郭又宁 Almee Kuo

从复杂业务中提炼共性，设计可扩展的产品框架

资深产品体验设计师｜系统平台型

企业级　SaaS　AI

简历 https://almee-kuo-sr-designer-resume-2026.figma.site





Hello！我是 又宁
8年 产品体验设计经验

专注复杂业务系统设计，
兼顾操作效率、业务扩展与长期维护

工业设计 → 产品体验设计，让我塑造「**先拆解问题结构，再进入设计**」的设计思维。

主导「任务自动化平台、资源调度扩展平台、多场景服务运营工作台、B 端设计系统治理」等项目经验，形成一套可复用的设计路径：**识别问题本质 → 抽象可复用结构 → 让系统持续演进**

ASUS NIO

复杂 B 端专精 | 0-1 平台搭建 | 设计系统治理与规模化 | 用户研究与画像抽象 | 跨团队机制建设

通过四个项目，展示关键设计判断与落地成果

核心能力 设计决策思路

洞察本质归因

精准定位 流程、信息、系统 与 协作 的瓶颈边界

本质识别

跳出单点诉求，挖掘底层业务矛盾

场景抽象

从高频路径提炼通用业务模型

结构化解法

不局限单点需求修补，建立场景模型信息层级组件能力

平台化框架

统一底层逻辑，减少跨业务重复建设

信息重构

梳理信息优先级，提升关键操作决策效率

长效治理与演进

赋予系统 可扩展、可协作 与 可持续迭代 的生命力

系统思维

贯通前后台逻辑，平衡体验与业务规则

策略落地

制定规范与机制，推动跨团队高效协同

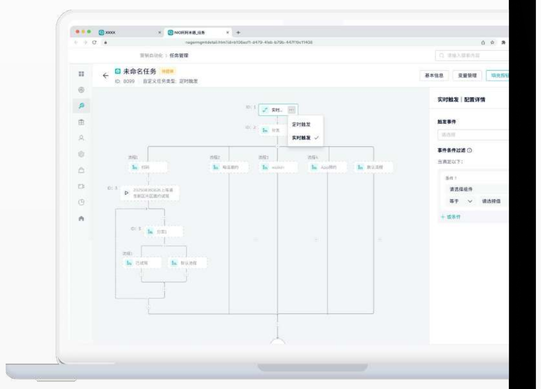
案例成果 应用与实践

1

企业级自动化
工作流搭建体验重构

以「限制自由度」换取「确定性」，
提升配置成功率与效能

主设计师 | 本质识别 场景抽象

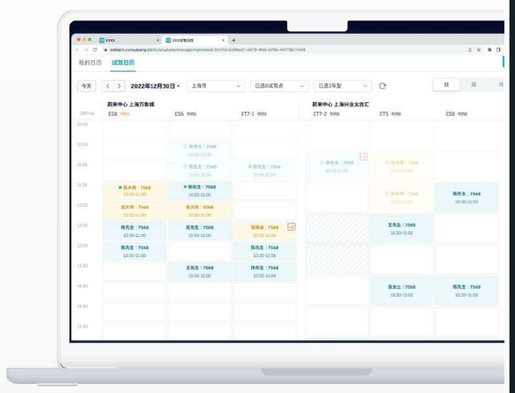


2

资源调度
可扩展框架

将资源调度规则拆解为可配置模块，
支持跨场景复用，不再重复造轮子

设计组长 | 平台化框架 策略落地

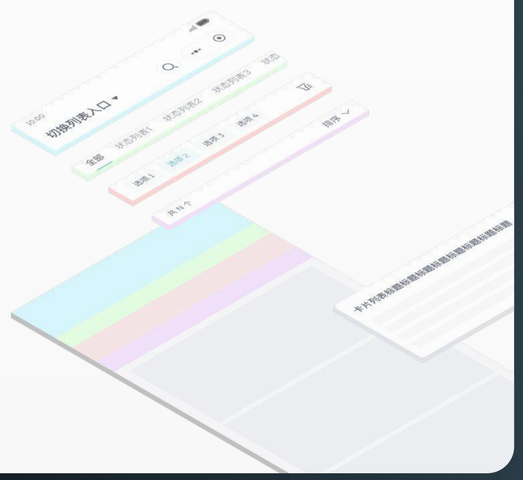


3

B端
移动设计系统治理

3品牌设计规范治理，并建立产设研协作机制，让规范贯穿全链路

设计组长 | 系统思维 策略落地



4

多场景
用户运营工作台

整合分散的用户信息与服务状态，
帮助员工更快判断服务优先级

设计组长 | 场景抽象 信息重构
系统思维



企业级自动化工作流体验重构

以「限制自由度」换取「确定性」，
提升配置成功率与效能

