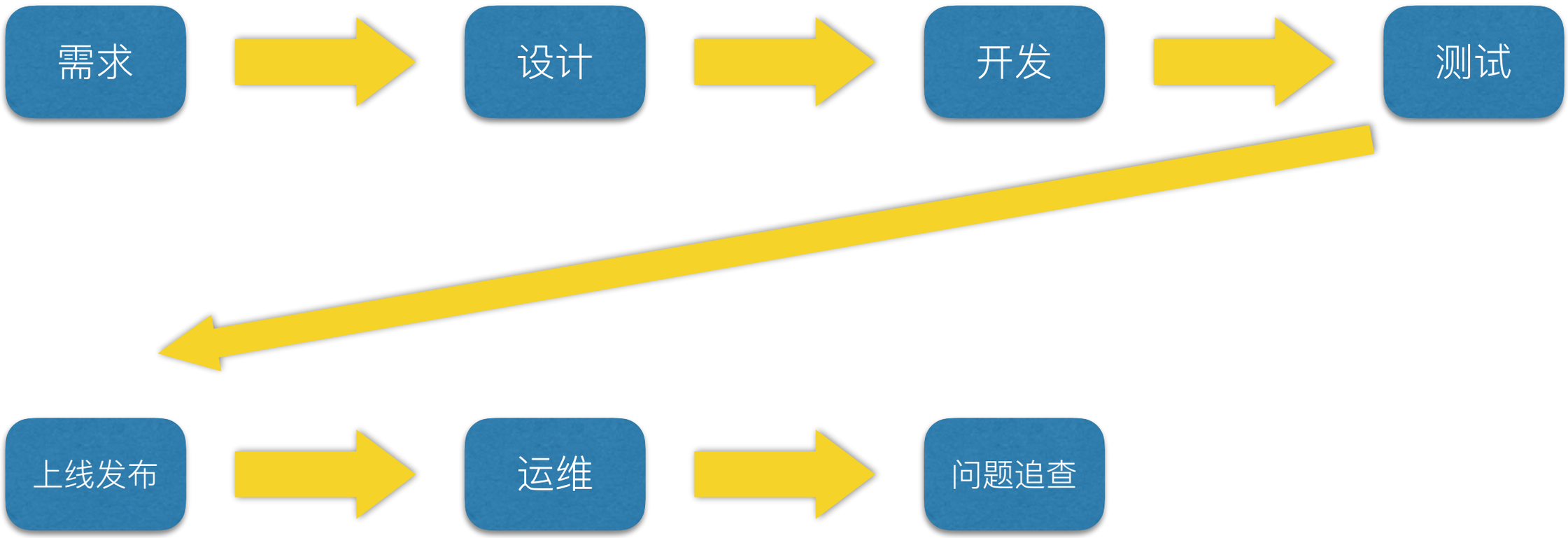
通用开发过程

General development pattern

唐义哲 @ 风控

2016-04

开发过程



你在完整执行这些过程吗？

开发过程

步骤可能从简

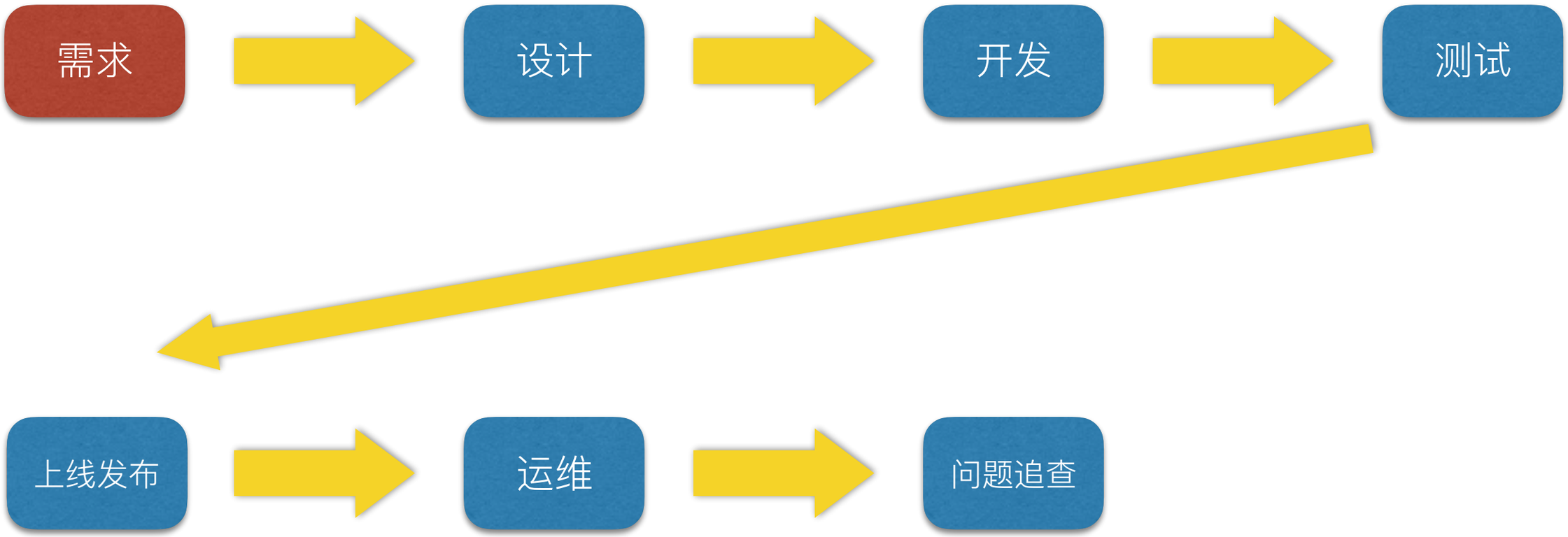
各步骤有很多注意点

细分目的——明确模式，了解取舍

【必须】

【建议】

【可选】


需求
设计
开发
测试
上线发布
运维
问题追查

需求阶段

“需求”

