# GCR 研发全域闭环治理 · 推行手册

| 项 | 内容 |
| --- | --- |
| **文档名称** | GCR 研发全域闭环治理 · 推行手册 |
| **版本** | V1.0 |
| **发布日期** | 2026 年 7 月 18 日 |
| **作者** | 周老师 |
| **定位** | 推行者的作战手册，不是执行者的操作说明书 |
| **适用范围** | 20–200 人研发团队（多项目并行）；超 200 人须叠加组合层治理 |
| **配套文档** | 《GCR 数字化落地指南（明道云 HAP）》 / 《GCR 方法论白皮书》 |
| **版权与转载** | 原创方法论，引用请注明出处与作者周老师 |

> 本手册不绑定任何工具。规则用白板、Excel、飞书表格甚至纸质看板都能跑；数字化只是把规则固化、让流转自动化的加速器，详见配套《GCR 数字化落地指南》。

## 目录

# 前言

这本手册只解决一个问题：**研发一团乱，怎么一步步把它治住**——做法是先让人站对位置，再让流程规范行为。

本手册不绑定任何工具。规则用白板、Excel、飞书表格甚至纸质看板都能跑；数字化只是把规则固化、让流转自动化的加速器，详见配套《GCR数字化落地指南》。没有数字化系统，这本手册照样成立。

【GCR】以【闭环】为治理内核：每个工作项从进入到关闭 / 上线，全程可追溯、可度量、可改进——这是贯穿全手册的红线。

## 这本手册是给谁写的（以及不是给谁写的）

【GCR】有两类核心读者，请先认领自己的位置，这决定了你该怎么用这本手册：

| 读者 | 该不该读这本手册 | 怎么读 |
| --- | --- | --- |
| **推行者（GCR 教练 / 顾问 / 研发效能负责人）** | **必须读，且是首要读者** | 通读全文，重点吃透每一章的为什么与判断口径——你不是来执行规则，你是来**把规则种进一家抵触的组织** |
| **产品负责人（判断型执行者）** | 应当读 | 重点读第二章需求治理与第三章迭代，懂为什么才填得对字段 |
| **执行层（研发 / 测试 / 设计 / 服务台）** | 不用读 | 规则编进数字化系统后，照着系统点就行；本手册对他们来说是黑话合集 |
| **需求方（被治理方）** | 不用读 | 靠推行者 / 产品负责人转述；本手册不面向他们写 |

> 一句话定位：**这本手册是推行者的作战手册，不是执行者的操作说明书。** 你拿到一家乱的研发组织，靠它知道"先打哪、怎么打、卡住了往哪松"。

## 为什么需要一本"推行者"手册

多数研发治理方法（Scrum、SAFe）的文档默认读者是"已经决定采用的人"，写的是"流程长什么样"。但真实推行现场最痛的不是"不知道流程"，而是：

- **老板随时插单**，流程挡不住绝对权威；
- **需求方绕过产品直接找开发**，唯一入口形同虚设；
- **需求池越堆越满**，没人敢关、没人会关；
- **团队说"我们情况特殊"**，每一条规则都被现场打折扣。

【GCR】 的设计哲学是：**系统让规则更透明，最终决策权始终在人。** 这本手册因此不追求"写完流程就完事"，而是把每一章拆成四段，专门服务推行者：

1. **为什么**——这条规则防的是什么痛、不治会烂在哪；

2. **怎么推**——在一家抵触的组织里，怎么一步步落地、怎么说才有人听；
3. **规则参考**——状态机、矩阵、公式是"标准答案"，执行层照此执行；
4. **判断口径**——什么时候该严、什么时候可松、踩线了往哪退，这是推行者真正的功力所在。

## 推行路线图：按这个顺序走，不要跳步

1. 团队职责规范（第一章）
2. 需求治理（第二章）
3. 迭代与交付（第三章）
4. 服务工单治理（第四章）
5. 效能度量（第五章）

> **不要跳步。** 团队没组建好就做收口，收口做了就排迭代——顺序乱了，每一步都在补前一步的坑。这本手册的章节顺序本身就是推行顺序。

**本手册编排对应【GCR】全域架构全景图的"四大治理域 + 一个横贯基座"：**

- **顶层 • 目标与商业价值**：【用户需求】是价值起点（第二章），【交互设计】作为需求画像期的就绪输入挂在其下；
- **基座 • 【GRT】**：统一时间轴 × 资源透明，横贯需求 / 交付 / 服务 / 度量四域，不是某一章的局部工具；
- **质量与风险管控域**：需求画像 + 【故障缺陷】质量门禁（第二、三章）；
- **交付管理域**：【迭代与交付】（第三章）；
- **服务响应域**：【服务工单治理】（第四章）；
- **效能度量域**：贯穿全图的度量基线（第五章）。

## 核心名词先对齐（贯穿全手册）

| 名词 | 定义 | 粒度 | 生命周期归属 |
| --- | --- | --- | --- |
| **用户需求** | 需求方提出的、偏价值的诉求，可能横跨多个迭代 | 大（月/周级） | 第二章 |
| **交付功能** | 用户需求拆解后、装入迭代执行的单元 | 中（迭代级） | 第三章 |
| **执行用例** | 针对单个功能点的最小测试单元（结果 通过/失败） | 小（日级） | 第三章（3.3.8） |
| **故障缺陷** | 验收或运行中发现的异常，需修复闭环（交付的质量门禁） | 小（日级） | 第三章（3.3.10） |
| **服务工单** | 一线用户/需求方提出的服务请求 | 服务入口 | 第四章 |

# 01 团队职责规范

## 1.1 为什么：先搭团队，再做收口

【GCR】 的一切流程都建立在"稳定的跨职能团队"之上。这是推行第一关，也是最容易被跳过的一关——很多推行者一上来就推需求收口、推迭代，结果需求进来了没人接、迭代排了没人测，流程在沙滩上。

**不治会烂在哪：**

- 人员按项目临时拼凑 → 没有稳定归属，需求"绕口"直接找人，唯一入口瞬间瓦解；
- 一个人跨多条业务线打杂 → 没有人对任何一条业务线的价值负责，优先级变成"谁嗓门大"；
- 研发兼任测试 → 没有独立验证，质量门禁形同虚设，"「已上线」"等于"研发说完了"；
- 服务台消失 → 工单没人接、没人回访，一线问题反噬需求池。

> **【GRT】（【双维全域治理】）是横贯四域的治理基座**：统一时间轴 × 资源透明，把团队、需求、交付、服务、度量串成一张网——它不是某一章的局部工具，而是后续所有流程跑在其上的底盘。本章先定"人"，后续章节在【GRT】 基座上跑流程。

### 四条红线（推行者必须向全员宣贯并守住）

1. **一条业务线 = 一支跨职能团队**，不接受"一个人跨多个业务线打杂"。
2. **产品、开发、测试是最小不可拆单元**，缺一个角色就不是研发团队。
3. **服务台可以兼职，但不能消失。**团队再小，也必须有人对工单服务负责。
4. **角色是职责的容器，不是人的标签。**同一个人可以承担多个角色，但职责边界必须分开、可追踪。

## 1.2 怎么推：在现有组织里长出第一支团队

推行者最常遇到的现实是：组织里根本没有"业务线团队"这个概念，大家按项目临时组队。不要试图一次性重组整个组织，按以下步骤落地：

**第一步：找到第一条值得单独治理的业务线。** 选体量足够、痛点最明显的一条（需求最乱、插单最多、交付最不准时）。说服老板：先拿这一条线做试点，跑通了再复制，不要求全网一步到位。

**第二步：从现有人员里划出"产品 / 开发 / 测试"三角。** 哪怕开发暂时只有 1 人、测试暂时兼任，也必须明确这三个职责的归属人。三角缺一，后续所有收口都是假动作。

**第三步：任命「研发效能负责人」作为边界守护者。** 这个人**不属于任何业务线**，他管的是团队之间的冲突与规范巡检。如果组织里暂时没有这个角色，推行者本人先顶上——这是推行能落地的关键支点。

**第四步：服务台先挂靠、再独立。** 最小团队里「产品负责人」可兼任服务台；一旦工单量超过其精力 20%，必须独立出服务台角色。不要等"完美编制"才起步，先有入口、再补编制。

> **推行话术（对老板）：** "我不是要加人，是要把现有的人按业务线归位。归位后需求有入口、交付有边界，您插单也有地方承接、有代价可看——比现在所有人救火强。"

**常见阻力与应对：**

- **"我们人少，一个人兼多角色行不行？"** → 可以兼，但角色边界必须分开、可追溯（见 1.7 Q1）。1-2 人的微型团队不在【GCR】 适用范围内，详见 1.3 适用范围说明。
- **"两条业务线共用一个研发团队行不行？"** → 不行。研发必须稳定归属一条业务线；体量都不够大就先合并成一条业务线（见 1.7 Q3）。

## 1.3 规则参考：角色、结构与组建标准

### 角色定义（7 个核心角色 + 组织级守护者）

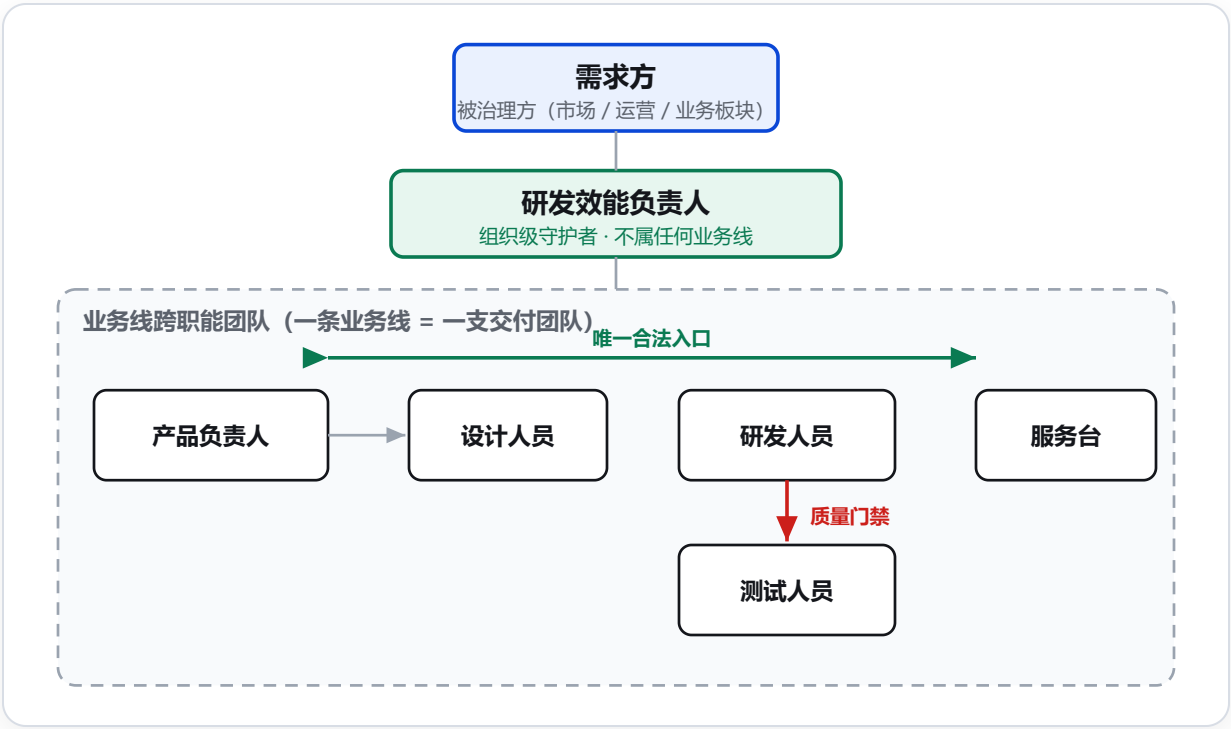



图 1.1　角色职责结构（7 个核心角色 + 组织级守护者）

| 角色 | 核心职责 | 日常做什么 | 对谁负责 |
| --- | --- | --- | --- |
| **需求方** | 提出真实业务问题、确认价值、验收结果 | 描述业务痛点、参与价值定义、验收交付物 | 业务结果 |
| **研发效能负责人** | 守住研发组织边界，对效能结果负责 | 监督团队规范、处理角色冲突/资源冲突、审批跨线借用、输出效能数据 | 组织效能 |
| **产品负责人** | 过滤需求、定义价值、决定优先级 | 每天过滤需求池、排优先级、主持评审会、与需求方沟通价值 | 研发效能负责人 + 需求方（双向） |
| **设计人员** | 把产品价值转化为界面/交互/流程方案；参与需求画像期交互概念设计 | 出交互概念方案、设计稿、原型、交互说明 | 产品负责人 |
| **研发人员** | 按设计实现功能、对代码质量负责 | 领取任务、编码、自测、提测 | 产品负责人+测试人员（质量） |
| **测试人员** | 验证功能是否满足验收标准 | 写用例、执行测试、提缺陷、出测试报告 | 产品负责人（验收标准） |
| **服务台人员** | 接收一线问题、反馈工单、初步分流 | 记录工单、分类、流转、回访 | 需求方 +产品负责人 |

**关键说明：**

- 「需求方」提出的任何诉求，**先经「研发效能负责人」确认归属哪条业务线**，再进入该线「产品负责人」或服务台的入口。「研发效能负责人」是组织级门禁，「产品负责人」/服务台是团队级入口，两层不能混为一谈。
- 「产品负责人」和「服务台人员」是需求进入研发团队的**唯一合法入口**。「需求方」不直接找开发，开发不直接接需求。
- 「研发效能负责人」不属于任何业务线团队，他是所有团队的"边界守护者"——管的是团队之间的冲突，不管单个团队内部的执行。
- 「设计人员」是产品价值和研发实现之间的翻译层，交互概念设计应前置到需求画像期（见第二章 2.2.1），避免迭代内前端等稿空转；没有「设计人员」时，

「产品负责人」必须承担"方案翻译"职责。

- ⚠️ **「工单负责人」是工单场景的专属指派角色**（非 7 核心角色之一）：由「服务台人员」派单，分配给谁即由谁负责处理工单问题，可能是客服、美工、研发等任一角色，不并入 7 角色体系。

### 团队组建标准

**最小团队配置（3 人起步）** — 适用于单条业务线、项目规模小、【迭代周期】稳定的场景。

| 角色 | 人数 | 说明 |
| --- | --- | --- |
| **产品负责人** | 1 | 可兼任服务台 |
| **研发人员** | 1 | 可前后端一体 |
| **测试人员** | 1 | 可兼任部分运维支持 |

**底线：** 少于 3 人不叫"业务线团队"，只能叫"支持小组"，必须尽快补到最小配置。

> **适用范围说明：** 本章"3 人起步"是单条业务线跨职能团队的**理论下限**（产品、开发、测试各至少一人）。【GCR】 白皮书定位适用规模为 **20-200 人**研发组织（组织级规模，非单业务线人数）——那是治理价值真正凸显的**目标市场**。1-2 人的微型团队（如老板兼任产品 + 1 名开发、不配测试）既无法建立三角制衡，也未到需要正式治理的规模，不在【GCR】 适用范围内；待团队增长到角色专业化、需求开始绕口、进度开始失焦时，再引入【GCR】。

**标准团队配置（5-7 人）**

| 角色 | 人数 | 说明 |
| --- | --- | --- |
| **产品负责人** | 1 | 不可兼任服务台 |
| **服务台人员** | 1 | 可兼任部分产品运营支持 |
| **设计人员** | 1 | 根据业务复杂度可 0.5 人共享 |
| **研发人员** | 2-3 | 可按前后端或模块拆分 |
| **测试人员** | 1-2 | 与研发人员比例不低于 1:3 |

**大型业务线配置（8 人以上）** — 在标准团队基础上按"交付小组"拆分，每个小组保持研发+测试成对出现：

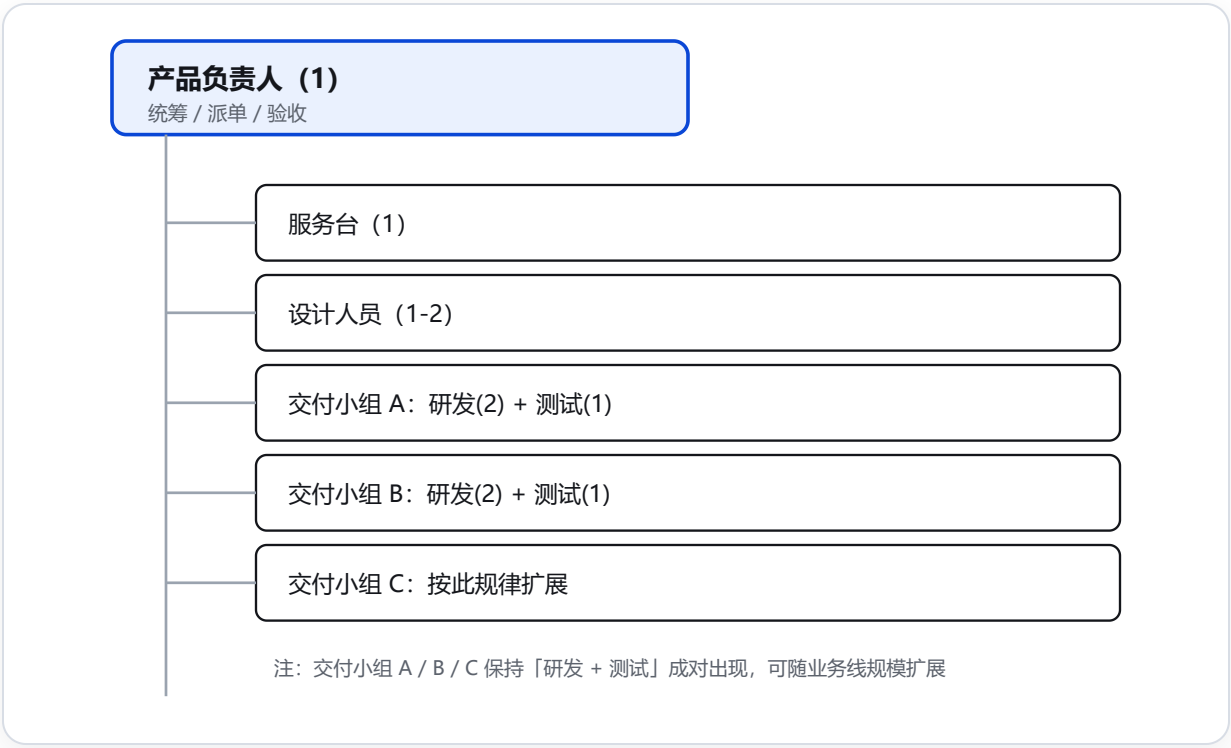



**图 1.2　大型业务线团队配置（8 人以上）**

## 红线：绝对不允许的事

### 角色禁止

1. **禁止开发人员长期兼任测试。**偶尔自测可以，但同一个人不能既写代码又签测试报告。

**→ 执行机制：「研发效能负责人」在日常巡检中核对测试报告签署人，发现自审自签即冻结该「研发人员」的新任务领取权限，直至整改。**

1. **禁止「产品负责人」同时承担多个业务线的产品决策。**一个「产品负责人」只能对一条业务线的产品价值负责。

**→ 执行机制：团队配置中「产品负责人」的归属业务线唯一，跨线需求必须经「研发效能负责人」重新指派。**

1. **禁止「研发人员」"被借调"到其他业务线冲刺。**临时支援走正式的资源借用流程，但主归属团队不变。

**→ 执行机制：人员借用须经「研发效能负责人」审批并录入借用记录，未审批的跨线任务不计入主团队迭代看板。**

1. **禁止服务台职责消失。**服务台可以兼职，但工单必须有明确的入口人和流转人。

**→ 执行机制：工单台账每周巡检空流转记录，发现工单无负责人超 48 小时即通报「产品负责人」和「研发效能负责人」。**

**组织禁止**

1. **禁止按项目拆分人员再按项目汇总。**人员稳定归属到业务线团队，项目是团队的工作内容，不是人员的组织归属。
2. **禁止"虚拟团队"长期存在。**跨职能团队必须有真实的日常协作机制，不能只在排期会上临时组队。
3. **禁止研发团队直接向「需求方」汇报。**「需求方」只通过「产品负责人」和服务台两个标准化入口与团队交互，避免【需求绕口】、口头插队。