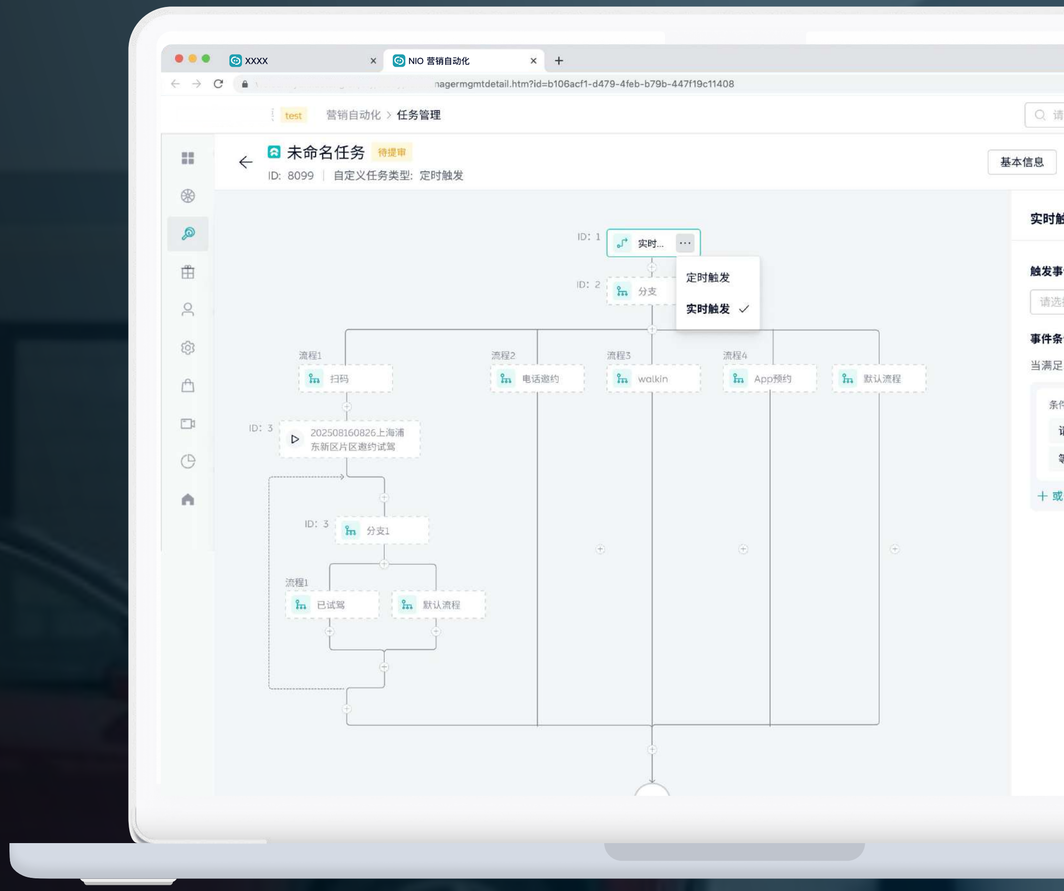
2周 → 3天
上线效率

-30%
新人上手时间

1 → 3个
支援品牌

60%
场景覆盖率

赋能场景	项目时长	主设计师
3品牌App、B端、企微	3次双周迭代	PM*2、RD*5



体验问题：过度自由、高报错率的工作流画布

为什么，不局部优化
Problem Statement

背景
问题

蔚来B端平台承接3个品牌App／Web终端，及企业微信单聊与群聊的营销场景，业务规模每季扩张。

平台越灵活，系统越脆弱

关键
判断

当「用户、产研、协作」三个视角问题，指向同个根源系统 缺乏结构性约束，我知道这 不能仅「局部优化」解决

以下三个视角印证这一判断：

	表象	核心原因
用户	操作依赖经验，效率随业务增长递减	**系统没有给用户「正确路径」的引导**
产研	框架缺乏约束，功能堆砌引发系统脆弱	**开发在没有边界的画布里无法建立结构**
协作	多方需求拉锯，高压复杂风险3重困境	**没有人知道「谁的判断是基准」**

为什么放弃开放式画布，选择限制操作自由度

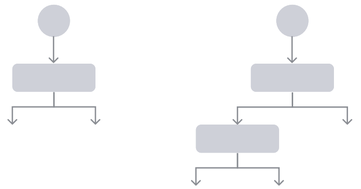
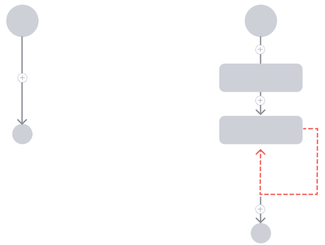
为什么这样判断
Design Trade off & Solution

当平台越灵活，
系统反而越脆弱

维护成本暴增、新人学习成本高、
操作成功率下降、跨团队无法协作

我们评估过保留开放式并优化的方案，但因为
无法拓展「循环式流程能力」，所以放弃了

我选择限制自由
来换取成功率与扩展性

分类	开放式画布	原版	封闭式画布	新版
抽象				
操作方式	重拖拽，手动对齐反覆		**重点击，精准有度**	
流程查看	控件摆放混乱，需「阅读」		**结构清晰，能快速「浏览」**	
开发成本	渲染动效要求高，还原度低		**控件类型明确，还原度高**	
扩展性	控件连接方式单一无法拓展		**连接方式有度易管理**	

我的3个设计决策，让工作流搭建，从「依赖经验者」变成「依赖系统」

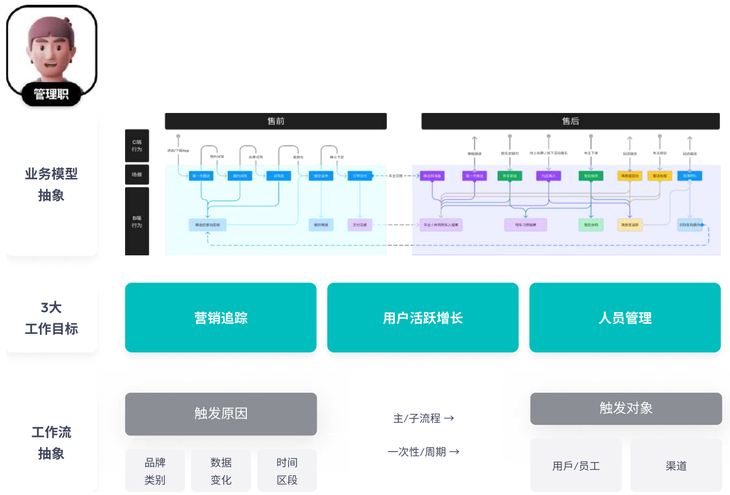
关键决策
Key Process

STEP 1：依赖「工作流 → 系统流」抽象

抽象业务模型，构建触发原因与对象

使用者依赖操作经验，随业务增长效率递减的问题，进行：

- 4大业务线共性流程抽象，提炼3类触发场景
- 隐性经验转化为**可复用的系统自动化流程**
- 据增长后的业态抽象工作流，明确触发原因与对象，并与产品经理、开发团队达成**整体产品交互框架共识**