功能模块	原型图	功能点说明
2.1、报表数据阅览下载功能（我的订阅）		• “我的订阅”菜单栏下为用户订阅所有报表明细情况 • 点击其中一张报表可查阅该报表的数据情况（日期未选择，默认为当日数据） • 通过选择日期可查看和下载改日期的报表数据
2.2、报表使用数据统计功能		该功能只展示给风控内部成员，包括两部分功能 A.报表使用数据详情 • 展示以下数据：报表名称，负责RD，上线时间，在线时间，当日点击数量，周均点击数量，月均点击数量 • 报表的负责人（RD&PM）只需要看到自己负责相关报表的数据情况 B.报表日点击变化 • 折线图当中，纵坐标代表报表点击量，横坐标代表日期，每一条折线代表一张报表的日点击数据变化情况（默认情况下为一周的点击变化情况） • 通过时间段的选择，可查看报表在该时间段内的点击情况 【备注】：每个mis账号每日点击数量只记录一次

需求阶段

“需求”

是：要解决的问题，不是：要做成某个样子

不要和设计混淆，RD要理解问题

一、背景

目前风控所有的报表通过邮件的形式实现，需求方通过接收邮件的方式关注报表数据。随着报表需求的增加，每天风控组内成员接收到的邮件就有几十封，混合其他业

务邮件，造成了目前报表邮件比较混乱的局面。报表的需求一直在做加法，几年前的报表需求可能还在线上维护，没有合理的上下线管理同样是目前邮件报表的痛点，

加上报表权限管理的基本依赖"人肉"添加，没有合理的报表数据异常通知渠道等等一系列问题，亟需要开发一个平台来解决目前"刀耕火种"的邮件数据报表形式。

二、产品要解决的问题