**红线违规的统一后果：**由「研发效能负责人」在日常巡检中发现，违规团队或相关个人**冻结新需求接入权限直至整改完成。**

### 多业务线管理规则

**每条业务线独立建队：**

> - 业务线 A 团队：产品负责人 + 服务台 + 设计 + 研发 + 测试 - 业务线 B 团队：产品负责人 + 服务台 + 设计 + 研发 + 测试 - 业务线 C 团队：产品负责人 + 服务台 + 设计 + 研发 + 测试

**共享资源怎么办** — 有些角色天然稀缺（设计、安全、运维），可以设立共享服务中心。三条规则：

- 共享角色必须**按【迭代周期】分配**到业务线，不能"谁催得急就帮谁"；
- 共享角色的工作必须进入业务线团队的迭代看板，接受「产品负责人」优先级排序；
- 共享角色不直接对「需求方」负责，只对业务线团队的「产品负责人」负责。

**「研发效能负责人」定位** — 不是业务线团队的成员，是所有业务线团队的"边界守护者"：监督各业务线团队是否按规范组建；处理角色冲突和资源冲突；审批跨业务线的人员借用；定期输出团队健康度和效能数据。

### 1.4 判断口径：何时可松、踩线了往哪退

推行者真正的功力在"「判断」"，不在"背规则"。以下是对推行现场最纠结几处的口径：

| 现场情形 | 判断口径 | 往哪退 |
| --- | --- | --- |
| **真就 2 个人，老板兼产品 + 1 开发** | 不在【GCR】 适用范围，先别硬套；等角色专业化、需求绕口时再引入 | 暂不治理，口头协调即可 |
| **测试暂时没人，研发自测** | **允许临时兼**，但测试报告不能自签；尽快补测试岗 | 红线 #1 触发即冻结新任务 |
| **业务线体量不够、养不起独立团队** | 先合并成一条业务线，配一个完整团队，不要"两条线共用一群人" | 红线组织禁止 #1 |
| **共享设计师被两条线同时抢** | 按【迭代周期】排期分配，进业务线迭代看板排序；谁催得急不算数 | 共享资源规则 |
| **研发被老板点名借去救火** | 走正式借用流程、研发效能负责人审批、录入记录；主归属不变 | 红线 #3 |
| **发现有人自审自签 / 需求绕口直达开发** | 立即冻结新需求接入权限，责任人书面说明，整改后恢复 | 红线违规统一后果 |

> **推行者心法：** 红线不是用来"抓人"的，是用来"划线"的。你划的每条线，都要让团队知道"越线会怎样、怎么整改回来"。冻结是手段，不是目的；目的是让团队自己长出纪律。

## 1.5 检查清单

每条业务线团队组建时，必须完成以下动作：

- ☐ 是否已向全员宣贯各角色的职责边界？
- ☐ 是否已向「需求方」宣贯"需求只找「产品负责人」/服务台，不找开发"？
- ☐ 是否建立了角色权限机制（无论用工具还是白板/表格，见《GCR数字化落地指南》第一章）？
- ☐ 是否向全员宣读并签署了四条角色红线 + 三条组织红线？
- ☐ 共享资源（设计/安全/运维）是否已纳入【迭代周期】分配机制？

- [ ] 是否已任命「研发效能负责人」作为组织级边界守护者（哪怕暂由推行者兼任）？

## 1.6 常见问题（推行阻力 Q&A）

**Q：我们人少，能不能一个人既做产品又做开发又做测试?**

不能，也不必要。【GCR】 的三大角色（产品、开发、测试）是互相制衡的治理闸门——产品定价值、开发实现、测试独立验证，三角缺一不可。少于 3 个人（典型如"老板当产品 + 招 1 个开发、不招测试"），既无法建立这种制衡，也没到需要正式治理的规模，口头协调就够了，用不上【GCR】。

【GCR】 白皮书定位适用规模为 **20-200 人**的研发组织（此为**组织级规模**，非单条业务线人数）；本手册第一章的"3 人起步"只是单条业务线跨职能团队的**理论下限**（产品/开发/测试各至少一人），绝不是鼓励 1-2 人硬套。团队增长到角色开始专业化、需求开始绕口、进度开始失焦时，才是引入【GCR】 的时机。

**Q：服务台和「产品负责人」兼任，会不会导致「产品负责人」太忙?**

所以最小团队配置只能短期存在。一旦业务线工单量超过「产品负责人」精力的 20%，必须独立出服务台角色。

**Q：我们有两个业务线，能不能共用一个研发团队?**

不能。研发团队必须稳定归属一条业务线。如果两个业务线体量都不够大，先合并成一条业务线，再配一个完整团队。

（角色与边界在工具中的固化方式，见配套《GCR数字化落地指南》第一章。）

# 02 需求治理

> 团队搭好了，接下来要解决的问题是：需求从哪里进来？谁有资格说"这个要做"？模糊的需求怎么变清晰？需求变了、老板拍桌子了，怎么办？

## 2.1 为什么：唯一入口，过滤优先

【GCR】 全景图中，我们说的需求叫【用户需求】。它偏业务、偏价值，不是开发视角的"任务"。【用户需求】来自需求方反馈、市场调研、产品规划、【服务工单】——它的特点是：**可能横跨多个【迭代周期】才能彻底完成。**

收口的第一条铁律：**只有「产品负责人」可以接收需求并录入系统——需求一经接收即在【需求池】中，以【用户需求】的"「创建」"态存在，由状态决定其形态，不另设"放入需求池"动作。**

这不是权力垄断，是秩序基础。如果老板可以直接找开发、运营可以直接找测试、客户可以直接找前端——第一章的跨职能团队架构瞬间瓦解。

**需求进入团队的路径只有一条：**

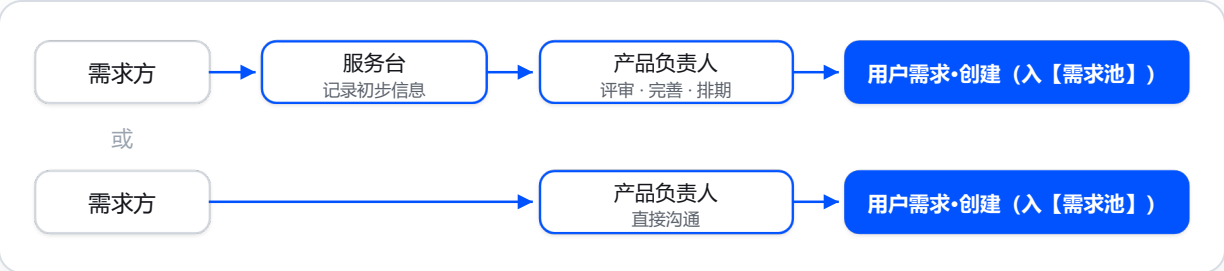
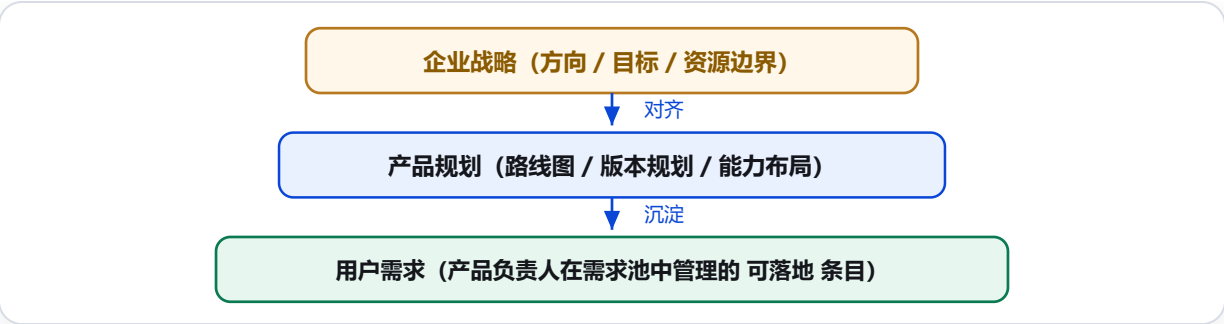



**图 2.1　需求进入团队的两条路径**

两条路径都是「产品负责人」为终点的漏斗。绕过「产品负责人」的需求，流程不认、不接、不排。

**需求的源头链路（从战略到需求）：**




- **图 2.2　需求源头链路（从战略到需求）**

**企业战略** 是顶层约束：今年公司往哪打、资源投在哪、什么不做。战略不清晰，【需求池】必然被各方塞满。

- **产品规划** 是「产品负责人」的主动编排：基于战略和市场的长期路线图。产品规划产生的需求，天然优先级高、信息完整——它们是"深思熟虑的结果"，区别于【服务工单】的"临时抱怨"。
- **【用户需求】** 是最终进入收口流程的条目。无论来自战略、规划还是一线工单，都必须经过画像完善和【状态机】。

「产品负责人」的职责之一，是保证【需求池】里的条目能向上追溯到产品规划、再向上追溯到企业战略。无法追溯的需求，要么是噪音，要么是老板拍脑袋——两者都需要在评审被拦截。

## 2.2 怎么推：让入口立住、让老板的火有地方接

**推行难点不在"流程画出来"，而在"没人听"。** 以下是推行者最常卡住的现场与打法：

**难点一：需求方习惯直接找开发，觉得走流程慢。**

→ 不要正面硬刚。让「产品负责人」主动上门："以后您的需求都发我，我半小时内给您排期结论，比您一个个找人快。"用"更快的响应"换"唯一的入口"。同时要求开发：收到需求方直接需求时，回复"收到，我让产品负责人评估一下排期"然后转交——不拒绝、不执行，把球传给「产品负责人」。

**难点二：老板随时口头插单，硬拒会得罪人。**

→ 【GCR】 的底线是**不拒绝、不堵嘴、走规则、留证据。**老板的需求必须被接收、被记录、走"「创建」→「待完善」→「待评审」→「待开始」"路径强制补全画像；填《插单代价声明》把延期影响通知所有受影响需求方；决策记录抄送老板。规则挡不住绝对权威，但能让每次插队的代价摆在桌面上，由决策者自己承担后果（详见 2.8）。

**难点三：画像字段太多，需求方嫌烦不肯填。**

→ 把"价值三维度"和"期望上线时间"做成 15 分钟内的轻量对话，由「产品负责人」主导填写，「需求方」只需拍板。不要用问卷轰炸；交互概念方案由设计前置，需求方只看低保真草图确认"做成什么样"。

**难点四：需求池越堆越满，没人敢关。**

→ 设"池子容量上限 = 迭代容量 × 3"，超了就进候补队列，倒逼需求方自己排序（详见 2.7）。关闭不是失败，是治理健康的体现——详见 2.4 判断口径。

## 2.3 规则参考：画像、评估、状态机与推导

以下为执行层照此落地的标准内容。

### 2.3.1 【用户需求】的完整画像

「产品负责人」接收一个需求时，它**已经在【需求池】里**——以"「创建」"态存在（见 2.3.6）。【需求池】是所有【用户需求】的容器，从"「创建」"到"「关闭」"全程都在池中，**形态由状态决定，不存在"丢进池"或"移出池"的动作。**所以接收后不能把它当成一个模糊条目堆着，必须补全画像、把它从"「创建」"推进到"「待完善」"，变成"可评估、可排序、可追溯"的结构化条目。

以下是每个【用户需求】必须填写的字段：

**基础信息**

| 字段 | 说明 | 填写人 |
| --- | --- | --- |
| **标题** | 一句话说清楚这个需求是什么 | 产品负责人 |
| **场景描述** | 谁在什么情况下遇到什么问题 | 产品负责人（与需求方共同完成） |
| **风险与依赖** | 有没有外部依赖？有没有技术风险？ | 产品负责人+ 研发风险评估后补充 |
| **交互概念方案** | 该需求的界面/交互概念草图（低保真），确认"做成什么样" | 设计人员（需求画像期） |
| **计划开始时间** | 待开始状态时的派生值 = 最早安排进入迭代的【交付功能】的迭代时间（日精度），用于拉甘特图；待开始前为空 | 系统自动派生（基于【交付功能】的迭代安排） |

> **交互概念方案是【用户需求】的就绪输入（对应全景图"交互设计 → 包含 → 用户需求"）：** 它前置到需求画像期完成，而非迭代内才启动——否则【交付功能】进【迭代周期】后前端会空等设计稿。迭代内只做高保真细化与切图配合（见第三章 3.5 / 3.6）。

**标题规范示例**

- ❌ "门禁息屏"
- ❌ "优化用户体验"
- ✅ "办公楼门禁屏幕息屏功能——实现闲置 30 秒自动熄灭，降低电费损耗"

**【业务价值】定义（必须与「需求方」共同确认）** — 从以下三个子维度给需求定价值，每个子维度按 10/7/3/1/0 五级打分：

| 子维度 | 说明 | 10 | 7 | 3 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- |
| **战略契合** | 与产品主线/年度目标的契合度 | 核心链路 | 重要方向 | 一般方向 | 边缘 | 无关 |
| **用户影响** | 受影响用户的范围与频次 | 全体高频 | 多数高频 | 部分 | 个别 | 无人 |
| **商业收益** | 可量化的收益（收入/成本/合规） | 显著 | 明显 | 中等 | 微弱 | 无 |

**业务价值分 = 战略契合 × 用户影响 × 商业收益**（三个子维度得分直接相乘）。

> **精益否决权：** 任一子维度为 0（无价值）→ 业务价值分直接归零，整条需求不进入重要序列——继承精益"消除浪费"，无业务价值的需求不配占用重要位。

**业务价值档（5 档对照矩阵）：** 业务价值分是连续乘积（0–1000），为便于沟通、看板展示与优先级矩阵联动，将其归约为 5 档（5 = 极高，1 = 极低）。任一子维度为 0 时业务价值分 = 0，落入「档 1（极低价值）」，且因精益否决权直接不进入重要序列。

| 业务价值分区间 | 业务价值档 | 标签 | 排期含义 |
| --- | --- | --- | --- |
| **700 – 1000** | 5 | 极高价值 | 战略级，优先保障资源、进核心迭代 |
| **300 – 699** | 4 | 高价值 | 正常高优先排期 |
| **100 – 299** | 3 | 中价值 | 常规排期 |
| **30 – 99** | 2 | 低价值 | 可插空、低优先 |
| **0 – 29** | 1 | 极低价值 | 暂缓、回退至待完善沉淀（仍在【需求池】中）；其中 0 = 精益否决权（任一子维度为 0，重要程度分 = 0，不进入重要序列） |

> 区间阈值（700 / 300 / 100 / 30）为经验基线，可按团队实际分布微调；档位是**展示与沟通层**，【重要程度】公式仍以「业务价值分」连续值参与计算（见 2.3.9），档位本身不直接进公式。

**规则：** 【业务价值】的分值由**「需求方」声明**，但**「产品负责人」有权质疑和下调**。双方无法达成一致时，提交「研发效能负责人」仲裁。

**优先级（四象限）**

| 象限 | 含义 | 处理方式 |
| --- | --- | --- |
| **紧急且重要** | 必须马上做，不做就出事 | 当期迭代最高优先级，可能触发插单评估（见 2.8） |
| **紧急不重要** | 时间紧但价值低 | 评估是否真的紧急，很多"紧急"是假的 |
| **重要不紧急** | 价值高但不赶时间 | 进入正常迭代排期，按 GRT 节奏推进 |
| **不紧急不重要** | 可做可不做 | 先放着，定期清理时考虑砍掉 |

**紧急轴的自动推导（重要）：** "紧急"不是一个靠「产品负责人」主观拍脑袋的标签，而是由【期望上线时间】自动推导的基线——

> 【期望上线时间】 − 今天 ≤ 28 天 → 紧急(2)；> 28 天 → 不紧急(1)；已过期 → 超期(3)。

「产品负责人」在受理阶段填好【期望上线时间】后，系统按 2.3.9 规则自动推导紧急程度（1/2/3 序数）。**紧急程度是【期望上线时间】距今天天数的函数，用 1/2/3 序数，绝不用 0/1**（否则会抹掉所有非紧急需求之间的差异，废掉重要程度轴）。它的客观依据是时间窗口，不取决于"老板说的"。**输出层不锁死**：系统给默认紧急程度，「产品负责人」可覆盖，但覆盖必须填写原因并留痕——这些覆盖记录正是未来校准阈值、决定标准化力度的数据燃料。

**需求类型**

| 类型 | 说明 | 典型例子 | 类型系数 |
| --- | --- | --- | --- |
| **核心/刚需** | 产品主线的必要功能 | 支付、登录、核心业务流程 | **1.2** |
| **体验/优化** | 已有功能的改进 | 界面调整、性能优化、交互改进 | 1.0 |
| **技术/探索** | 技术债偿还或新技术验证 | 架构重构、安全加固、人工智能能力引入 | 0.9 |
| **数据/分析** | 数据采集、报表、分析能力 | 运营看板、用户行为分析、审计日志 | 1.0 |

> 类型系数用于 2.3.9 计算【重要程度分】（业务价值分 × 类型系数 × 来源系数）。

**需求来源**

| 来源 | 说明 | 来源系数 |
| --- | --- | --- |
| **产品规划** | 产品负责人主动规划的长期路线图中的需求（优先级天然较高、信息完整） | **1.1** |
| **运营反馈** | 运营团队从用户使用中收集的问题 | 1.0 |
| **内部提出** | 研发/测试/设计团队提出的技术改进或效率需求 | 0.9 |
| **服务工单** | 通过服务台收集的一线用户问题（转化而来，见 2.3.8） | 0.8 |

> 来源系数用于 2.3.9 计算【重要程度分】。**为什么记录来源？** 因为不同来源的需求有不同的可信度和紧迫度。产品规划是深思熟虑的结果，【服务工单】可能是单个用户的临时抱怨。来源帮助「产品负责人」在做优先级判断时有据可依，也是 2.3.9 自动推导中"来源权重"的输入。

**【期望上线时间】（业务视角的价值兑现点）** — 这是【用户需求】层的时间字段之一（另一个是排期后回填的【计划开始时间】），也是整个优先级推导与需求卫生的锚点。

| 字段 | 说明 | 精度 | 填写人 | 填写时间 |
| --- | --- | --- | --- | --- |
| **期望上线时间** | 需求方希望这个功能**真正上线能用**的时间，不是研发"做完了"的时间 | **日**（精确到日，仅作排期锚点标识） | 产品负责人（与业务板块负责人在待评审共同确认） | 待评审 → 待开始（受理阶段） |

**为什么叫"上线"而不是"截止"：** 对需求方来说，研发"做完了"没有意义，只有"上线了、能用上"才有价值。**不上线的需求对需求方是垃圾。** 所以【GCR】站在业务侧，用"【期望上线时间】"而非"期望截止时间"——需求的终点对齐价值兑现点，不是研发的内部里程碑。

**为什么精确到日：** 【期望上线时间】是需求方对"价值兑现点"的承诺，用**精确到日**的日期表达，作为排期锚点标识——它让 2.3.9 的紧急程度（期望上线时间 − 今天）算得精确，也作为甘特图的目标线（【计划开始时间】 + 工期 → 期望上线时间）。

它和第三章整周【迭代周期】不冲突：期望上线时间是需求层的"目标日"（契约锚点），迭代内任务才用日级排期（执行粒度）。

**这个字段驱动两件事：**

1. **紧急轴自动推导**：`【期望上线时间】 − 今天 ≤ 28 天` → 紧急(2)；否则不紧急(1)；已过期 → 超期(3)。系统给默认，「产品负责人」可覆盖但须填原因留痕
2. **过期检测**：`今天 > 【期望上线时间】 且 状态 ≠ 「已上线」` → 标红，倒逼关闭或重估

### 2.3.2 【风险评估】（四个维度）

需求经过「产品负责人」的业务侧完善后，必须与技术团队进行一次**大致评估**。注意是"大致"——不需要精确到人天，目的是让「产品负责人」知道这个需求的技术体量大概在哪一档，以便合理排期。

评估由「产品负责人」发起，「研发人员」（或技术负责人）配合，从以下四个维度各选一档（每个维度按"顶格→最轻"映射 10/7/5/3/1 五档得分：第 1 档=10，第 2 档=7，第 3 档=5，第 4 档=3，第 5 档=1）：

