出内容审计。

预测效率：一次推理要多久、吞吐多少、显存占多少。测法：压测、批量与单条对比、长尾输入的耗时分布。**注意 AI 的耗时对输入长度极敏感**，测的时候要覆盖极端长度。

可靠性：输入扰动下输出稳不稳。下一页那个熊猫的例子就是典型。测法：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（性别、地域、年龄）的表现有没有系统性差异。测法：分组指标对比，看各组召回率、误判率是否差得离谱。**公平性测试最容易被忽略，也最容易引发舆情。**

可解释性：模型能不能给出人理解的理由。测法：特征重要性是否与业务常识一致、局部解释（比如"因为 A 特征高所以拒贷"）是否合理。

安全性：会不会被恶意输入攻击——对抗样本、提示注入、数据投毒。测法：红队测试、异常输入轰炸。

六个专项有一条共同点：**它们都不是"功能对不对"的问题，而是"这个 AI 能不能被信任"的问题**。这也回到本章开头那句话：AI 测试的终点是**可信**。

> **板书：**六项竖排：数据隐私｜预测效率｜可靠性｜公平性｜可解释性｜安全性。每项右侧一句测法要点。底部："AI 测试的终点是可信"。

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，因为很多团队觉得"我们的数据是客观的"。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，尤其在"效果优先"的团队。后果是出了问题无法向监管和用户解释，金融、医疗场景直接不满足合规要求。

- 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，通常不是因为它不重要，而是因为它不好测**——这恰恰是测试人员该补位的地方。

易错点 / 考点：① 六项名称要能默写；② 每项的典型测法至少记一个；③ 简答高频："AI 系统的专项测试包括哪些方面"；④ 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，而是**不存在系统性偏差**。

过渡：六项专项靠什么方法验？下一页给五类常用方法，正好一一对应。

【14.35】常用的测试方法（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页把 AI 测试的五类方法摆开，每一类都配了"用在什么对象上"和"具体手段"。我按顺序讲。

第一，蜕变测试（metamorphic testing）。这是 AI 测试里最重要的方法之一，思路很妙：**我不知道正确答案是什么，但我知道"输入按某种方式变化后，输出应该怎么变"**。课件给的例子全在这：**类别标签乱序**——把标签的顺序打乱，预测结果不该变；**属性乱序**——把特征列顺序调换，结果不该变；**增加无信息属性**——加一列随机噪声，结果不该变；**一致重复预测**——同样输入多跑几次，结果应该一致。这四条都是"**根据公式特点，特定变换数据，结果比较**"。**蜕变成分的价值在于：它绕开了"没有预期结果"这个死结。**

第二，效果测试。用的是**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"。

第三，数据测试。对象是数据：**ETL 质量、特征质量、用 Fuzz 构造异常数据**。看的是**分布、diff、性能时长**。

第四，常规逻辑。就是我们学了一学期的传统测试：**业务逻辑、性能**。别以为讲了 AI 就可以不测这些——**接口该测还得测，性能该压还得压**。

第五，小样本试验／A/B。**样本验证评估**，小流量先试，看真实业务指标。

五类方法的分工很清楚：**蜕变测试解决"没有 oracle"，效果测试解决"好不好"，数据测试解决"原料行不行"，常规逻辑解决"工程对不对"，A/B 解决"上线有没有用"**。

板书：五类：蜕变测试（标签乱序・属性乱序・加无信息属性・重复一致）｜效果测试（KS・PSI・精确・召回・F1・AUC）｜数据测试（ETL・特征・Fuzz；分布/diff/耗时）｜常规逻辑（业务・性能）｜小样本/AB。底部："蜕变治没有 oracle，效果治好不好"。

提问：为什么"把特征列的顺序打乱，预测结果不该变"能算一种测试?

预设回答：

- 答"因为模型不该依赖列的顺序"——正是。如果打乱顺序结果变了，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，**这正是蜕变测试的精髓**：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，多次打乱才能暴露问题。

- 收口：记住蜕变测试的公式：**输入变换 + 输出关系断言 ＝ 无需 oracle 的测试**。

易错点 / 考点：① 蜕变测试的四类变换与原理，是本节最高频考点；② 五类方法的分工要能对上号；③ "没

有 oracle（预期结果）"是 AI 测试的根本困难，蜕变测试是对它的解法；④ 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，看一个让人印象深刻的例子——一张熊猫的照片。

【14.36】示例：可靠性测试（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是 AI 可靠性测试最经典的一个例子，课件描述得很清楚：**用 FGSM 方法生成的一个对抗样本——一张干净的熊猫图片，被模型正确分类为熊猫；中间是添加的扰动；扰动后的熊猫图片，被模型识别成长臂猿。**

三句话，但信息量极大。第一，**模型原本是对的**——它认识熊猫，这说明模型不是没学好。第二，**扰动是"看不见"的**——人眼看过去还是熊猫，甚至可能觉得只是噪点。第三，**结论完全变了**——从熊猫变成长臂猿，而且置信度往往还很高。

这说明什么？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。**只要在梯度方向上做一点点精心设计的偏移，就能让模型的高维决策边界翻转**。FGSM 的做法就是沿着损失函数梯度方向加扰动，一步到位。

这对测试的启示有三条：**第一，准确率高≠可靠**——在测试集上 99% 的模型，可能被一张加了噪点的图骗过去；**第二，必须做对抗测试**——把对抗样本作为常规测试集的一部分，而不是等出了安全事故才想起；**第三，对抗训练是防御手段之一**——把对抗样本混进训练数据，能提升鲁棒性，但要持续更新，因为攻击方法也在进化。

再延伸一句：这个例子不只适用于图像。语音加一点人耳听不见的噪声、文本加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"**，这就是可靠性专项存在的理由。

板书：三步：干净熊猫图（判对）→ 加扰动（人眼不可见）→ 判成长臂猿（结论翻转）。启示三条：准确率高≠可靠｜必须做对抗测试｜对抗训练可提升鲁棒性。旁注："凡是模型，都有对抗样本"。

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，扰动本来就是为人眼不可见而设计的。

- 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：所以对抗测试必须工具化、自动化，靠人工抽查是发现不了的。

> **易错点 / 考点：**① 这个例子的三句话要能复述（干净判对→加扰动→判成另一种）；② FGSM 的定位：**基于梯度的对抗样本生成方法**；③ 简答高频："对抗样本说明了什么"——模型决策依赖的特征与人不同，准确率不等于鲁棒性；④ 别把对抗样本与"数据噪声"混为一谈，前者是**恶意构造**的。

> **过渡：**14.2 讲完——被测对象是 AI。现在掉个头：让 AI 来帮我们做测试。

【14.37】14.3 AI 助力软件测试（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 这一页是 14.3 的地图，标题六个字：**AI 助力软件测试**。请注意和上一节的区别：上一节 **Test for AI**，AI 是被测对象；这一节 **AI for Test**，AI 是工具。**方向一反，整个立场就变了。**
>
> 五小节：**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，我用一句话概括：**把测试中最"机械"的部分交给机器，把最"判断"的部分留给人。** 你听的时候留心一件事：每一种 AI 应用，解决的其实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。

> **板书：**标题写"14.3 AI for Test（AI 是工具）"，与上节"Test for AI"并排对比。五小节竖排；右侧一行："机械的交机器，判断的留给人"；下方："痛点在前，AI 在后"。

> **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，考试里方向混淆会直接丢分。

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

- 收口：所以读论文、看工具时，先判断"AI 在这篇文章里是主语还是宾语"。

> **易错点 / 考点：** ① 两节方向必须分清，这是本章第一号易错点；② 五小节名称要能列举；③ 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、工具五方面答；④ 记忆抓手："**Test for AI 找它的 bug，AI for Test 用它找 bug**"。

> **过渡：** 先看全景——测试的智能化到底包含哪些环节。

【14.38】测试的智能化（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

> 这一页给了一张"智能化测试"的全景清单，我按测试流程的顺序重排一遍，你就看得懂它的野心了。
>
> **智能的单元测试**——自动生成单元测试、自动补 mock、自动找边界。**智能的 UI／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与冗余，推荐该补哪些。**质量风险评估的自我调整**——根据历史缺陷分布、代码改动热度，动态调整测试重点。**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、自动修复依赖。
>
> 这一段清单说明什么？说明智能化的目标不是"替掉某一个环节"，而是**把整条测试链路的每个环节都加上一层智能**。课件给了一句英文口号，很值得抄在笔记上：**Test smarter, not harder**——**更聪明地测，而不是更拼命地测**。
>
> 课件还列了几个工具名：**Applitools、Appvance IQ、Eggplant AI、Test.AI、Functionize**。这些名字不用背，但你要知道它们各自代表的方向：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 主打从用户行为学习，Functionize 主打 AI 驱动的端到端测试。
>
> 提醒一句：**这些工具解决的是"降低门槛、提高覆盖"，不解决"该测什么"**。选题、定标准、判风险，还是人的活。

> **板书：** 七行：智能单元测试｜智能 UI/API｜脚本生成｜用例优化补充｜质量风险自调｜缺陷诊断机器人 DDB｜环境智能运维。中间一行大字："Test smarter, not harder"。工具：Applitools・Appvance IQ・

Eggplant AI・Test.AI・Functionize。底部："工具降门槛，不替你定标准"。

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"——这是最需要纠正的判断。工具不知道这个版本的业务重点在哪，也不知道哪个风险不能冒。

- 答"剩判断：测什么、测到什么程度、能不能上线"——准确，这正是"**测试设计**"与"**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：所以能力模型要往"**懂业务 + 会设计 + 能判断**"上走，而不是往"会点工具"上走。

易错点 / 考点：① 七类智能化应用要能列举；② 记住口号 "Test smarter, not harder" 及其中文意思；③ 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与判据；④ 工具名不必全记，但要知道它们代表的**方向**。

过渡：逐个看案例。第一个是最"朴素"的 AI——图像识别。

【14.39】AI 应案例：SikuliX（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲 **SikuliX**，它是"用图像识别做 UI 测试"的经典代表，也是很多同学第一次接触"AI 帮测试"的地方。

先说它解决什么问题。传统 UI 自动化靠什么定位元素？靠 **DOM 结构、XPath、控件 ID**。这套办法有三个死穴：**一是脆**——开发改了个 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，比如 **Canvas 绘制的界面、远程桌面、老式桌面客户端、游戏画面**；**三是跨技术栈难**——Web、桌面、移动三套定位方式完全不同。

SikuliX 的思路是把问题反过来：**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。**定位的依据从"代码结构"变成了"视觉效果"**，于是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：一是它的 API 完全实现为**由目标机器上运行的服务支撑的 REST-API**——这句话的意思是它做成了**服务化**形态，测试机与被测机可以分离，这对后面讲"测试服务化"是个伏笔；二是 **SikuliX1 演进到**

SikuliNG，说明这个技术路线一直在维护。

它的局限也在这里：**图像匹配怕分辨率变化、怕主题换色、怕字体渲染差异**，所以工程上要控制运行环境的一致性。

板书：痛点三行：定位脆（改 class 就废）｜无 DOM 可依（Canvas・远程桌面・桌面客户端・游戏）｜跨栈难。SikuliX 思路：截图 → 屏上找图 → 点坐标；＝"用眼睛定位"。形态：REST-API 服务化（测试机/被测机分离）；版本：SikuliX1 → SikuliNG。局限：分辨率・主题・字体差异。