1.监控报表使用情况，解决目前报表没有上下线标准的问题，减少不必要的报表维护成本

2.报表使用和查看入口统一，解决邮件报表杂乱，打扰率高，不方便查看问题

3.标准化，流程化权限管理，解决目前报表查看权限没有统一管理问题

4.报表指标数据出现异常大象提醒，解决指标数据异常，无法及时通知使用人员查看问题

5.报表数据指标，自主拖拽生成，解决数据报表开发依赖开发人员且比较耗时的问题

需求阶段

需求文档

【可选】小而稳定的团队，无需外部合作，或自己是需求方，小需求

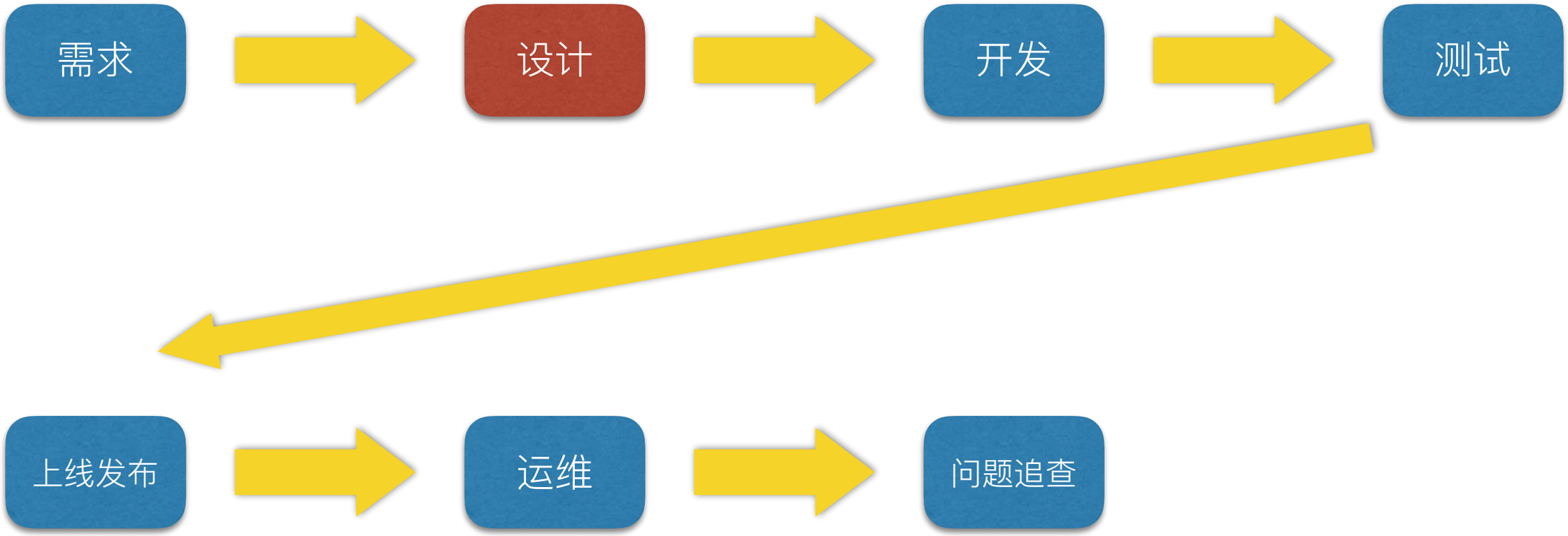
【必须】反之

需求确认

【必须】理解初始问题，理解推衍过程，确认，复述给需求提出者

结果衡量*

【推荐】尤其高并行化团队，关系到优先级，推衍出观测需求


需求
设计
开发
测试
上线发布
运维
问题追查

设计阶段

功能设计

用什么方法解决这个问题——是需求理解的延续

技术设计

技术实现细节

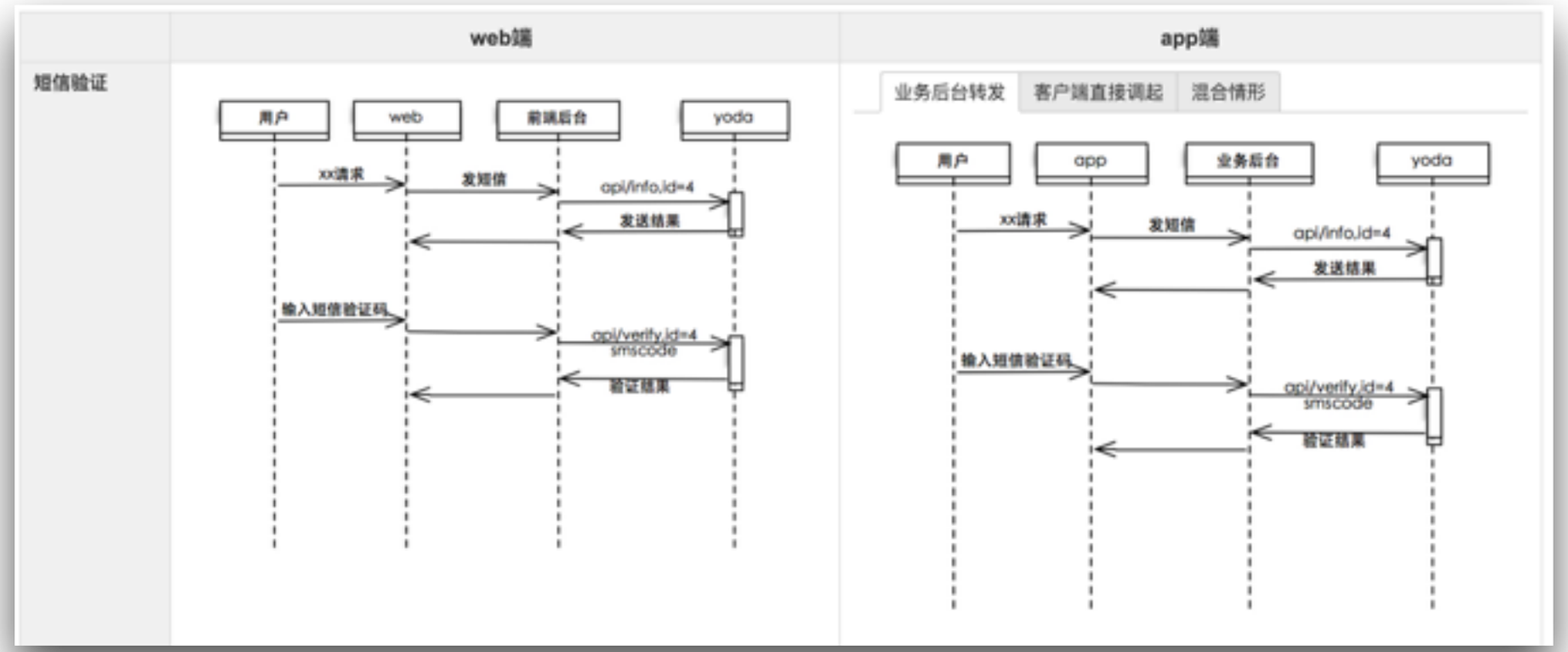
设计阶段——功能设计

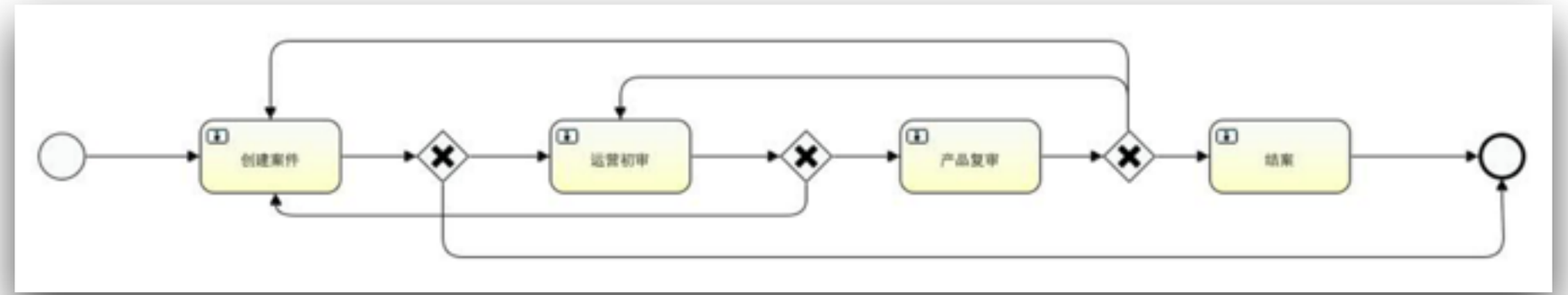
方法

- 衡量反作弊的效果——？
- 运营处理无存档——？
- 规则快速上线——？

设计阶段——功能设计

交互


web端
app端
短信验证
用户
web
前端后台
yoda
xx请求
发短信
api/info,id=4
发送结果
输入短信验证码
api/verify,id=4
smscode
验证结果
业务后台转发
客户端直接调起
混合情形
app
业务后台


