

# AI 驱动 软件研发 全面进入数字化时代

中国·北京 08.18-19

AI+  
software  
Development  
Digital  
summit



## 百度单元测试智能生成实践

刘晓娟 百度

# 科技生态圈峰会 + 深度研习 ——1000+ 技术团队的选择



2023K+  
全球软件研发行业创新峰会  
上海站

会议时间 | 06.09-10



2023K+  
全球软件研发行业创新峰会  
北京站

会议时间 | 07.21-22



2024K+  
全球软件研发行业创新峰会  
深圳站

会议时间 | 05.17-18



K+峰会详情



会议时间 | 08.18-19

AiDD AI+软件研发数字峰会  
北京站



会议时间 | 11.17-18

AiDD AI+软件研发数字峰会  
深圳站



AiDD峰会详情

# ▶ 演讲嘉宾



## 刘晓娟

百度资深软件工程师

2014年加入百度，负责百度研究院CI工作。2017年担任覆盖率平台技术负责人，从0开始构建了百度内部测试覆盖率度量的工具体系及平台。2018年负责了百度单测构建基础设施的建设，注重单测提效，建设了精准单测及分布式单测。2022年底着手单测智能化生成，截止目前在百度内部已支持了top研发语言的单测生成，并广泛落地在研发的日常研发工作中。