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，后者需要阈值、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）"——非常到位，这是图像方案落地时最大的坑。

- 收口：所以工程上是**互补**：能用 DOM 用 DOM，DOM 不可靠时用图像——**技术选型看场景，不看时髦**。

易错点 / 考点：① SikuliX 的定位原理（截图匹配 + 坐标点击）；② 它最适合的场景：**无 DOM 可依**（Canvas、桌面客户端、游戏、远程桌面）；③ "REST-API 服务化"这个点可用来答"测试工具形态演进"；④ 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：图像识别之上还有一层更强的能力——不光认出"这是一张图"，还能认出"这是一个按钮"。

【14.40】AI 应用案例：目标检测技术（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲**目标检测**在测试里的用法，课件没有配文字，我讲清"它比图像匹配强在哪"。

先说区别。上一页的图像匹配是"**找这张特定的图**"——你给它一张按钮截图，它去找这一张。目标检测是"**找出这一类东西**"——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的按钮框出来，还给出位置和置信度。

这个能力对测试意味着什么？**三点。**

第一，**脚本不再依赖"这张截图"**。按钮换了颜色、换了文案、换了尺寸，只要它还是按钮，目标检测照样能认出来——**这叫泛化**，图像匹配做不到。

第二，**可以做界面结构分析**。把一屏的元素都框出来，就能自动判断"这个页面上有几个可点区域""布局有没有错位""有没有元素被遮挡"。这类检查人工做很费眼，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"。

但它的代价也要说清：**目标检测需要训练数据**——你得先标注大量界面截图，这是不小的投入；而且**识别有置信度**，低置信度的结果需要人工复核。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**——这也正是"人机协同"的典型形态。

> **板书：**图像匹配（找"这一张"）↔ 目标检测（找"这一类"）。三个用途：脚本不依赖具体截图（泛化）｜界面结构分析（数量・错位・遮挡）｜视觉回归（多／少／移）。代价：需标注训练・有置信度。策略："高置信自动、低置信转人工"。

提问：目标检测识别出一个按钮，置信度只有 0.4，脚本该不该点？

预设回答：

- 答"该点，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"不该点，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"——非常专业：**操作的危险等级决定阈值高低**，删除类操作阈值该设得极高。
- 收口：所以"AI 辅助测试"的落地关键，往往不是模型多准，而是**阈值与兜底策略设计得好不好**。

易错点 / 考点：① 图像匹配与目标检测的区别（特定 vs 类别）；② 三个典型用途要能列举；③ "置信度阈值 + 人工兜底"是人机协同的标准答案；④ 判断题陷阱："识别准确率 95% 就能全自动跑"——还要看错误代价与兜底机制。

过渡：把目标检测、游戏 AI 这些能力打包，就有了一套现成的开源工具包。

【14.41】AI 驱动的智能测试（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页介绍腾讯的 **GameAISDK**，课件的定义是：**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**。

我把它拆成三层能力讲，这样你能看清它为什么叫"工具包"而不是"一个工具"。

第一层，UI 检测与 UI 元素识别。就是上一页讲的目标检测能力——从游戏画面里找出按钮、血条、对话框、道具。**没有这一层，一切自动化都无从谈起**，因为游戏界面根本没有 DOM。

第二层，AI 算法。工具包内置了训练与推理能力，你可以用它训练自己的识别模型。**这就是"工具包"和"工具"的区别**：工具给你结果，工具包给你造结果的能力。

第三层，面向游戏的测试与评估。游戏测试的难点在于：**状态空间巨大**（一局游戏的可能走法无穷多）、**判定标准主观**（好不好玩）、**复现困难**（同一个操作序列结果可能不同）。所以游戏测试必须靠 **bot 自动探索**——让 AI 去玩，玩出人想不到的路径，玩出崩溃和卡死。

这一页的方法论价值，我给你提炼成一句：**当被测系统的界面不可编程、状态空间又巨大时，"让 AI 去操作、让 AI 去探索"是唯一可行的自动化路线。** 这句话不只适用于游戏，也适用于移动 App、IoT 设备、车载系统。

还有一个现实提醒：游戏 AI 测试出来的问题，往往不是"功能错了"，而是"**在某些极端操作序列下崩了**"。这类缺陷用传统用例永远发现不了，**只能靠大规模随机探索 + 异常检测**。

> **板书：**三层：UI 检测与元素识别（无 DOM 的界面）→ AI 算法（训练与推理，可自定义）→ 面向游戏的测试评估（bot 自动探索）。右侧写游戏测试三难：状态空间巨大｜标准主观｜复现困难。底部："界面不可编程 + 状态巨大 → 让 AI 去探索"。

提问：为什么游戏测试特别依赖"机器人自动玩"，而不是录脚本回放？

预设回答：

- 答"因为游戏状态太多，脚本覆盖不全"——核心原因：录脚本只能走一条路，而游戏的可能路径是爆炸性的。
- 答"因为脚本一改版本就失效"——实情，游戏更新频繁，录制的脚本极脆。
- 答"因为要模拟真人操作习惯"——也对，bot 能模拟各种操作节奏，暴露时序类缺陷。
- 收口：所以"随机探索 + 异常检测"是这类系统的标准打法，**覆盖靠随机，判定靠监控**。

易错点 / 考点： ① GameAISDK 的三层能力要能列举；② 游戏测试三难（状态空间、标准主观、复现困难）是简答素材；③ "界面不可编程时用 AI 探索"这一结论可迁移到 App、IoT、车载；④ 判断题陷阱："AI 探索能替代人工试玩"——体验类判断仍要靠人。

过渡： 游戏画面里还有一种更常见的东西——文字。识别文字，有专门的技术。

【14.42】AI 应用案例：OCR（2 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页讲 **OCR**，课件给了一句非常具体的话：**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**。这句话看着朴素，其实是移动端测试的一个大痛点，我给你讲透。

先说 **webview 是什么**。移动 App 里嵌网页很常见——活动页、支付页、客服页，很多都是 webview 加载的 H5。问题来了：**自动化工具拿不到里面元素的属性**。原生控件的 ID、层级都能读，webview 里的内容对工具来说往往是一整块"黑盒"——**你能看到，但读不到**。这就是"元素无法识别"。

OCR 怎么解决？**既然读不到属性，那就读像素上的字。** 把 webview 区域截图，做文字识别，得到"这里写着'确认支付'"，再按文字位置去点击。**定位依据从"控件属性"换成了"屏幕上的文字"**，跟 SikuliX 的思路一脉相承，只是把"图像匹配"换成了"文字识别"，更稳、更通用。

它还有三个衍生价值：**一是跨技术栈**——原生、webview、小程序、Flutter 画出来的界面，只要上面有字，就能认；**二是可做内容断言**——不只是点，还能验"页面上有没有出现'支付成功'"；**三是可做多语言适配测试**——同一套脚本，换语言包照样跑。

局限也要讲：**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**；**纯图标按钮没有文字**，得退回图像匹配。所以工程上常见组合拳：**能读属性就读属性，读不到就 OCR，OCR 不行再图像匹配，三层兜底。**

板书： 痛点：webview 元素读不到属性（原生可读、H5 是黑盒）。解法：截图 → OCR 取文字与位置 → 按文字点击。三价值：跨技术栈｜内容断言｜多语言适配。局限：字体/背景/手写体/纯图标。兜底三层次：属性 → OCR → 图像匹配。

提问： 一个纯图标按钮（只有图形没有文字），OCR 能定位吗？该怎么办？

预设回答：

- 答"不能，没有字可识别"——正确判断。

- 答"退回图像匹配，或者用目标检测认这个图标"——标准兜底方案，就是上面说的三层策略。

- 答"让开发加个无障碍标签（contentDescription）"——非常专业：**推动被测系统可测性，也是测试人员的职责**，这属于"可测试性设计"。

- 收口：所以好的测试方案不只选技术，还会**反过来要求产品提供可测的接口**。

易错点 / 考点：① OCR 解决的核心问题：**webview 元素无法识别**（这句话是课件原话，常直接考）；② 三种定位手段的适用与优先级（属性 → OCR → 图像）；③ "可测试性设计"是加分概念；④ 判断题陷阱："OCR 只用于识别验证码"——用途远不止，界面元素定位、内容断言都在用。

过渡：文字之后是语言。NLP 能把"人话"变成"用例"。

【14.43】AI 应用案例：NLP（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲 **NLP** 在测试里的应用，课件给了四个动作，我按顺序讲，这四个动作其实是一条完整的流水线。

第一步，约定测试用例描述规范。这是整件事的前提，也是最容易被跳过的一步。你要让机器读懂用例，用例就得有**结构**：前置条件、操作步骤、预期结果，用相对固定的句式写。**为什么这一步是前提？** 因为 NLP 再强，也架不住"随便写"——同一件事五种说法，机器很难稳定抽取。所以行业里的做法是**先定模板，再上模型**。

第二步，建立专业知识图谱。这一步解决"机器不懂业务"的问题。测试用例里全是业务名词——"授信额度""逾期天数""T+1 对账"，模型不认识就没法理解意图。知识图谱把这些概念、它们之间的关系、以及同义词都建起来，**机器才能把"额度超限"和"超过授信上限"当成一回事**。

第三步，与用例管理系统打通。这一步是工程落地的关键。生成的用例不能躺在某个脚本目录里，必须回到团队真正在用的系统（比如 Jira、TestLink、PingCode），否则没人会去用它。**工具的价值取决于它嵌入了多少现有流程。**

第四步，一键批量转化。这是最终的用户体验：选中一批需求文档或文字用例，一键转成可执行的测试脚本或结构化用例。

四个动作合起来，就是**"规范 → 知识 → 集成 → 转化"**的闭环。给你一个判断：**NLP 转用例的准确率，八成取决于前三步做得好不好，而不是模型选得大不大。**

板书：四步流水：约定用例描述规范（结构化句式）→ 建立专业知识图谱（概念・关系・同义词）→ 与用例管理系统打通（落回现有流程）→ 一键批量转化。底部："规范 → 知识 → 集成 → 转化"；"准确率八成取决于前三步"。

提问：为什么"约定用例描述规范"要放在用 NLP 之前，而不是之后？

预设回答：

- 答"因为机器需要稳定输入"——对，输入的规范性决定了抽取的稳定性，这是**数据决定上限**。

- 答"因为先有规范才能批量"——也对，规范是批量的前提，不然每条都要人工校正。

- 答"规范太死会限制表达"——这是真实顾虑，所以要**在结构化与自然表达之间取平衡**：关键字段结构化，描述部分自由。

- 收口：所以这件事本质是**流程改造**，不是**模型选型**——很多团队搞反了顺序，才觉得"AI 不好用"。

易错点 / 考点：① 四个动作要能按顺序列举；② "先规范、后模型"这个顺序是核心观点，简答高频；③ 知识图谱在测试中的作用要能解释（解决业务术语理解）；④ 判断题陷阱："上了 NLP 就能自动生成全部用例"——生成的是草稿，评审与补充仍是人的活。

过渡：再换一种 AI——这次不是识别，是"进化"。让机器自己进化出会玩的 bot。

【14.44】AI 应用案例：神经元演化算法（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲的案例很有意思，课件原文我念一遍：**采用 Monte Carlo 树搜索算法、自动启发式构建算法、增强拓扑的神经元演化算法来训练 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**。作者署名是 Alexander Andelkovic，King／Midasplayer AB——就是那家做 Candy Crush 的公司。