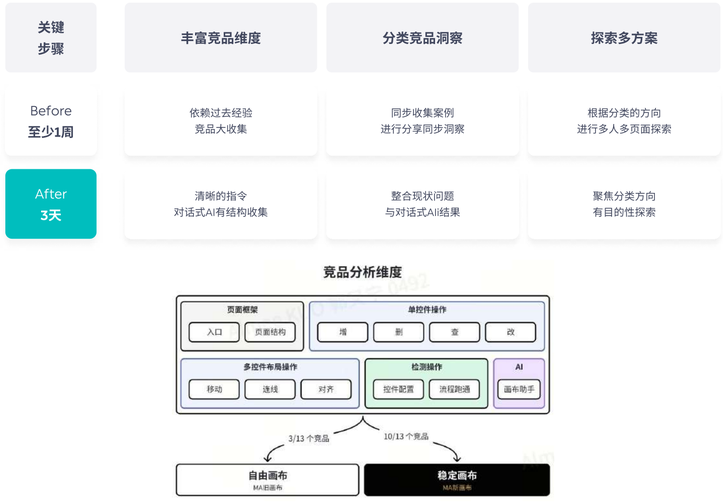


STEP 2：依赖「交互特征」抽象

对话式 AI 分析结果，提升5倍交互决策效率

用对话式AI系统性拆解 13个竞品的画布结构与控件逻辑：

- 压缩原人工分析时间，2周至**3天**
- 人力减少，至少5人分工至**1人作业**
- **操作效率、开发还原度**，封闭式画布 > 开放式画布



STEP 3：依赖「操作控件」抽象

限制操作自由度，提升配置成功率

开放式，无法准确控件连接，仅在执行后识别，导致需反复对齐控件导致操作费时；

封闭式，能确保控件连接，且在**操作即整齐布局**整体流程，借此提高**增删改查的操作效率**（抽象3类控件拓展逻辑：串行、内嵌套、多流程类型控件）



布局框架｜清晰流程顺序，不中断不遮蔽当前所需信息为原则

1 提高画布可视区

行为建立：当前我在配置哪个控件，它处于第几步/当前处在哪个阶段

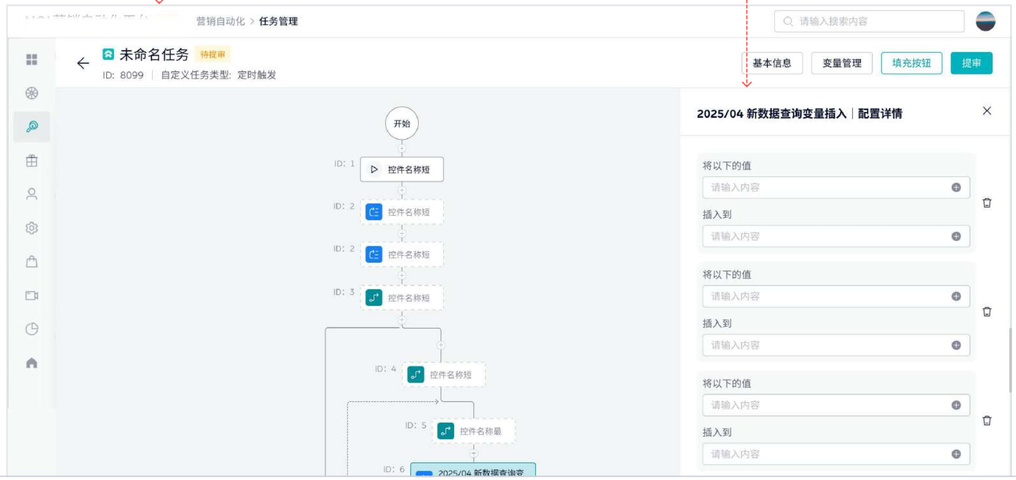
Before



移除固定式控件工具栏位

原来表单宽度收窄至屏幕1/3
并去除不必要的蒙层

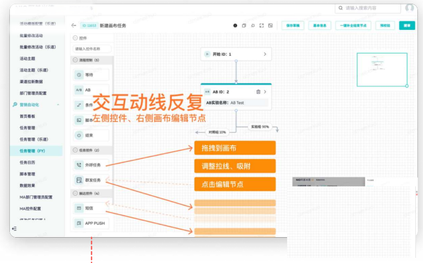
After



1 提高配置操作流畅度

行为建立：明确在哪里，添加什么控件，配置什么控件与数据内容

Before



移除拖拽交互方式，调整为点即添加，并刷新表单

After



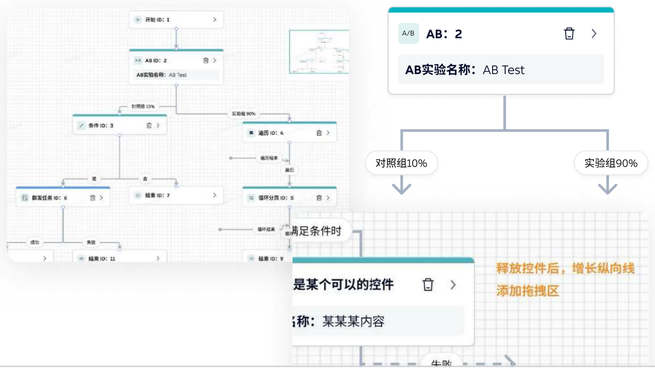
控件模组｜抽象逻辑，拓展有理有据

控件问题

控件互相遮蔽
→ 整体流程不易监测（查）
→ 在原有流程里，不易编辑（增删改）



类型单一
→ 控件间连接适合简易串形流程