**维度一：界面影响**

| 档位 | 定义 | 分值 |
| --- | --- | --- |
| **界面彻底重构（含多端）** | 整套界面重新设计，涉及网页端/移动端/小程序等多端同步改版 | 10 |
| **多页面多端更新** | 涉及多个页面、可能跨端的界面变动 | 7 |
| **单页面单端更新** | 只改一个页面、一个端口的界面 | 5 |
| **仅文案/样式优化调整** | 改字、改颜色、改间距等局部调整 | 3 |
| **无界面改动** | 纯后端逻辑，前端无需动 | 1 |

**维度二：数据库影响程度**

| 档位 | 定义 | 分值 |
| --- | --- | --- |
| **核心表重构（如删表、数据迁移）** | 涉及数据库结构性变更，需要数据迁移脚本、停机窗口 | 10 |
| **核心表变更（如字段变更、数据清洗等）** | 主表增删字段、历史数据清洗、索引重建 | 7 |
| **新增表/非核心表变更** | 新建表、修改配置表或日志表 | 5 |
| **仅新增字段** | 在现有表上加几个字段，不影响已有数据结构 | 3 |
| **无更新** | 不涉及数据库操作 | 1 |

### 维度三：功能依赖程度

| 档位 | 定义 | 分值 |
| --- | --- | --- |
| **依赖 3 个及以上核心功能** | 这个需求改动了 底层基础 模块，波及面广 | 10 |
| **依赖 1-2 个核心功能** | 影响主要功能模块，但范围可控 | 7 |
| **依赖 3 个及以上非核心功能** | 影响辅助功能较多，但不在主干上 | 5 |
| **依赖 1-2 个非核心功能** | 影响范围小，基本独立 | 3 |
| **无依赖，完全独立** | 一个独立的模块或功能点 | 1 |

### 维度四：第三方集成复杂度

| 档位 | 定义 | 分值 |
| --- | --- | --- |
| **3 个以上接口深度集成** | 涉及加密、算法、回调、联调、对账等多个环节的外部系统对接 | 10 |
| **1-2 个接口深度集成** | 少量但复杂的第三方对接，涉及安全认证和数据一致性 | 7 |
| **3 个以上接口简单集成** | 多个外部系统的数据获取/推送，但交互简单（仅读/写 接口） | 5 |
| **1-2 个接口简单集成** | 少量外部数据对接，无复杂交互逻辑 | 3 |
| **无接口集成** | 纯内部系统，不涉及任何第三方 | 1 |

### 2.3.3 评估结果的综合判定

四个维度的评估完成后，「产品负责人」得到的是一个**技术体量画像**，不是一个精确工时。这个画像的目的很单一：**为 2.3.3 风险等级矩阵的两条轴打分。**

> **技术评估 → 风险等级的唯一链路：** 四个维度得分先各归为"高/中/低"档（单维映射见 2.3.3），再**取每轴较高档**聚成两条轴（影响面、耦合复杂度），最后查矩阵得风险等级（高/中/低）。**风险等级不进入【重要程度】公式**——重要程度只回答"值不值得做"。

**风险等级矩阵（高 / 中 / 低）** — 采用行业标准**二维风险矩阵（影响面 × 耦合复杂度）**（ISO 31000 / PMI 风险矩阵范式）：

**第一步：把四个维度归为两条综合轴**（取该轴两维度中较高的一档——风险视角下"任一偏高则整体偏高"）：

- **影响面（Impact）** = 界面影响 与 数据库影响 的较高档
- **耦合复杂度（Complexity）** = 功能依赖 与 第三方集成 的较高档

> 单维度档位映射：得分 10 / 7 → 高，5 → 中，3 / 1 → 低。

**第二步：两条轴交叉，查风险等级矩阵：**

| 影响面 \ 耦合复杂度 | 低 | 中 | 高 |
| --- | --- | --- | --- |
| **低** | 低 | 低 | 中 |
| **中** | 低 | 中 | 高 |
| **高** | 中 | 高 | 高 |

> 逻辑：风险 = 影响面 × 耦合复杂度。两轴都高 → 高风险；一高一低 → 中风险；都低 → 低风险。

**风险分值（1–9，与矩阵 1:1 对应）：** 将两条轴的档序（低 = 1 / 中 = 2 / 高 = 3）相乘：

| 影响面 \ 耦合复杂度 | 低(1) | 中(2) | 高(3) |
| --- | --- | --- | --- |
| **低(1)** | 1 | 2 | 3 |
| **中(2)** | 2 | 4 | 6 |
| **高(3)** | 3 | 6 | 9 |

分值档位：**低(1–2) / 中(3–4) / 高(6–9)**（分值 5 为空档，理论上不存在）。

**第三步：按等级决定排期动作：**

| 风险等级 | 排期处置 |
| --- | --- |
| 🔴 **高** | 拆分到多个迭代；指派资深研发；甘特图标红风险条；需技术负责人评审后方可排期 |
| 🟡 **中** | 正常排期；迭代内常规评审 |
| 🟢 **低** | 可快速插入；轻量评审即可 |

**示例：门禁息屏功能**

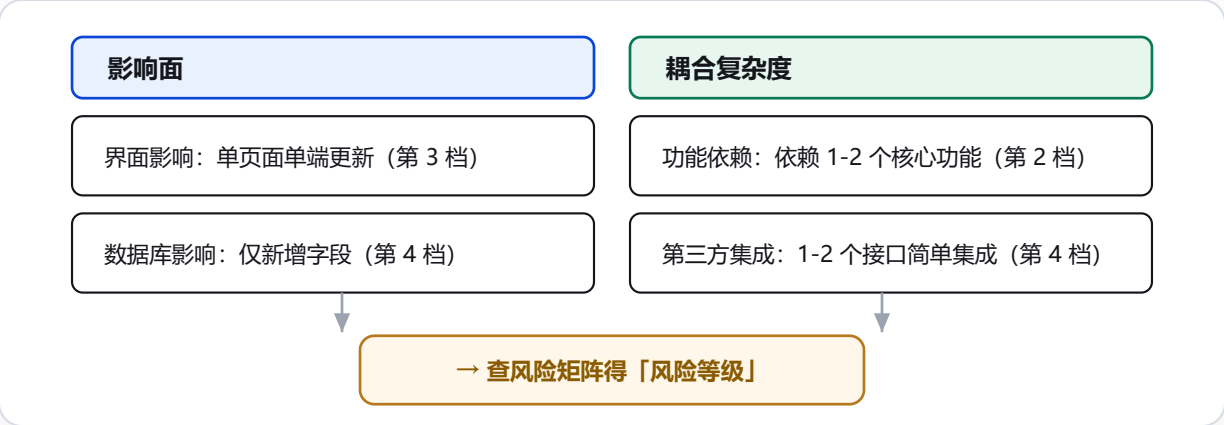



**图 2.3　风险评估四维示例（门禁息屏）**

归轴：影响面 = max(中, 低) = 中；耦合复杂度 = max(高, 低) = 高

查矩阵：影响面中 × 耦合复杂度高 → 风险等级 = 高

风险分值：影响面序(2) × 耦合序(3) = 6 → 属"高"档

综合判断：界面改动小，但依赖核心模块（耦合高），按高风险处理（拆分迭代 + 资深研发 + 甘特红条 + 技术负责人评审）

**这个画像的作用：** 帮助「产品负责人」判断这个需求应该放在哪个迭代（大需求拆分到多个迭代）；帮助「产品负责人」在与需求方对话时有底气（"这个涉及核心表重构，至少需要一个完整迭代"）；为后续的插单评估（2.8）和【交付功能】拆解（第三章）提供基线数据。

### 2.3.4 【需求漏斗】与过滤机制

收口不是一次动作，而是一道持续过滤的漏斗。【GCR】 用"【用户需求】漏斗"表达这个机制——**量在逐层减少，质量在逐层提高**：

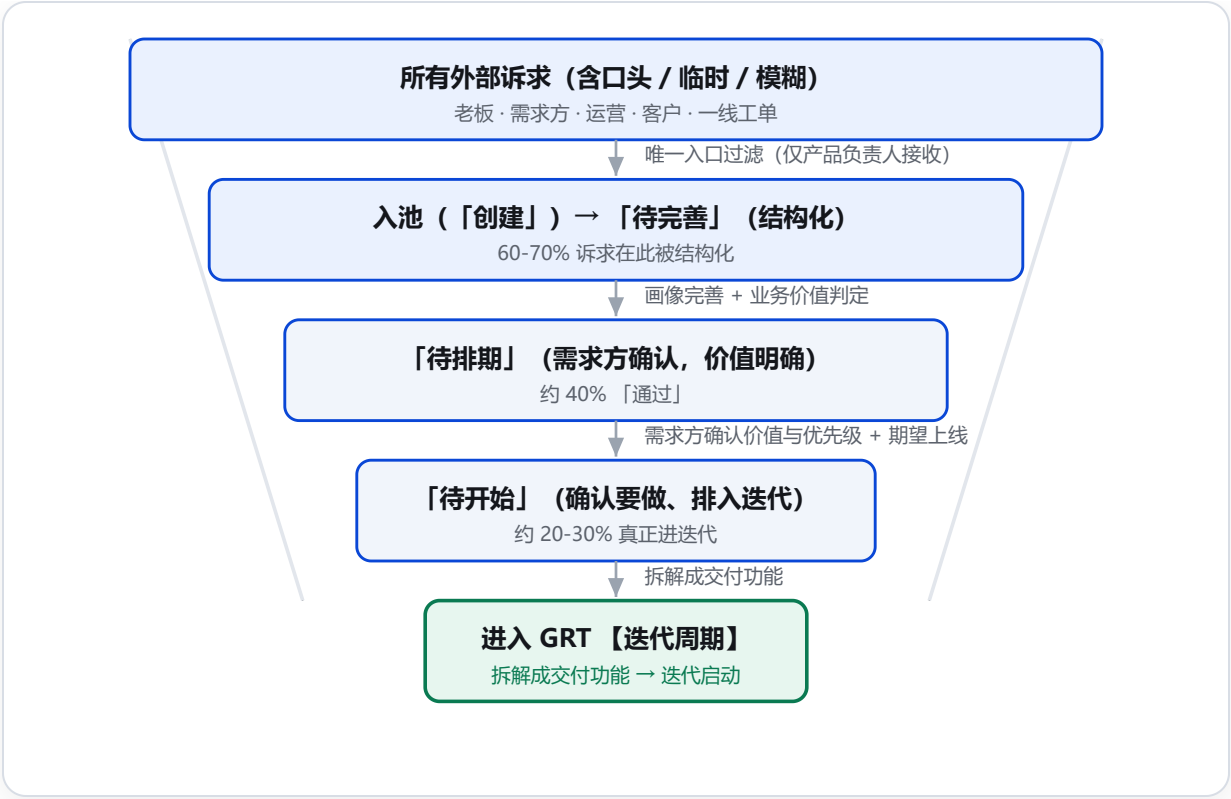



**图 2.4　【需求漏斗】——量逐层减少、质量逐层提高**

**漏斗的健康度本身就是指标：** 如果"「待评审」→「待开始」"通过率长期低于 30%，说明「产品负责人」在画像阶段没拦住噪音；如果"【外部诉求】→「待完善」"通过率过高（接近 100%），说明入口过滤形同虚设（可能有人绕过「产品负责人」直接灌需求）。这两个比率纳入第五章【效能度量】。

### 2.3.5 【用户需求】生命周期【状态机】

**图例说明**

| 符号 | 含义 |
| --- | --- |
| **黄色菱形 ◆** | 决策 / 审批节点（如是否确认上线时间变更审批） |
| 🔴 **红圆** | **重度审批关卡**（三方及以上会签，如插单审批关闭审批） |
| **灰圆角矩形** | 关闭终态（暂停 / 放弃，可重新打开） |
| 🟢 **绿圆角矩形** | 正常完成终态（已上线） |
| **红虚线** | 跨模型自动化触发（数字化后由工作流实现；手工模式下对应角色按条件手动建单 / 推进） |

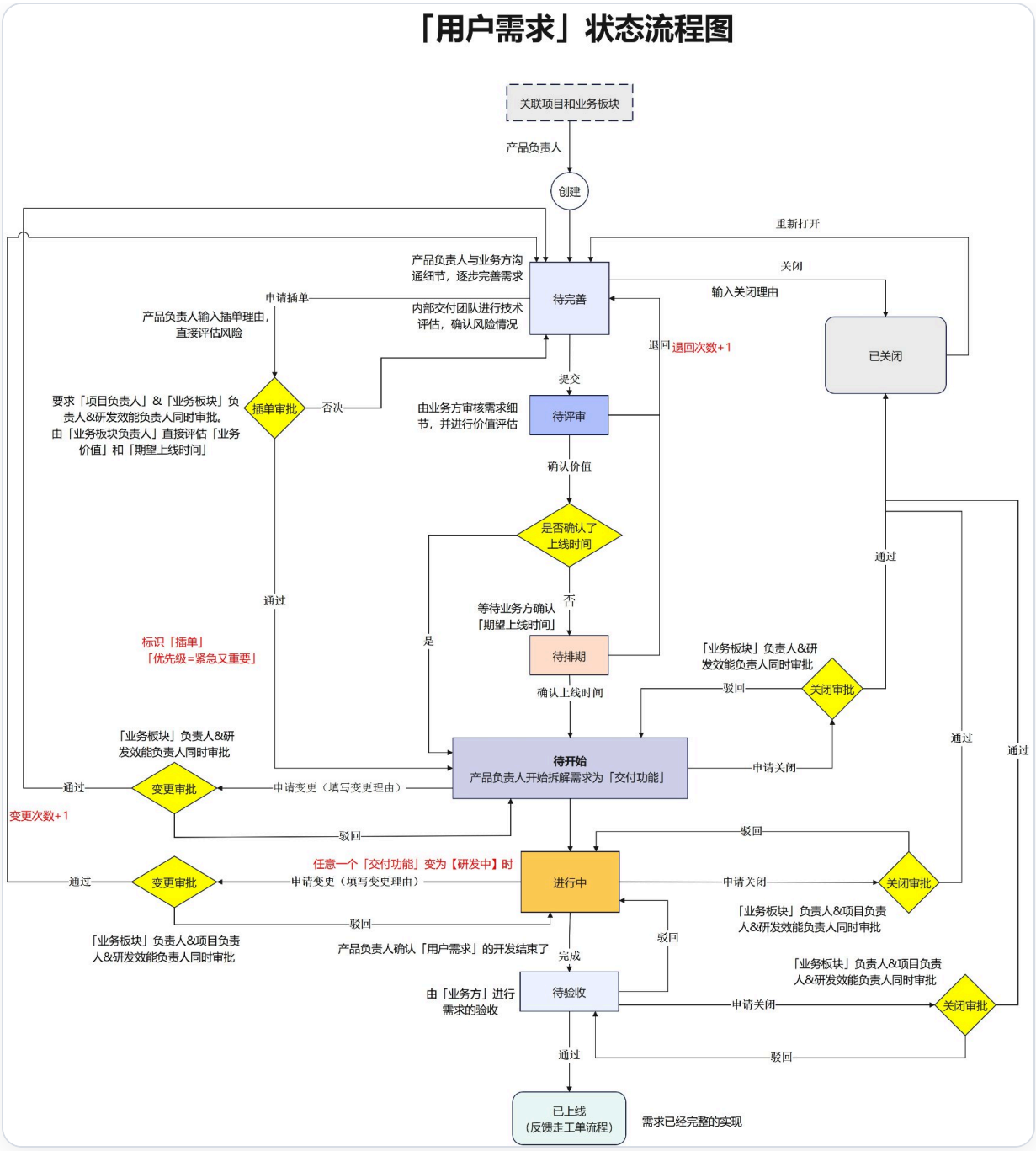
### 【用户需求】状态机矩阵

> 本状态机与「5 大模型的数字化联动」总图（最高权威源）的用户需求列逐列一致。下表中「业务板块」负责人 = 该业务线的「需求方」决策代表。所有「插单审批 / 变更审批 / 关闭审批」关卡的定义与分层审批人见 2.3.6。

| 状态 | 参与对象 | 做什么 | 流转至 |
| --- | --- | --- | --- |
| **创建** | 需求方（提出）；产品负责人（受理录入） | 关联项目与业务板块；提出原始诉求，录入【需求池】，填基础字段（标题、描述、【需求来源】、价值初判） | 待完善 |
| **待完善** | 产品负责人（主责）；设计人员（前置交付交互概念方案） | 与需求方沟通细节、逐步完善需求细节；内部交付团队进行技术评估、确认风险情况；补全画像字段 | ① 提交 → 待评审；② 申请插单 → ◆插单审批；③ 输入关闭理由 → 关闭 |
| **待评审** | 产品负责人 + 技术团队（内部先评）；业务板块负责人（需求方后评） | 由需求方审核需求细节，并进行价值评估 | ① 退回 → 待完善；② 确认价值 → ◆是否确认上线时间? |
| **待排期** | 产品负责人 + 业务板块负责人 | 等待需求方确认【期望上线时间】 | ① 确认上线时间 → 待开始；② 超时未定 → 回退待完善（沉淀，仍在【需求池】中）或预警升级 |
| **待开始** | 产品负责人 + 研发效能负责人（风险复核定稿） | 【期望上线时间】已定；研发效能负责人复核【风险评估】**定稿**（覆盖暂定须留痕）；拆解【交付功能】并安排进入迭代；**最早进迭代的【交付功能】迭代时间 = 【计划开始时间】** | ① 任一【交付功能】→研发中（联动自动触发）→ **进行中**；② 申请关闭 → 🔴关闭审批（待开始级）；③ 申请变更（填变更理由）→ ◆变更审批（待开始级） |
| **进行中** | 研发人员 + 测试人员 + 设计人员（执行）；产品负责人（跟踪） | 【交付功能】开发 / 测试 / 设计协作；产品负责人确认【用户需求的开发有结果了】；按风险等级排期；跟踪质量门禁（3.11） | ① 完成 → 待验收；② 申请关闭 → 🔴关闭审批（进行中级）；③ 申请变更（填变更理由）→ ◆变更审批（进行中级） |

| 状态 | 参与对象 | 做什么 | 流转至 |
| --- | --- | --- | --- |
| **待验收** | 产品负责人（提交）；业务板块负责人（验收） | 功能可用且质量达标（3.11 门禁）；业务板块负责人按【验收评分矩阵】打分验收 | ① 通过 → 已上线；② 驳回 → 回退进行中；③ 申请关闭 → 🔴关闭审批（待验收级） |
| **已上线** | 业务板块负责人 | 按【验收评分矩阵】确认闭环；价值回环；后续反馈可能触发新工单 | 终态（正常完成） |
| **关闭** | 产品负责人（申请）；对应触发级的审批人（审批） | 关闭审批通过后的归档态；需求不再推进 | 终态（暂停 / 放弃）；可重新打开 → 通过 → 回**待完善** |

下图为「用户需求」完整状态流转图（含决策/审批节点、分支路径与计数器），是本矩阵的可视化权威源。图中： - 黄色菱形 ◆ = 决策/审批节点（与上方 2.3.6 表一一对应） - 🔴 红圆 = 三方及以上会签的重度审批关卡 - 「退回次数」「变更次数」为计数器，异常信号见 2.3.6 后续说明 - 关闭可重新打开 → 回**「待完善」**（需重新走完整流程）