三个算法名字听着唬人，我翻译成大白话：

Monte Carlo 树搜索——**边试边算**。不把所有可能都算完，而是随机采样一些路径，根据结果反过来把资源集中到有希望的路径上。围棋 AI 就是这么干的。

自动启发式构建——自己总结经验规则。人写启发式是"我觉得该先点这里"，它是让算法从数据里自己长出这类规则。

增强拓扑的神经元演化算法（NEAT）——不光调参数，还改结构。普通神经网络结构是人定的，NEAT 让网络自己增删节点和连接，边进化边变复杂。

然后是那六步训练流程，这是本页的骨架：**获得训练数据：预测成绩 → 创建随机的机器人（bots）→ 竞争测试（计算成绩）→ 创建新的机器人 → 选择最好的机器人（基于 ANN）→ 基于 ANN 选择移动、继续操作游戏**。你看，这是个标准的**进化循环**：随机生成一批 → 让它们互相竞争打分 → 留下好的、变异出新的 → 用神经网络把"好 bot 的行为"学下来 → 用它继续玩。

最后产出的**多个 bots 里挑出最合适的 bot**。为什么要有多个？因为不同类型的游戏、不同关卡需要不同策略。

这个案例的方法论价值一句话：**当"正确的操作序列"无法人工定义时，就用"成绩"作为进化目标，让机器自己搜索出来。**

板书：三算法：Monte Carlo 树搜索（边试边算）｜自动启发式构建（自长规则）｜NEAT（连结构一起进化）。六步循环：取训练数据（预测成绩）→ 建随机 bot → 竞争测试算分 → 造新 bot → 按 ANN 选优 → 按 ANN 选动作继续玩。产出：多个 bot → 最合适的 bot。

提问：这套"进化出 bot"的方法，能直接搬到 Web 系统的功能测试上吗？

预设回答：

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

- 答"不能，因为 Web 测试有明确判据，不需要进化"——说得对：**进化搜索适用于"没有明确操作路径、只有结果评分"的场景**，比如游戏、探索式测试。

- 答"可以用在探索式测试里"——很专业的迁移：把"覆盖率""崩溃数"作为成绩，让 bot 去探索。

- 收口：所以方法迁移的关键不是技术，是**先问"这个场景有没有可计算的成绩函数"**。

易错点 / 考点：① 三个算法名称及各自作用；② 六步训练流程的顺序（竞争测试 → 选优 → 学习 → 继续操作）；③ 该方法的适用前提：**有可计算的评分、缺少人工可定义的路径**；④ 判断题陷阱："强化学习/进化算法可以替代测试设计"——它替代的是探索，不是设计与判断。

过渡：自动化跑完一堆用例，最烦的是"红了 300 条，从哪儿看起"。AI 在这里能帮大忙。

【14.45】AI 应用案例：失败用例分析（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲**失败用例分析**，这是我认为对一线测试人员最实用的一页。场景你肯定遇到过：一晚上回归跑了 3000 条用例，红了 300 条，你打开报告，满屏红色，从第一条开始看，看到第 20 条已经不知道自己在看什么了。

课件给的解法分两条线：**离线分析**和**在线分析**。

离线分析先做**粗清理**：拿**用例全量执行日志**和**报错日志**做**相同失败用例聚类**——300 条红色，可能只有 7 个真实原因。聚类把"同一原因导致的失败"归成一堆，你的工作量立刻从 300 降到 7。然后再做**精细清理**：对每一堆深入分析，找到根因。这一步用**智能化算法辅助分析**，产出**日志聚类信息**。

在线分析是靠**失败用例分析服务（Service）** 加 **JavaAgent** 做的，粒度是**单用例级**。它的核心动作是**建立用例级多源数据画像**——把跟这条用例有关的一切都收集起来：**执行环境、用例执行日志、业务日志、用例脚本、用例覆盖代码、调用链日志、业务配置、流水线部署参数**，还要区分**服务自身**的问题和**服务依赖系统**的问题。

有了画像，就能给出**失败原因推荐**：**可疑点 1、可疑点 2、可疑点 3、可疑点 4**——注意它是给"可疑点"而不是给"结论"，这是个很诚实的设度。再配合**人工反馈**，把人的判断回灌，模型越来越准。

价值在哪？**它把"300 条红"变成"7 个原因 × 每条的可疑点排序"**。测试工程师的时间，应该花在判断"这是不是真 bug"上，而不是花在"这条为什么红"的检索上。

> **板书：**两条线。离线：全量执行日志＋报错日志 → 粗清理（相同失败用例聚类）→ 精细清理（智能算法辅助）→ 日志聚类信息。在线：失败用例分析服务＋JavaAgent（单用例级）→ 用例级多源画像（执行环境・执行日志・业务日志・脚本・覆盖代码・调用链・业务配置・部署参数；服务自身 vs 依赖系统）→ 失败原因推荐（可疑点 1–4）→ 人工反馈。

提问：为什么系统给的是"可疑点 1 2 3 4"而不是直接给"根因"？

预设回答：

- 答"因为不准，怕误导"——实在的原因。失败原因常常是环境、数据、代码交织，机器给唯一答案反而危险。

- 答"因为要人做最终判断"——正是，这就是"人机协同"的边界：**机器缩小范围，人下结论**。

- 答"可以给根因，模型够强就行"——过于乐观，测试失败的原因分布极其长尾，模型很难覆盖。

- 收口：所以这类工具的设计哲学是"**给排序，不给判决**"——这也是我们后面对 AI 工具应有的期待。

> **易错点 / 考点：** ① 离线与在线两条线的分工（离线做聚类，在线做画像与推荐）；② "用例级多源数据画像"包含哪些数据源，是简答高频；③ 粗清理与精细清理的区别；④ 判断题陷阱："AI 能直接定位根因"——它给可疑点排序，结论要人定。

> **过渡：** 最后一个案例，把 AI 用在接口测试的"精准调度"上。

【14.46】基于形式化语言的 API 智能测试（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 这一页信息密度很高，讲的是**怎么用 AI 让 API 测试"又准又省"**。我按数据流给你串一遍：左边进来的是各种"素材与反馈"，中间是两组模型库，右边出去的是"精准的测试任务"。
>
> **先说输入**：**问题反馈、执行日志、在线数据、交付场景、测试数据**——注意这几样里最值钱的是**问题反馈和执行日志**，它们是"真实世界告诉你的信息"，比任何人工设计都贴近实际。
>
> **再说中间的加工**：一条线是**特征数据清洗**——用**特征提取规则库**和**数据清洗模型库**做清洗，再**去量纲化**（把不同量纲的指标标准化，否则没法放在一起比较）。另一条线是**模型参数训练**——用**数据样本集**做**真实数据学习**，再加**强化学习**和**加速学习**，并**周期性同步**。**周期性同步**这四个字很关键：模型不是训一次就完，要跟着线上变化滚动更新。
>
> **然后是两组库的产出**：**测试脚本库**（**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**）和**测试模型库**。这几项里，"**变更-测试脚本关联关系**"是精准测试的核心——**代码改了哪一行，能映射到哪些脚本**，这样就不必全量回归。
>
> **最后是执行侧**：**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**，采集分析的结果又回灌到输入，形成闭环。
>
> 一句话总结这一页：**精准测试的本质是"建立变更与用例之间的映射"，AI 只是让这个映射更自动、更准。**

> **板书：** 输入（问题反馈・执行日志・在线数据・交付场景・测试数据）→ 加工（特征提取规则库＋清洗模

> 型库→去量纲；样本集→真实学习＋强化学习＋加速学习→周期性同步）→ 两库（测试脚本库：覆盖度组合・参数关联・变更-脚本关联・相似度聚类；测试模型库）→ 执行（筛选与模式选择 → 精准调度执行 → 管理平台 → 采集分析）→ 回灌闭环。

> **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"因为能只跑相关的用例，省时间"——正是它的直接价值。

- 答"因为全量回归跑不完"——现实约束，尤其在微服务架构下，一个改动牵动的服务可能几十个。

- 答"因为怕漏测"——这个担心也对：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：精准测试永远有一对矛盾：**省时间 vs 怕漏测**，靠"映射质量 + 定期全量兜底"来平衡。

> **易错点 / 考点：**① 这一页的数据流主线要能复述（输入→清洗/训练→两库→执行→回灌）；② "变更-测试脚本关联"是精准测试核心，必考；③ 去量纲化的作用要能解释；④ 判断题陷阱："精准测试可以完全取消全量回归"——需要定期全量兜底来校验映射完整性。

> **过渡：**到这里 AI 帮测试的五类案例都看完了。下一节换个角度：工具本身，未来会变成什么样。

【14.47】低代码测试平台（1 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

> 这一页讲**低代码测试平台**，课件只有标题，我讲清一件事：**"低代码"到底低的是什么**。
>
> 先看传统自动化的门槛。写一套 UI 自动化，你要会：一门语言、一个测试框架、一套定位方式、一套断言库，还要会搭环境、会做数据准备。这套技能的门槛不低，结果是——**能做自动化的人少，能维护自动化的人更少**。行业里有个很常见的现象：自动化项目第一年搭起来很漂亮，第二年没人维护，第三年全废——**不是技术不行，是维护成本压垮了它**。
>
> 低代码要解决的正是这个问题。**"低"的不是能力，是"写代码的量"**：把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；把定位方式做成配置而非脚本；把数据驱动做成表格而非代码。测试人员拖拽编排，平台负责生成与执行。
>
> 它带来的三个变化：**一是参与面变宽**——业务测试人员也能上手，不必等开发支援；**二是维护成本下降**——

改一个步骤改配置就行，不用改代码；**三是标准化**——大家都用同一套组件，脚本风格统一，交接容易。

但代价也要说清：**平台能力有边界**，遇到复杂逻辑（复杂的前置数据构造、非标准协议、特殊加密）就绕不过去，还是要写代码；**平台本身是新的依赖**——平台升级、平台 bug、平台锁定，都是新风险。所以我的判断是：**低代码适合覆盖"大量、标准、重复"的场景，不适合覆盖"少量、特殊、复杂"的场景。**

> **板书：**传统自动化四道门槛：语言｜框架｜定位｜维护。低代码"低"什么：封装操作组件、配置化定位、表格化数据。三个变化：参与面宽｜维护成本降｜标准化。边界：复杂逻辑仍要写代码；平台本身是新依赖。底部："低代码覆盖大量标准重复，不覆盖少量特殊复杂"。

提问：低代码平台让"不会编程的人也能做自动化"，这是好事还是有隐患？

预设回答：

- 答"好事，参与的人多了"——是好事，但要配套：**组件与用例规范**必须跟上，否则会造出大量无人维护的脚本。
- 答"有隐患，脚本质量参差"——真实风险，尤其断言写得随意，会出现"跑绿了但没验到东西"。
- 答"看平台有没有强制规范"——很到位：好的平台会强制断言、强制命名，把规范做进工具。
- 收口：所以选低代码平台，第一看**断言能力**，第二看**复用与分层能力**，第三才看拖拽好不好看。