创建案件
运营初审
产品复审
结束

设计阶段——功能设计

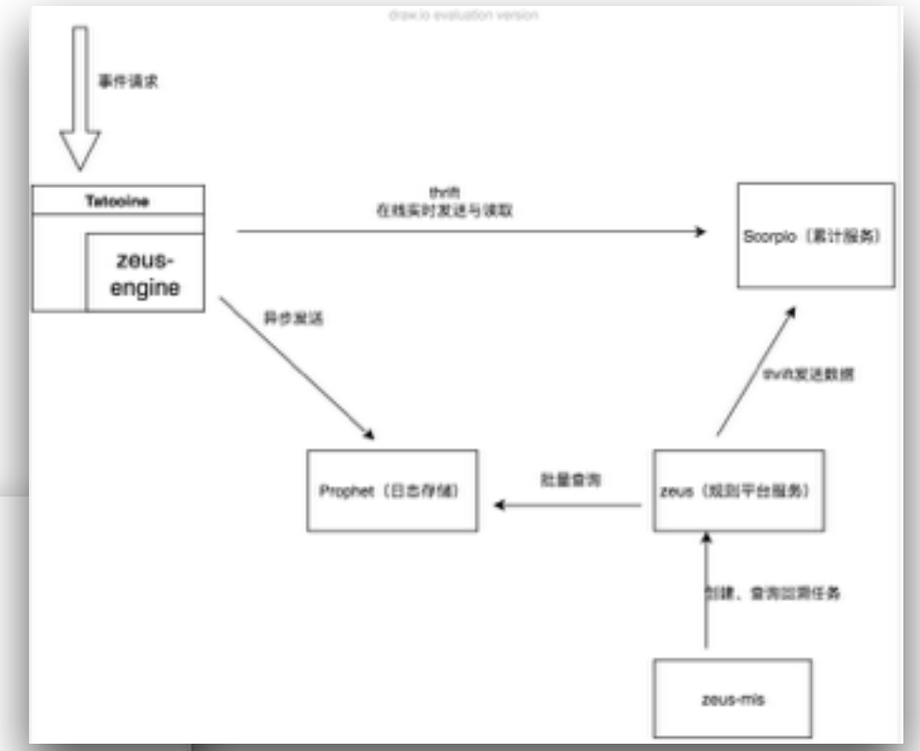
界面

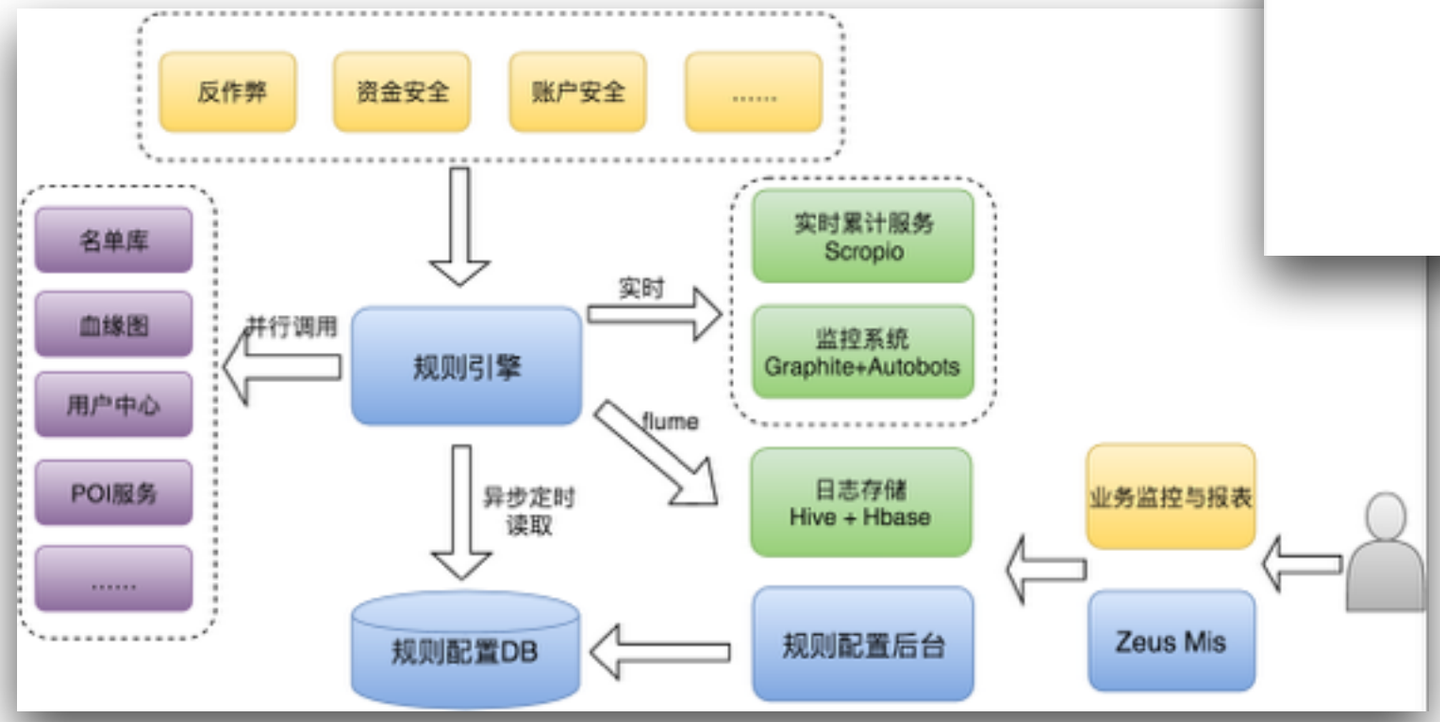
详细功能

设计阶段——技术设计

构架设计

【必须】设计公开





设计阶段——技术设计

容易忽略的：

技术选型

部署设计

【必须】思考部署位置（机房，是否分布式）

【可选】业务隔离

降级设计

【推荐】外部依赖、数据库、网络——无法访问

【推荐】快速降级

安全设计

【必须】可以推迟，但不可不做

信息安全、漏洞（验证码、短信轰炸、DDOS等）

【推荐】访问认证

设计阶段——技术设计

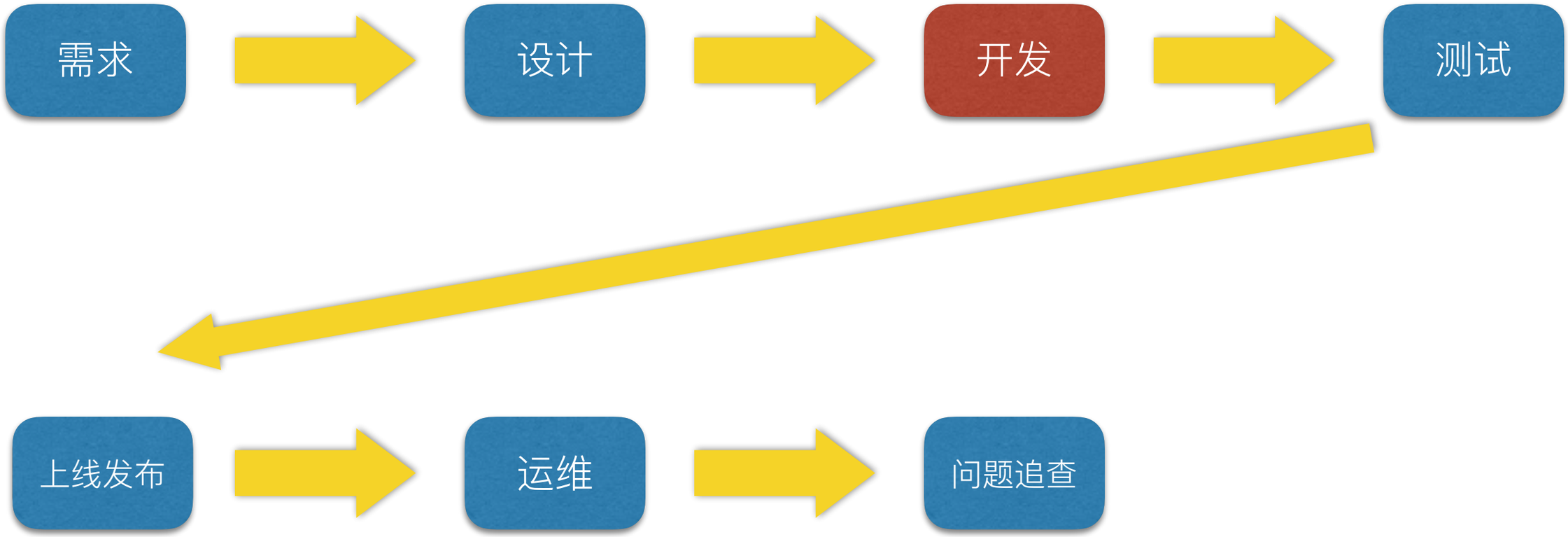