「用户需求」状态流程图

## 决策与审批节点（含分层审批人）

> 治理原则：**沉没成本越大，审批门槛越重**——越靠后的状态，「关闭」 / 变更的审批人越多。

| 节点 | 符号 | 触发位置 | 审批人（分层） | 判定内容 | 出口 |
| --- | --- | --- | --- | --- | --- |
| **是否确认【期望上线时间】？** | ◆黄菱（决策） | 待评审 → 待排期 / 待开始 | 业务板块负责人 | 需求方在待评审时是否已给出【期望上线时间】 | 否 → 待排期；是 → 待开始 |
| **插单审批** | 🔴红圆（重度审批） | 待完善 → 待开始 | 项目责任人&业务板块负责人&研发效能负责人（**三方同时审批**） | 由业务板块负责人直接评估【业务价值】与【期望上线时间】，确认是否插单 | 否决 → 回退待完善；全体通过 → 待开始（跳过待评审 / 待排期） |
| **变更审批（待开始级）** | ◆黄菱（审批） | 待开始 → 进行中 | 业务板块负责人&研发效能负责人（**2 人**） | 未开工前的范围 / 时间变更是否合理 | 驳回 → 回退待开始；通过 → 进行中 |
| **变更审批（进行中级）** | ◆黄菱（审批） | 进行中 → 继续 / 回退 | 业务板块负责人&项目负责人&研发效能负责人（**3 人**） | 开发过程中的范围 / 时间变更，门槛更重 | 驳回 → 回退进行中；通过 → 继续（变更次数 +1） |

| 节点 | 符号 | 触发位置 | 审批人（分层） | 判定内容 | 出口 |
| --- | --- | --- | --- | --- | --- |
| **关闭审批（待开始级）** | 🔴红圆（重度审批） | 待开始→关闭 | 业务板块负责人&研发效能负责人（**2 人**） | 是否确实应关闭（不可行 / 不再做 / 重复等） | 驳回 → 回退待开始；通过 → 关闭 |
| **关闭审批（进行中级）** | 🔴红圆（重度审批） | 进行中→关闭 | 业务板块负责人&项目负责人&研发效能负责人（**3 人**） | 同上，沉没成本更大、门槛更重 | 驳回 → 回退进行中；通过 → 关闭 |
| **关闭审批（待验收级）** | 🔴红圆（重度审批） | 待验收→关闭 | 业务板块负责人&项目负责人&研发效能负责人（**3 人**） | 同上 | 驳回 → 回退待验收；通过 → 关闭 |

### 两个计数器

| 计数器 | 触发点 | 含义 | 异常信号 |
| --- | --- | --- | --- |
| **退回次数** | 待评审 → 待完善（不确认 / 需重定） | 每次需求方不通过 +1 | 同一需求被退 3 次仍未过审 →产品负责人或业务板块负责人沟通机制有问题 |
| **变更次数** | 进行中 → 变更审批通过 | 每次范围 / 时间变更通过 +1 | 变更次数 > 迭代内需求数均值 2 倍 → 需求在收口阶段没想清楚 |

**关键约定**

1. **【计划开始时间】是派生值**：不由手动填写，而是"最早安排进入迭代的【交付功能】的迭代时间"。在「待开始」状态时自动确定。
2. **「进行中」由【交付功能】联动触发**：任一关联的【交付功能】进入"「研发中」"时，【用户需求】自动变更为"「进行中」"；全部【交付功能】完成后由「产品负责人」手动提交至"「待验收」"。
3. **「待评审」双段式**：内部技术评审先做（暂定风险评估），需求方评审后做（定价值 / 定时间）。两段可同一天完成，也可分天。
4. **关闭可重新打开**：重新打开后回到**「待完善」**（需重新走完整流程），防止"关了就丢"。
5. **【业务价值】验证发生在"「已上线」"那一刻**：不是"已提测"或"已开发完"。超过【期望上线时间】仍未进入"「已上线」"的需求，按 2.3.7 【过期铁律】处理。

### 2.3.6 【需求池】管理规则

**谁拥有【需求池】？「产品负责人」。**

【需求池】是「产品负责人」的核心资产。「产品负责人」对【需求池】有以下权利和义务：

**需求池是什么：** 【需求池】是容纳**所有**【用户需求】的**容器**——一条需求从"「创建」"到"「关闭」"全程都在池中，**需求池不对应任何单一状态**；一条需求"长什么样"由它当前的**状态**决定（见 2.3.5）。因此本文凡说"进池 / 出池 / 丢进池"，都指"被接收为【用户需求】（创建态）/ 被关闭归档"，而非在池内外移动。

**权利**

- 决定接收哪些外部诉求为【用户需求】（以"「创建」"态入池）
- 决定池内需求的优先级顺序
- 决定哪些需求关闭 / 归档（状态终态，仍留痕于池）

**义务**

- 定期（建议每两周）审查【需求池】，清理过时的需求

- 保持每个池中需求的信息完整性（字段不缺）
- 向需求方透明化【需求池】的状态（让「需求方」知道自己的需求排在哪儿）
- 主动关闭超过【期望上线时间】仍未上线的需求（见下方铁律）

**【需求池】的上限**

- 建议每条业务线的活跃【需求池】规模不超过**迭代容量的 3 倍**。也就是说，如果一个团队每两周能交付 5 个需求，池子里不应该超过 15 个活跃需求。
- 超出上限意味着「产品负责人」的过滤机制失效了——要么加强过滤，要么申请扩编。

**过期未上线铁律：** 需求以"上线"为价值兑现终点（见 2.3.5）。超过"【期望上线时间】"仍未进入"「已上线」"状态的需求，视为失效需求：

- 由「研发效能负责人」巡检标红、或由数字化工具自动标红并通知「产品负责人」与「需求方」（见《GCR数字化落地指南》）
- 「产品负责人」必须在下一个评审周期内做出决定：**「关闭」**（取消/方向变更）或**重估**（调整【期望上线时间】并书面说明原因）
- **不允许"做了 80% 但永远不上线"的需求在池中腐烂**——那是业务视角的垃圾，也是团队的隐形债务。

### 2.3.7 变更与插单治理

需求进入"「待开始」"之后，老板拍桌子、需求方催命，是最真实的战场。【GCR】不假装能消除这类压力，而是**让压力可见、让代价透明、让责任归位。**

**核心场景（实战案例）：**

> 老板："去年我就提了门禁息屏，到现在还没做！你知不知道我每个月浪费多少钱！赶紧做了！"——该功能评估 1 个月工时，涉及全城硬件更新和物业管家协调，当前迭代已满。

**【GCR】 的判定逻辑（三步）：**

**第一步：变更还是插单**

| 情形 | 定义 | 走哪条流程 |
| --- | --- | --- |
| **变更** | 已排期的需求本身改了范围/时间 | 待开始 → 变更审批（2.3.5 状态机） |
| **插单** | 一个全新的、未排期的需求要挤进当前迭代 | 走 2.3.5 的"创建→待完善→待评审→待开始"完整路径，或触发应急插单评估 |

门禁息屏属于**插单**：它不在当前迭代排期里，是老板口头提出的全新诉求。

**第二步：是否满足紧急级** — 【GCR】 对"紧急"有客观标准（不是老板说急就急）：

> **紧急级 = 同时满足以下三项：** ① 线上已崩溃 / 用户已无法使用 ② 无临时替代方案 ③ 不处理会造成可量化的重大损失

门禁息屏已存在一年——说明业务可容忍，**不符合紧急级。**因此**禁止中途插单**，应纳入下一迭代缓冲窗口集中处理。

**第三步：老板强制要求怎么办** — 【GCR】 的底线是：**不拒绝、不堵嘴、走规则、留证据。**

| 动作 | 说明 |
| --- | --- |
| **不拒绝需求** | 老板的需求必须被接收、被记录 |
| **不绕过流程** | 走"创建→待完善→待评审→待开始"路径，强制补全画像字段 |
| **填《插单代价声明》** | 按 2.3.9 规则计算：强行插入 1 个月工时 → 当前迭代交付率归零、3 个需求方功能延期、需求波动率从 15% 飙到 40% |
| **自动通知受影响方** | 将延期影响通知所有受影响需求方，让代价透明 |
| **共同决策** | 拉上需求方一起协商延期方案（门禁息屏当时延期 3-5 天，由各需求方自行向客户解释） |
| **留痕** | 决策记录抄送老板，扣钱时有据可查 |

**关键结论：** 【GCR】 不是用来让老板闭嘴的，是用来**让每次插队的代价摆在桌面上，由决策者（老板）自己承担后果。**规则挡不住绝对权威，但规则能让你在离开时

手里有证据——证明你不是搞不定，是有人拒绝面对物理规律。

### 2.3.8 【跨模型联动】：需求 ↔ 功能 ↔ 工单

【用户需求】不是孤立的。它是五大模型联动网络的起点。以下是需求与其他模型的自动化/人工关联（红色虚线=自动化触发）：

| 触发源 | 动作 | 目标 | 类型 |
| --- | --- | --- | --- |
| **需求•待排期** | 拆解（人工） | → 交付功能·创建 | 人工（产品负责人主导） |
| **服务工单•待接单** | 转需求（判断为产品诉求） | → 用户需求·创建 | 自动/半自动 |
| **服务工单•判断** | 转缺陷（判断为缺陷） | → 故障缺陷·创建 | 自动/半自动 |
| **需求•某节点** | 触发（一线问题反馈） | → 服务工单·创建 | 自动化 |

**拆解规则（需求→功能）**

- 一个【用户需求】在"「待排期」"阶段，由「产品负责人」拆解为一个或多个**【交付功能】**
- 【交付功能】才是装入【GRT】【迭代周期】的执行单元（第三章）
- 【用户需求】跨多个迭代时，每个迭代装一部分【交付功能】——这就是"【用户需求】横跨多个迭代"在排期上的落地方式
- 【用户需求】的"已完成"= 其下所有【交付功能】均已"「已发布」"且需求方验收达标
- 【交互概念方案】已在需求画像期随【用户需求】完成，【交付功能】拆解时直接带上，迭代内只做高保真细化与切图配合——设计不成为前端的前置阻塞（见第三章 3.4 / 3.6）

## 2.3.9 优先级的自动推导与校验

第二章所有结构化选型都是**枚举型字段**，按治理角色分三类：【业务价值】/类型/来源构成【重要程度】维度（2.3.9.1），【期望上线时间】构成【紧急程度】维度（2.3.9.2），【风险评估】属于排期/风险维度、**不参与重要程度**——这正是【GCR】可被规则引擎执行、也可由「产品负责人」手算的基础。本节定义这些字段如何组合成"建议优先级"与"矛盾检测"。

**原则：规则给基线，人可修正，修正即数据。** 自动推导的【优先级】由系统按 2.3.9.1–2.3.9.3 合成，给出默认基线；「产品负责人」可覆盖紧急程度与重要程度分，但覆盖必须填写原因并留痕。这与【GCR】"教练式而非命令式"的哲学一致：**规则提供可解释的默认决策，人保留最终判断权，每次偏离都被记录为校准标准的燃料。**

**优先级四轴模型（对应 2.3.9.1–2.3.9.3）**

| # | 维度 | 计算公式 | 说明 |
|---|---|---|---|
| ① | 业务价值分 | 战略契合 × 用户影响 × 商业收益 | 任一子维度 = 0 即整体归零；经 2.3.1 对照矩阵归约为业务价值档1–5，用于看板与优先级矩阵联动 |
| ② | 重要程度分 | 业务价值分 × 类型系数 × 来源系数 | 重要程度 = 重要程度分 ≥ 阈值_T（默认 **120**） |
| ③ | 风险等级 | 影响面 × 耦合复杂度 → 查矩阵唯一判定（见 2.3.3） | 风险分值(1–9) = 影响面档序 × 耦合档序；排期/风险轴，**不进重要程度公式** |
| ④ | 紧急程度 | 期望上线时间 − 今天 ≤ 28 天 | → 紧急(2) / 不紧急(1) / 超期(3) |
| → | **优先级** | **重要程度 × 紧急程度 → 四象限** | 同象限按分降序；PO 可覆盖须留痕 |

### 2.3.9.1 重要程度分（自动推导"重要程度"轴）

**核心公式**

> **重要程度分 = 业务价值分 × 类型系数 × 来源系数**

其中 **业务价值分** = 战略契合 × 用户影响 × 商业收益（各 10/7/3/1/0；任一为 0 → 整体归零）

**类型系数表**

| 需求类型 | 类型系数 |
| --- | --- |
| **核心刚需** | 1.2 |
| **体验优化** | 1.0 |
| **技术探索** | 0.9 |
| **数据分析** | 1.0 |

**来源系数表**

| 需求来源 | 来源系数 |
| --- | --- |
| **产品规划** | 1.1 |
| **运营反馈** | 1.0 |
| **内部提出** | 0.9 |
| **服务工单** | 0.8 |

> **判定规则：** 重要程度分 ≥ 阈值_T（默认 **120**，经验基线、可按团队校准；阈值仅冻结认知，非客观真理）→ 重要

**重要程度分 → 是否重要 对照表（阈值_T = 120，可在工具中配置）：**

| 重要程度分 | 是否重要（重要程度轴） | 处置 |
| --- | --- | --- |
| ≥ 120 | 重要（true） | 进入重要序列，参与四象限排序 |
| < 120 | 不重要（false） | 不进重要序列，回退至待完善沉淀（仍在【需求池】中） |

> **【风险评估】不进入本公式**：它衡量技术难度、工作量与风险，其乘积【风险程度分】属排期 / 风险维度（见 2.3.2 / 2.3.3），不进重要程度。重要程度只回答"值不值得做"，"难不难做"由排期阶段另行评估。当【业务价值分】= 0（任一子维度为 0）时重要程度分直接归零，继承精益"消除浪费"：无业务价值的需求不配进入重要序列。

#### 2.3.9.2 紧急程度推导（单层序数）

紧急程度由【期望上线时间】距今天的天数自动推导，用序数（不紧急=1 / 紧急=2 / 超期=3），绝不用 0/1——否则会抹掉所有非紧急需求之间的差异，废掉重要程度轴。

- 剩余天数 = 期望上线时间 − 今天
- ≤ 28 天 → 紧急(2)
- > 28 天 → 不紧急(1)
- 已过期望上线时间（剩余天数 < 0）→ 超期(3)
- **这是经验阈值，非客观真理**：`28 天` 是当前默认紧急阈值，可按团队交付节奏微调，待跨团队覆盖数据积累后校准。
- **输出层不锁死**：系统给出默认紧急程度，「产品负责人」可覆盖，但覆盖必须填写原因并留痕——这些覆盖记录正是未来校准阈值、决定标准化力度的数据燃料。
- **【期望上线时间】的权威性（治理闸门之一）**： 它由「需求方」提出、「产品负责人」受理时确认，是双方对交付节奏的契约。「产品负责人」不得为调度便利擅自改日期来操纵紧急程度——若确须调整，须由「需求方」重新确认或走覆盖留痕流程。

### 2.3.9.3 建议优先级（四象限合成）

> **建议优先级 = 重要程度分（2.3.9.1） × 紧急性（2.3.9.2）** → 紧急且重要 / 紧急不重要 / 重要不紧急 / 不紧急不重要

规则将建议优先级填入需求卡片（或「产品负责人」依据公式手算）。**紧急程度为默认基线，可覆盖须填原因留痕**；重要程度分由 2.3.9.1 输入，系统合成后「产品负责人」确认或修正。

**2.3.9.4 矛盾检测（【GCR】 区别于普通需求管理法的核心能力）** — 依据结构化字段做一致性校验，标红自相矛盾的需求：

| 矛盾组合 | 处置动作 |
| --- | --- |
| **业务价值分=0（无价值），但四象限=紧急重要** | 🔴 标红："无价值需求为何紧急重要？请产品负责人确认" |
| **业务价值分极高（核心级），但四象限=不紧急不重要** | 🔴 标红："核心需求被压到低优先级，确认是否遗漏" |
| **来源=服务工单（单用户），但业务价值=核心** | 🟡 提示："单用户工单通常不至核心级，建议复核" |
| **风险程度=高（风险程度分极高、四维度全顶格），但业务价值分极低** | 🔴 标红："高成本低价值需求，确认是否值得做" |
| **期望上线时间已过，状态仍非"已上线"** | 🔴 触发 2.3.7 过期铁律流程 |

### 2.3.9.5 过期检测（需求卫生的自动守门员）

> **过期检测（需求卫生的自动守门员）：** 周期性扫描（人工或自动化）：今天 > 期望上线时间 且 状态 ≠ 「已上线」 → 标红 + 通知产品负责人与需求方 → 产品负责人在下个评审周期内关闭或重估（见 2.3.6）

这一条让"做了 80% 但永远不上线"的僵尸需求无处藏身，从源头保持【需求池】的健康度。

（需求治理全部规则的工具固化方式，见配套《GCR数字化落地指南》第二章。）

### 2.3.10 红线：需求收口的绝对禁区

1. **禁止口头需求直接进入开发。** 任何"老板让我加一个 XX""客户说要改一下 YY"——必须走完待完善→「待评审」→待开始流程。例外情况走 2.3.7 的插单流程，不是绕过流程。
2. **禁止「需求方」直接指定「研发人员」。** "你去找老王把这个做了"——不行。所有需求必须经「产品负责人」分配，「研发人员」只能从待开始状态的任务队列中领取。
3. **禁止"先做再补流程"。** "先做着，回头补个需求文档"——这是所有项目混乱的根源。先有需求，后有代码。顺序不可逆。
4. **禁止「产品负责人」独自决定高价值需求的方向而不通知需求方。** 核心和重要级需求的优先级变更，必须与「需求方」确认。「产品负责人」有排序权，但没有独断权。
5. **禁止绕过变更审批直接改已排期需求。** 待开始之后的任何变更，必须走变更审批并记录变更次数。

**违规后果：** 由「研发效能负责人」在日常巡检中发现违规需求（未走流程直接进入"「进行中」"状态的），该需求立即暂停，责任人（「产品负责人」或「研发人员」）需提交书面说明，整改后方可恢复。

## 2.4 判断口径：推行者最该拿稳的几把尺

| 现场情形 | 判断口径 | 往哪退 |
| --- | --- | --- |
| **需求方把某子维度打了 0，但坚持"这需求很重要"** | **精益否决权生效**：业务价值分归零、不进重要序列。请需求方重新确认——是真的不重要，还是没想清楚 | 归零即落地，不纠结；若确属填写错误，重填后重算 |
| **阈值_T=120 把一半需求判成"不重要"** | 阈值是经验基线、可配置。若团队普遍价值偏低，先调阈值让 Pareto 显形，别让"全不重要"掩盖真问题 | 工具中调阈值_T，记录调整原因 |
| **老板口头插单，且确实火烧眉毛** | 先按 2.3.7 三步判：真·紧急级（线上崩+无替代+可量化损失）才走应急插单；否则填《插单代价声明》走正规路径 | 绝不为"老板急"破闭环；代价透明化 |
| **待完善直接输关闭理由 → 关闭（零审批）** | 这是信任出口：需求还没占资源，关闭零审批合理。但**关闭可重开**，不会丢 | 若关闭后被发现是"假装关掉规避评审"，属红线#1，冻结接入 |
| **退回次数 3 次仍过不了** | 不是需求问题，是产品负责人/业务板块负责人沟通机制问题，推行者介入复盘 | 见计数器异常信号 |
| **产品负责人擅自改期望上线时间操纵紧急度** | 越权。日期是双方契约，改须需求方重确认或走覆盖留痕 | 红线外的行为偏差，巡检发现即纠正 |

> **推行者心法：** 这条链路上，你最大的武器是"留痕"。每一次覆盖、每一次插单、每一次关闭，都留下记录。这些记录短期是"博弈弹药"（老板拍桌子时你有数据），长期是"校准燃料"（阈值该不该调、流程哪里卡，全靠它）。**不要怕流程慢，怕的是流程没有记忆。**

## 2.5 检查清单

需求治理机制上线前，逐条确认：

- ☐ 是否已建立需求台账（工具不限：白板 / Excel / 明道云均可），并定义必填字段?
- ☐ 是否锁定了唯一入口——只有「产品负责人」能接收并录入需求?
- ☐ 【业务价值】三子维度（10/7/3/1/0）标准是否已对齐并可供「产品负责人」选用?
- ☐ 【风险评估】四维度档位是否已定义并可供选用?
- ☐ 【期望上线时间】字段是否已约定（日级精度，仅作排期锚点）？
- ☐ 【计划开始时间】是否已在「待开始」时自动派生（基于【交付功能】迭代安排，用于甘特图）？
- ☐ 状态流转规则是否已明确（必填字段齐全 → 进入「待评审」，由「产品负责人」或系统推进）？
- ☐ 驳回 / 变更计数机制是否已建立（人工记录或系统累加）？
- ☐ 变更审批、关闭审批节点是否已定义（审批不通过则回退）？
- ☐ 【需求漏斗】通过率是否有人定期统计（【外部诉求】→「待完善」、「待评审」→「待开始」）？
- ☐ 「产品负责人」是否已经向所有「需求方」宣讲过"需求必须找我，不要直接找开发"?
- ☐ 是否建立了【需求池】定期审查机制（建议双周一次）？
- ☐ 【需求池】上限预警机制是否建立（活跃数 > 迭代容量 × 3 时预警）？