易错点 / 考点：① "低代码"的准确含义（降低编码量，不是降低能力）；② 三类收益与两类风险要能列举；③ 简答高频："低代码测试平台适合什么场景"；④ 判断题陷阱："低代码可以完全替代编码自动化"——复杂场景仍需要代码能力。

过渡：低代码平台最典型的能力，是把"用例"画成一张图。

【14.48】低代码测试平台- 续（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页给低代码平台的两个核心能力，课件说得很准，我逐句讲。

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。这句话讲的是**用例建模**——不是"写一条条步骤"，而是"画一张流程图"。节点是操作，边是流转，条件分支画成岔路。为什么这对复杂场景特别有用？因为**复杂用例的难点不在执行，在"想清楚"**：有几种分支？异常路径怎么走？前置状态有几种？写成文字步骤，这些关系是隐性的；画成图，**关系变成显性的**，漏掉的分支一眼就能看出

来。这其实和我们第 2 章讲的"因果图"、第 3 章讲的"判定表"是同一个思路——**用结构化图形管住组合爆炸**。

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。 这句话的价值比第一句还大，因为它把测试的介入点**左移到了需求阶段**。你想：需求文档里写"用户可以修改订单"，测试把它拆成步骤时会立刻问出问题——改到什么时候不能改？改了要不要重新计价？已支付的呢？这些问号，**在需求评审时提出来，成本是一句话；在测试执行时才发现，成本是一次返工**。

所以低代码平台真正的定位，不是"省事的脚本工具"，而是**"让需求变得可讨论的建模工具"**。这句话你记住，面试时讲出来会很加分。

再补一个工程判断：**模型可以复用**。同一张模型换一套数据就能跑多种场景，这就是模型化带来的效率——也为下一节的 MBT 埋了伏笔。

板书： 能力一：用例建模（节点＝操作、边＝流转、分支显性化）→ 适合复杂场景；与因果图／判定表同源。能力二：需求阶段建步骤 → 澄清需求（改单到什么时候？要不要重算价？已支付的？）→ 测试左移。定位："让需求可讨论的建模工具"；模型可复用（模型＋数据＝多场景）。

提问： 为什么"在需求阶段画测试步骤"能减少返工？

预设回答：

- 答"因为问题发现得早"——对，这就是左移的核心收益：**同一个问题，越早发现越便宜**。

- 答"因为能逼需求写清楚"——更准确：画步骤的过程就是**把模糊需求逼成明确规则**的过程。

- 答"因为开发能照着步骤实现"——也对，步骤图常常变成开发的验收依据，双方理解一致。

- 收口：所以测试在需求阶段最有价值的产出，不是用例，是**问出来的那些问题**。

易错点 / 考点： ① 两个能力（用例建模、需求阶段建步骤）要能列举；② "测试左移"在这里的具体体现，是论述题的好素材；③ 用例建模与因果图／判定表的思想同源，可作知识串联；④ 判断题陷阱："低代码平台只是给不会编程的人用的"——它的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）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲**零代码智能测试工具 APIAuto**，课件只有标题，我讲清"零代码"和"低代码"差在哪，以及为什么这类工具会出现在**接口测试**领域。

先说差别。低代码是"少量配置 + 少量代码"，零代码是"**不写代码，只填信息**"。APIAuto 这类工具的做法是：你给它**接口文档**（Swagger、OpenAPI、或者一份规范的接口说明），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。你不用写一行脚本。

为什么这个思路在接口测试上特别成立？因为**接口是"有契约"的**——参数类型、必填项、取值范围、响应结构，文档里都写着。**有契约，就能自动化推导**。UI 测试做不到这一点，因为界面没有契约，只有像素。这是我这门课要反复强调的一个判断：**测试自动化的容易程度，取决于被测对象的形式化程度。**

它的能力边界也说清楚：**能自动发现"结构性问题"**——响应字段缺失、类型不符、状态码异常、必填参数不传会怎样；**不能自动发现"业务性问题"**——这个金额算得对不对、这个状态流转合不合理。前者靠契约，后者靠人。

还有一个隐藏价值：**它顺手做了"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一，零代码工具天天跑，等于天天在验文档与实现的一致性。

板书：低代码（少量配置＋少量代码）↔ 零代码（只填信息，不写代码）。APIAuto 路线：接口文档 → 自动生成请求／推断参数／校验响应结构 → 批量执行。成立前提："接口有契约"。能力边界：结构性问题上自动，业务性问题上靠人。隐藏价值：契约校验（文档 vs 实现）。

提问：为什么零代码测试工具在接口测试上比在 UI 测试上更容易落地?

预设回答：

- 答"因为接口有文档、有契约"——核心答案：**形式化程度决定自动化程度**。

- 答"因为 UI 变化太快"——也成立，界面改版频繁，脚本维护成本高。

- 答"因为接口测试更重要"——不能这么讲，两者都重要，只是**自动化难度不同**。

- 收口：所以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：① 零代码与低代码的区别；② "形式化程度决定自动化程度"这个判断很有分量，论述题可用；③ 零代码工具的能力边界（结构 vs 业务）；④ 判断题陷阱："零代码工具生成的用例不需要人审核"——结构对了不等于业务对了。

过渡：低代码、零代码都是"工具形态"。还有一种更根本的变化——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）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲 **MBT＋AI**。MBT 是 **Model-Based Testing，基于模型的测试**——这个词你其实已经接触过了：上一页"为测试用例创建一个模型"，就是 MBT 的雏形。

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**：把系统行为画成状态机、流程图、或者更形式化的规约，然后**由工具从模型自动生成测试路径**。

这么做的好处有三个：**一是生成能力强**——人能想到 20 条路径，工具能把模型里所有合法路径都走一遍，包括你没注意到的组合；**二是可维护**——需求变了改模型，路径自动重生成，不用手改用例；**三是可验证**——模型本身可以被检查（比如检查有没有走不到的状态、有没有死锁）。

那 AI 加进来解决什么？解决 MBT 最大的痛点：**建模太贵**。手画状态机，几百个状态画到崩溃。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。这就是"MBT＋AI"的组合逻辑：**MBT 提供严谨性，AI 提供建模效率。**

一句话收口：**MBT 是把测试从"经验驱动"变成"模型驱动"，AI 是让这个转变变得付得起。**

> **板书：**MBT＝Model-Based Testing（模型驱动）。依据从"用例"换成"模型"（状态机・流程图・形式化规约）→ 工具自动生成路径。三好处：生成能力强｜可维护｜模型可验证。AI 解决"建模贵"：自动抽取模型｜补全迁移｜按风险排序路径。底部："MBT 给严谨，AI 给效率"。

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"因为建模成本太高"——头号原因，也是 AI 要解决的问题。

- 答"因为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"因为团队不会画模型"——也对，MBT 需要状态机、形式化方法的基础，门槛在人的能力上。

- 收口：所以 MBT 落地的前提是"**模型能自动或半自动维护**"，这也解释了为什么 AI 一进来，MBT 就又被提起。

易错点 / 考点：① MBT 的全称与核心思想（依据从用例换成模型）；② 三个好处与 AI 解决的三件事要能对

上；③ 简答高频："MBT 的优缺点"；④ 判断题陷阱："MBT 生成的用例一定比人工设计的好"——模型不全，生成的路径也不全。

过渡：MBT＋AI 具体能做什么？下一页给一个很具体的例子。

【14.51】MBT+AI 合力相助测试 – 续（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页给了一个具体应用，课件原文是：**使用 NLP 进行 E2E Web 测试依赖项检测，验证所有依赖关系，并使用一个新的恢复算法来确保在最终的测试依赖关系图中没有遗漏任何真正的依赖关系，提高回归测试覆盖率。**

这句话里塞了四个动作，我拆开讲，你会发现它是一个完整的技术方案。

第一，用 NLP 做 E2E 测试的依赖项检测。 端到端测试是一长串操作：登录 → 搜索 → 加购 → 下单 → 支付。这些步骤之间是有依赖的——**没登录就不能下单，没加购就不能支付**。人工梳理这些依赖很费劲，而且容易漏。NLP 的作用是**从测试脚本、页面文本、接口描述里自动抽取"谁依赖谁"**，把依赖关系挖出来。

第二，验证所有依赖关系。 挖出来还不够，要验：这个依赖是真的吗？是不是巧合？会不会随版本变化？这一步是**去伪**。

第三，用新的恢复算法确保最终依赖图里没有遗漏真正的依赖。 这是这一页最精妙的地方。"恢复算法"的意思是：**从已知的依赖出发，反推可能被漏掉的依赖，再回去确认**。为什么要这么做？因为**漏掉依赖的后果是"看起来跑了、其实没验到"**——比如支付用例其实依赖了登录态，但脚本里没写，某天登录失效，这条用例就变成了"假通过"。

第四，结果落到"提高回归测试覆盖率"。 依赖图清楚了，就能知道"改 A 会影响到 B、C、D"，从而**精准地选出该回归的用例**——这跟上一节讲的"变更-脚本关联"是同一个目标。

一句话总结这一页：**依赖关系是 E2E 测试的隐性骨架，NLP 把它显性化，恢复算法保证它完整，覆盖率就是最终收益。**

板书：四步：NLP 抽取依赖（从脚本・页面文本・接口描述）→ 验证依赖（去伪）→ 恢复算法补全（防遗漏）→ 提高回归覆盖率。核心风险："漏依赖 → 假通过"。底部："依赖图是 E2E 的隐性骨架"。

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"因为前置条件碰巧满足了"——正是：在测试环境里登录态可能还留着，用例就过了；换个干净环境就红。

- 答"因为它没真的验到业务"——对，跳过了本该验的步骤，等于用一条空用例充数。

- 答"因为环境是共享的"——这是重要诱因，**共享环境会掩盖依赖问题**，所以要做环境隔离与前置清理。

- 收口：所以 E2E 用例的第一条纪律是——**每一条都必须能在干净环境里独立跑通**。

易错点 / 考点：① 四个动作要能按顺序列举；② "漏依赖 → 假通过"这个后果是案例题核心；③ 恢复算法的作用（反推补全依赖图）；④ 判断题陷阱："E2E 用例依赖越多越好（更真实）"——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 就讲工具本身往哪儿变。

【14.52】14.4 软件测试工具的未来（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是 14.4 的地图，只有两小节，但都是硬骨头：**14.4.1 软件测试工具的发展趋势**、**14.4.2 Codeless 测试自动化**。

我先把这一节的问题立起来：**过去二十年，测试工具的主旋律是"更强大的脚本能力"——更好的录制、更灵活的断言、更多的协议支持。但这条路的收益已经见顶了。** 为什么？因为脚本越强，会用的人越少，维护成本越高。行业里大量的团队不是"不想自动化"，是"自动化维护不起"。

所以工具的未来方向发生了变化：**从"让人写更强的脚本"，转向"让人少写甚至不写脚本，但覆盖更多"**。这就是 **Codeless（无代码）测试自动化** 的由来。

这一节你要抓三件事：**趋势是什么（几条主线）、Codeless 到底怎么做、以及这些变化对测试人员意味着什么**。第三件事最重要——工具越"傻瓜"，对使用者的**设计能力**要求反而越高，因为工具只负责执行，**测什么、断言什么、怎么判**还是人定。这也回到我们这门课的主线：**方法比工具长久。**