# 目录

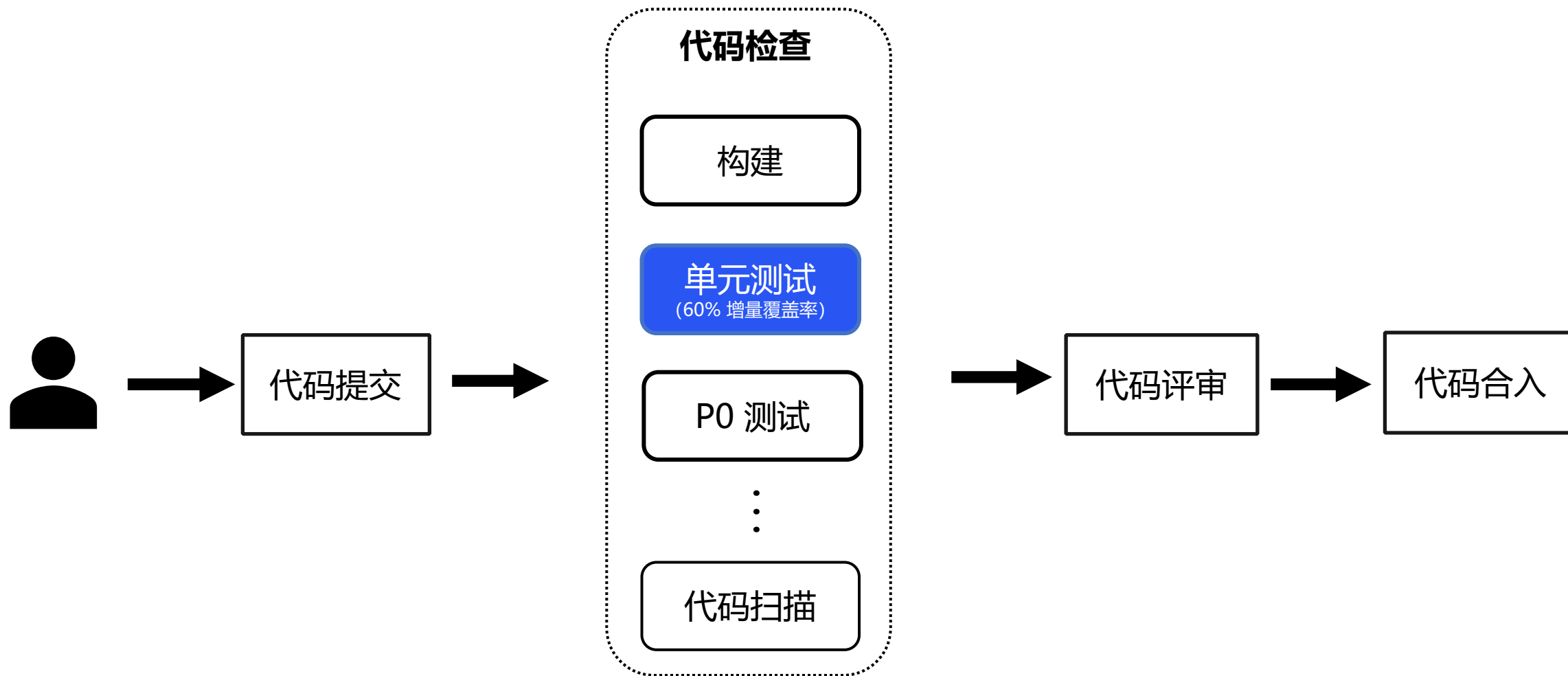
## CONTENTS

1. 背景介绍
2. 单测生成的探索
3. 百度单测智能生成的具体实现
4. 当前效果、挑战和未来展望

# **PART 01**

## **背景介绍：百度的单测**

# ▶ 背景介绍 - 单测保障代码质量



# ▶ 背景介绍 - 平台化建设路径

## 全方位度量

全量覆盖率

增量覆盖率

用例结果

## 全自动执行

自动配置

自动用例选择

分布式执行

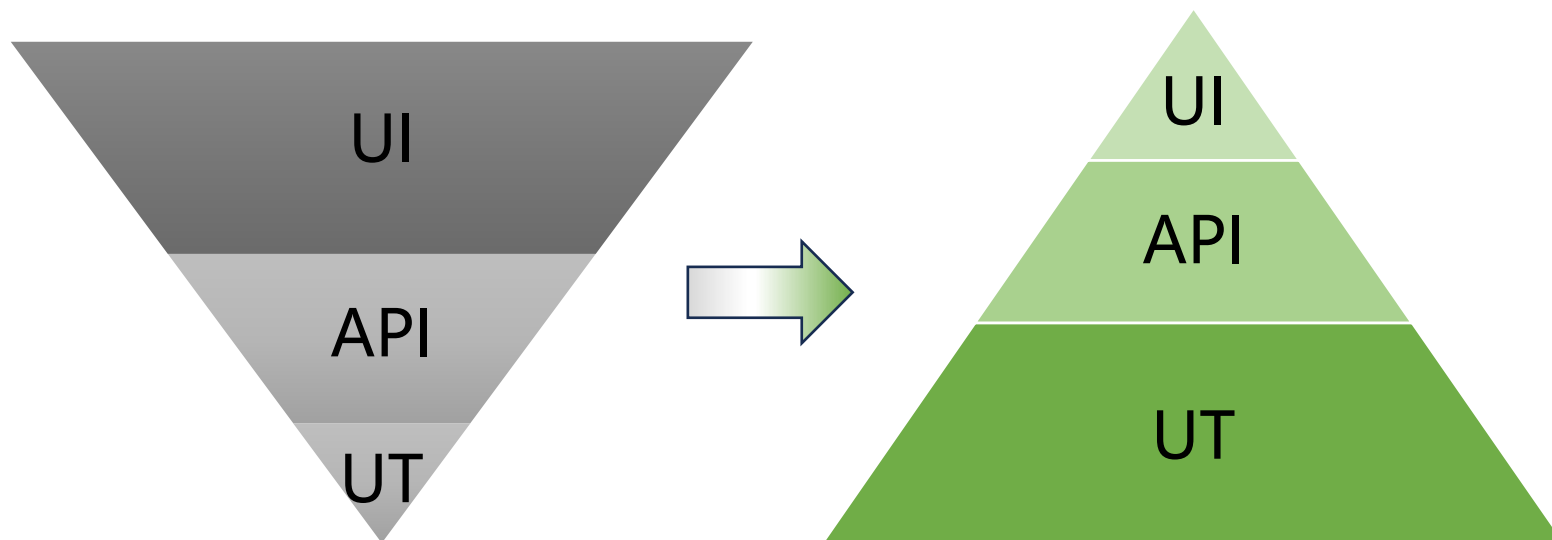
## 智能用例生成

人机交互

执行反馈修复

## ▶ 背景介绍 – 单测推广面临的问题

- ▶ 【书写成本高】据调研，单测的研发时间约占总研发时长的30%~50%
- ▶ 【项目开发周期紧】来不及写单测
- ▶ 【影响研发迭代效率】单位时间内交互的需求变少