## 2.6 常见问题（推行阻力 Q&A）

**Q：老板直接在群里 @ 开发说"帮我加个功能"，算不算违规?**

算。但这不代表你要当面怼老板。正确的做法是：开发收到后回复"收到，我让「产品负责人」评估一下排期"——然后把消息转给「产品负责人」。「产品负责人」去

跟老板走正规流程。开发不拒绝，也不执行，而是把球传给「产品负责人」。这就是"规则挡住需求"的实际操作。

**Q：「需求方」说"我很急，来不及走流程了"，怎么办？**

"急"不等于"跳过流程"。【GCR】 有应急通道（2.3.7），但应急通道本身也是流程的一部分——它需要填《插单代价声明》，需要「研发效能负责人」审批，需要通知所有受影响的需求方。所以答案永远是：**越急越要走流程，因为流程就是用来处理"急事"的。**

**Q：【风险评估】四个维度，每次都要填吗？**

是的。但不需要开评审会。「产品负责人」在收集需求信息的日常过程中就可以跟研发随口问一句"这个大概影响几个表？""有没有外部接口？"——研发凭经验给个档位就行。四个维度的评估总耗时不应超过 15 分钟。它的精度不需要达到"精确到人天"的程度——只需要区分"这是一个小改动"还是"这是一个大工程"即可。

**Q：【需求池】满了（超过 3 倍迭代容量），新的需求怎么办？**

先不录入为【用户需求】（进入候补队列）。「产品负责人」告诉「需求方」："当前【需求池】已满，您的新需求将进入候补队列。我们会每两周清理一次池子，届时优先处理。"这倒逼「需求方」自己想清楚：这个需求真的那么重要吗？如果不重要，可能自己就撤回了。

**Q：【用户需求】被驳回到待完善了，「需求方」不满意怎么办？**

驳回不是拒绝，是"信息不足、请补全"。「产品负责人」/「业务板块」负责人在「待评审」退回时必须写明原因（缺场景描述？价值说不清？依赖未确认？）。「产品负责人」补完再提交，**退回次数** +1（人工记录或工具累加）——如果这个需求被退回 3 次，「产品负责人」和「研发效能负责人」要复盘：是需求方不会提需求，还是评审门槛过高。

# 03 迭代与交付

> 需求收口完成后，【用户需求】在"「待排期」"被拆解成"【交付功能】"，【交付功能】才真正进入【GRT】【迭代周期】。本章讲清楚：【交付功能】长什么样、迭代怎么跑、团队怎么协作、功能怎么发布。

## 3.1 为什么：节流阀，而不是无限并行

【GCR】 解决中国式多线并行痛点的结构性答案，叫**节流阀**：

- **需求侧（上游）多线并行无瓶颈**——【需求池】全收不拦，所有业务线的需求都进来；
- **执行侧（下游）同期只跑单一迭代**——资源有限，同一时间只跑一个必要迭代。

这就是"兼容多线 + 保质量"的答案。很多团队乱，是因为上游注水、下游也注水——需求无限进来、迭代也无限开。结果每个迭代都塞满、每个都延期、质量全靠个人英雄主义兜底。【GCR】 的做法是：上游敞开收，下游用固定【迭代周期】节流。资源/时间有限，须遵守规则、牺牲个人英雄主义。

> 推行者要反复向团队讲清这句话：**可以多项目并行，但同一时间只跑一个必要迭代。** 迭代 = 节流阀。

## 3.2 怎么推：把"大需求"切成"能装盒的功能"

**难点一：需求方/老板理解不了"为什么一个需求要拆好几个迭代"。**

→ 用关系层级模型图（见 3.3）当面讲："用户需求是大颗粒，可能横跨多个迭代；交付功能是装进迭代的执行单元。"不要强行解释状态机，先让对方接受"粒度两层"这个事实。

**难点二：团队习惯"一个迭代塞所有事"，怕盒子装不满。**

→ 盒子装不满不是失败，是健康。盒内只装已排期的【交付功能】，其余进下期或走缓冲窗。缓冲窗口（10-20% 容量）用来吸收插单和紧急修复，是安全垫不是浪费。

**难点三：迭代内还想要站会、周会、评审会一大堆。**

→ 【GCR】 的协作会议**只有两个**（功能评估会 + 功能交付会），不设独立站会。日常阻塞由「产品负责人」在协作任务看板实时跟进。会议少而精，是为保住迭代的执行节奏。

**难点四：设计稿总在迭代内才出，前端空等。**

→ 交互概念方案已前置到需求画像期（第二章 2.3.1）。推行者要检查：需求进「待开始」前，低保真方案是否就绪。没就绪就不许拆功能进盒。

## 3.3 规则参考：两层粒度、状态机与迭代

### 3.3.1 两层粒度模型

【GCR】 把"要做的事"分成两层，粒度不同、生命周期不同、管理视图不同：

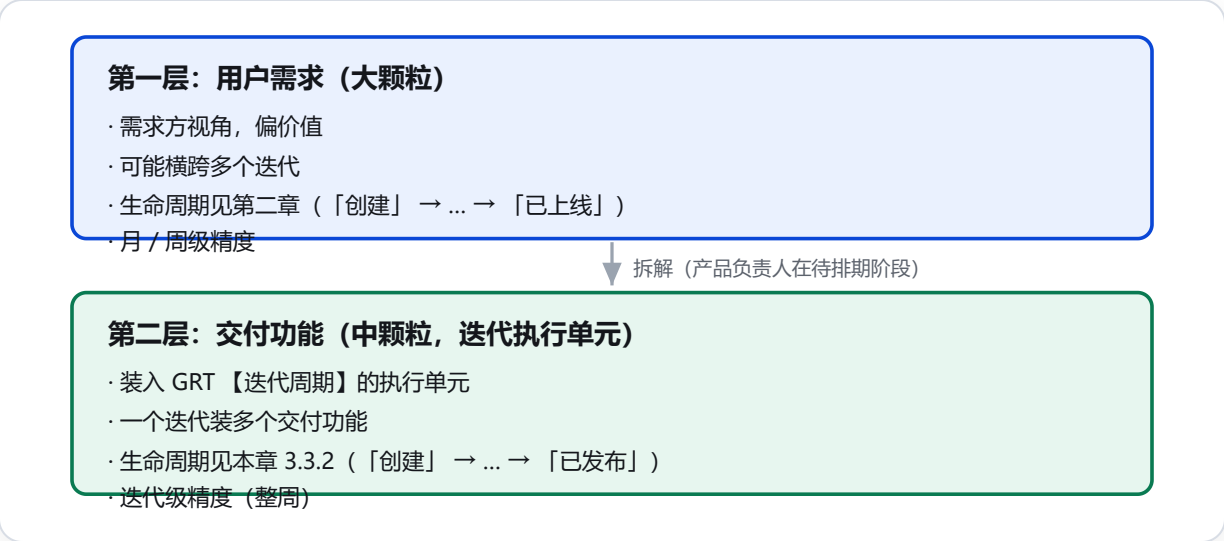
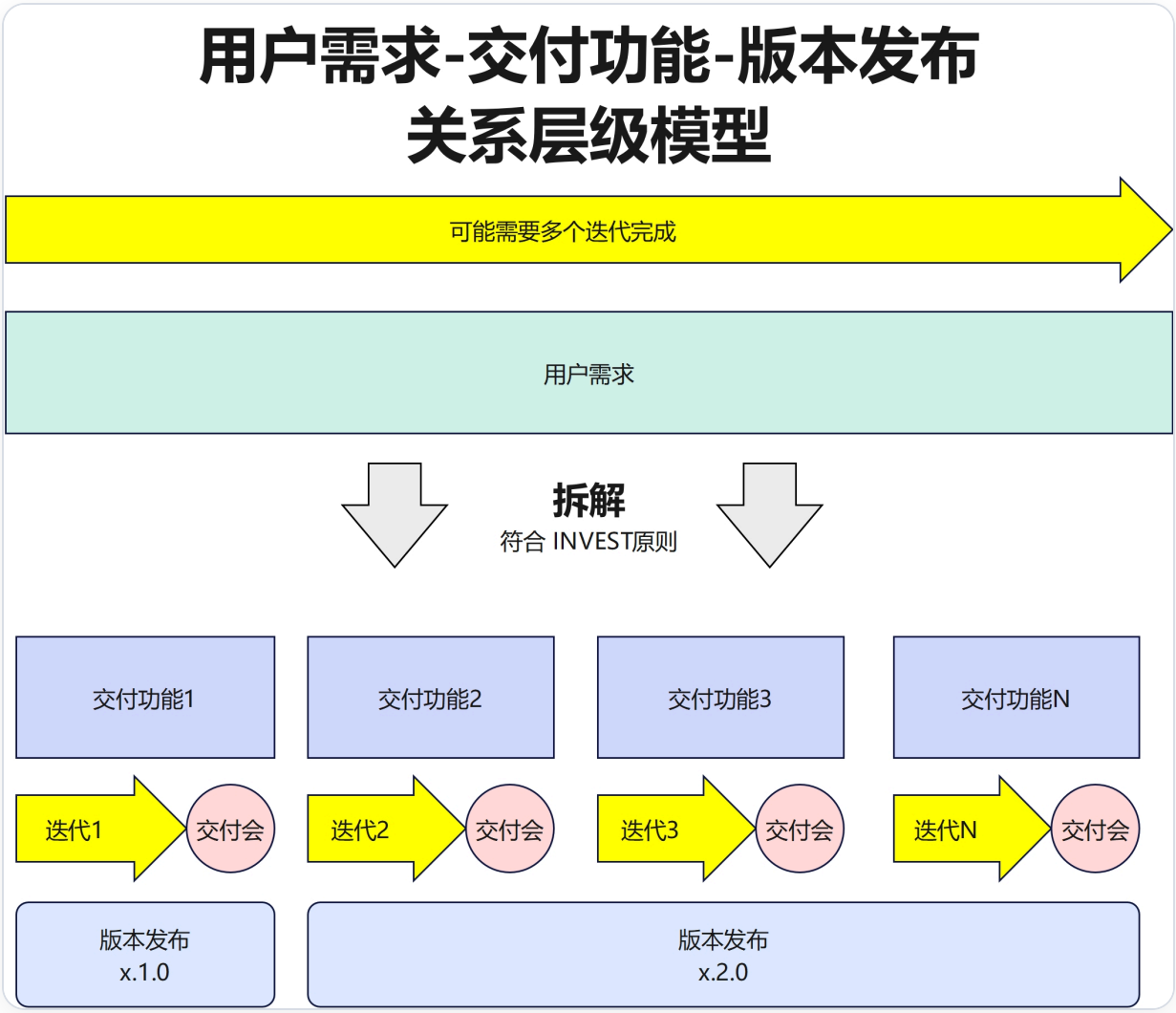



**图 3.1　两层粒度模型：用户需求 → 拆解 → 交付功能**

**关键边界：** "【迭代计划】只对【交付功能】生效。" 【用户需求】不进【迭代周期】——它太大；【迭代周期】管的是这层"【交付功能】"。

> 下图直观展示「用户需求 → 交付功能 → 迭代 → 交付会 → 版本发布」的拆解与层级关系。一个【用户需求】可能需要多个迭代才能完成（横向箭头），每个迭代内装若干【交付功能】，每次交付会后产出版本发布。




**用户需求-交互功能-版本发布 关系层级模型**

### 3.3.2 【交付功能】完整【状态机】

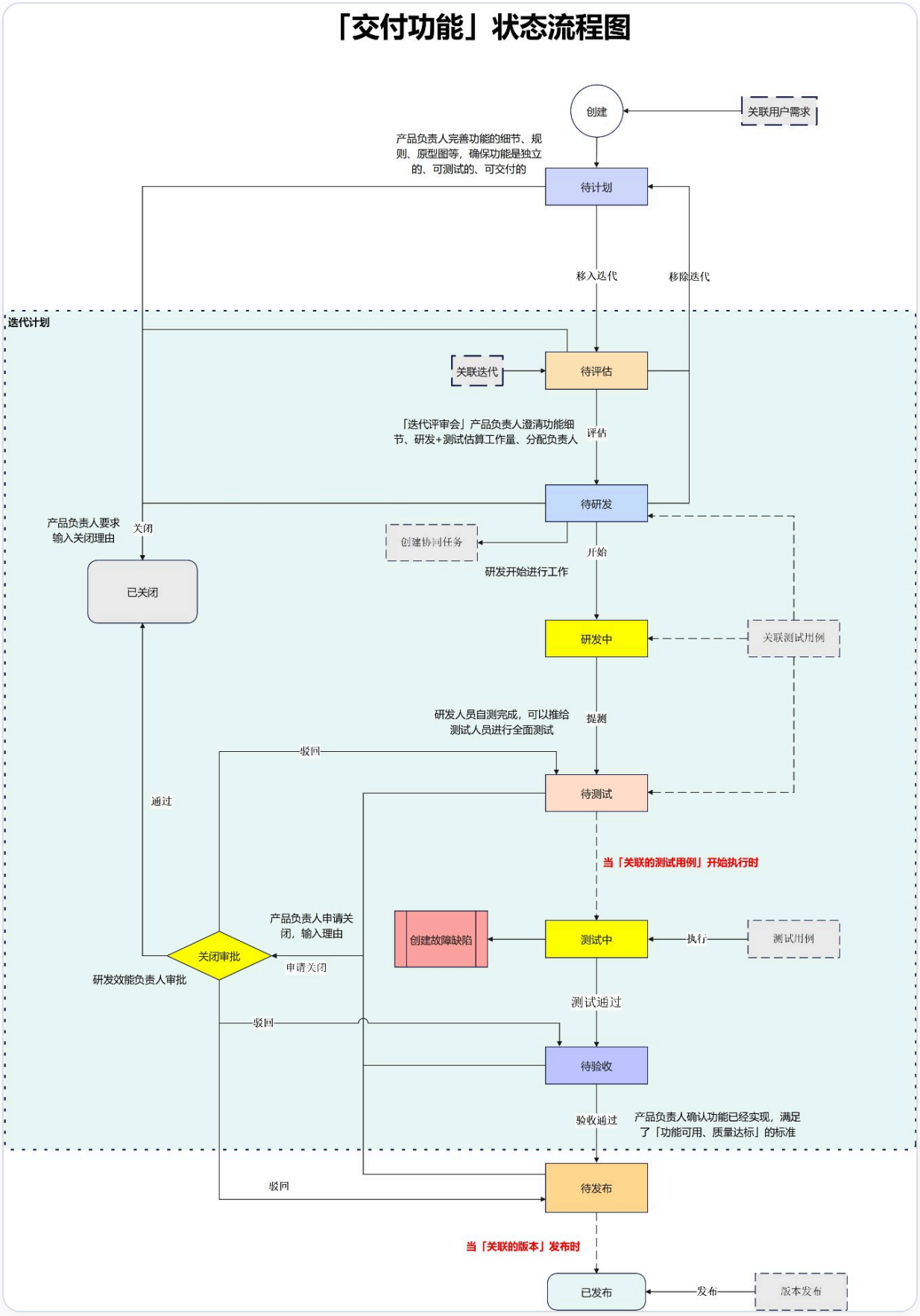
【交付功能】从"【用户需求】拆解"而来，经历以下状态：

**【交付功能】状态机矩阵**

> 本状态机与「5 大模型的数字化联动」总图的交付功能列逐列一致。

| 状态 | 参与对象 | 做什么 | 流转至 |
| --- | --- | --- | --- |
| **创建** | 产品负责人 | 从【用户需求·待开始】拆解生成；填写基础信息（标题、描述、关联需求、预估工作量） | 待评估 |
| **待评估** | 产品负责人 + 研发效能负责人（可选） | 评估是否值得做、工作量多大、技术可行性 | 确认要做 → 待研发；不值得 → 关闭 |
| **待研发** | 产品负责人 + 研发效能负责人 | 已确认要做，等迭代排期；纳入【迭代周期】（3.3.3） | 迭代启动、任务分配 → 研发中 |
| **研发中** | 研发人员 | 开发中；**此状态的任一【交付功能】触发关联【用户需求】→进行中**（跨模型联动自动） | 开发完成 → 待测试；执行用例发现缺陷 → 跨列联动到故障缺陷·创建 |
| **待测试** | 测试人员 | 开发完，等测试；研发提测 | 测试人员领取 → 测试中 |
| **测试中** | 测试人员 | 执行测试；执行用例通过/失败 | 通过 → 待验收；执行用例发现新缺陷 → 跨列联动到故障缺陷·创建 |
| **待验收** | 产品负责人 + 设计人员（可选） | 测试完，等验收 | 通过验收 → 待发布；拒绝验收 → 回退研发中 |
| **待发布** | 产品负责人 | 验收通过，等发布窗口；准备版本发布材料 | 版本发布（功能交付会议） → 已发布 |
| **已发布** | 产品负责人 | 已上线发布；需求方打分（交付质量 / 感知度）汇入第五章【验收评分矩阵】 | 终态 |
| **关闭** | 产品负责人 | 评估后认为不值得做 / 取消 | 终态 |

下图为「交付功能」完整状态流转图（含迭代周期虚线框、审批节点、分支路径），是本矩阵的可视化权威源。图中： - 虚线框 = 【迭代周期】容器，围住待研发→「研发中」→「待测试」→「测试中」→待验收五态 - 关闭审批由「研发效能负责人」审批；驳回回退至待评估 - 当【关联的测试用例】开始执行时触发测试阶段 - 创建协作任务 / 关联测试用例为跨模型联动（红虚线）


「交付功能」状态流程图
创建
关联用户需求
产品负责人完善功能的细节、规则、原型图等，确保功能是独立的、可测试的、可交付的
待计划
移入迭代
移除迭代
迭代计划
关联迭代
待评估
「迭代评审会」产品负责人澄清功能细节、研发+测试估算工作量、分配负责人
评估
待研发
产品负责人要求输入关闭理由
关闭
已关闭
创建协同任务
开始
研发开始进行工作
研发中
关联测试用例
研发人员自测完成，可以推给测试人员进行全面测试
提测
驳回
待测试
通过
当「关联的测试用例」开始执行时
产品负责人申请关闭，输入理由
创建故障缺陷
测试中
执行
测试用例
关闭审批
申请关闭
研发效能负责人审批
测试通过
驳回
待验收
验收通过
产品负责人确认功能已经实现，满足了「功能可用、质量达标」的标准
驳回
待发布
当「关联的版本」发布时
已发布
发布
版本发布

「交付功能」状态流程图

**【迭代周期】虚线框：** 「待研发」 → 「研发中」 → 「待测试」 → 「测试中」 → 「待验收」，这五个状态在【GRT】【迭代周期】内部运行。【迭代周期】是它们的容器（3.3.3 详述）。

### 3.3.3 【GRT】迭代计划（【迭代周期】）

【GRT】（【双维全域治理】）用固定【迭代周期】约束交付节奏。**默认 2 周一个迭代，可选 1-4 周。**

| 迭代周期 | 适用场景 |
| --- | --- |
| **1 周** | 超高不确定性的探索型业务、紧急救火期 |
| **2 周（默认）** | 绝大多数稳态研发团队 |
| **3 周** | 业务节奏偏慢、评审周期长的团队 |
| **4 周** | 强合规/强评审环境（金融、政企） |

**【迭代周期】内部节奏（双阶段拆分）：**


**图 3.2 【迭代周期】内部节奏（双阶段拆分，默认 2 周）**

**【GRT】【迭代周期】的三条铁律**

1. **盒内只装已排期的【交付功能】。** 不在本期迭代清单里的功能，不进盒——要么进下期，要么走插单评估（第二章 2.3.7）。
2. **盒不可被口头打破。** 老板临时需求不能直接塞进进行中的迭代，必须走 2.3.7 的插单流程（填代价声明、通知受影响方）。