→ 对于循环、重复类型控件无法添加





旧控件，需阅读理解

串形控件（旧）

控件上包含成功与失败连接线（每新增一个连接就会产生一个分支，依此类推）

新控件，从「轮廓、添加控件方式」来识别

串形控件（新）

- 添加时按顺序上下连接，不含连接线。
- 整体流程仅一条主线
- 适用于「顺着走」的流程

内嵌套控件

- 添加时不断向内堆叠，包含循环类连接线。
- 保持唯一主流程
- 适用于「重复走」的流程

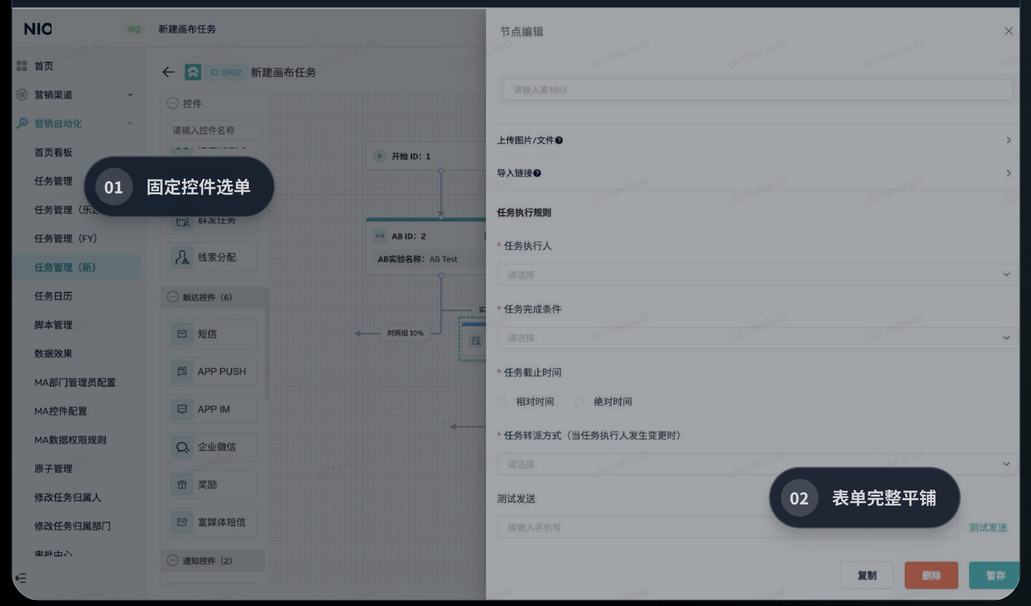
多流程控件

- 添加时不断向旁边扩展，包含分支类连接线。
- 在唯一流程中衍生多个流程
- 适用于「分岔走」的流程

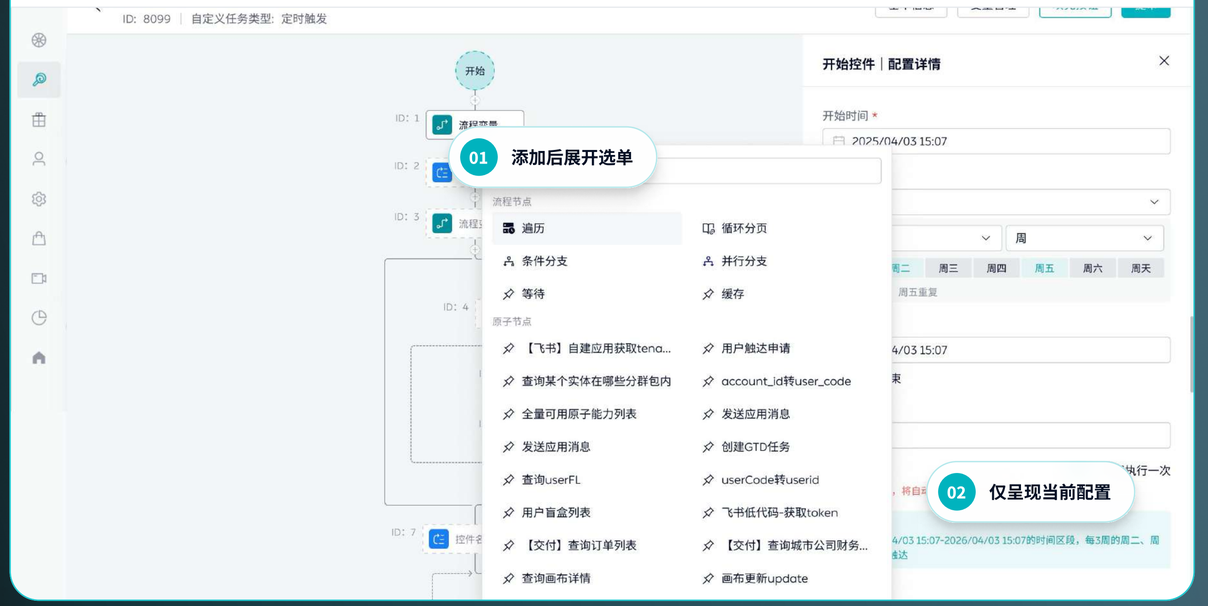
企业级自动化工作流，体验改版对比

效果展示
Project Overview

Before　**单一控件，灵活布局**　操作入口与配置内容同时平铺



After　**3 类控件模块，限制布局**　按需选择控件，再进入对应配置　OPTIMIZED



从固定、平铺的配置模式 → 按需、聚焦的任务编排体验

1　**固定控件选单**
控件常驻左侧，挤压流程画布空间
→ **按需添加控件**
点击添加后展开选单，选择路径更聚焦

2　**配置表单平铺**
完整表单一次展开，信息密度过高
→ **随选中控件展开**
仅显示当前配置，层级清楚、操作连续

企业级自动化
工作流搭建体验重构
影响力

画布优化

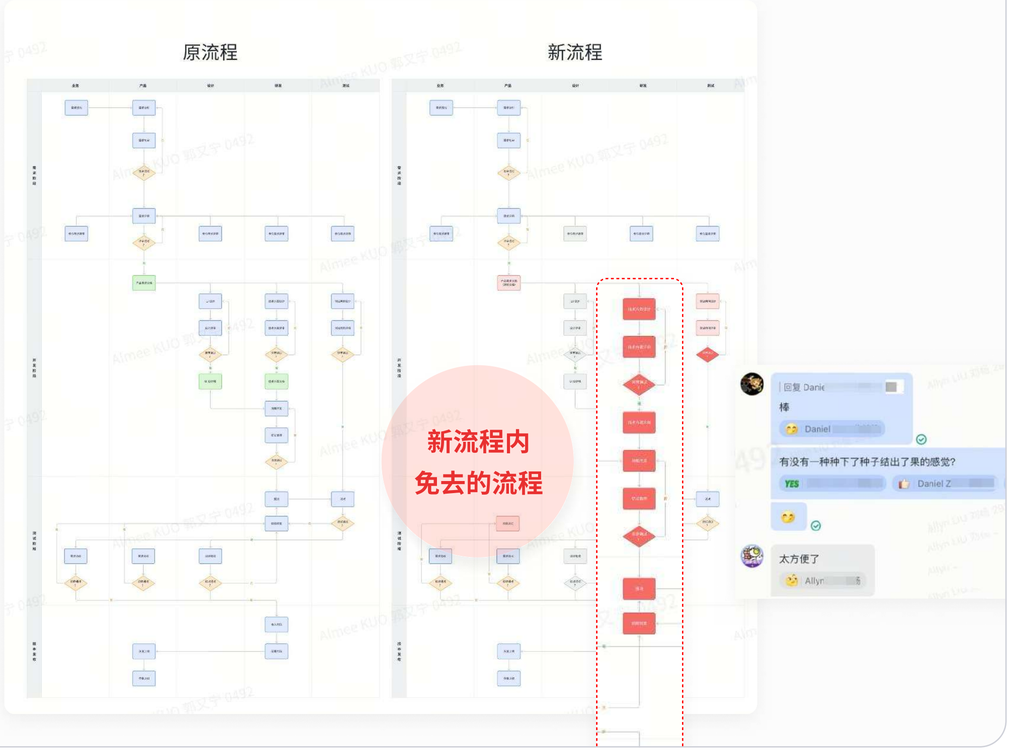
体系化效率
赋能团队
公开分享



2周 → 3天

上线效率

原：一个业务需求，需2个迭代满足
新：一个业务需求，需0.5/1个迭代