# **PART 02**

## **单测生成的探索**



# 传统的单测生成

➤ 以追求代码高覆盖为目标，依托于代码解析技术 + 遗传变异算法，以追求代码高覆盖为目标

➤ 明显的问题：

1. 代码可读性差
2. 断言质量差
3. 生成性能差
4. 支持语言单一
5. 开发维护成本高
6. 生成环境要求高

```
@Test(timeout = 1000)
public void testCreateIssue() throws Throwable {
    // mock逻辑处理
    IssueField mockIssueField = new IssueField();
    Mockito.when(fieldService.getIssueFieldBySpaceIdAndDisplayName(Mockito.anyLong(), Mockito.anyString()))
        .thenReturn(mockIssueField);
    IssueFieldValue mockIssueFieldValue = new IssueFieldValue();
    Mockito.when(issueFieldValueService.getIssueFieldValueByIssueIdAndIssueFieldId(Mockito.anyLong(),
        Mockito.anyLong())).thenReturn(mockIssueFieldValue);
    PlanBox mockPlanBox = new PlanBox();
    Mockito.when(planBoxService.getById(Mockito.anyLong())).thenReturn(mockPlanBox);
    // 数据准备
    IssueBean issue = new IssueBean();
    LinkedHashMap<String, String[]> parameters = new LinkedHashMap();
    User opUser = new User();
    opUser.setId(1L);
    opUser.setName("xiaoming");
    Issue srcIssue = new Issue();
    srcIssue.setId(2L);
    srcIssue.setSpaceId(1L);
    srcIssue.setIssueTypeId(22L);
    srcIssue.setIssueStatusId(1L);
    IssueOperSource operSource = IssueOperSource.EMAIL_CRAWL;
    // 被测方法调用
    ServerResult actualResult = issueServiceImpl.createIssue(issue, attachmentIds: "sameName", relIssueIds: "xiaoming", path: "sameName", parameters);
    System.out.println("actualResult=" + (actualResult == null));
    // TODO: 请根据实际运行情况添加断言
}
```