3. **盒有缓冲窗口。** 每个迭代预留 10-20% 容量作缓冲，用于插单、紧急修复。缓冲不是"浪费"，是吸收波动的安全垫。

### 3.3.4 迭代内协作（交付团队内部）

> **边界：** 迭代内协作指**交付团队内部**协作，**需求方不参与**。需求方仅在【用户需求】确认阶段（第二章「待评审」→「待开始」）参与；上线后的验收打分属价值回环，见 3.3.7。

一个【迭代周期】内，交付团队四方角色按以下泳道协作：

| 角色 | 在迭代内的核心动作 |
| --- | --- |
| **产品负责人** | 维护迭代清单、主持两个协作会议、澄清需求、验收交付功能 |
| **研发人员** | 认领并推进【协作任务】、编码、自测、提测 |
| **测试人员** | 写测试用例、执行测试、提缺陷、验证修复 |
| **设计人员** | （前置）提供交互概念方案；迭代内交付高保真稿 / 切图配合、参与设计评审 |

> **【协作任务】：** 交付团队内部为推进【交付功能】而协作的事项（如研发-测试联调、设计交付评审、跨功能接口对齐），在协作任务看板中跟踪，不单独排迭代——协作语境一律用【协作任务】。

**协作节奏（两个会议）：** 【迭代周期】内的正式协作会议**只有两个**，其余为日常执行流：

1. **迭代内功能评估会议**（迭代启动）：评估本期【交付功能】的范围、技术可行性与协作分工，明确各【协作任务】的归属与依赖。
2. **功能交付会议**（迭代结束）：演示已完成的【交付功能】、验收、「发布」、复盘。

> 日常开发测试为执行流，不设独立站会会议；阻塞由「产品负责人」在协作任务看板中实时跟进。

> **协同任务可分解出【故障缺陷】：** 协作过程中（如跨功能接口对齐、设计交付评审）发现的质量问题，应分解出缺陷单跟进，对应全景图"协同任务 → 分解 → 故障缺陷"。

### 3.3.5 交付功能内的工作分解

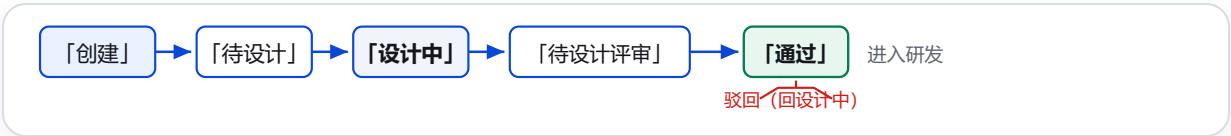
【交付功能】进入迭代后，由研发 / 测试 / 设计按角色认领并推进，**不再设独立的"功能任务"实体**——这些工作直接挂在【交付功能】上跟踪：

| 工作类型 | 说明 | 归属 |
| --- | --- | --- |
| **开发任务** | 具体编码工作（如"门禁息屏后端接口"） | 研发人员领取 |
| **测试任务** | 具体测试工作（如"息屏功能回归测试"） | 测试人员领取，见 3.3.8 |
| **设计任务** | 高保真界面细化 / 切图配合（概念方案已在需求期前置，见 2.3.1） | 设计人员领取，见 3.3.6 |

上述工作从【交付功能】的"「研发中」"状态派生，跟随【交付功能】的生命周期。它们全部完成，【交付功能】才能从"「研发中」"推进到"「待测试」"。

### 3.3.6 交付设计

涉及界面的【交付功能】，需经过设计环节（对应下方"交付设计"子流程）。概念方案已在需求画像期确定（见 2.3.1），此处是细化与评审关卡——设计未通过评审，研发不启动，避免"做完了才发现界面不对"的返工：



设计评审由「产品负责人」主导，研发参与（确认技术可行性）。设计未通过评审，研发不启动——避免"做完了才发现界面不对"的返工。

### 3.3.7 版本发布

【交付功能】通过验收后，进入发布环节（对应下方"版本发布"子流程）。版本发布是全景图顶层"版本发布"节点——需求价值兑现的顶层产出动作，向下包含到【交付功能】：

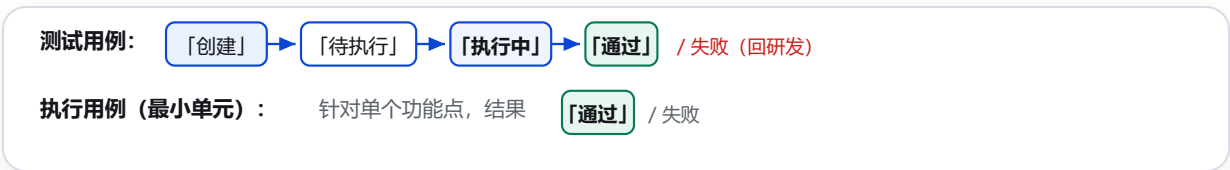

- **版本发布** 是技术动作（打包、部署、灰度）
- **「发布」** 是业务动作（通知需求方可用）
- **「需求方打分」** 是价值回环（交付质量、满意度），分数汇入【验收评分矩阵】（第五章）

发布节奏与【迭代周期】对齐：每个迭代结束时统一发布，避免"随时发"造成的环境混乱。

### 3.3.8 测试用例与执行

「测试人员」 在【交付功能】"「待测试」"阶段同步编写**测试用例**，测试阶段**【执行用例】**。

**测试计划（迭代前的质量就绪规划）：** 对应全景图"测试计划 → 包含 → 测试用例 → 关联 → 执行用例"独立纵向链。测试人员在迭代内功能评估会议后、迭代启动前，针对本期【交付功能】制定测试计划（含用例范围与优先级），作为迭代就绪（DoR）的一部分前置完成——而非迭代内才临时想测什么。



- 测试不通过（执行用例失败）→ 自动触发【交付功能】回"「研发中」"（红虚线联动）
- 修复后重新生成【执行用例】 → 再次执行 → 通过才进"「待验收」"
- 这就是总图中"「待验收」→拒绝验收→回研发中"的自动化闭环

### 3.3.9 【跨模型联动】：功能 ↔ 缺陷 ↔ 评分矩阵

| 触发源 | 动作 | 目标 | 类型 |
| --- | --- | --- | --- |
| **交付功能·待验收** | 自动触发（红虚） | → 故障缺陷·创建（验收发现缺陷） | 自动化 |
| **交付功能·已发布** | 结果汇总（红虚） | → 验收评分矩阵 | 自动化 |
| **用户需求·已上线** | 达标（红虚） | → 验收评分矩阵 | 自动化 |
| **故障缺陷·待复测** | 结果汇总（红虚） | → 验收评分矩阵 | 自动化 |

**联动说明：** 【交付功能】在验收阶段发现缺陷，应建立缺陷单（数字化后自动建，手工模式由「测试人员」建单），缺陷修完才能重新验收。已发布的【交付功能】，其质量/满意度打分自动汇入第五章的【验收评分矩阵】——这是【GCR】"闭环"二字的最终落点。缺陷的完整【状态机】与分级见 3.4。

### 3.3.10 质量门禁与【故障缺陷】治理

> **缺陷是交付的"质量达标"维度。** 一个【交付功能】要真正闭环，必须同时满足**功能可用 + 质量达标**。缺陷就是"质量不达标"的显形，是交付的【质量门禁】——它与【交付功能】紧耦合：验收发现缺陷→自动回退研发中；修复完才能重新验收 / 「发布」；致命缺陷清零才允许上线（3.4.2）。

**【故障缺陷】完整【状态机】**

【故障缺陷】从"【交付功能·测试中/待验收】发现"（跨列联动）或"【服务工单·判断】转缺陷"（跨模型联动）而来：

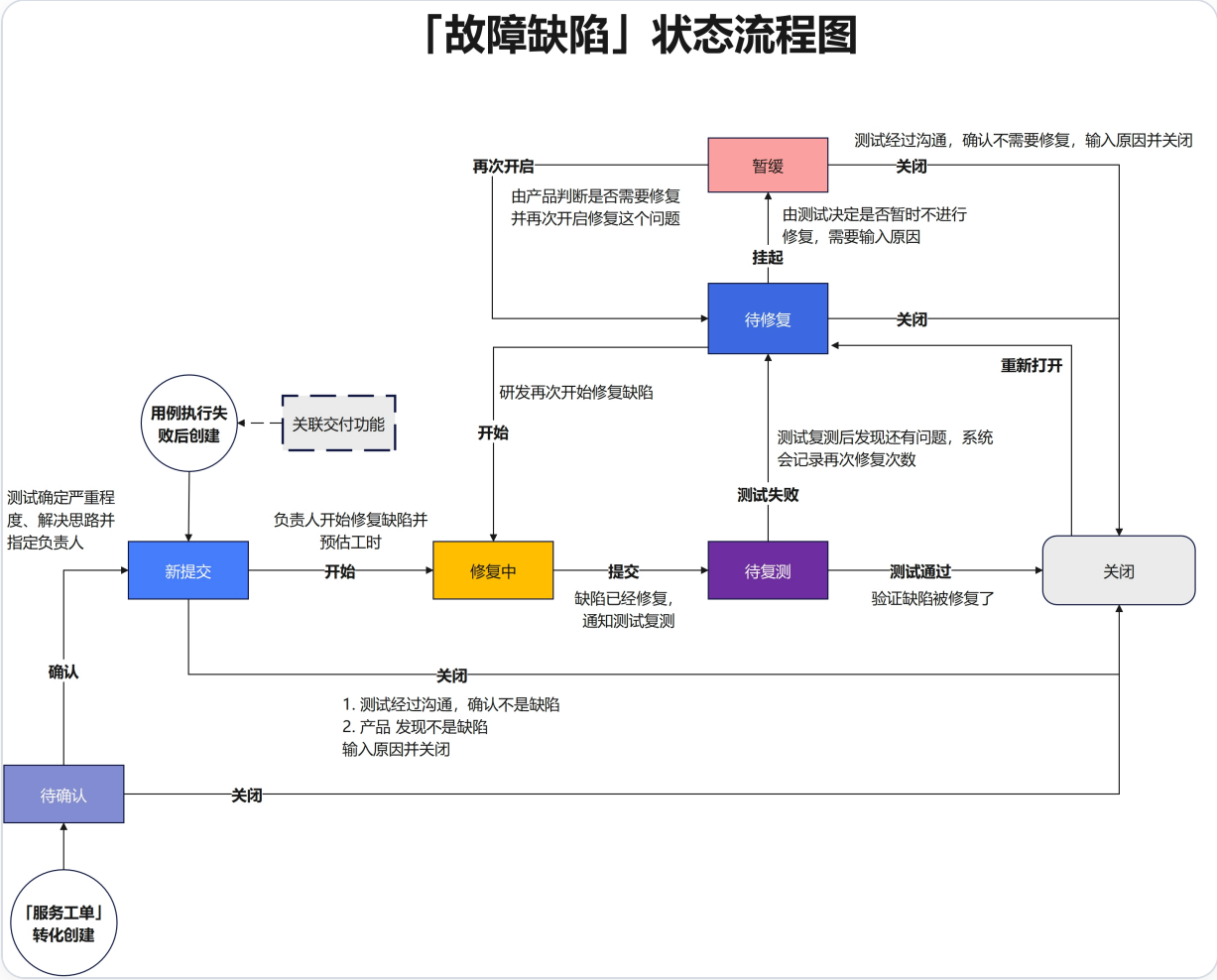
**【故障缺陷】状态机矩阵**

本状态机与「5 大模型的数字化联动」总图的故障缺陷列逐列一致。⚠️ 白皮书高清图仅展示 **5 态主链路**（「新提交」→「待确认」→「修复中」→「待复测」→「关闭」）；下表保留「创建/待修复」作为系统内部辅助态并注明其与图示的区别。

| 状态 | 参与对象 | 做什么 | 流转至 |
| --- | --- | --- | --- |
| **新提交（起始态）** | 测试人员（交付功能触发）/ 服务台人员（工单转缺陷） | 提交缺陷，填写标题、描述、复现步骤、关联源（【交付功能】或【服务工单】）、初步分级 | 待确认 |
| **待确认** | 相关方（研发 / 产品 / 测试） | 确认是否为有效问题；无效则直接关闭 | 确认有效 → 修复中；关闭（非缺陷：测试通过沟通不须修 / 产品发现不是缺陷，输入原因并关闭） |
| **修复中** | 研发人员 | 接收缺陷，定位根因并实施修复；预估工时 | 修复完成提交 → 待复测 |
| **待复测** | 测试人员 | 回归验证（执行用例）；确认是否真正修好 | 通过 → 关闭；失败（复测未通过 / 发现新问题）→ 回退修复中 |
| **暂缓（条件分支）** | 产品负责人 | 由产品判断是否暂时不修（挂起），需输入原因 | 挂起，可后续再次开启回修复中；关闭（确认不需要修复，输入原因） |
| **关闭** | 测试通过 / 待确认判非缺陷 | 缺陷验证无误，正式闭环 | 终态；可重新打开（如后续重现）→ 回新提交 |

> **说明：** - 主链路为 5 态：**「新提交」** → **「待确认」** → **「修复中」** → **「待复测」** → **「关闭」**——这与白皮书状态流程图完全一致。 - 「创建」在系统内部作为录入动作存在，但白皮书与流程图以「新提交」为起始态，不设独立「创建」态。 - 「待修复」在某些系统实现中作为"修复完成等待分配测试"的中间态存在，但权威流程图中修复中直连待复测，无此中间态。若所用工具含此态，视为修复中→待复测之间的系统内部过渡。 - 「暂缓」为条件分支（非主链路）：由产品判断暂不修复时进入，可后续重新打开。

> 下图为「故障缺陷」完整状态流转图（含暂缓分支、关联交付功能虚框、重新打开路径），是本矩阵的可视化权威源：




「故障缺陷」状态流程图

## 关键节点

| 节点 | 说明 |
| --- | --- |
| **新提交** | 起始态，无独立创建态；由用例执行失败后创建，或由【服务工单】转化创建 |
| **待确认** | 相关方确认是否为有效缺陷；确定非缺陷则直接关闭（两种情形：①测试沟通后不需修 ②产品发现不是缺陷） |
| **修复中** | 研发人员定位根因并实施修复；预估工时 |
| **待复测** | 修复完成后提交测试回归验证；测试失败则回退修复中（系统记录再次修复次数） |
| **暂缓** | 条件分支：产品判断暂时不修时挂起，需输入原因；可后续再次开启回修复中，也可直接关闭 |

### 缺陷分级（白皮书 13.3 标准）

| 级别 | 定义 | 响应时效 |
| --- | --- | --- |
| **致命** | 核心功能不可用、数据丢失、安全/合规底线被击穿 | 立即（当前【迭代周期】内优先修，且禁止带致命缺陷上线） |
| **严重** | 主流程受阻，有替代方案 | 本迭代内修 |
| **一般** | 非核心功能异常 | 下个迭代排期 |
| **轻微** | 体验类、性能类可优化的非缺陷问题 | 待办清单，定期清理 |

> ⚠️ **命名纪律**：废弃旧词"优化类"——白皮书与 SOP 统一用"**轻微**"，避免与"重要/严重"混淆。

**致命缺陷清零原则：** 任何环境、任何版本，禁止携带致命级缺陷交付与上线（白皮书 13.3 硬性质量底线）。

**缺陷修复闭环：** 缺陷创建即关联源【交付功能】，修复后必须回归测试（【执行用例】通过才关闭）；缺陷关闭率、平均修复时长纳入第五章【效能度量】；同一【交付功能】反复出缺陷 → 提示该模块质量风险，需在迭代复盘中处置。

### 3.3.11 事务工作（非研发）

交付团队成员承担的非研发类事务（如客服回复、文档审核、线下协调、会议组织），对应全景图"事务工作(非研发)"实体。

- **不进入【迭代周期】**，不占迭代容量；
- 「通过」"关联"挂到相关【交付功能】做追溯（如"此客户投诉关联【交付功能】X"）；
- 轻量记录即可，避免与研发任务混淆排期。

## 3.4 判断口径：迭代现场的松与严

| 现场情形 | 判断口径 | 往哪退 |
| --- | --- | --- |
| **老板要往进行中的迭代塞需求** | 盒不可被口头打破。走 2.3.7 插单流程；非紧急进下一期缓冲窗 | 【迭代周期】铁律 #2 |
| **迭代只装了 60% 容量，团队焦虑"浪费"** | 健康信号。盒内只装已排期功能，余量留给缓冲窗吸波动 | 铁律 #3 |
| **功能卡在"待测试"没人测** | **待测试是硬卡点**：没测完不许进待验收。测试资源不足时先补测试岗，别让研发自测放行 | 质量门禁底线 |
| **验收发现致命缺陷** | **零容忍**：禁止带致命缺陷上线，立即修、本【迭代周期】内优先 | 致命清零原则 |
| **严重缺陷本迭代没修完** | 严重级管控率目标 ≥98%（缓冲周期内修复复测通过占比）；没修完不许带病发布 | 质量底线 |
| **缺陷被标"暂缓"** | 产品判断暂不修须输入原因，可重开；但"暂缓"不能变成"永远不修"——定期清理 | 暂缓是条件分支非垃圾桶 |
| **设计稿迭代内才出、前端空等** | 概念方案应前置到需求期；没就绪不许拆功能进盒 | 3.2 难点四 |

> **推行者心法：** 节流阀的精髓是"下游守得住"。你可以让上游需求池再满，但只要【迭代周期】守得住、致命缺陷清零守得住，团队就不会被洪流冲垮。守不住盒子，前面所有治理白做。

## 3.5 检查清单

（【迭代与交付】的全部工具配置，见配套《GCR数字化落地指南》第三章。）

- ☐ 是否已建立【交付功能】台账，并定义【状态机】（含待研发→「研发中」→「待测试」→「测试中」→「待验收」→「待发布」→已发布的完整流转）？
- ☐ 是否已建立【迭代周期】（默认 2 周，可在 1-4 周间选择），并约定容量?
- ☐ 【交付功能】与迭代的归属关系是否已记录（哪个功能装进哪个盒）？
- ☐ 开发 / 测试 / 设计任务是否已按角色认领并跟踪?
- ☐ 测试用例是否已编写，执行结果（「通过」 / 失败）是否已记录?
- ☐ 验收发现缺陷时，是否能建立缺陷记录并关联源功能（见 3.4）？
- ☐ 是否建立了缓冲窗口（10-20% 容量）用于吸收插单与紧急修复?