板书：写"14.4 测试工具的未来"；两小节：发展趋势｜Codeless 自动化。问题立起来："脚本越强 → 门槛越高 → 维护不起"。方向转变："写更强脚本 → 少写不写但覆盖更多"。右侧："工具越傻瓜，设计要求越

高"。

提问：如果工具已经能做"录制即用例"，为什么还需要学用例设计方法？

预设回答：

- 答"因为录制出来的用例又长又脆"——实情：录一条下单流程可能 80 步，其中 70 步是无关操作。

- 答"因为录不出来异常场景"——关键：**录制只能录"走过的路"，异常、边界、并发它录不到**。

- 答"因为不知道该验什么"——最本质的回答：工具能执行，但**判据要人给**。

- 收口：所以工具越强，越显得"设计能力"值钱——这也是本课程存在的理由。

易错点 / 考点：① 两小节名称；② 工具演进的转折点（从增强脚本到降低门槛）；③ 简答高频："为什么录制回放式的自动化难以长期维护"；④ 判断陷阱："工具智能化后测试设计方法过时"——恰恰相反。

过渡：先看趋势——工具这几十年是怎么一步步变过来的。

【14.53】软件测试工具的发展趋势（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲工具发展趋势，课件没有铺文字，我按"五代演进"给你梳理，每一代都对应一个"被解决的痛点"。

第一代，手工与半自动：靠人点、靠表格记。痛点是**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。痛点是**可重复执行**，但代价是**要会编程**。

第三代，关键字驱动与数据驱动：把操作抽成关键字，把数据抽到表格里。痛点是**脚本复用**——同一套流程换数据跑多场景。

第四代，平台化与低代码：可视化编排、统一调度、统一报告。痛点是**协作与维护**——让不写代码的人也能参与。

第五代，智能化：AI 生成用例、AI 定位元素、AI 分析失败。痛点是**覆盖率与效率的瓶颈**。

这条演进线有一个清晰的规律：**每一代都在降低"使用门槛"，同时提高"覆盖能力"**。但请注意一个反直觉的现象：**门槛降低的同时，工具能做的判断并没有变多**。工具永远不知道"这个业务这个版本最怕什么"，它只

能执行你给的判据。

所以看趋势要看出两层：**表层是形态变化（脚本 → 关键字 → 平台 → 智能），底层是不变的东西（判据仍由人定义、风险仍由人评估）**。这句话你记下来，答任何"趋势类"论述题都能用上。

板书：五代演进：手工 → 脚本化 → 关键字/数据驱动 → 平台化/低代码 → 智能化。每代右侧标"解决的痛点"：重复劳动｜可重复执行｜脚本复用｜协作维护｜覆盖与效率。底部两行："门槛在降、覆盖在升；判据仍由人定"。

提问：这五代演进里，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，它把测试分成了"会写代码"和"不会写代码"两类人。

- 答"第四代平台化，因为它改变了协作方式"——很到位：平台化让测试从"个人技艺"变成"团队资产"。

- 答"第五代，因为要懂 AI"——部分对，但更准确的是**要懂怎么用 AI 的产出做判断**。

- 收口：技能结构的变化方向是——**从"会操作"到"会设计"，从"写脚本"到"定标准"**。

易错点 / 考点：① 五代演进的大致阶段与每代解决的痛点；② 底层不变的两件事（判据、风险）是论述题核心；③ 简答高频："测试工具发展趋势对测试人员的影响"；④ 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低代码平台，我们再看它在"平台化"这个大方向上的位置。

【14.54】低代码测试平台（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这里再讲一次低代码平台，但角度换了：上一轮讲"它是什么"，这一轮讲"**它在整个研发体系里的位置**"。这个角度更重要，因为它牵扯到三个你一定会遇到的概念：**平台化、DevOps、云原生**。

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，全在一个平台上。为什么企业愿意投入做平台？因为**个人脚本无法沉淀为组织能力**——人一走，脚本就废；平台把能力沉下来，人能换，能力不换。这就是"平台化"的本质：**把个人技艺变成组织资产**。

再说 DevOps。 DevOps 的核心是"**流水线**"：代码提交 → 构建 → 测试 → 部署 → 监控，全自动串起来。低代码平台要嵌进这条流水线，就必须提供 **API、触发器、质量门禁**——测试不通过，流水线就拦住。所以低代码平台的"平台化"不是为了好看，是为了**能被流水线调用**。这也解释了一个现象：**不能接 CI 的测试工具，在企业里活不长。**

最后说云原生。 云原生带来两个变化：**环境是弹性的**（需要时拉起，用完销毁），**服务是拆分的**（微服务、容器）。对测试的影响是：**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"**。于是测试平台的职责扩展了——它要能对接 K8s、能按需造环境、能做服务间的契约测试。

三句话收口：**平台化解决"能力沉淀"，DevOps 解决"嵌进流程"，云原生解决"环境与服务形态"。低代码是这三件事在测试侧的具体形态。**

板书：三概念三句：平台化＝个人技艺→组织资产（能力沉淀）；DevOps＝流水线（API・触发器・质量门禁，"不能接 CI 活不长"）；云原生＝弹性环境＋服务拆分（声明式环境・契约测试）。底部："低代码是这三件事在测试侧的形态"。

提问：为什么说"不能接 CI 的测试工具在企业里活不长"?

预设回答：

- 答"因为没人会手动去跑"——对，流水线是研发的主干道，脱离主干道的工具会被遗忘。

- 答"因为质量门禁需要它"——更本质：**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"因为它没法沉淀数据"——也对，跑不在流水线里，就没有稳定的质量趋势数据。

- 收口：所以选型时先问一句："**它能被流水线调用吗？**"——这比功能列表重要。

易错点 / 考点：① 三个概念与测试的关系（平台化、DevOps、云原生）；② "个人技艺 → 组织资产"这个说法很适合答平台化；③ 云原生对测试的两个影响（环境弹性、服务拆分 → 契约测试）；④ 判断题陷阱："平台化就是买个商业工具"——平台化的核心是能力沉淀与流程嵌入。

过渡：低代码走到极致，就是零代码。再回看 APIAuto 这类工具，你会看到另一层含义。

【14.55】零代码智能测试工具 APIAuto（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

APIAuto 再讲一次，这次讲它在**服务化**和**云原生**背景下的位置——因为接口测试恰恰是云原生时代最主流的测试形态。

先说为什么。云原生把系统拆成一堆微服务，服务之间靠接口通信。这时候**界面测试只能覆盖很小一部分**，真正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。所以行业里有个说法：**微服务时代，接口测试是主战场，UI 测试是补充。**

在这种背景下，零代码接口工具的价值就被放大了，因为它能做到三件事：

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，一跑就露馅。**

第二，批量覆盖。 微服务动辄几百个接口，人工写用例不现实，工具能在分钟级铺开。

第三，嵌入流水线做门禁。 每次服务发布前跑一遍契约测试，保证"我改的东西没破坏别人的调用"。

但请一定记住它的边界，这也是我要反复强调的：**零代码工具验的是"形式"，不是"语义"**。它能告诉你"返回了 200、字段齐全、类型正确"，但**不能告诉你"这个金额算得对"**。语义正确性，要靠业务用例、要靠数据比对、要靠人。

所以成熟的测试体系是"**三层叠加**"：**契约层自动化（零代码工具）＋ 业务层用例（人工设计）＋ 数据层比对（diff 工具）**。三层各管一段，谁也替不了谁。

> **板书：**云原生 → 微服务 → 接口是主战场（UI 是补充）。零代码三价值：契约即测试（文档 vs 实现）｜批量覆盖（几百接口）｜流水线门禁（不破坏调用方）。边界：验"形式"不验"语义"。三层叠加：契约层（零代码）＋业务层（人工用例）＋数据层（diff）。

提问：接口全部返回 200、字段都齐全，能说明这个系统没 bug 吗？

预设回答：

- 答"能，接口都通了"——错答。电商系统里"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，形式正确与业务正确是两件事。

- 答"要看有没有数据比对"——对，这正是"数据层比对"存在的理由。

- 收口：所以接口测试的完整链条是"**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：① 零代码接口工具的三个价值；② "验形式不验语义"是核心判断，案例题常用；③ 三层叠加

的测试体系结构，是论述题的好框架；④ 判断题陷阱："接口测试就是测 HTTP 状态码"。

过渡：工具讲完，接下来是更大的变化——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）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是 14.5 的地图，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**。

先把"持续测试"这个词的分量说清。我们前面十三章的测试，本质上都是"**阶段式**"的：需求阶段做评审，开发阶段做单元测试，提测后做系统测试，上线前做验收。这种模式的隐含假设是——**测试有起点和终点**。

持续测试打破的正是这个假设：**测试没有终点，它跟着流水线一直跑**。代码一提交就跑单测，合并前跑接口测试，部署到预发跑 E2E，上线后跑巡检和线上验证。测试从"一道关卡"变成"**一条持续的反馈带**"。

这个转变带来三个连锁反应，你听的时候留意：**一是节奏**——测试必须快到能在流水线里跑完，否则就是阻塞；**二是选择**——跑不完所有用例，必须精准选；**三是组织**——测试不再是测试组的事，开发、运维都得参与。这三条在后面的"特点"和"成熟度模型"里会展开。

一句话预告：这一节要回答的是"**怎么让测试真正跟得上交付速度**"——这是全书最后一组工程问题。

板书：写"14.5 彻底实现持续测试"；四小节竖排。左侧对照：阶段式测试（有起点终点）↔ 持续测试（跟着流水线一直跑）。三个连锁反应：节奏（要快）｜选择（要准）｜组织（要协同）。底部："测试从一道关卡，变成一条反馈带"。

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，会直接堵住流水线。
- 答"不是，还要求快、准、和流程融合"——准确。
- 答"还要看跑的东西对不对"——补充得好：**没有精准选择的"多跑"，只是把噪声放大**。
- 收口：所以持续测试是"**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：① 四小节名称；② 持续测试与"阶段式测试"的根本差别（有无终点）；③ 三个连锁反应（节

奏、选择、组织）是后面内容的纲；④ 判断题陷阱："持续测试＝持续集成里加个测试步骤"——远不止，它涉及流程、组织与成熟度。

过渡：先把"持续测试"这四个字重新定义清楚。

【14.57】持续测试（2 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页给持续测试下了三条定义，一条一条讲，这三条合起来就是它的完整画像。

第一条：测试可以随时开展起来，且具有连续性。"随时开展"是关键——不是"等提测"，而是**任何时刻、任何一次代码变更，都能立刻拉起测试**。这背后要求环境随时可用、用例随时可跑、数据随时可造。**连续性**则意味着结果是可以连起来看的：今天和昨天的通过率能不能对比，趋势是变好还是变坏。

第二条：测试和开发、运维能很好地融合、匹配和同步起来。这条讲的是**组织层面**。传统模式下，测试是"下游部门"，开发交什么就测什么；持续测试要求测试**往前走**，参与需求与设计；同时**往后走**，参与线上监控。**"融合"三个字，说白了就是测试不再是独立环节，而是研发流程里的一层能力。**

第三条，也是最有分量的一条：以最少的测试、最快的速度覆盖交付所面临的业务风险。请注意这里的措辞——不是"覆盖最多的代码"，不是"跑最多的用例"，而是"**覆盖业务风险**"。这是一个立场性的转变：**测试的目标不是"测得多"，是"把风险挡住"**。所以持续测试必然是**基于风险的测试**：改动大的地方多测，改了没风险的地方少测，从没出过问题的老代码可以只做冒烟。