## PART 03

# 百度单测智能生成的具体实现

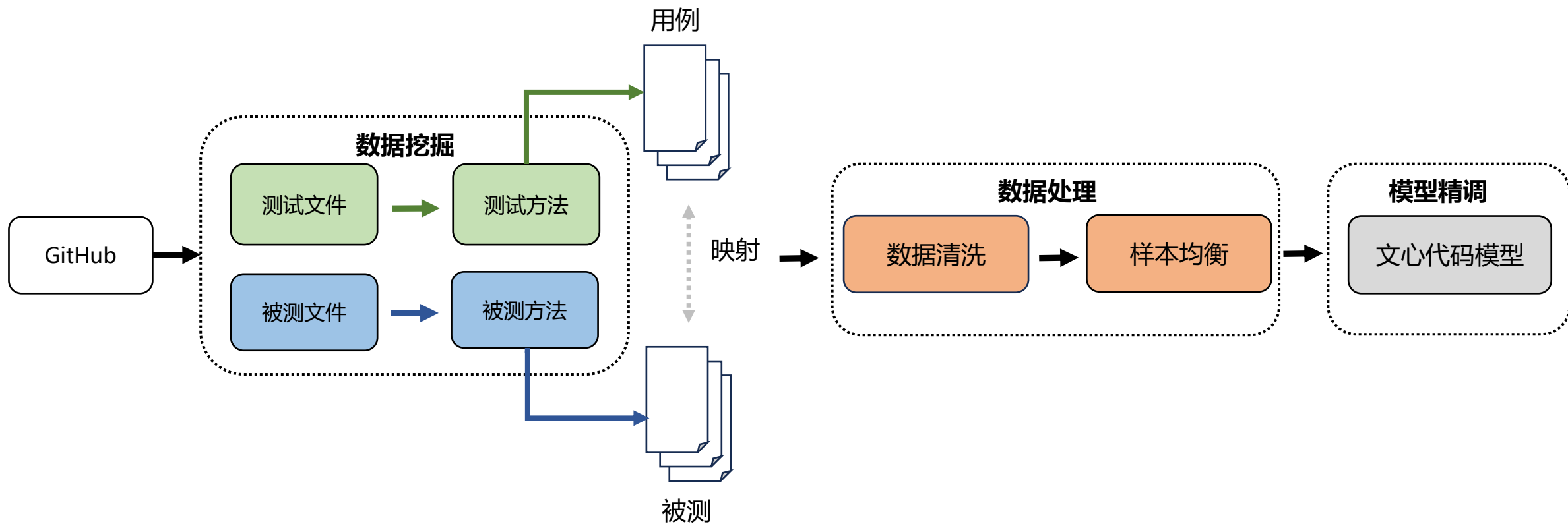
# ▶ 一、实现路径

## 用AI原生思想重构单元测试的生成



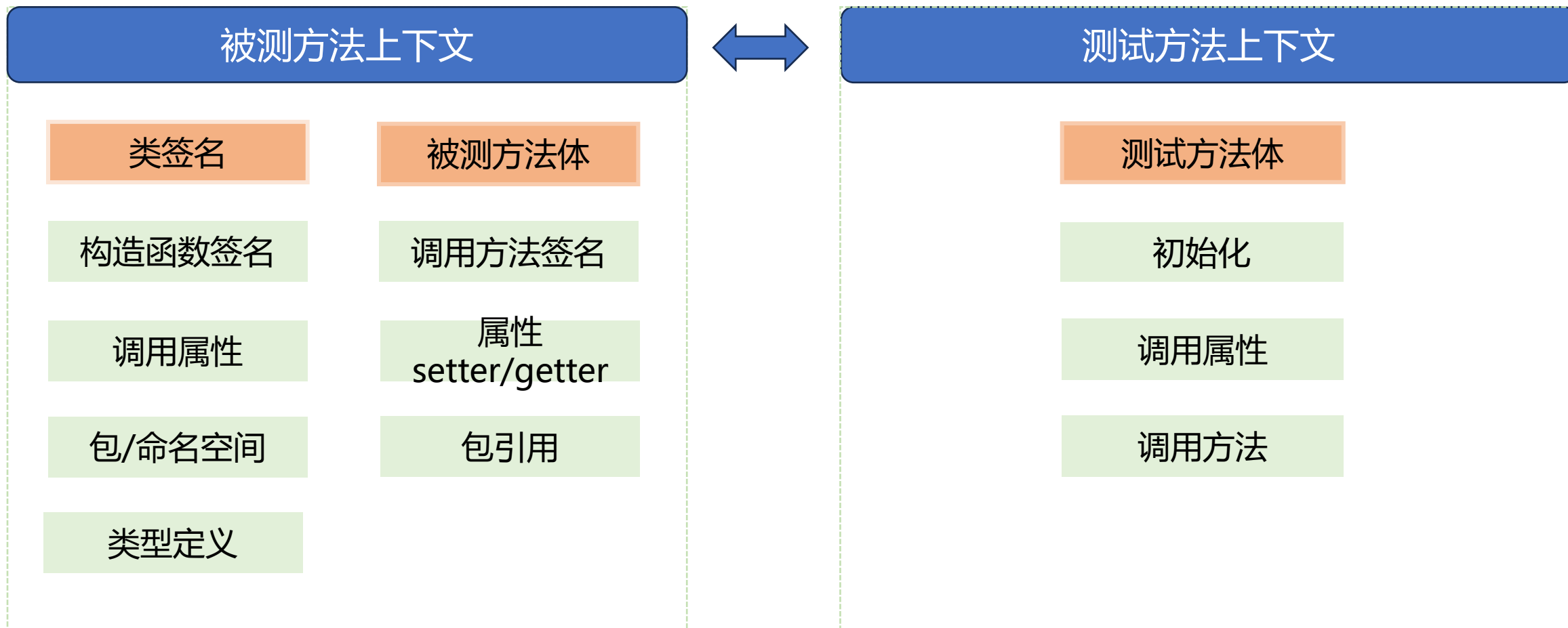
# ▶ 单测模型：模型精调整体流程

让模型会写单测，写好单测



# ▶ 单测模型：数据挖掘

数据要完善，包含足够的上下文



# ▶ 单测模型：数据处理

数据质量要高，要有各框架各场景用例书写样本

## 数据清洗

单测的基本结构

断言

被测函数调用

代码格式

无效注释

body体大小

## 样本均衡

代码复杂度

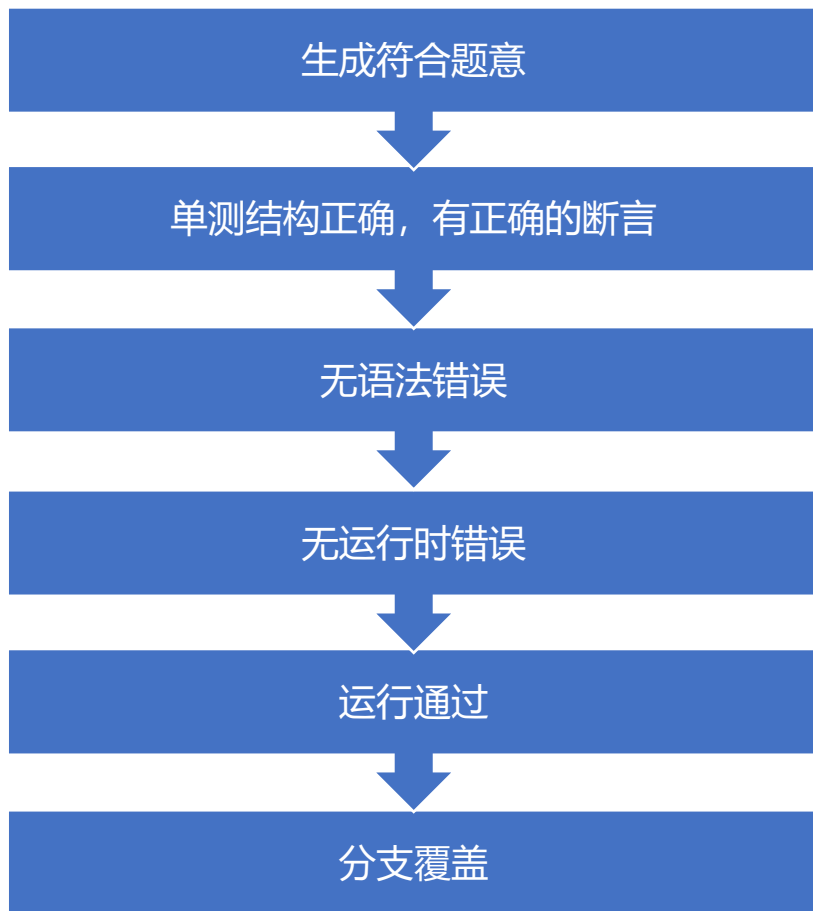
单测框架

Mock框架/业务

代码语言

# ▶ 单测模型：模型效果评估

## 评估指标：



## 优点

- 定制性：满足更多的书写风格，支持特定框架及库
- 生成能力：内容更丰富，mock处理更好
- 性能：响应更快

## 缺点

- 多场景支持
- 用例的解释能力



# ▶ 单测模型 – 提示词

## 遵循规则

- 与精调数据格式一致，提供足够的上下文
- 在token限制范围内

## 自适应上文下提示词

- 代码解析获取被测方法相关信息
- 根据优先级与token限制，构造提示词内容

```
1 -----
2 算法: 自适应上下文生成
3 -----
4 Input: 被测类签名 (FCsig), 被测类构造方法签名 (FCctor), 被测类属性 (FCflds), 被测类Getter/Setter方法签名 (FCgs), 被测方法体 (FMbody), 调用方法(IMsig), 最大提示词长度(MaxPrtLen)
5 Output: 提示词
6 context ← FCsig + FCctor + FMbody
7 prompt ← context
8 if 用到类属性 then
9     context ← context + FCflds + FCgs
10 tokens ← TokenCount(context)
11 if tokens > maxPrtLen then
12     return prompt
13 prompt ← context
14 if 有方法调用 then
15     context ← context + IMsig
16 tokens ← TokenCount(context)
17 if tokens > maxPrtLen then
18     return prompt
19 prompt ← context
20 return prompt
```

# ▶ 单测模型：提示词【无上下文信息】

## 被测方法

```
public Edge addEdge(Vertex source, String label, Vertex target,
                    Object... properties) {
    return this.addEdge(source.id(), label, target.id(), properties);
}
```

只给出被测方法信息

## 生成的用例

```
@Test
public void testAddEdge() {
    // Create a sample graph and vertices
    Graph graph = new Graph();
    Vertex source = new Vertex("source");
    Vertex target = new Vertex("target");
    // Add an edge with properties
    Edge edge = graph.addEdge(source, "label", target, "property1",
                              "property2");
    // Verify that the returned edge is not null
    Assertions.assertNotNull(edge);
    // Verify that the edge is correctly added to the graph
    Assertions.assertTrue(graph.containsEdge(edge));
    // Verify the properties of the edge
    Assertions.assertEquals("label", edge.getLabel());
    Assertions.assertEquals("property1", edge.getProperty(0));
    Assertions.assertEquals("property2", edge.getProperty(1));
    // Verify that the source and target vertices are correct
    Assertions.assertEquals(source, edge.getSource());
    Assertions.assertEquals(target, edge.getTarget());
}
```