✓ **服务业务从 2个 → 5个；1个品牌至3个品牌**

迭代后的操作逻辑、配置方式提升需求满足
新人同类产品上手时间缩短30%

✓ **控件配置化，新业务接入不需重新开发，直接复用框架即扩展**

✓ **画布体验优化**

流程配置过程，单控件减少至少4个步骤，且降低反复调整次数

✓ **3类控件交互拓展逻辑**

产品团队据业务需求，从3类里进行排列组合配置，快速上线

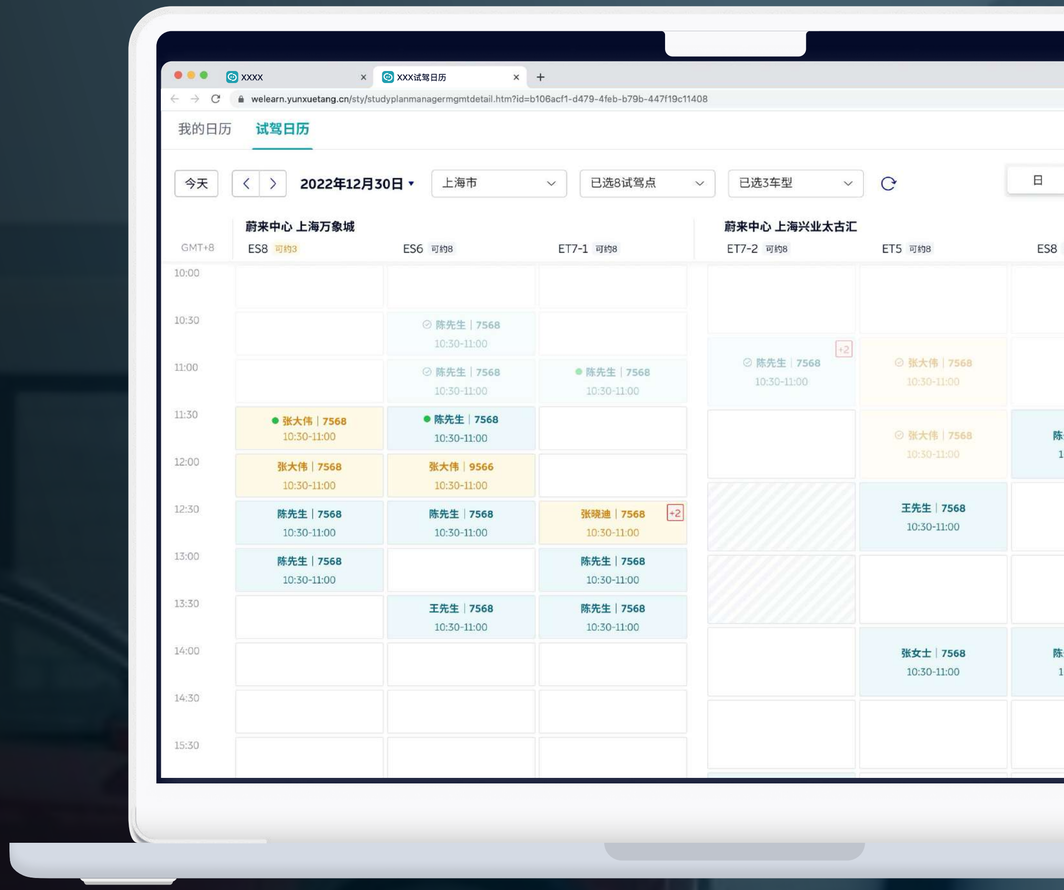
资源调度可扩展平台

将资源调度规则拆解为可配置模块，
支持跨场景复用，不再重复造轮子

85 → 15 天
新场景接入效率

1 → 4 个
支援场景

+50 %
框架组件复用效率



赋能场景
试驾安排、交付交车、
排班、维修维保

项目时长
搭建期8周
导入8次双周迭代

主设计师
PM*1、UX*1、
UI*2、RD*3

「识别业务多场景，重复建设资源损耗」的问题

为什么，值得 0 → 1 自研
Problem Statement

核心
原因

多个B端资源 与 C端服务量不匹配

如果用现成产品：表格/日历（它们解决的是时间管理问题），仅能解决单一业务场景（试驾预约）问题；且来一个场景，造一个。

关键
判断

需自研框架，解决蔚来所有 资源调度场景 的底层逻辑。
关于「解决问题的范围」的设计判断，不仅是技术问题。

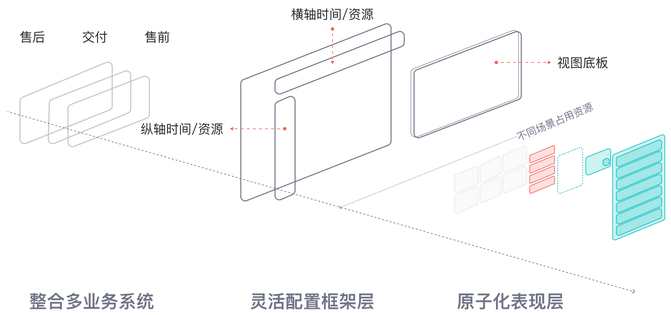
	表象	本质
线上客服	试驾车被重复预约，用户等待时间过长	同资源在同时间被多门店争抢
管理者	排班看不清忙闲，影响人力调度	人力资源可视化与调配没有统一语言
交付专员	交付时间撞单，人员支援不及时	时间、空间、人、车，4个维度没有共同框架