# 04 服务工单治理

> 一线用户 / 需求方报来的问题、请求、咨询，先进【服务工单】这个"分诊台"。它多数会转成【用户需求】或【故障缺陷】，少数原地闭环。本章讲【服务工单】的完整【状态机】与分流规则。

## 4.1 为什么：一线问题不治理，会反噬需求池与交付

很多团队把工单当成"客服的事"，跟研发治理割裂。结果有两个烂法：

- **工单越积越多、没人知道在干嘛**——问题在服务台里腐烂，需求方满意度崩盘；
- **一线问题绕开治理直接找研发**——本该转需求 / 转缺陷的，变成口头插队，前面积累的入口纪律一次性崩掉。

【服务工单】是【GCR】 五大生命周期的"服务响应域"，也是需求与缺陷的**入口分诊台**。把工单正确分流，是避免上面两个烂法的关键。它和【需求池】、【故障缺陷】通过跨模型联动自动打通——工单不是终点，是源头之一。

## 4.2 怎么推：让工单有人接、有人分流

**难点一：工单进来没人接，超时了怪谁?**

→ 用响应 SLA（「创建」→「待接单」 ≤1 小时）兜底；超时由「研发效能负责人」**强行分配接单人员**（见 4.4 判断口径）。推行者要先把"接单责任人"这一格填死，否则工单永远在空转。

**难点二：服务台分不清"这是需求还是缺陷还是该直接处理"。**

→ 不要指望人肉判断永远对。用 4.3 的五向分流决策表当检查单：派单 / 转需求 / 转缺陷 / 源单 SLA 超时自动转需求。把"「判断」"变成可勾选的动作，降低对个人的依赖。

**难点三：用户说"没解决"但工单被关了，扯皮。**

→ 关闭不是由处理人单方面决定。「待反馈确认」环节由「需求方」或「服务台人员」确认；确认未解决 → 回退待处理、**未解决次数+1**。同一工单未解决超 3 次 → 升级「产品负责人」复盘。反馈 SLA 超时系统自动标记已解决并关闭，是兜底不是常规。

**难点四：工单转需求 / 转缺陷后，原工单算不算完?**

→ 转出去后原工单进入对应生命周期，工单本身闭环（「关闭」）。但关联要建——转需求挂到【用户需求·创建】，转缺陷挂到【故障缺陷·创建】，可追溯。

## 4.3 规则参考：状态机、SLA 与分流

### 4.3.1 【服务工单】完整【状态机】

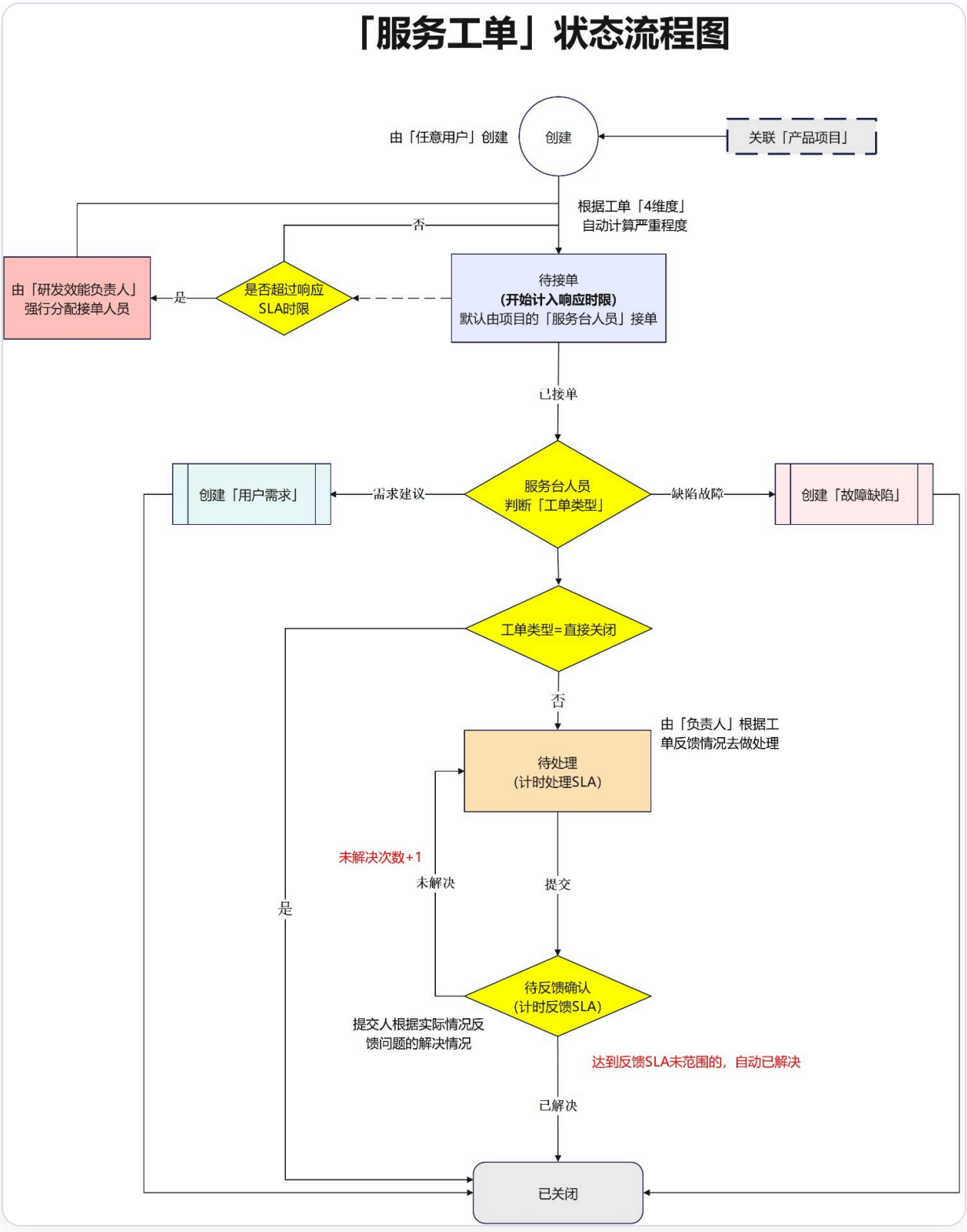
【服务工单】来自一线用户/需求方的服务请求（可能由【用户需求·已上线】反馈触发创建）：

**【服务工单】状态机矩阵**

> 本状态机与「5 大模型的数字化联动」总图的服务工单列逐列一致。

| 状态 | 参与对象 | 做什么 | 流转至 |
| --- | --- | --- | --- |
| **创建** | 需求方 / 服务台人员 / 系统自动触发 | 录入工单：标题、描述、来源（用户 / 系统 / 已上线反馈）；**填写 4 维智能分类问卷（4.3.2.1）**；系统自动推导严重程度 | 待接单 |
| **待接单** | 服务台人员 | 接收并初步分类；确认工单信息完整；查看问卷结果与自动推导的严重程度 | 信息完整 → 判断 |
| **判断（◆黄菱）** | 产品负责人或服务台负责人 | 结合问卷结果判断**工单类型**（缺陷故障/需求建议/技术支持/问题咨询/直接关闭），定义优先级与负责人，执行**五向分流**（4.3.2 详述） | 派单 SLA 限时 → 待处理；转需求（跨模型）→【用户需求·创建】；转缺陷（跨模型）→【故障缺陷·待确认】；直接关闭 → 关闭 |
| **待处理** | 被派单人（研发 / 产品 / 服务等，按判断结果） | 处理工单请求（配置 / 咨询 / 排查等）；受"处理 SLA 限时"约束 | 处理完成提交 → 待反馈确认；可主动关闭 → 关闭 |
| **待反馈确认** | 服务台人员 | 向用户确认问题是否解决；受"反馈 SLA 限时"约束 | 确认 → 回访解决问题 |
| **回访解决问题（◆黄菱）** | 需求方或服务台人员 | 最终确认是否真正解决 | 是 → 关闭；否 → 回退待处理（重新处理） |
| **关闭** | 服务台人员 / 需求方 | 工单闭环终态 | 终态；可重新打开 ↔ 待处理（双向路径） |

> 下图为「服务工单」完整状态流转图（含 SLA 时限判断、五向分流、未解决计数器），是本矩阵的可视化权威源。图中： - 黄色菱形 = 决策节点（是否超过响应 SLA 时限 / 工单类型判断） - 「未解决次数+1」为循环计数器，未解决的待处理工单重新进入处理 - 达到反馈 SLA 未范围时，系统自动标记已解决




「服务工单」状态流程图

#### 4.3.1.1 SLA 时限规则（流程图中决策节点的量化标准）

流程图中出现多处"SLA 时限"判断与计时约束，以下是具体量化标准（经验初定，按团队校准）：

| SLA 类型 | 触发位置 | 默认阈值 | 超时处置 |
| --- | --- | --- | --- |
| **响应 SLA** | 创建 → 待接单 | 按严重程度分级：致命 ≤ 1h / 严重 ≤ 4h / 一般 ≤ 8h / 轻微 ≤ 24h（见 4.3.2.2） | 超时由研发效能负责人强行分配接单人员 |
| **派单 SLA** | 判断节点 → 待处理 | 按严重程度分级：致命 ≤ 4h / 严重 ≤ 8h / 一般 ≤ 24h / 轻微 ≤ 72h（见 4.3.2.2） | 超时升级至产品负责人介入 |
| **处理 SLA** | 待处理 → 解决 | 按严重程度分级：致命 ≤ 2h / 严重 ≤ 4h / 一般 ≤ 24h / 轻微 ≤ 48h（见 4.3.2.2） | 记入效能度量；超时触发升级通知 |
| **反馈 SLA** | 处理完成 → 反馈确认 | 按严重程度分级：致命 ≤ 1 天 / 严重 ≤ 3 天 / 一般 ≤ 5 天 / 轻微 ≤ 7 天（见 4.3.2.2） | 超时自动标记"已解决"并关闭（系统兜底） |

**关键逻辑：** - 响应 SLA 超时时，由「研发效能负责人」**强行分配接单人员**——这是流程图中"否→由研发效能负责人强行分配接单人员"分支的落地。 - 未解决循环：用户在「待反馈确认」环节确认问题未解决时，工单回退**「待处理」**，**未解决次数+1**。同一工单未解决次数超过 3 次 → 升级至「产品负责人」复盘。 - 关闭后可**重新打开** ↔ **「待处理」**（双向路径），适用于"关了又出问题"的场景。

### 4.3.2 工单分流规则（判断节点的四→五向决策）

【服务工单】的"「判断」"节点（黄色菱形，判断类型 / 优先级 / 派单）是分流中枢，决定工单去向。

**分流依据来自 4.3.2.1 的智能分类问卷**——创建工单时由提交人填写 4 维枚举问卷，系统自动推导严重程度；接单时由「服务台人员」或「产品负责人」结合问卷结果判断工单类型并执行对应分流。

| 判断结果 | 去向 | 触发条件与说明 |
| --- | --- | --- |
| **转缺陷故障** | → 故障缺陷·待确认 | 反映已有功能异常（如"导出按钮点了没反应"、"系统报错"）；问卷表现维度选红/橙 |
| **转需求建议** | → 用户需求·创建（入【需求池】） | 反映产品能力缺失（如"希望导出表格"、"能不能加个筛选"）；问卷表现维度选绿/蓝 |
| **派单（技术支持）** | → 待处理（需定义优先级 + 负责人） | 需要技术人员介入排查/配置/修复但不构成缺陷（如环境问题、账号权限）；按优先级四象限排序 |
| **派单（问题咨询）** | → 待处理（需定义优先级 + 负责人） | 纯操作/流程咨询（如"怎么导出数据"、"这个字段什么意思"）；可由非研发角色处理 |
| **直接关闭** | → 关闭 | 无效/重复/已解决/超出服务范围；由服务台人员在判断节点直接闭环 |

> 注：原"派单（确认处理）"拆分为「技术支持」和「问题咨询」两行——两者都进入待处理，但处理人角色不同（前者通常为研发，后者可为产品/运营/服务台）。

**分流原则：** 工单是"入口"不是"终点"。大多数工单最终会转化为需求或缺陷（通过问卷自动推导方向），少数纯服务动作（技术支持、咨询）在原地闭环并定义优先级。把工单正确分流，是避免"工单越积越多、没人知道在干嘛"的关键。

#### 4.3.2.1 工单智能分类问卷（创建时自动触发）

工单**创建阶段**即由提交人填写以下 4 维枚举问卷。问卷有两个作用：

1. **自动推导严重程度**（致命 / 严重 / 一般 / 轻微）→ 驱动处理 SLA 时长和升级策略
2. **为「判断」节点的类型判定提供结构化依据** → 降低人肉判断偏差

> **问卷设计原则：** 1 个工单只解决 1 个问题，多个问题请分开提单；先说明现状，再描述期望。

### 问卷四维定义

| # | 问题 | 选项 A（红/致命级） | 选项 B（橙/严重级） | 选项 C（蓝/一般级） | 选项 D（绿/轻微级） |
|---|---|---|---|---|---|
| ① **影响范围** | 问题影响范围有多大? | 全公司核心业务 | 整个部门 / 多人遇到 | 小团队 / 少数人遇到 | 仅个人遇到 |
| ② **业务影响** | 是否影响正常业务工作? | 业务彻底中断，完全无法工作 | 工作效率明显下降，无法顺畅操作 | 有不便，但可以绕开凑合使用 | 完全不影响，可正常工作 |
| ③ **风险维度** | 是否存在数据、资金、合规、对外风险? | 涉及结算金额 / 核心数据丢失泄露 / 对外客户投诉 / 合规违规高风险 | 部门数据错乱 / 权限异常 / 统计错误 | 仅个人账号 / 数据轻微异常 | 无任何风险，仅界面 / 体验问题 |
| ④ **实际表现** | 问题实际表现是什么? | 系统崩溃、直接报错、功能完全不可用 | 频繁出错，严重影响日常使用 | 偶尔报错，不影响核心使用 | 界面瑕疵、文案错误、优化建议 |

> 四维均采用**枚举单选**（非打分），降低提交人的认知负担。颜色标注仅用于后台严重度计算逻辑，前端可隐藏。

### 严重程度自动推导规则

系统根据 4 维选择自动计算严重程度：

| 严重等级 | 推导条件 | SLA 影响 |
| --- | --- | --- |
| **致命** | 任一维度选 **A（红档）** | 处理 SLA ≤ 2h；立即升级产品负责人+研发负责人双通知 |
| **严重** | 无 A，但 ≥ 2 个维度选 **B（橙档）** | 处理 SLA ≤ 4h；优先派资深人员 |
| **一般** | 其余情况（以 B/C 混合为主） | 处理 SLA ≤ 24h（默认） |
| **轻微** | 全部选 **D（绿档）** 或 3 个以上 D | 处理 SLA ≤ 48h；可排队处理 |

> 严重度排序：致命(4) > 严重(3) > 一般(2) > 轻微(1)

> **注意：** 严重程度是**工单的属性**，不是「故障缺陷」的属性。缺陷有自己独立的缺陷严重度（见第三章 3.4），两者不混用。工单转缺陷时，工单严重度作为参考传入缺陷创建表单，但「产品负责人」可在缺陷侧重新评定。

#### 4.3.2.2 处理 SLA 分级时效矩阵（按严重程度）

上述各 SLA 以**严重程度**为基准分档，具体时效阈值如下（默认推荐值，可按团队实际承载能力校准）：

| SLA 名称 | 致命 | 严重 | 一般 | 轻微 |
| --- | --- | --- | --- | --- |
| **接单响应时效（创建 → 待接单）** | ≤ 1h | ≤ 4h | ≤ 8h | ≤ 24h |
| **派单时效（判断节点 → 待处理）** | ≤ 4h | ≤ 8h | ≤ 24h | ≤ 72h |
| **处理时效（待处理 → 解决）** | ≤ 2h | ≤ 4h | ≤ 24h | ≤ 48h |
| **反馈时效（处理完成 → 反馈确认）** | ≤ 1 天 | ≤ 3 天 | ≤ 5 天 | ≤ 7 天 |

- 表中时效为**默认值**，推行者应在首次部署前根据团队实际承载能力调整并固化到工具中；调整原则：先紧后松——初期用较紧的阈值暴露问题，跑通 2~3 个迭代后再根据数据放宽或收紧。

### 工单类型判断矩阵（接单时执行）

「服务台人员」接单后，结合问卷结果 + 工单标题/描述，判断工单类型：

| 工单类型 | 判断依据 | 后续动作 | 负责角色 |
| --- | --- | --- | --- |
| **缺陷故障** | 表现维度选 A 或 B + 反映已有功能异常 | 转故障缺陷·待确认（跨模型联动） | 研发人员 |
| **需求建议** | 表现维度选 D 或 C + 反映能力缺失/新功能诉求 | 转用户需求·创建入【需求池】 | 产品负责人后续接管 |
| **技术支持** | 需技术人员介入但不构成缺陷（环境/配置/权限/部署） | 派单→待处理，定义**优先级（四象限）+ 被指派人** | 研发/运维 |
| **问题咨询** | 纯操作/流程/使用方法咨询 | 派单→待处理，定义**优先级+被指派人** | 产品/运营/服务台均可 |
| **直接关闭** | 无效/重复/已解决/超出服务范围/信息不全无法跟进 | 直接→关闭 | 服务台人员 |

**技术支持与问题咨询必须定义优先级和负责人**——否则会重蹈"接了没人管"的覆辙。优先级沿用 GCR 四象限模型（重要程度 × 紧急程度），由「服务台人员」或「产品负责人」在派单时评定。

### 分流路径汇总图

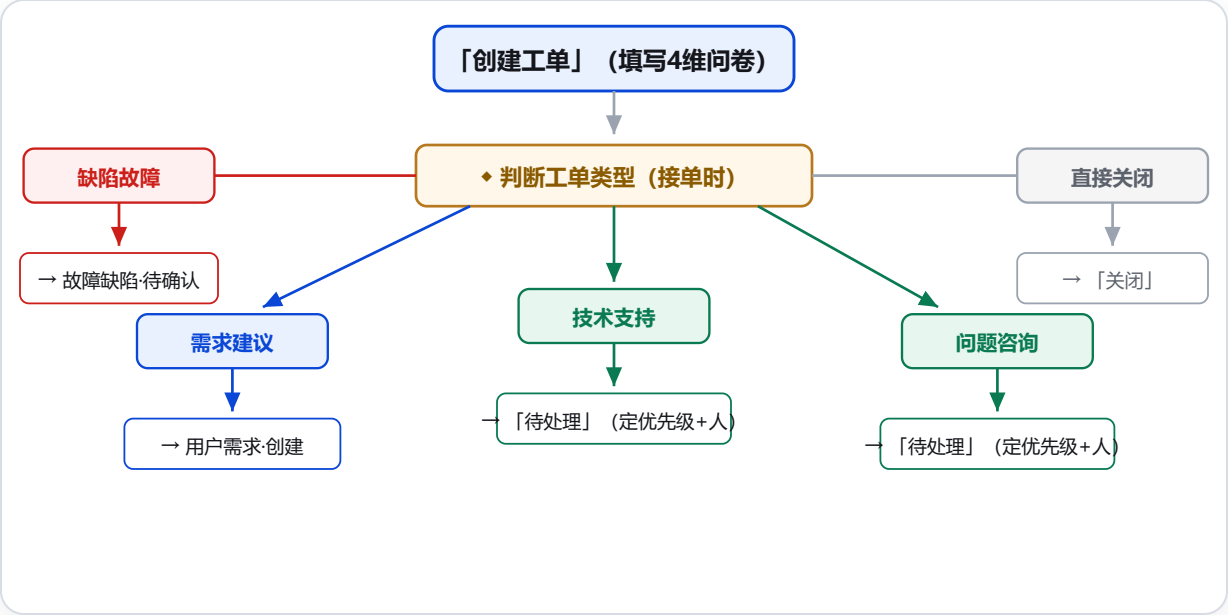



**图 4.1　工单分流路径——五向决策**

### 与原有状态机的衔接

上述问卷和分流机制嵌入现有状态机的两个节点：

| 状态机节点 | 新增机制 |
| --- | --- |
| **创建** | 提交人填写 4 维问卷 → 系统自动推导严重程度（写入工单字段） |
| **判断（◆黄菱）** | 服务台人员结合问卷结果判断工单类型 → 执行五向分流 |

严重程度字段贯穿工单全生命周期，影响：

- **处理 SLA** 的时长阈值（致命 2h / 严重 4h / 一般 24h / 轻微 48h）
- **效能度量** 中按严重度分布统计（第五章）
- **升级通知** 的触发条件（致命级双通知）

### 4.3.3 【跨模型联动】汇总（工单）

| 触发源 | 动作 | 目标 | 类型 |
| --- | --- | --- | --- |
| **服务工单•判断（转缺陷）** | 转 | → 故障缺陷·创建 | 自动化 |
| **服务工单•判断（转需求 / 源单 SLA 超时）** | 转 | → 用户需求·创建 | 自动化 |
| **用户需求•某节点** | 触发 | → 服务工单·创建 | 自动化 |

## 4.4 判断口径：工单现场的松与严

| 现场情形 | 判断口径 | 往哪退 |
| --- | --- | --- |
| **工单创建超过 1 小时没人接** | 响应 SLA 超时 →研发效能负责人强行分配接单人员，不纠结"谁该接" | SLA 兜底机制 |
| **派单超 30 分钟没派出去** | 升级产品负责人介入，别让工单卡在判断节点 | 派单 SLA |
| **用户说没解决，处理人坚持已解决** | 以待反馈确认环节的用户确认为准；回退待处理、未解决次数 +1；超 3 次升级复盘 | 未解决计数器 |
| **反馈 SLA 超时自动关闭，但用户其实没满意** | 系统兜底是防止工单腐烂的底线，不是免责金牌；若用户后续重开，重新走流程 | 关闭↔重开双向路径 |
| **工单该转需求还是转缺陷拿不准** | 判断标准：反映"能力缺失"→转需求；反映"已有功能异常"→转缺陷。拿不准时让产品负责人在判断节点拍板 | 五向分流 |
| **无效 / 重复工单** | 不在判断节点驳回，在待处理阶段由服务台人员主动关闭 | 关闭路径 |