【必须】设计评审

不同部分由不同角色评审

- 功能评审
- 构架评审
- 技术方案评审

评审什么

- 功能完整性
- 可扩展性（功能、技术）
- 可监控？
- 安全性
- 容量规划
- …


需求
设计
开发
测试
上线发布
运维
问题追查

开发阶段

开发规范

【必须】文档：开发规范、接口、代码表、数据表说明等

【必须】日志

【推荐】单测：先确定基线，持续提高

【推荐】异常、降级

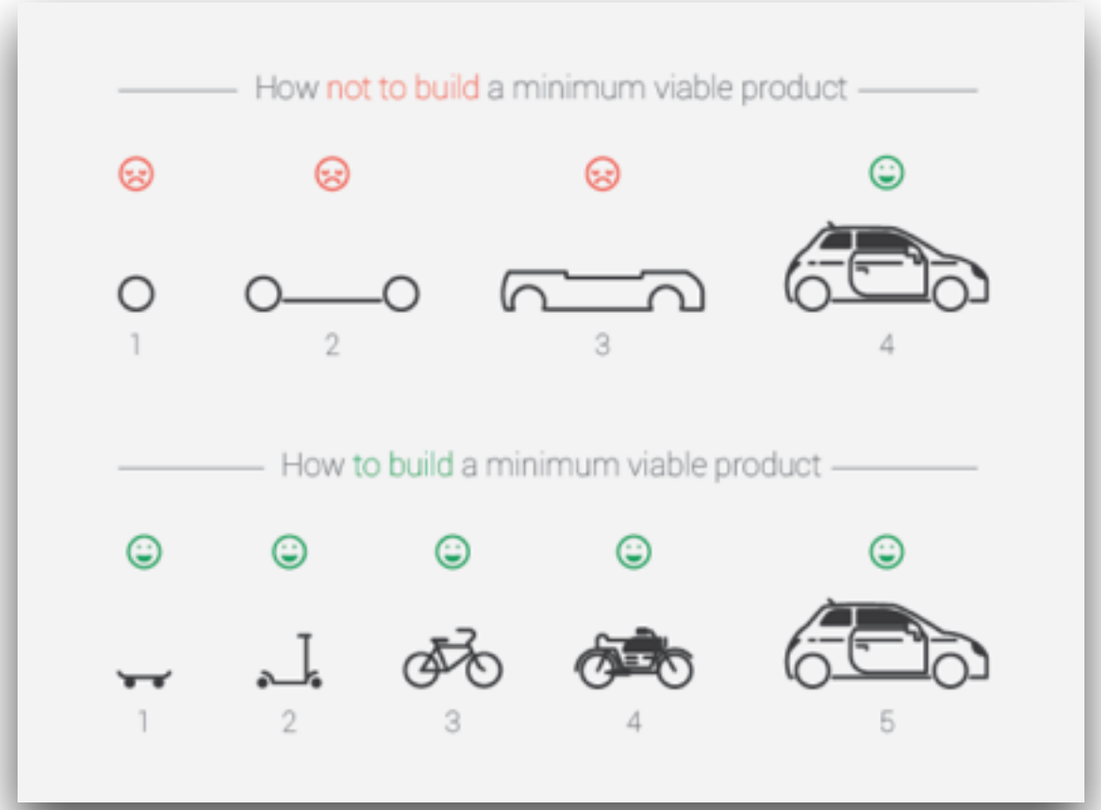
开发阶段

代码设计

抽象、层次划分、复用性、可扩展性

快速迭代

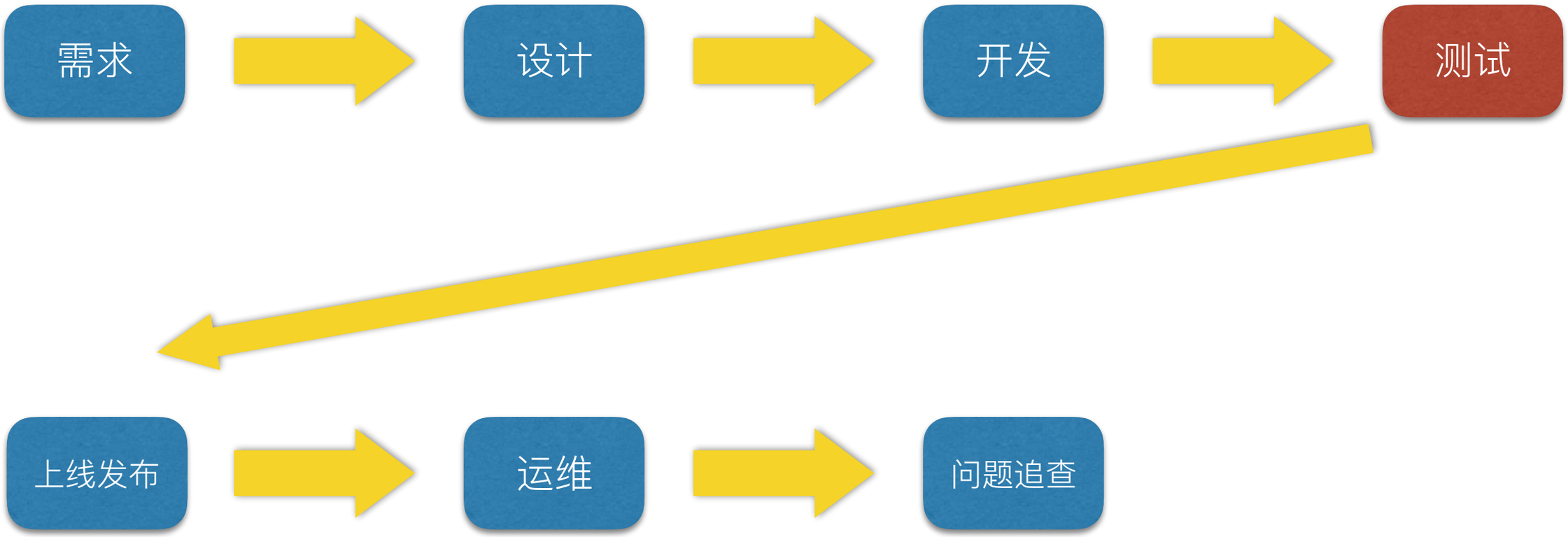
尽早成型可Review，Review范围从小及大——减少返工



开发阶段