提炼底层资源公式，支撑业务拓展的原子结构

STEP 1：原子抽象

抽象问题，四个场景，一个本质

试驾 · 排班 · 维修 · 交付的底层冲突一致：
资源占用 = 时间 × 空间 × 资源（人 × 物）
这个公式是整个设计框架的起点。



STEP 2：原子堆叠

三层架构设计，支撑四个业务场景

① 整理「车辆、人力、时段」，资源为系统可识别的原子单位

② 灵活配置框架
横轴时间 × 纵轴资源，两轴独立配置

③ 原子化表现
先综述后细节 · 材质颜色区分忙闲、角标显示总量、不堆叠



STEP 3：堆叠规则

主动解构视图逻辑，消除协作摩擦

将 UX 层级拆解为开发可直接对照的组件结构：底板 → 视图类型 → 筛选器 → 单元格 → 事件卡片。
每层交互关系明确定义，还原度问题在沟通前就解决。



我提炼且验证，适用高复杂度B端系统的 4 步骤

先厘清「人与场景」再定义「工具边界」，最后才进入设计。

先定义	识别	建立	最后才是：交互与UI
真正要优化的是什么	哪些是高频关键链路	可扩展但有限度的框架	打磨体验细节 + 呈现可交付的方案

项目拆解
时间规划

从1v1访谈
提炼场景痛点

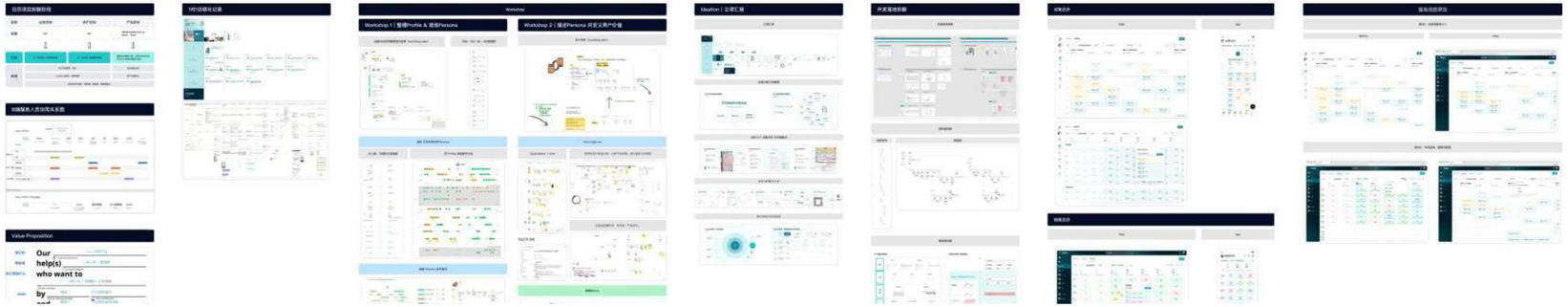
提炼设计原则
激发可能性

汇报立项
产品业务团队

拆解颗粒度
获取开发资源

赋能多场景
维修、试驾、排班、交付

精准效率迭代
场景细化、屏效优化



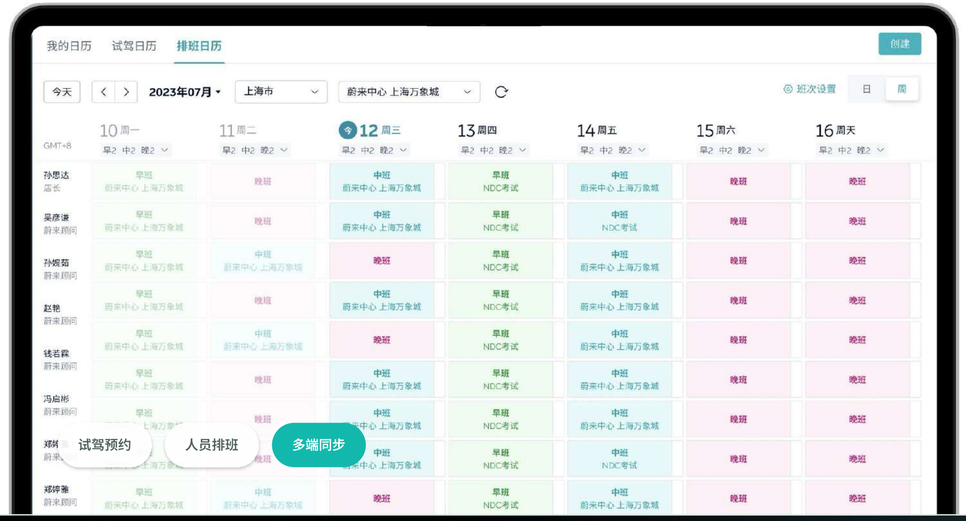
可套用至：0→1 项目，为项目落地指引；1→1+项目，为缺漏补查清单

资源调度可扩展平台，一套能力底座，组合出多种业务场景

效果展示
Project Overview

1 统一框架，覆盖桌面端与移动端

排班管理 × 用户预约 · 同一资源状态实时映射





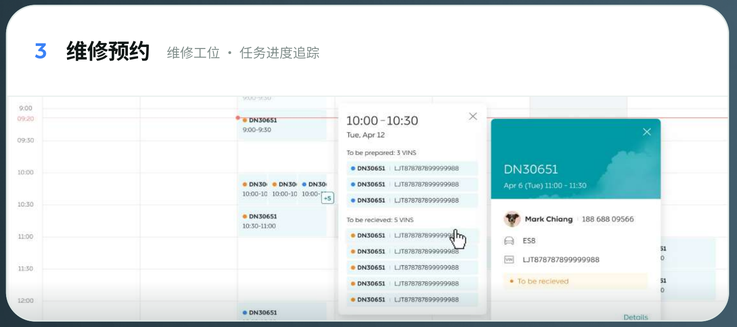
2 交付预约

交车专员 · 车辆状态聚合



3 维修预约

维修工位 · 任务进度追踪



统一能力底座

1 框架 | 2 视图底板 | 3 视图状态 | 4 预约详情 | 5 业务筛选

按需组合，持续扩展

从单点页面到可复用的
多场景产品能力

1 套统一框架
建立一致体验

5 个核心模块
支持自由组合

4 类业务场景
覆盖多端流程

资源调度可扩展平台影响力

框架可复用性被真实业务验证的意义：

85天→15天不只是一个效率数字。

它说明：

框架的抽象层级是对的

足够稳定可以被复用

足够灵活可以被扩展

新场景接入周期从 85 天缩短至 15 天，
验证了框架的复用能力

✓ **日历框架提炼与落地规范**

App/Web端日历规范化，赋能产品经理与开发落地

✓ **4大场景落地** （App/Web端）

试驾日历：提高试驾车使用率
排班日历：效率调配跨门店人员支援
交付日历：调配交付量与调配人员
车模日历：调配单量与技师人员

视图原子化

推进产品化

推进开发

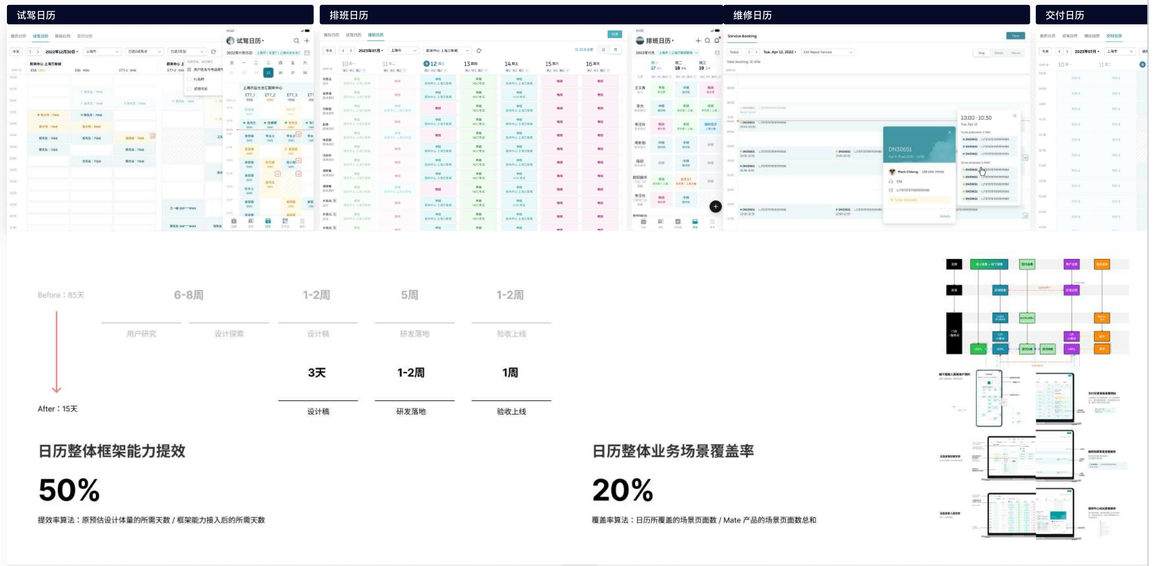
赋能业务



85→15 天

日历框架原子化，
新场景接入从平均85天缩短至15天

从单一试驾场景，扩展至排班、维修、交付，覆盖率提升说明框架的可复用性被验证了。