三条合起来，我给你一句总结：**持续测试＝随时能测（能力）＋与研发运维同步（组织）＋用最小代价挡最大风险（策略）。**这三条也正好对应下一页的四个特点。

板书：三条：① 随时开展 + 连续性（环境/用例/数据随时可用；结果可对比）② 与开发运维融合匹配同步（往前参与需求，往后参与监控）③ 最少测试 + 最快速度覆盖业务风险。底部："持续测试＝能测（能力）＋同步（组织）＋挡风险（策略）"。

提问："以最少的测试覆盖业务风险"，这句话和"测试要尽量充分"矛盾吗？

预设回答：

- 答"矛盾，应该尽量多测"——这是把"充分"理解成"数量"。**充分是针对风险的，不是针对用例数的。**

- 答"不矛盾，充分是指风险覆盖充分"——准确，这是行业里"基于风险的测试"（RBT）的核心思想。

- 答"资源有限，必须取舍"——现实约束，也是这条定义存在的理由。

- 收口：所以判断测试做得好不好，标准不是"跑了多少条"，而是"**该挡的风险挡住了没有**"。

> **易错点 / 考点：** ① 三条定义要能复述；② "覆盖业务风险"而非"覆盖代码"这个立场转变，是简答与论述高频；③ 与第 7 章"基于风险的测试"思想呼应，可作知识串联；④ 判断题陷阱："持续测试要求 100% 回归"——恰恰相反，它要求精准选择。

> **过渡：** 三条定义落到工程上，就变成四个特点。

【14.58】持续测试的特点（3 分钟）

☐ 已讲 ☐ 需回看　　课堂记录：_____

这一页给持续测试的四个特点，每一个都对应一条工程约束，我逐条讲，最后收一句组织层面的话。

第一，足够快。 课件说得很具体：**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**。注意这里的两条腿——**高度自动化**保证"机器能跑的都快跑掉"，**端到端探索式测试**保证"人该看的还看"。为什么探索式测试不能省？因为自动化只能验"你想得到的"，**想不到的问题只能靠人去碰**。持续测试不是"全自动"，是"**自动化兜底 + 人做探索**"。

第二，准确且有效。 课件解释得很到位：**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**。这句话是持续测试的技术核心——**精准测试**。它的前提是"变更与用例的映射"（前面 14.3 讲过），它的目标是"改哪儿测哪儿，且不漏"。

第三，平滑有序。 课件说：**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**。这三个"到"非常有分量：**时间上从左到右**（需求阶段到线上），**层次上从下到上**（单元到系统），**手段上从静到动**（评审到执行）。**"打通"意味着这些环节不再是各自为政的孤岛，而是一条连续的链**——需求里的验收标准，要能一路传到线上的监控指标。

第四，与 CI/CD 集成。 与研发的持续构建、持续集成、持续部署、运维等环境的集成。这条是技术前提：测试必须能被流水线调用，能给出明确的通过／失败信号，能作为质量门禁拦住不合格版本。

四个特点我用一句话收口：**快（自动化＋探索）、准（精准测试）、通（左移右移打通）、接（嵌入**

CI/CD）。

板书：四点：足够快（高度自动化＋端到端探索式）｜准确有效（复杂系统做不了全回归 → 精准测试）｜平滑有序（左移→右移、单元→系统、静态→动态，三个"打通"）｜与 CI/CD 集成（构建・集成・部署・运维）。底部："快・准・通・接"。

提问：为什么"精准测试"在持续测试里是必需的，而不是可选的优化？

预设回答：

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

- 答"因为等不起就等于没测"——更本质：**跑得太慢的测试，会被团队想办法绕过**，那就彻底失效了。

- 答"因为要做质量门禁"——也对，门禁必须分钟级给出结论。

- 收口：所以精准测试不是"偷懒"，是**让测试能在真实节奏下存在**的前提。

易错点 / 考点：① 四个特点要能默写；② "三个打通"（左移右移、单元系统、静态动态）是简答高频；③ 精准测试的必要性要能从"节奏"角度论证；④ 判断题陷阱："持续测试等于全自动化"——必须保留探索式测试。

【思政落点 2】：课件一边要求"高度自动化"，一边坚持保留端到端的探索式测试——为什么？因为机器只验人写下的判据，而事故常出在人没想到的地方。这就是质量工程的含义：不是一个部门的职责，而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。人人都是防线的一环，这种文化比任何工具都更能挡住事故。

过渡：四个特点知道"要什么"，接下来看"怎么搭"——实施框架。

【14.59】持续测试实施框架（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是持续测试的实施框架，课件没有铺文字，我讲清"一个能落地的持续测试体系，必须回答五个问题"，你按这五问去对照任何一家公司的实践，都能立刻看出它成熟到哪一步。

第一问：测什么？也就是**测试范围与风险选择**——哪些用例进流水线，哪些跑夜间，哪些只在发布前跑。这一层解决"精准"，也是框架的入口。

第二问：在哪儿测？ 也就是**环境与数据**——环境要能按需拉起、数据要能按需构造。这一层是最容易被低估的：**大量"自动化跑不起来"的问题，根子都在环境和数据上**，不在脚本上。

第三问：谁来跑、什么时候跑？ 也就是**调度与触发**——提交触发、定时触发、发布触发，以及各层测试的时间预算（单测几分钟、接口十几分钟、E2E 半小时以上）。

第四问：跑完怎么判？ 也就是**结果判定与门禁**——通过标准是什么、失败怎么分级、门禁卡在哪个环节。这一层决定测试有没有"牙齿"。

第五问：判完怎么改？ 也就是**反馈闭环**——失败原因能不能快速定位（回到 14.3 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。

五问连起来就是一句话：**持续测试框架＝范围、环境、调度、门禁、反馈**。缺哪一环，体系就在哪儿断。而在这五环里，**最难的不是技术，是"门禁"**——因为门禁意味着"测试说不许发，就真的不发"，这需要组织授权。

> **板书：**五问五环：测什么（范围与风险选择）→ 在哪儿测（环境与数据）→ 何时谁跑（调度与触发）→ 怎么判（结果判定与门禁）→ 判完怎么改（反馈闭环）。底部："范围・环境・调度・门禁・反馈"；"最难的是门禁，因为它需要组织授权"。

提问：一个团队自动化脚本写了几千条，但流水线里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，第四问缺失，整套体系没有"牙齿"。

- 答"得看失败率高不高"——补得好：如果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：所以判断测试体系成熟度，先看**"红灯能不能拦住发布"**。

易错点 / 考点：① 五环（范围、环境、调度、门禁、反馈）要能列举；② "环境与数据是被低估的一环"这个判断很实用；③ 门禁需要组织授权，是论述题的高级观点；④ 判断题陷阱："自动化用例数量是成熟度的核心指标"——不是，门禁与反馈才关键。

过渡：五环之外还有一条时间线——成熟度是分阶段长出来的。

【14.60】14.6 软件测试发展趋势（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页是本章最后一节的地图：**14.6 软件测试发展趋势**，两小节——**14.6.1 MBT 的应用前景**、**14.6.2 软件测试六大趋势**。

到这一节，我们从"具体技术"上升到"整体判断"。前面五节讲的是"某一种新技术怎么用"，这一节要回答的是"**整个行业往哪儿走**"。

为什么把 MBT 单独拎出来讲前景？因为在六大趋势里，**"模型化"是最不显眼、但可能最深刻的一条**——它改变的不是工具，是**测试的依据**：从"人想出来的用例"变成"从模型推导出来的路径"。这条一旦走通，测试的**系统性**会上一个台阶。

六大趋势则是全章的收口：**敏捷化、高度／超级自动化、云化、服务化、模型化、智能化**。这六条你要能背，而且要能各举一个例子——这是本章最标准的论述题。

提前说一句：这一节最后我会把整门课收起来，所以现在你带着一个问题听——**这六条趋势里，哪一条对我们的课程内容冲击最大？** 我的答案是"敏捷化"，因为它把测试从阶段变成了常态，逼着前面所有方法都要"提速"。

板书：写"14.6 发展趋势"；两小节：MBT 应用前景｜六大趋势。右侧预告六条：敏捷化・高度自动化・云化・服务化・模型化・智能化。底部："模型化改的是测试依据；敏捷化改的是测试节奏"。

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，用例会从手写变成生成"——很有洞见，MBT 正是这个方向。
- 答"智能化，AI 直接生成用例"——也成立，AI 生成 + 模型约束是最可能的组合。
- 答"敏捷化，用例会变少"——部分对：敏捷要求"够用的用例"，而不是"全的用例"。
- 收口：所以未来的"用例"可能是**模型 + 数据 + 断言**三件套，而不是一份步骤清单。

易错点 / 考点：① 两小节名称；② 六大趋势是本章第一名必背内容；③ "模型化改依据、敏捷化改节奏"这个提法便于记忆；④ 论述题要求"每条趋势举一例"，平时就要准备好例子。

过渡：先看我们身处一个什么样的时代——这决定了趋势的走向。

【14.61】一个新时代（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 这一页讲时代背景。课件有一句话我觉得写得很好：**我们可以用很多词汇来形容这个时代：移动互联、智能、Cloud、数字化等等，但更是软件定义一切的时代。**
>
> 四个词，我一个一个说它对测试的影响。
>
> **移动互联**：终端从一台电脑变成几亿部手机，**碎片化**成为常态——系统版本、屏幕尺寸、网络状况、厂商定制。测试从"测一个环境"变成"测一片环境"，这才有了兼容性专项。
>
> **智能**：软件从"按规则执行"变成"自己判断"，**输出不再是确定的**，判据从"对不对"变成"够不够好"，这就是 14.2 讲的内容。
>
> **Cloud**：基础设施从"买机器"变成"租资源"，**环境变成弹性的、可编程的**，测试环境可以按需拉起、用完销毁，这直接支撑了持续测试。
>
> **数字化**：业务从"线下为主"变成"线上为主"，**软件故障直接等于业务中断**，质量的经济权重被放大了——一次宕机不是"系统慢了点"，是"生意停了"。
>
> 四个词最后收成一句：**软件定义一切**。这句话的分量在于：**当软件定义一切，软件质量就定义一切**。所以测试这个岗位的重要性是上升的，但要求也变了——**它不再是"点点点"，而是"在高速变化中守住质量底线"。**
>
> 这也带出我想跟你说的一件正事：**你要给自己建一个能力模型**。我建议三条腿：**一条是技术深度**（会写代码、会看架构、会用工具）；**一条是业务理解**（懂这个行业的规则和风险）；**一条是工程判断**（知道什么该测、什么可以放过、什么叫"可以发"）。三条腿缺一条，走不远。

> **板书：**四词四影响：移动互联（碎片化 → 兼容性）｜智能（输出不确定 → 判据变"够不够好"）｜Cloud（环境弹性可编程 → 支撑持续测试）｜数字化（软件故障＝业务中断 → 质量权重上升）。底部："软件定义一切 → 质量定义一切"；能力模型三条腿：技术深度・业务理解・工程判断。

> **提问：**这三个能力方向里，你觉得在校期间最该先补哪一个？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，但只补技术会变成"工具人"。