代码审查

- 你相信Code Review吗？为什么？
  - Code Review减少了多少问题？
  - 你的Code Review有效吗？
  - 必要时当面讲解


需求
设计
开发
测试
上线发布
运维
问题追查

测试阶段

测试环境

【必须】独立测试环境（或方法）

验证

【必须】线下、线上分别验证——反面案例？

【必须】测试用例的管理

【推荐】固化测试流程（尤其是联调）

集成测试

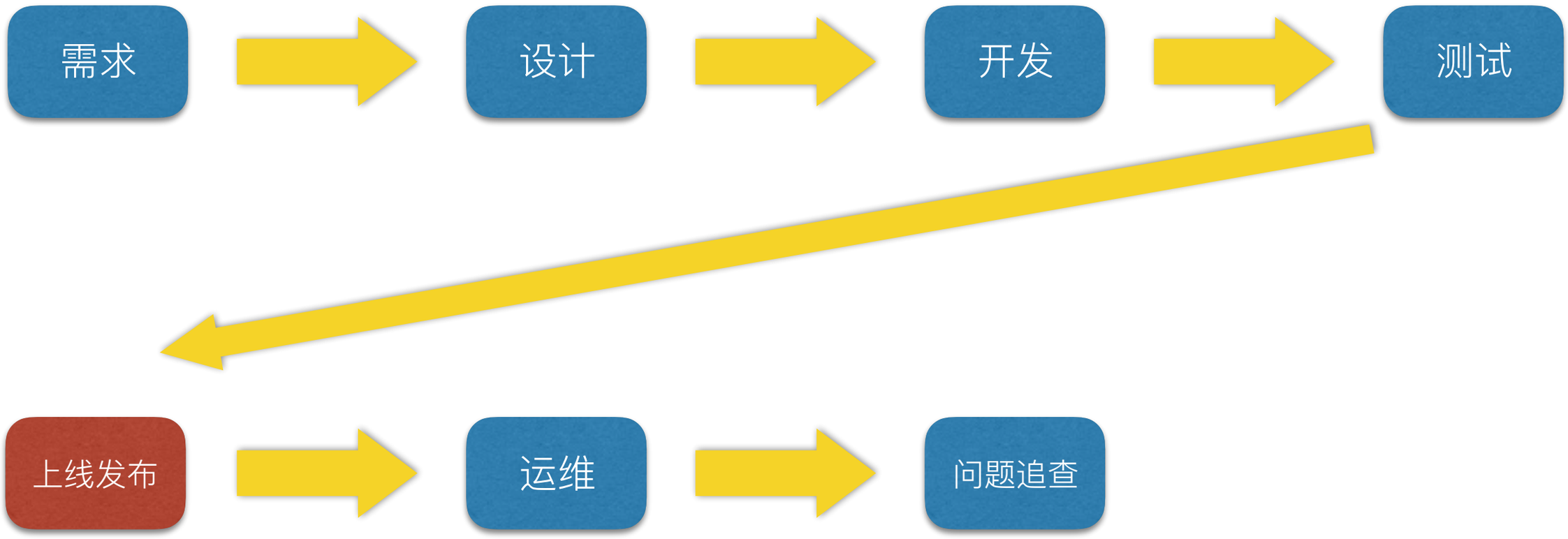
【推荐】自动化，结果清晰可观测

压测

【可选】

Bug跟踪

【可选】


需求
设计
开发
测试
上线发布
运维
问题追查

上线/发布

发布流程

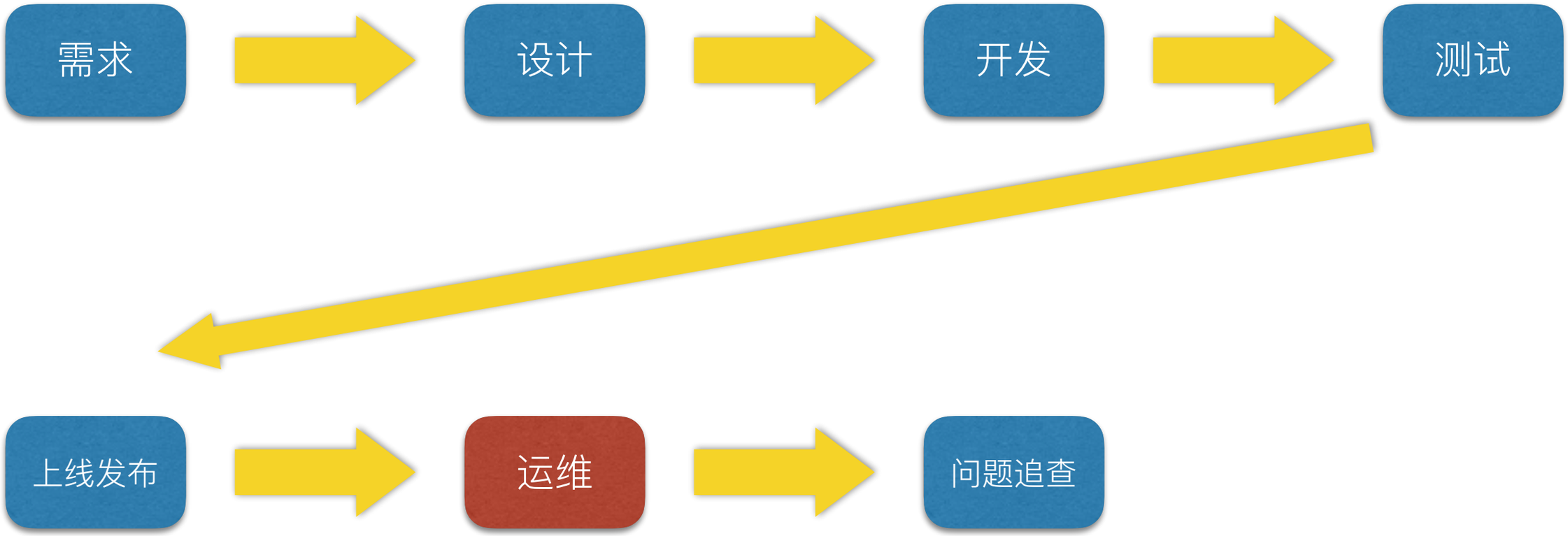