内容幻想严重

# ▶ 单测模型：提示词【完善的上下文信息】

## 被测方法

```
public Edge addEdge(Vertex source, String label, Vertex target,
                    Object... properties) {
    return this.addEdge(source.id(), label, target.id(), properties);
}
```

## 被测类及构造函数签名

```
public class GraphManager
public GraphManager(RestClient client, String graph)
```

## 入参相关方法签名

```
public Vertex(@JsonProperty("label") String label)
public RestClient(String url, String username, String password,
                  int timeout)
```

## 入参相关方法签名

```
public String id()
public Object sourceId()
public Object targetId()
public String sourceLabel()
public String targetLabel()
```

## 生成的用例

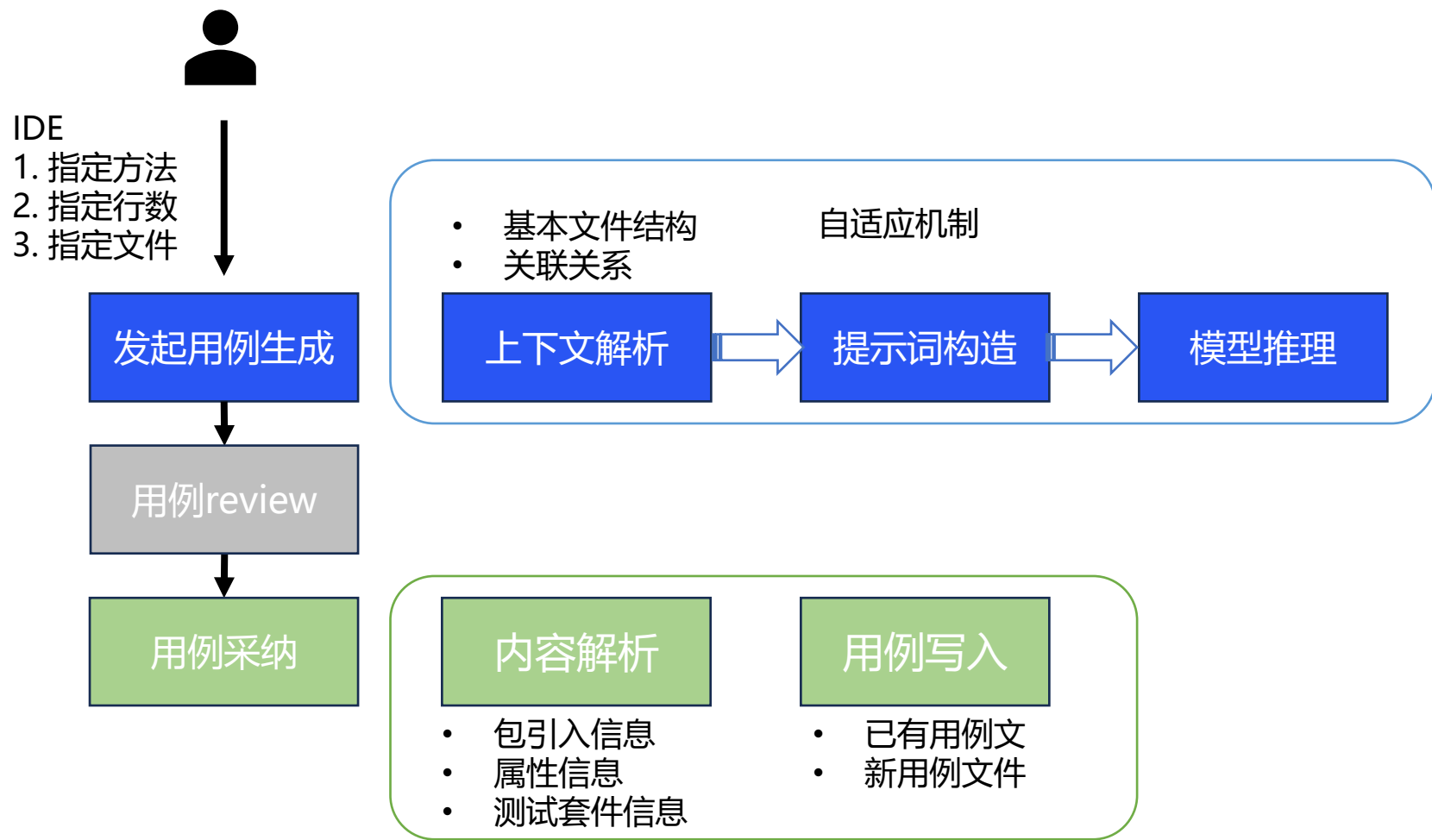
```
public class GraphManagerTest {
    private GraphManager graphManager;
    private String graph = "testGraph";

    @BeforeEach
    public void setup() {
        RestClient client = new RestClient("http://example.com", "username",
                                          "password", 5000);
        graphManager = new GraphManager(client, graph);
    }

    @Test
    public void testAddEdge() {
        // Create the necessary objects
        Vertex source = new Vertex("sourceLabel");
        Vertex target = new Vertex("targetLabel");
        Object[] properties = new Object[]{"property1", "property2"};
        // Call the addEdge method
        Edge result = graphManager.addEdge(source, "label", target,
                                          properties);
        // Verify that the addEdge method returned a non-null result
        Assertions.assertNotNull(result);
        // Verify that the source and target vertices were used correctly
        Assertions.assertEquals(source.id(), result.sourceId());
        Assertions.assertEquals(target.id(), result.targetId());
        Assertions.assertEquals("sourceLabel", result.sourceLabel());
        Assertions.assertEquals("targetLabel", result.targetLabel());
    }
}
```