- 答"业务理解，学校没教"——很清醒：业务理解往往要在实习里补，越早接触真实业务越好。

- 答"工程判断，最难"——最难也最值钱，它只能靠"做过、错过、复盘过"积累。

- 收口：建议是"**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；② "软件定义一切"这句话的含义与推论；③ 能力模型三条腿是简答与面试高频；④ 判断题陷阱："测试岗位会因自动化而消失"——岗位形态会变，判断责任不会消失。

> **【思政落点 3】**：技术变化这么快，唯一不会过期的能力是"学习的能力"。你在学校学的工具，五年后可能一半没人用；但"看到一个新技术，能判断它解决什么问题、有什么代价、该不该用"这套本事会跟你一辈子。终身学习不是一直考证，而是保持"敢问、会学、能判断"的状态。把学习当习惯，你就不会被淘汰。

> **过渡：**时代讲完，回到具体技术——MBT 为什么被看好？

【14.62】MBT 的应用前景（2 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页给 MBT 的三条前景，课件写得很扎实，我逐条讲。

第一条：通过使用 MBT 工具会让这个建模过程更加规范和可视化，增加了模型的可读性和重用性。 这条讲的是"**模型资产化**"。手画的流程图躺在文档里，没人维护；工具化的模型是**活的资产**——能版本管理、能复用、能被工具读取生成用例。**可读性**是对人的（新人能看懂系统行为），**重用性**是对机器的（同一模型生成不同覆盖策略的用例集）。

第二条：在敏捷开发中采用 MBT 可以促进持续交付，有效的应对敏捷测试中的风险。 这条把 MBT 和敏捷连起来了，逻辑是：敏捷要"快速迭代"，而快速迭代最大的风险是"**改一处、崩一片**"。MBT 的价值在于**模型是变更的影响分析器**——需求变了，改模型，工具自动告诉你哪些路径受影响，该回归哪些。**这就把"精准测试"从人工判断变成了模型推导。**

第三条：将 AI 技术和 MBT 结合会更强大，使用 AI 技术创建训练测试模型，利用算法为不同的测试覆盖率基于风险推荐测试路径。 这条是前面 14.3 的呼应：**AI 负责建模型和排序，MBT 负责生成和验证**。特别是"基于

风险推荐测试路径"——同样的模型，优先级不同，先跑高风险路径，这在时间不够时特别有用。

三条我给一句总结：**MBT 的前景不在"多一个工具"，而在"把测试依据从经验升级为模型"**。模型一旦成为一等公民，建模、变更分析、路径生成、风险排序就都能自动化——这才是它被称为"趋势"的原因。

也要说清现实：**MBT 落地的前提是团队愿意画模型、并且模型能跟着代码走**。这两条做不到，MBT 就只是 PPT 上的概念。

板书：三条前景：① 建模规范化可视化 → 模型可读、可重用（资产化）② 敏捷中促进持续交付 → 模型＝变更影响分析器（自动告诉你回归哪些）③ AI＋MBT → 自动建模型、按风险推荐路径。底部："前景不在多一个工具，在把测试依据从经验升级为模型"；前提：愿画模型、模型跟着代码走。

提问：为什么说"模型是变更的影响分析器"？

预设回答：

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

- 答"因为可以自动生成回归集"——这是它的直接产出，也是精准测试的实现路径。

- 答"因为模型能查出走不到的状态"——很好的补充：**模型本身可被静态检查**，能发现死状态、死锁、不可达路径。

- 收口：所以 MBT 的价值是"**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＝变更影响分析器"这个提法很适合答题；③ AI 与 MBT 的分工（AI 建模与排序、MBT 生成与验证）；④ 判断题陷阱："MBT 只适用于大型嵌入式系统"——Web、API、业务流程同样适用。

过渡：说了半天模型，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）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页列了一批**模型检测工具**，都是学界与业界经典。参数不用背，但你要记住"**每种工具代表一种建模与验证的思路**"。

SPIN（Simple Promela Interpreter）：LTL 模型检测工具，用 Promela 语言对**并发系统**建模，检测一个有限状态系统是否满足 **PLTL／CTL** 性质。看家本领是**并发**——死锁、竞态、活锁靠人工评审很难发现，靠模型检测能系统性穷举。

SMV（CMU）：CTL 模型检测工具，采用二叉图（BDD）表示状态转换关系，以计算不动点的方法检测状态的可达性及其所满足的性质。BDD 的价值是**压缩状态空间**——状态爆炸是模型检测的天敌，BDD 让"看起来算不完"的问题变得可算。

CWB（爱丁堡大学及美国北卡大学）：mu-演算工具，检测**系统间的等价关系、PRE-ORDER 关系**及是否满足 mu 演算公式。特色是**比较两个系统**——"重构后行为是否一致"就靠它。

NuSMV：符号模型检测器（SMC），基于 **BDD 和 SAT** 的模型检验。它是 SMV 的进化版，SAT 方法的引入让它能处理更大的状态空间。

NuXmv：也是 SMC，但**可以支持同步的有限状态或无限状态系统的分析**。"无限状态"是重要突破——真实系统的变量取值往往无限，传统方法处理不了。

Murphi（斯坦福大学、犹他大学）：是许多模型检测的基础，在此之上开发了 **CMurphi、Preach** 等。

它们的意义在于：**把"验证"从"跑用例"提升到"数学证明"**。用例只能证明"这条路上没问题"，模型检测能证明"**所有可能路径上都没问题**"（前提是模型准确）。这就是 MBT 的理论根基。

板书：六行：SPIN（LTL・Promela・并发）｜SMV（CTL・BDD・不动点）｜CWB（mu 演算・等价与 pre-order）｜NuSMV（SMC・BDD+SAT）｜NuXmv（SMC・有限/无限状态）｜Murphi（许多工具的基础 → CMurphi・Preach）。底部："用例证明'这条路上没问题'；模型检测证明'所有路径都没问题'"。

提问：模型检测能证明"所有路径都没问题"，那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，但有个致命前提：**模型必须准确**。
- 答"不是，模型错了证明也没意义"——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，所以才需要 BDD、SAT 这些技术。
- 收口：所以模型检测与测试是**互补**关系：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 六个工具与各自代表的技术路线（LTL/CTL/mu 演算/SMC/BDD/SAT）要能对上；② 模型

检测的两大局限（模型准确性、状态爆炸）是简答高频；③ "验证 vs 测试"的关系可作论述素材；④ 判断题陷阱："模型检测可以替代测试"——不能替代，二者互补。

过渡：工具与模型都讲完了，最后收成六条趋势——这是本章的骨架。

【14.64】软件测试六大趋势（3 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是本章最该背下来的一页：**软件测试六大趋势**。我一条条讲，每条给你一个"为什么会出现"和"对你意味着什么"。

第一，敏捷化：测试左移到位（ATDD）、测试右移显著增强。 左移是"往前站"——ATDD（验收测试驱动开发）让测试在写代码前就把验收标准定下来；右移是"往后站"——上线后继续做灰度、监控、线上验证。**测试的边界从"提测到上线"扩展到"需求到运维"。** 对你意味着：**要会和产品、开发一起写验收标准**。

第二，高度／超级自动化：让自动化测试成为流水线。 注意"成为流水线"这个说法——自动化不是"跑脚本"，而是**流水线的一个组成部分**，有触发、有门禁、有报告。

第三，云化：基础设施支持自动化运行。 测试环境、测试数据、测试执行都可以跑在云上，**按需扩容、按需销毁**。这解决的是"环境不够、排队等环境"这个老问题。

第四，服务化：一切测试皆 API，提升自动化测试效率。 把测试能力包装成服务——造数服务、比对服务、环境服务、执行服务。**谁需要谁调用**，不必每个人都重造轮子。这一条其实是"测试中台"的思想。

第五，模型化：MBT 让自动化测试更彻底。 前面讲过了，它改的是**测试依据**。

第六，智能化：基于大数据与 AI，让自动化测试更智能。 用例生成、元素定位、失败分析、风险预测，都属于这一条。

六条我给你一个记忆结构：**两"化"管节奏（敏捷化、自动化），两"化"管形态（云化、服务化），两"化"管内核（模型化、智能化）**。答题时按这个结构展开，条理就出来了。

板书：六条竖排：敏捷化（左移 ATDD・右移增强）｜高度/超级自动化（自动化成为流水线）｜云化（基础设施支持自动化运行）｜服务化（一切测试皆 API）｜模型化（MBT 更彻底）｜智能化（大数据与 AI）。右侧标注：节奏（敏捷・自动化）｜形态（云・服务）｜内核（模型・智能）。

提问：六条趋势里，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 会替代人"——短期影响有限：AI 替代的是重复执行，**判断与设计仍缺人**。

- 答"服务化，能力集中后不需要那么多人"——有可能：重复建设的岗位会减少，**平台型与专家型岗位会增加**。

- 答"敏捷化，测试融进开发，专职测试变少"——这是行业真实发生的：很多团队叫"质量工程师（QE）"而不是"测试工程师"。

- 收口：所以趋势不是"人变少"，是"**人的结构变了**"——执行型岗位收缩，设计型与平台型岗位扩张。

易错点 / 考点：① 六条趋势**必须能默写**，这是本章第一名考点；② 每条要能举一个具体例子（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"很好用；④ 判断题陷阱："六大趋势是先后关系"——它们是并行发生的，不同企业进度不同。

【思政落点 4】：六条趋势摆在这里，最容易犯的错是追热点：听说智能化火就扎进 AI，明年换个平台又重来。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，这些只能一年一年攒。选一条主线扎下去，扎到能解决真实问题为止，热点会自己找上你。

过渡：六大趋势里，"云化"最容易被误解成"把测试搬到云上"。看一个更本质的例子。

【14.65】示例：云化（1 分钟）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页是"云化"的示例，课件没有配文字，我讲清"云化对测试到底改变了什么"，重点是**云原生**带来的三个深层影响。

第一个改变：环境从"资产"变成"能力"。 以前申请一套测试环境要提工单、等排期，环境是稀缺资产；云上可以**声明式拉起**——写一份配置，几分钟就有一套隔离环境，用完销毁。这直接解决了持续测试最大的物理瓶颈：**并行**。你可以同时跑十个分支的测试，各自有独立环境。

第二个改变：被测对象的形态变了。 云原生把单体拆成微服务、装进容器、跑在编排平台上。于是测试面临三个新问题：**服务发现**（我调的那个服务现在在哪个实例上）、**配置外置**（同一份镜像在不同环境行为不同）、**有状态与无状态混合**（数据库、缓存、消息队列怎么造）。这就是为什么"云原生测试"不是"把老测试

搬到云上"，而是**要重新设计测试策略**。

第三个改变：故障变成常态。 云上实例会被调度、会被驱逐、网络会抖动。所以测试的重点从"功能对不对"扩展到"**故障时能不能自愈**"——这就是混沌工程（Chaos Engineering）的由来：**主动注入故障，验证系统的韧性**。

三句话收口：**云化让环境变便宜、让架构变复杂、让故障变常态**。对应的测试能力是：**环境即代码、契约测试、混沌实验**。

> **板书：**三个改变：① 环境＝资产 → 能力（声明式拉起・并行）② 对象形态变（服务发现・配置外置・有状态/无状态）③ 故障成常态（→ 混沌工程）。对应能力：环境即代码｜契约测试｜混沌实验。底部："云化＝环境变便宜、架构变复杂、故障变常态"。