【必须】自动化发布过程

【必须】灰度发布

【必须】有观测指标：日志、数据、反馈、客诉等

Case：打包问题

Case：误上线


需求
设计
开发
测试
上线发布
运维
问题追查

运维阶段

观测指标

【必须】可用性、响应时间、报错

【必须】业务指标（核心数据、客诉等）

【推荐】观测指标的便捷性

是否汇总、相关人是否知道

反馈渠道

如何接收问题和建议？

运维阶段

问题处理

【推荐】固化处理流程：问题定位、回滚、联系人列表

步骤

判断现状

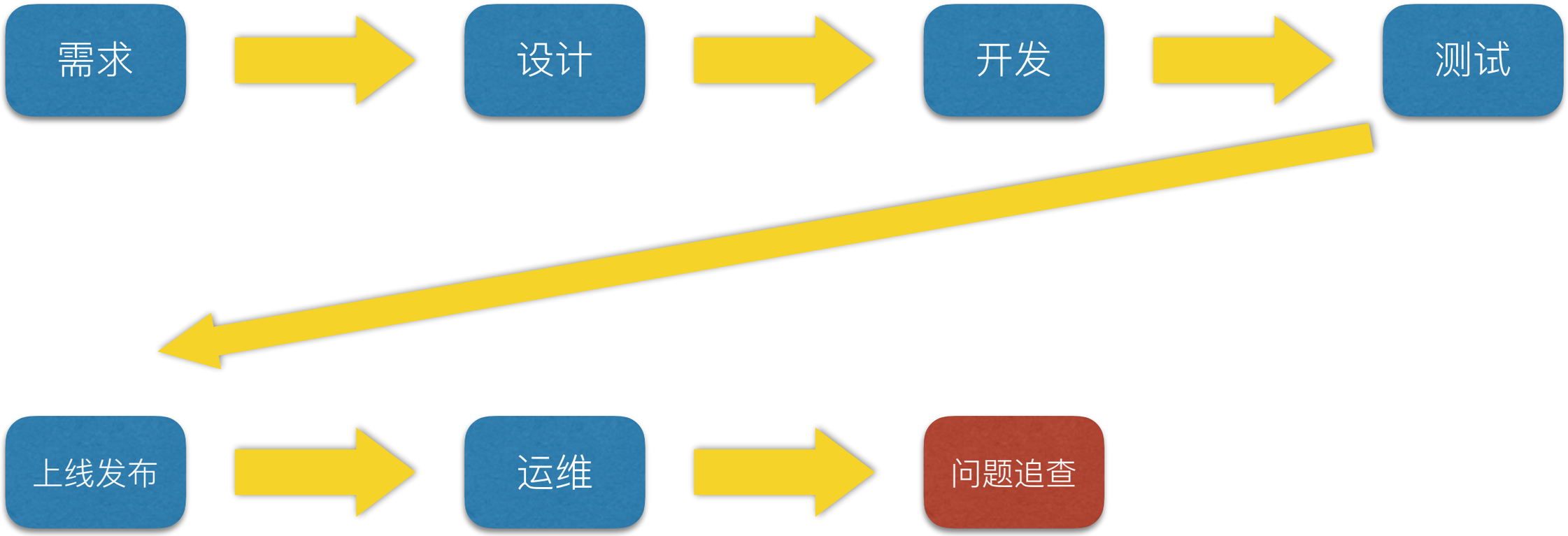
【必须】周知影响

与上下游跟进

e.g. 与上游、依赖方、DB、SRE、网络跟进

现象（上下文、版本、报错、恢复与否）、影响（紧急否）、猜测、需要的数据和印证