B端移动设计系统治理

3品牌设计规范治理，并建立产设研协作机制，
让规范贯穿全链路



-30 %

使用规范与模版的需求
其设计开发工时降低

15 套

高频业务场景模版

约 100 人

跨团队业务规模

赋能场景
车企售前至售后
全场景转化

项目时长
6周重构
并导入12次双周迭代

主设计师
UX*1、UI*2
跨团队设计师*3

「组件库完成度 90%，体验仍然割裂」的问题

为什么，设计系统失效
Problem Statement

关键判断

有设计系统，不代表被使用

已有设计系统、组件库、Figma文件等，但还有体验问题；
组件是原子，但业务使用需要的是分子：

缺，指导性内容

核心原因

设计系统提供了原子，却 没有说清怎么组合

以下分析来印证此观点

失效原因	本质
组件存在 ≠ 团队会使用	设计师面对业务场景，不知道该用哪个组件
缺少业务语义	原子组件没有场景化说明，设计师只能靠猜
缺少场景化指引	同一个操作目标，4条业务线各自有一套做法
PM/RD无法理解	规范文件只有设计师看得懂，产研无法自行对齐

为什么不重建组件库，而是在现有系统上补充业务应用层

当时团队

第一反应是要**建立「业务自己的设计系统」**，
试图绕过「建设容易维护难」的窘境

关键判断

重构解决的是系统问题，但 失效的根源是「使用问题」

我决定：在原有系统上叠加「业务应用层」，并推动团队内部与跨团队协同

以下三个视角印证这一判断：

	重构	叠加业务应用层
产研	能解决当前的设计系统过于灵活问题	**在原有基础上，说明到位**
风险	全部推倒重来，周期半年起步 业务线开发需要同步配合迁移，无法正常交付	**不影响现状开发，计划有度的迭代上线**
收益	三个月后，可能会回到原点	**从 MVP 扩大规模，拉通并优化使用方式**

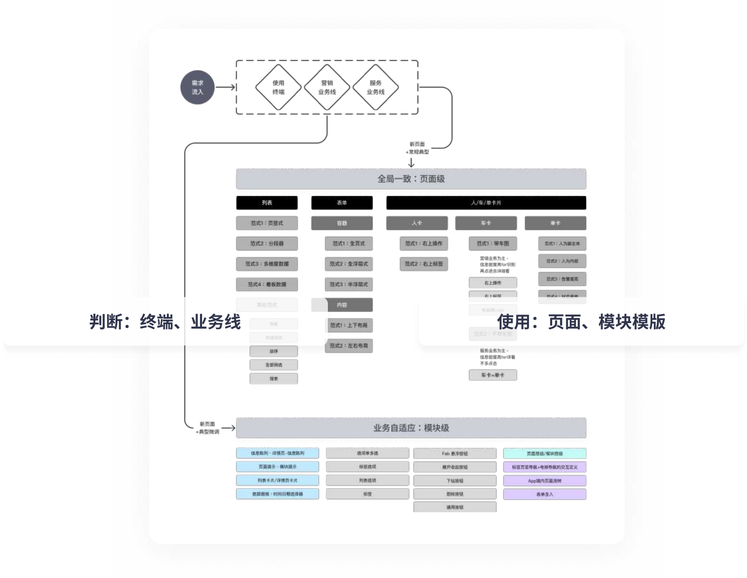
推动约 100 名跨团队成员采用统一规范，重点解决规范使用与协作问题

关键过程
Key Process

STEP 1：建立判断逻辑与机制

避免全面重构，典型页面模块抽象

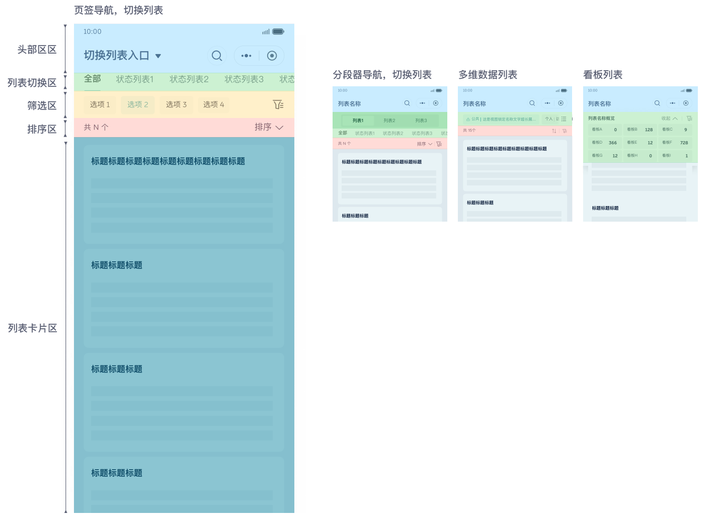
全面对齐成本极高，优先从影响面最大的地方动：高频页面与跨业务线重复出现的标准操作流程。
确立一致性基线后，再按 交互→视觉 的顺序向外扩展，每个阶段都有可验收的交付边界。



STEP 2：建立组合规则

降低导入成本，规范「自带说明书」

加入「模组封装层」后，设计师不需要理解底层逻辑，照路径走就能做出一致结果。每个模组容器定义了完整使用路径：入口 → 接续操作 → 可替换内容 → 回显，解决了原系统「先懂逻辑才能用」的门槛问题。



STEP 3：大规模覆盖实际工作场景

组织 Design All Hands 提高接受度

规范治理的终点是：没有设计师说明下也能运转。通过跨业务线 Design All Hands，用「典型页面 + 常用模组范式」替代抽象规则，通过设计稿（设计师）、文档（非设计师）的方式传递，让规范从被动接收变为主动共识。



B端移动设计系统治理影响力

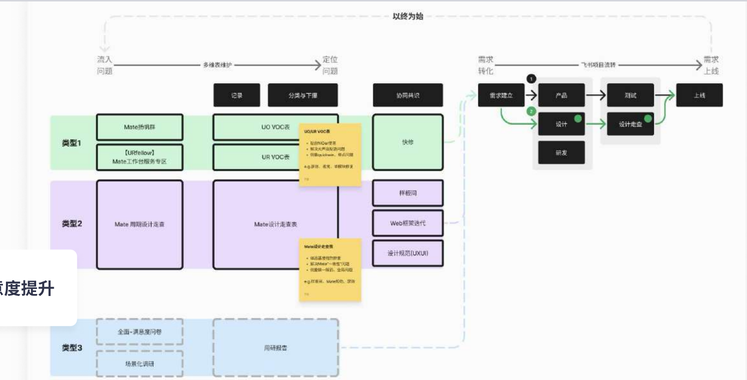
✓ **这个治理框架，3个层面闭环：**
用户体验对齐、产研效能提升、设计资产沉淀