提问：微服务架构下，为什么"端到端测试"要少做、"契约测试"要多做？

预设回答：

- 答"因为服务太多，E2E 组合爆炸"——核心原因：N 个服务的组合数远超可测范围。

- 答"因为 E2E 慢且脆"——实情，一个 E2E 用例要拉起一整套环境，失败原因还难定位。

- 答"契约测试能保证调用方不被破坏"——准确，契约测试把"服务间一致性"这件事单独拎出来验，成本低、定位准。

- 收口：所以微服务的测试策略是"**大量契约测试 + 少量关键 E2E**"，这跟测试金字塔是一个道理。

易错点 / 考点：① 云化对测试的三个改变；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、E2E 少）是高频简答；④ 判断题陷阱："云化就是把本地测试脚本放到云主机上跑"。

过渡：云化之外还有一条更"务虚"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）

☐ 已讲　☐ 需回看　　课堂记录：_____

这一页讲**测试服务化**，课件给了一组关键词，我按"基础设施—能力—使用体验—数据"四层给你归拢。

第一层，基础设施层：基础设施即代码（IaC）。 环境不再靠人手搭，而是用代码描述、用工具创建。这一条

是服务化的地基——**环境都不能自动创建，谈什么服务化。**

第二层，能力层：基于 API、服务的云平台。 把测试能力做成 API：造数 API、比对 API、执行 API、报告 API。**"一切测试皆 API"**，这句话是这一页的核心。它的价值在于**复用**：一个团队做好的造数服务，十个团队都能调，不必各自重写。

第三层，使用体验层：支持一键部署、自动监控、高度可视化、在线实时收集数据。 这一层讲的是"**门槛**"——能力再强，用起来麻烦就没人用。一键部署降低启动成本，可视化降低理解成本，实时数据降低判断成本。

第四层，数据层：大数据处理能力。 测试会产生海量数据——执行日志、覆盖率、性能指标、失败聚类。要有能力存下来、算出来、看出来。**没有数据层，测试平台就只是个"跑脚本的地方"。**

四层合起来，就是"**测试中台**"的雏形。它解决的核心问题是：**把测试能力从"每个项目各自造"变成"一次建设、多次复用"**。

但我要提醒一个副作用：**服务化会让测试离被测系统更远**。你调一个造数服务，可能不知道它内部怎么造的；出问题时，"服务"就成了新的黑盒。所以服务化的同时要配套**服务自身的可观测性**——不然问题会被推来推去。这句话，是很多团队踩过坑才明白的。

> **板书：**四层：基础设施（IaC）→ 能力（API／服务，一切测试皆 API）→ 体验（一键部署・自动监控・可视化・实时数据）→ 数据（大数据处理）。底部："测试中台＝一次建设、多次复用"；副作用："服务化让测试离系统更远 → 服务自身要可观测"。

提问：测试服务化最大的收益是什么？最大的风险是什么？

预设回答：

- 答"收益是复用，风险是依赖"——简明准确。
- 答"收益是效率，风险是黑盒"——更贴切：**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，所以要给关键服务做 SLA 与降级方案。
- 收口：所以服务化的配套是"**可观测 + 可降级**"，这两条不写进方案，就别急着推。

易错点 / 考点：① 服务化四层内容；② "一切测试皆 API"这句口号及其含义；③ 服务化的收益与风险要成对答；④ 判断题陷阱："服务化只对大公司有意义"——小团队用开源组件同样能做轻量服务化。

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）

☐ 已讲 ☐ 需回看 课堂记录：_____

这一页讲**手淘 RXT 测试机器人**，这是一个非常"工程"的案例，它解决的是一个特别具体的体验问题：**点击消息 Tab，到底要等多久才能看到内容**。

先说这个问题为什么难测。传统做法是打点——在代码里埋时间戳，算差值。但打点有三个毛病：**一是侵入**，要改代码；**二是不准**，打点只能覆盖到代码层，覆盖不到渲染、合成、上屏这些环节；**三是和用户感受不一致**，用户感受到的"加载完"是**屏幕上真的出现了内容**，不是"接口返回了"。

RXT 的思路是**换个测量工具——用眼睛，而且是高速眼睛**。课件给的参数很硬：**240fps 高速摄像头（误差 < 5ms），无需参照目标，自动识别准确率 ⩾ 95%**，方法是**基于视频特征分析的加载时长自动化过程**。我拆开讲这三点：

240fps 意味着每 4 毫秒一帧，所以能捕捉到毫秒级的画面变化，误差控制在 5ms 以内——**这个精度足够判断"是不是达标"**。

无需参照目标 意味着不需要在界面上做标记、不需要埋点，**对被测应用零侵入**。这一点特别重要：**零侵入才能测真实版本**。

自动识别准确率 ⩾95% 意味着它用视觉算法自动判断"内容出来了没有"，而不是靠人盯屏幕数帧。

这个案例的方法论价值，我给你三条：**第一，测试的"测量工具"可以跳出软件本身**——摄像头、示波器、功耗仪都是合法手段；**第二，用户体验指标要用用户视角测**——用户看的是屏幕，那就测屏幕；**第三，零侵入是测量真实性的前提**。这三条，比记住那几个数字值钱得多。

> **板书：**问题：点击消息 Tab 的加载时长（打点法三毛病：侵入・不准・与感受不一致）。RXT：240fps 高速摄像头（误差<5ms）・无需参照目标・自动识别准确率⩾95%・基于视频特征分析。三条方法论：测量工具可跳出软件｜用用户视角测体验｜零侵入才真实。底部："用户看屏幕，就测屏幕"。

> **提问：**为什么"接口返回了"不等于"用户看到了"？

> **预设回答：**

- 答"因为还要渲染、布局、绘制、合成"——准确，从数据到像素中间有好几道工序。

- 答"因为可能有图片、字体要加载"——也对，资源加载往往是体验瓶颈。

- 答"因为接口快不代表页面快"——这就是为什么"端到端体验指标"必须单独测。

- 收口：所以体验类指标要用"**用户视角的终点**"来定义——**内容出现在屏幕上的那一刻**。

> **易错点 / 考点：** ① RXT 的三个关键参数与"零侵入"的意义；② 打点法的三个局限；③ "测量工具可以跳出软件"这个观点能显著提升论述题层次；④ 判断题陷阱："性能测试只能用软件工具做"——硬件测量（摄像头、功耗仪）是重要补充。

> **过渡：** 全章讲完了。最后一页把 14 章收起来。

【14.68】小结（3 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

> 这一页是本章小结，也是**整门课的收口**。我先把本章收成四句，再把十四章串成一条线，最后给你两条实在的建议。
>
> **本章四句**：一，**大数据的测试贯彻数据清洗、转换、存储、加载等全过程**，靠抽样验证、数据对比分析和数据映射验证来兜底；二，**智能应用的测试是为了保证数据质量、特征质量和模型质量**，目前已有成熟的评价指标体系；三，**AI 技术（图像识别、NLP、OCR 等）已经用在单元测试、API 测试、UI 测试和日志分析上**，还能与 MBT 结合产生更好效果；四，**未来六大趋势：敏捷化、高度／超级自动化、云化、服务化、模型化、智能化**。
>
> **现在收全书。** 十四章是一条线六个台阶：**观念（1–2 章）给你立场**——测试不是为了证明对，是为了发现错；**方法（3 章）给你工具箱**——把无穷输入变成有限用例；**流程（4 章）给你节奏**——什么时候做什么事；**技术（5–9 章）给你手艺**——单元集成、系统功能、专项、缺陷与度量；**项目实践（10–13 章）给你工程能力**——把方法用进一个有 deadline 的真实项目；**展望（14 章）就是今天**：对象在变、手段在变、交付在变，判断力不变。
>
> **两条建议。** 给毕业设计：选一个小而完整的系统，把"需求—设计用例—执行—缺陷—报告"完整走一遍，哪怕只测三个模块；答辩看的是**结论有没有证据支撑**。给求职：面试八成问三个问题——怎么设计用例、怎么判断能不能上线、发现过什么有价值的缺陷；**前两个考方法，第三个考你有没有真做过**。

> **板书：** 本章四句（大数据全过程｜数据・特征・模型三质量｜AI 应用于测试各方面｜六大趋势）。全书六台

> 阶竖排：观念（1–2）→ 方法（3）→ 流程（4）→ 技术（5–9）→ 项目实践（10–13）→ 展望（14）。底部两条建议：毕设"一条链走完 + 证据支撑"；求职"三个问题：怎么设计用例・怎么判能不能上线・发现过什么缺陷"。

提问：如果只允许你用一句话向别人介绍这门课学了什么，你会怎么说？

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"——这句话很到位，包含了方法、流程与判断。

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

- 收口：建议的答案是——**"我学会了在有限时间里，用可复现的证据判断一个软件的风险在哪里、能不能上线。"**

易错点 / 考点：① 本章四句小结要能复述；② **六台阶主线**是综合论述题的标准框架（观念—方法—流程—技术—项目实践—展望）；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，**不编造市场数据与时间点**。

【思政落点 5】：十四周走下来，我最想留下的不是某个方法的名字，而是一种习惯：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。往小里说是职业素养，往大里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，工匠精神决定你肯不肯做干净。**

过渡：最后一页，我把话说完。

【14.69】感 谢 聆 听（1 分钟）

☐ 已讲　☐ 需回看　　课堂记录：_____

课到这里就结束了。我把最后几句话留给你。

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，甚至"测试"这个岗位的名字都可能变。但有一件事不会过时：**当你面对一个软件，能问出"它可能在哪儿出错、我凭什么说它没错"这两个问题。** 这就是这门课真正想给你的东西，它不是知识，是**一种看系统的方式**。

我也想提醒你：**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

时坚持把该测的测掉——这些动作很小，加起来就是用户愿不愿意信任我们的软件。**质量不是测出来的，是每个人一点一点做出来的。**

最后，谢谢这十几周坐在下面的你们。有同学问过很好的问题，有同学在实验里较过真——**较真在工程里是最值钱的品质**，希望你别弄丢它。

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。**我们江湖再见。**

> **板书：**留两行字："它会可能在哪儿出错？我凭什么说它没错？"；"质量不是测出来的，是每个人做出来的。"

提问（收尾式提问，不要求回答）：请在你自己的笔记上写一句——这门课里，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"——很实用，它在你写任何一个测试方案时都用得上。

- 答"用证据说话的习惯"——这是最有价值的答案，它超越了测试岗位本身。

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

- 收口：不管你写哪一句，请在毕业设计里用它一次——**用过一次，它才真的属于你**。

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

> **过渡：**课程结束，学习不止——把这本书留着，遇到问题时回来翻，比任何笔记都管用。

四、思政落点清单（共 5 处）

序号	所在页	落点主题
1	14.20	AI 测试最容易被误解的一句话是"让工具去判"。工具能算准确率、能跑覆盖率，却不会
2	14.58	课件一边要求"高度自动化"，一边坚持保留端到端的探索式测试——为什么？因为机器只
3	14.61	技术变化这么快，唯一不会过期的能力是"学习的能力"。你在学校学的工具，五年后可能

4	14.64	六条趋势摆在这里，最容易犯的错是追热点：听说智能化火就扎进 AI，明年换个平台又重
5	14.68	十四周走下来，我最想留下的不是某个方法的名字，而是一种习惯：说话要有证据。说"没