【必须】周知处理动作、是否恢复


需求
设计
开发
测试
上线发布
运维
问题追查

问题追查

五个为什么——如何落地？

从直接原因向前推衍

问题追查

运维

多长时间发现的、为何不容易观测、反馈渠道是否通畅、是否有人值班、人工干预便捷性

上线发布

是否有灰度？业务隔离？事前压测？容量规划？是否执行了流程？流程是否合理？

测试

新功能是否验证过？集成测试？压力测试？对比测试？

开发

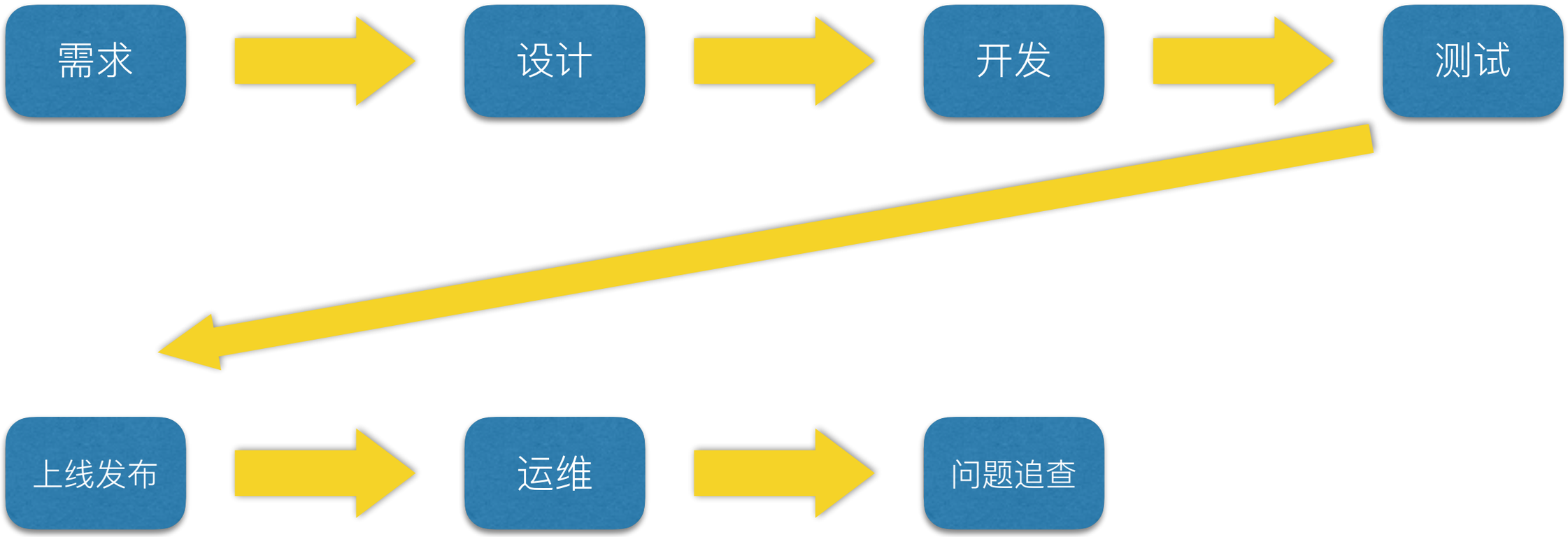
Code Review有效性？是否符合规范？

设计

功能设计 是否合理，容易被用户理解？技术设计 是否有缺陷？观测指标是否完备？

需求

理解是否正确？这个需求是不是合理？



你在正确执行这些过程吗？

Q & A