**推行者心法：** 工单治理的目标不是"关得快"，是"分得对、接得住、可追溯"。SLA 是兜底绳，不是鞭子——用它防止工单腐烂，别用它逼团队造假数据。

## 4.5 检查清单

（服务工单的全部工具配置，见配套《GCR数字化落地指南》第四章。）

☐ 是否已建立工单台账，并定义"「判断」（类型/优先级/派单）"节点的**五向分流**（转缺陷故障 / 转需求建议 / 派单技术支持 / 派单问题咨询 / 直接关闭）？

☐ **工单智能分类问卷（4 维枚举）**是否已配置？（影响范围 / 业务影响 / 风险维度 / 实际表现）

☐ 严重程度自动推导规则（致命/严重/一般/轻微）是否已落地？处理 SLA 是否按严重度分级？

☐ 技术支持/问题咨询类工单的**优先级定义（四象限）+ 负责人指派**机制是否已明确？

☐ 工单转需求 / 转缺陷时，是否能建立对应记录并关联？

☐ 「待处理」→「待反馈确认」→「回访解决问题」→「关闭」（否→回退待处理）的流转是否已记录？

☐ 关闭后的"重新打开"路径（「关闭」↔「待处理」）是否已支持并记录？

☐ 各流转的 **SLA 时限**（响应 / 派单 / 处理 / 反馈）是否已定义？（见 4.3.1.1）

☐ 响应 SLA 超时的"强行分配接单人员"机制是否已明确责任人？

☐ "未解决次数+1"计数器与升级阈值（建议 3 次）是否已配置？

☐ 反馈 SLA 超时自动关闭（系统兜底）逻辑是否已在工具中实现？

# 05 效能度量与验收评分

> 治理的尽头是度量。没有度量，团队不知道自己运转得好不好；没有闭环，度量只是数字。本章用"【验收评分矩阵】"把前四章汇聚，用 18 项指标验证团队健康度（结构见白皮书 16.2，v1.2 重定义）。

## 5.1 为什么：没有度量，前面所有治理都是"感觉良好"

推行者最容易在最后一关松劲——流程跑起来了，就觉得"治好了"。但"觉得"不可持续：

- 没有度量，你无法向老板证明治理值不值；
- 没有度量，你无法发现"哪条线在悄悄腐烂"；
- 没有度量，阈值（28 天、T=120）永远停在"经验初定"，无法校准。

【GCR】 的度量哲学是：**决策增强，非决策替代。** 指标是给推行者和管理者看的"治理仪表盘"，不是给个人打分的通缉令。数据服务秩序，而非绑架效率。

> ⚠️ **使用红线：** 度量用于体系优化，不用于个人奖惩；用于透明共识，不用于问责追责；用于治理校准，不用于绩效排名。所有指标的核心价值是发现治理短板、优化流程规则，而非单纯追求数字达标。

## 5.2 怎么推：让度量成为"下一轮治理的起点"

**难点一：团队怕被数字考核，抵制采集。**

→ 推行者第一句话必须是"这些指标不评人、只评流程"。把采集与绩效解绑，团队才肯让数据流动。

**难点二：指标太多，不知道看哪个。**

→ 先盯三个"腐烂信号"：需求漏斗通过率（<30% 说明画像没拦住噪音）、迭代交付率（<90% 说明盒子没守住）、致命缺陷数（>0 说明质量门禁失守）。这三个红

了，其余再看。

**难点三：老板要"提升 X%"的漂亮结论。**

→ 诚实定位：当前指标为**经验初定、待校准**基线，是可行性观察、非统计结论。**绝不写"提升 X%"等伪精确**——样本期短，说这种话反而削弱权威。

**难点四：复盘会变成批斗会。**

→ 复盘只问三件事：**什么做得好（保留）、什么做得差（根因）、下期改什么（动作）。** 每次功能交付会议后 30 分钟，对照 18 项指标看本期变化。

## 5.3 规则参考：评分矩阵、18 项指标与采集

### 5.3.1 【验收评分矩阵】（三模型汇聚点）

图 5.1 橙色区域标注的"【验收评分矩阵】"是【GCR】 的汇聚节点——【用户需求】、【交付功能】、【故障缺陷】三条生命周期的结果在此汇总，形成对团队交付质量的综合评分。

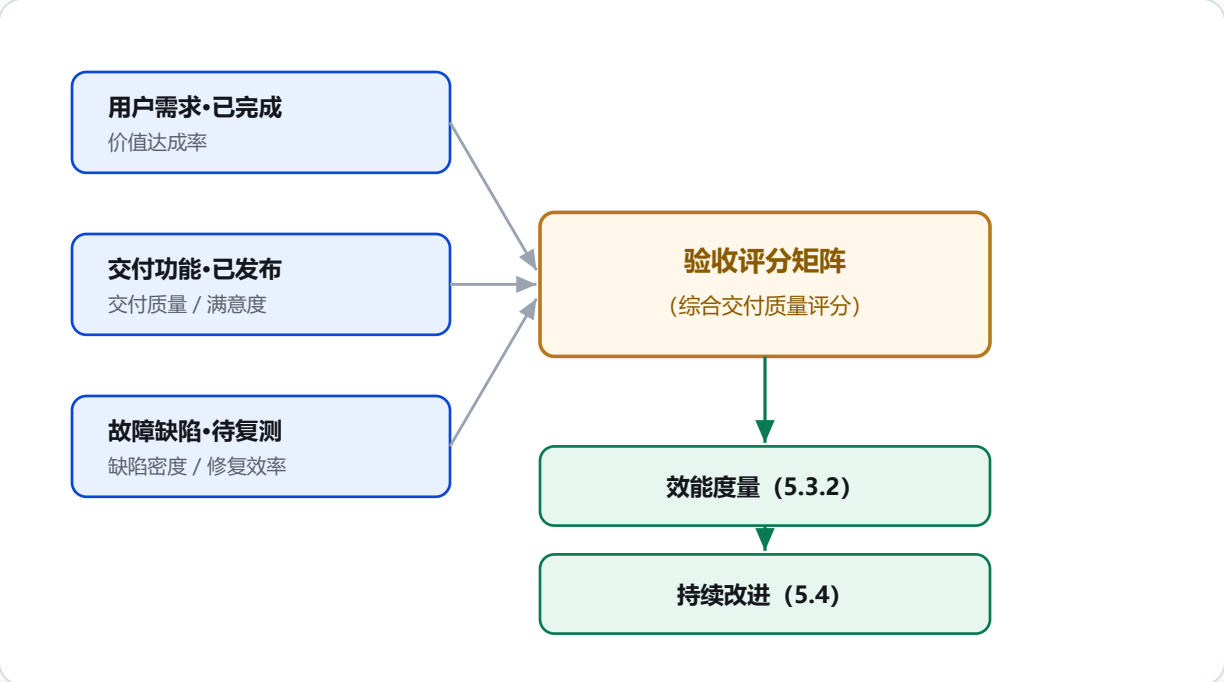



**图 5.1　【验收评分矩阵】——三模型结果汇聚点**

**矩阵的三个输入维度**

| 输入源 | 贡献的评分维度 | 说明 |
| --- | --- | --- |
| **用户需求** | 价值达成率 | 期望上线时间内上线的比例 |
| **交付功能** | 交付质量、需求方满意度 | 验收通过率、需求方打分 |
| **故障缺陷** | 缺陷密度、修复效率 | 千行缺陷率、平均修复时长 |

**评分矩阵的作用：** 它把"做了什么"（需求/功能/缺陷）翻译成"做得怎么样"（质量/满意度/效率），是团队向需求方、向老板、向自己证明价值的最终凭据。

### 5.3.2 效能健康度指标（18 项，结构见白皮书 16.2）

【GCR】 用 18 项指标刻画团队健康度，分五个维度，均设有理想值作为治理校准基准（非考核红线）。**指标严格按归属实体归类——用户需求类与交付功能类不得混填同一列**（迭代装载交付功能，用户需求不进迭代）：

**A. 【用户需求】流转健康度（价值源头）**

1. M1 需求交付周期(中位)（录入→已上线，取中位抗卡滞）— 理想值：越低越好
2. M2 外部依赖等待占比（等待业务方评审/验收时长占比，外部归因）— 理想值：越低越好
3. M3 需求评审通过率 — 理想值：不低于 90%
4. M4 需求按期交付率（规划窗口内验收通过比例）— 理想值：不低于 85%
5. M5 用户需求变更率（需求定义层变更占比）— 理想值：越低越好（建议 ≤20%，待校准）
6. M6 插单率（违规新增紧急/临时需求占比，非次数）— 理想值：不超过 5%（建议值，待校准）

**B. 迭代节奏健康度（交付节拍，主体 = 交付功能）**

1. M7 迭代交付率（迭代内完成交付功能占规划比例）— 理想值：不低于 90%
2. M8 准时交付率（迭代结束前完成交付功能比例）— 理想值：不低于 85%

**C. 交付功能波动健康度（变更管控，主体 = 交付功能）**

1. M9 交付功能变更率（迭代内变更功能占比，与 M5 分两层）— 理想值：不超过 15%
2. M10 风险评估偏差率（评估功能点与实际交付功能点偏差比例）— 理想值：不超过 10%

**D. 交付质量健康度（质量底线，主体 = 故障缺陷）**

1. M11 线上致命缺陷数（上线后致命级缺陷数）— 理想值：0
2. M12 严重缺陷管控率（缓冲周期内修复并复测通过比例）— 理想值：不低于 98%
3. M13 测试回归通过率（缺陷修复后回归验证通过比例）— 理想值：不低于 95%
4. M14 测试覆盖率（已覆盖测试点 ÷ 应测测试点，测试点级）— 理想值：不低于 85%
5. M15 缺陷逃逸率（发布后发现缺陷占比）— 理想值：不超过 1.5%

**E. 服务运维健康度（末端服务，主体 = 服务工单）**

1. M16 工单响应时长(中位)（创建→接单）— 理想值：不超过 1 小时
2. M17 工单处理时长(中位)（接单→待反馈）— 理想值：不超过 24 小时
3. M18 工单关闭时长(中位)（创建→关闭全流程）— 理想值：不超过 48 小时

> 原 v1.1 的「需求确认/排期/验收/上线平均时长」「需求变更平均次数」「拒绝验收平均次数」「迭代内需求变更次数」「平均反馈时长」等已按 v1.2 重定义合并或下沉：时长类统一改中位、变更分用户需求层与交付功能层两层、补测试覆盖率与缺陷逃逸率。详参见《效能度量指标科学重定义 v1.2》。 **度量使用原则（白皮书 16.3）：** 用于体系优化，不用于个人奖惩；用于透明共识，不用于问责追责；用于治理校准，不用于绩效排名。数据服务秩序，而非绑架效率。所有指标的核心价值是发现治理短板、优化流程规则，而非单纯追求数字达标。

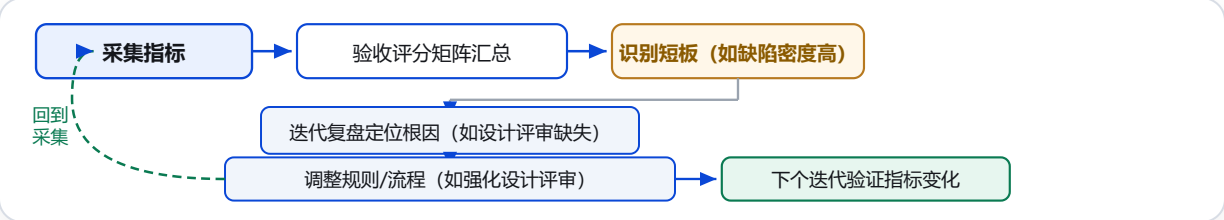
### 5.3.3 度量数据采集

| 指标 | 数据来源 | 采集方式 |
| --- | --- | --- |
| **需求漏斗/池健康/过期** | 需求台账 | 定期统计（数字化后仪表盘自动） |
| **按期发布/交付率/波动率** | 交付功能+迭代台账 | 状态流转记录（数字化后自动累计） |
| **验收通过率/缺陷指标** | 缺陷+用例台账 | 状态流转累计 |
| **工单转化/满意度** | 工单+评分矩阵 | 流转动作记录 |

**原则：指标应尽量基于状态流转记录采集，减少人工填报。** 无数字化时由专人定期汇总流转记录；数字化后由规则引擎自动计算（见《GCR数字化落地指南》第五章）。

### 5.3.4 持续改进闭环

度量不是终点，是下一轮治理的起点：




**复盘节奏：** 每个功能交付会议后进行 30 分钟复盘，对照 18 项指标看本期变化。复盘只问三件事：**什么做得好（保留）、什么做得差（根因）、下期改什么（动作）。**

## 5.4 判断口径：指标的松与严

| 现场情形 | 判断口径 | 往哪退 |
| --- | --- | --- |
| **某指标长期不达标（如需求评审通过率 70%）** | 指标是"经验初定、待校准"基线，先看是流程真有问题还是阈值不合理；复盘定位根因再调规则 | 持续改进闭环 |
| **老板要求"指标提升 30%"的结论** | 诚实拒绝伪精确：样本期短，只能给可行性观察。写"提升X%"会削弱权威 | 度量使用原则 |
| **有人拿指标问责个人** | 立即纠正：指标用于体系优化，不用于个人奖惩 | 度量使用红线 |
| **致命缺陷数 > 0** | 零容忍硬线，不是"待优化项"；立即查质量门禁哪里失守 | 质量底线 |
| **迭代交付率波动大** | 先看是否盒子被口头打破（插单未走流程）；波动是治理失效的先行信号 | 节流阀铁律 |

> **推行者心法：** 度量是你推行工作的"仪表盘"，不是"成绩单"。它告诉你哪条线在腐烂、哪个阈值该调，而不是用来证明你多厉害。把指标当武器对准流程，别对准人。

## 5.5 检查清单

（【效能度量】的全部工具配置，见配套《GCR数字化落地指南》第五章。）

- ☐ 是否已建立【验收评分矩阵】，汇总【用户需求】/【交付功能】/【故障缺陷】三源数据?
- ☐ 18 项效能指标是否有人定期统计（无论自动采集还是人工汇总）？
- ☐ 指标采集是否尽量基于状态流转记录，减少人工填报?
- ☐ 是否建立迭代复盘机制（每迭代一次）？
- ☐ 复盘结论是否落入下期改进动作并跟踪?

- [ ] 是否明确"指标不用于个人奖惩"的使用红线并已向全员宣贯？

# 附录

## A. 推行落地总检表（按推行顺序）

这本手册的章节顺序就是推行顺序。以下是一张"推行者飞行清单"——照着打勾，不要跳步：

| 顺序 | 阶段 | 关键交付物 | 完成标志 |
| --- | --- | --- | --- |
| 1 | 团队职责规范（第一章） | 至少一支业务线跨职能团队就位；研发效能负责人任命；红线宣贯签署 | 四条角色红线 + 三条组织红线全员签署；共享资源纳入迭代周期 |
| 2 | 需求治理（第二章） | 需求台账 + 唯一入口落地；需求池机制运行；插单/变更流程跑通 | 需求方已知"需求只找产品负责人"；漏斗通过率有人统计 |
| 3 | 迭代与交付（第三章） | 迭代周期运行（默认 2 周）；两层粒度拆解落地；质量门禁生效 | 致命缺陷清零守得住；迭代交付率 ≥90% |
| 4 | 服务工单治理（第四章） | 工单台账 + 五向分流 + SLA 兜底 | 响应 SLA 超时有人强行接单；转需求/转缺陷可追溯 |
| 5 | 效能度量（第五章） | 验收评分矩阵+ 18 项指标定期统计 | 每迭代有 30 分钟复盘；指标用于校准流程而非考核人 |

> 每一阶段未"完成标志"达标前，不急于进下一阶段。顺序乱了，每一步都在补前一步的坑。

## B. 名词对照表

| 名词 | 定义 | 出处 |
| --- | --- | --- |
| **GCR** | 研发全域闭环治理方法论，覆盖需求 / 迭代 / 交付 / 缺陷 / 工单五大生命周期 | 全手册 |
| **GRT** | 双维全域治理节奏体系（统一时间轴 × 资源透明） | 全手册 |
| **用户需求** | 需求方提出的、偏价值的诉求，可能横跨多个迭代（大粒度） | 第二章 |
| **交付功能** | 用户需求拆解后、装入迭代执行的单元（中粒度） | 第三章 |
| **执行用例** | 针对单个功能点的最小测试单元（通过/失败） | 第三章（3.3.8） |
| **故障缺陷** | 验收或运行中发现的异常，需修复闭环 | 第三章 |
| **服务工单** | 一线用户 / 需求方提出的服务请求 | 第四章 |
| **产品负责人** | 需求的唯一入口与需求池负责人 | 第一章 |
| **研发效能负责人** | 跨业务线治理者，负责模式切换、违规巡检、效能度量 | 第一章 |
| **需求绕口** | 需求方绕过产品负责人直接找开发的行为，GCR 红线禁止 | 第二章 |
| **迭代周期** | 固定周期（默认 2 周）的时间容器，盒内装交付功能 | 第三章 |
| **验收评分矩阵** | 汇总需求 / 功能 / 缺陷三源数据，验证价值兑现的评分机制 | 第五章 |
| **闭环** | 每个生命周期从进入到关闭 / 上线全程可追溯、可度量 | 全手册 |
| **节流阀** | 需求侧多线并行无瓶颈、执行侧同期只跑单一迭代的结构性机制 | 第三章 |
| **过期铁律** | 超期望上线时间仍未上线的需求必须关闭或重估 | 第二章 |
| **精益否决权** | 业务价值任一子维度为 0 即整体归零、不进重要序列 | 第二章 |

> 术语卫生硬约束：**「业务方」一词已废弃，全文统一称「需求方」**；闭环仅在开头声明处加【】，正文描述一律不加；专有名词（GCR/GRT/明道云/SAFe 等）保留不译。

## C. 推行阻力 TOP Q&A（汇总）

**Q：老板不认可"流程"，觉得我们小团队不需要。**

→ 先讲适用范围：1-2 人微型团队确实不在GCR 范围内，等角色专业化、需求绕口、进度失焦时再引入。对达标团队，用"节流阀"故事切入：不是要加流程，是让插单有代价、让交付可预期。

**Q：需求方不愿意走流程，觉得慢。**

→ 用"更快的响应"换"唯一的入口"：产品负责人主动上门半小时内给排期结论。开发收到直接需求时转交产品，不拒绝不执行。

**Q：老板随时口头插单，硬拒得罪人。**

→ 不拒绝、不堵嘴、走规则、留证据。填《插单代价声明》让代价透明，决策抄送老板。规则挡不住权威，但让你手里有证据。

**Q：团队怕指标被用来考核人。**

→ 第一句话就解绑：指标不评人、只评流程。把采集与绩效切开，数据才流得动。

**Q：指标不达标，老板要"提升 X%"的结论。**

→ 诚实定位"经验初定、待校准"，样本期短只能给可行性观察，绝不写伪精确。

**Q：服务台说工单太多接不过来。**

→ 先确认「产品负责人」精力 20% 红线：超了必须独立服务台角色；响应 SLA 超时由「研发效能负责人」强行分配接单。

## D. 配套与差异文件

- **GCR数字化落地指南（明道云 HAP）**：本手册所有规则的工具固化方式（角色权限、状态机工作流、指标仪表盘），分章对应。

- **GCR 方法论白皮书**：面向推行层 + 产品负责人的"是什么 / 为什么 / 模型长什么样"，本手册是其操作层落地；白皮书只呈现主链路，分支与异常管控在此。
- **《GRT 双维全域治理体系》白皮书**：GRT 双维机制与四步法则的专门论述，本手册仅引用其作为横贯基座。
- **与禅道 / ONES / SAFe 的差异对照**：见独立文档，本手册不展开。
- **真实案例集**：多家企业脱敏实战案例，持续补充。

**本手册基于GCR 五大模型状态机与数字化联动图编写，面向推行者（GCR 教练 / 顾问 / 研发效能负责人），持续迭代中。**

---

GCR 推行手册 v1.0 · 纯 SOP · 配套《GCR 数字化落地指南》使用效果更佳 · 版权归作者周老师所有