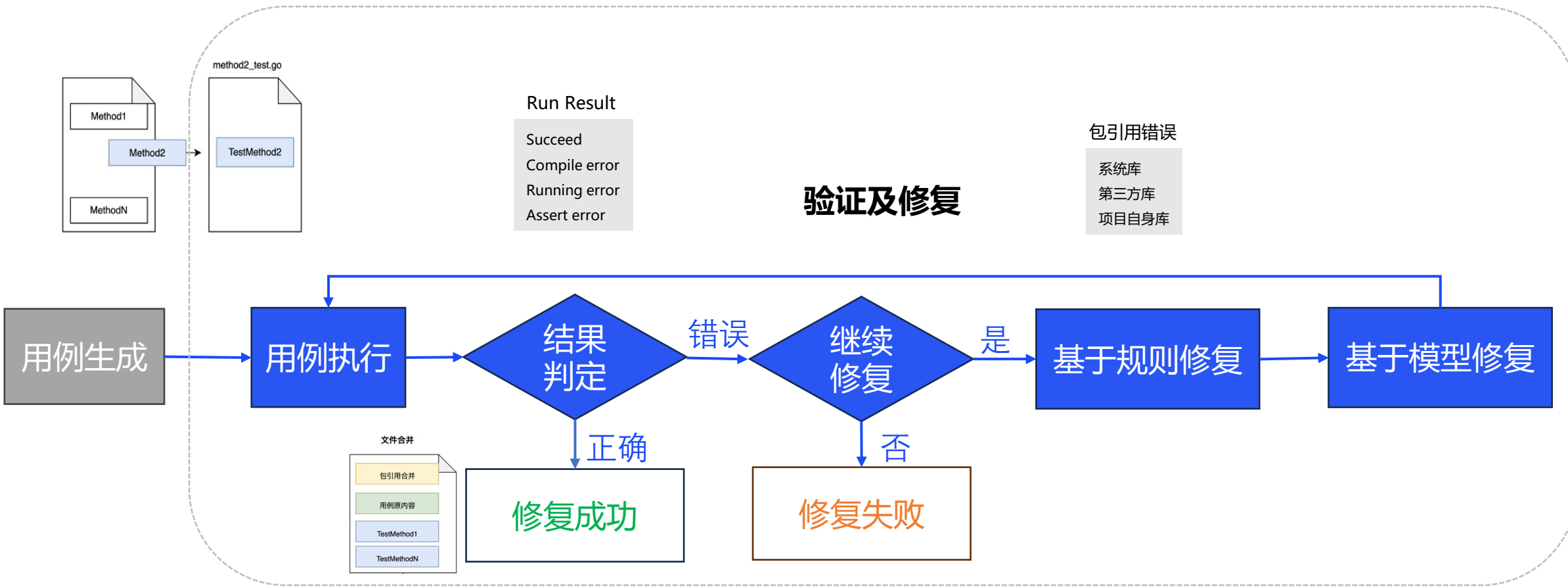
内容书写正确

# ▶ 产品化：IDE人机协同模式



# 工程化：批量可用用例生成

## 自验证自修复



# **PART 04** **当前效果、挑战和未来展望**

# ▶ 当前效果 – IDE编码场景

## ➤ 覆盖主流研发语言



## ➤ 覆盖主流IDE

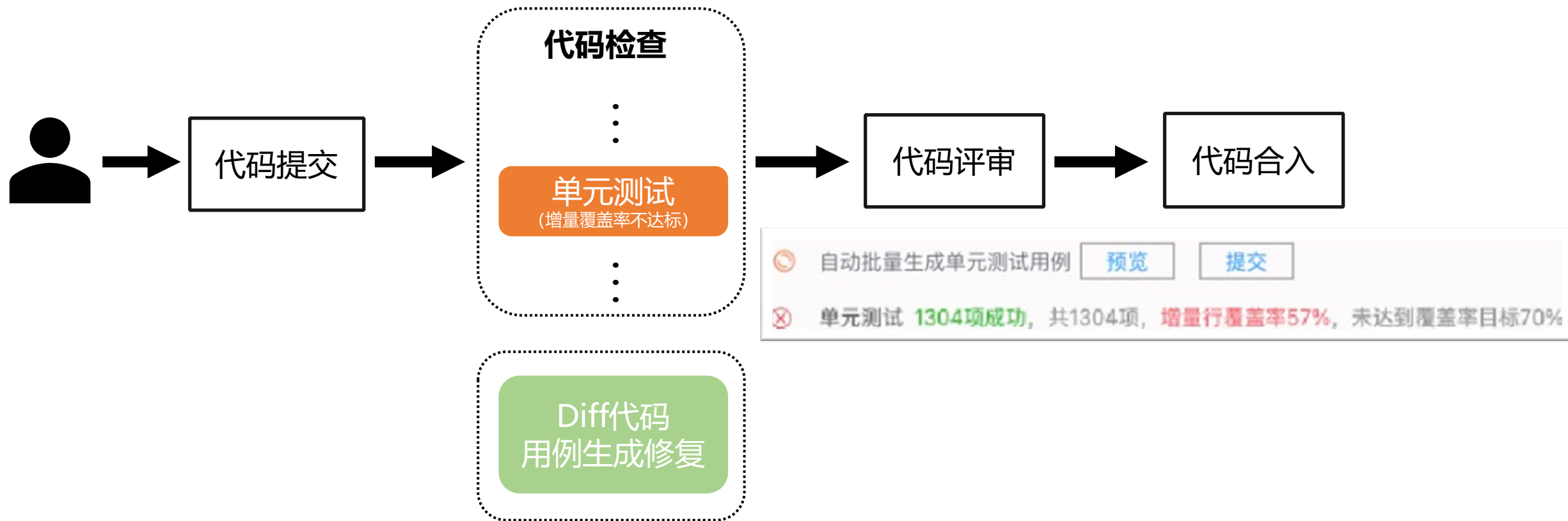
- IntelliJ 家族
- Vscode

## ➤ 落地全百度日常的研发工作

A screenshot of a code editor window titled "smartUT-evaluator - strings.go". The editor shows Go code for a function named "Compare". The code includes package declarations, comments explaining the function's behavior and performance considerations, and the function implementation using if-else statements to return 0, -1, or +1 based on string comparison. The function signature is "func Compare(a, b string) int". The editor interface includes a top toolbar with various icons, a sidebar on the right with "百度 Comate" and "API Security Audit" panels, and a status bar at the bottom showing "14:18 LF UTF-8 制表符 正在变基 master".

```
1 package helper
2
3 // Compare returns an integer comparing two strings lexicographically.
4 // The result will be 0 if a == b, -1 if a < b, and +1 if a > b.
5 //
6 // Compare is included only for symmetry with package bytes.
7 // It is usually clearer and always faster to use the built-in
8 // string comparison operators ==, <, >, and so on.
9 func Compare(a, b string) int {
10 // NOTE(rsc): This function does NOT call the runtime cmpstring function,
11 // because we do not want to provide any performance justification for
12 // using strings.Compare. Basically no one should use strings.Compare.
13 // As the comment above says, it is here only for symmetry with package bytes.
14 // If performance is important, the compiler should be changed to recognize
15 // the pattern so that all code doing three-way comparisons, not just code
16 // using strings.Compare, can benefit.
17 if a == b {
18     return 0
19 }
20 if a < b {
21     return -1
22 }
23 return +1
24 }
```

# ▶ 当前效果 - CR阶段用例批量生成





# ▶ 挑战

- 正确性的提升，更少的幻想
- Mock技术的合理应用
- 断言的正确性
- 高场景（分支）覆盖的用例生成
- 多用例的写入文件合并

# 感谢聆听