这是设计系统治理真正的价值，不只是规范文件；
最终输出PM / RD可直接执行的多品牌、App、Web规范细则，小型需求设计师只需走查验收，不必全程介入。

用户满意度数据治理

体验闭环

下钻提炼Voc
至逐条体验优化
用户满意度 3.4 → 4.0

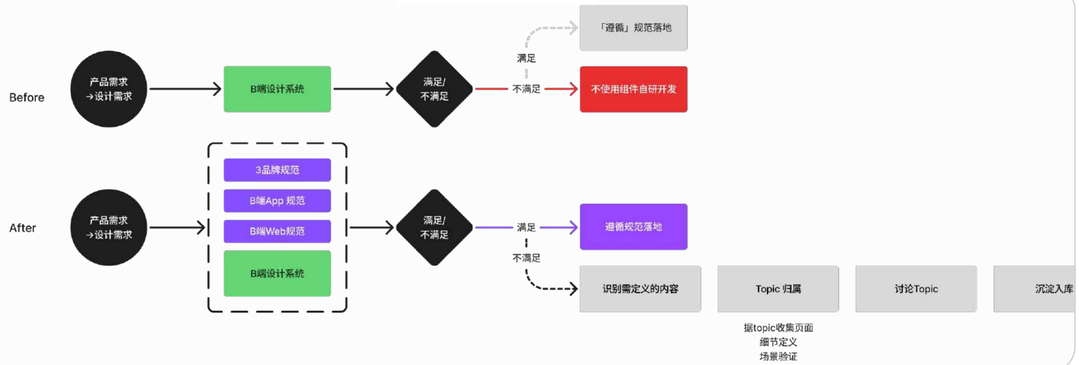


VOC 来源 → 需求转化 →满意度提升

协作工作流治理

效能闭环

落实规范使用
共识组件模块
工时 −30%



设计系统治理

资产闭环

品牌规范应用
App/Web端模块组件拓展
沉淀 15+ 套场景模版



多场景
用户运营工作台

整合分散的用户信息与服务状态，
帮助员工更快判断服务优先级

-60 秒
服务响应时间

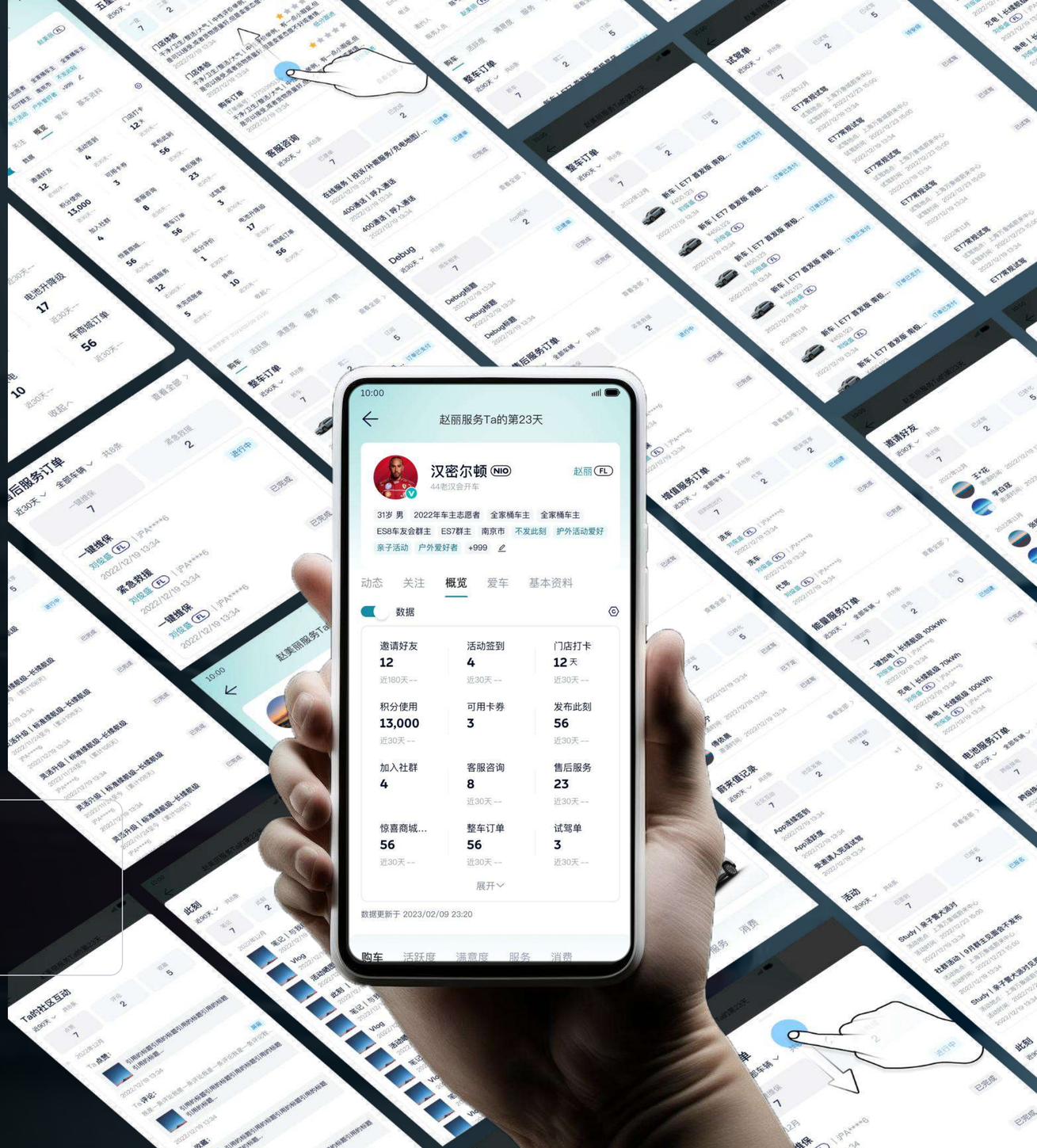
+40 %
数据可视化占比提升

0 天
新数据无需设计即上线

赋能场景
满意度跟进、活跃度运营、售后服务管理

项目时长
6周重构
12次双周迭代

主设计师
PM*1、UX*1、
UI*2、RD*3



为什么不继续增加字段，而是调整信息架构

为什么，高密度服务信息无法快速决策
Problem Statement

背景
问题

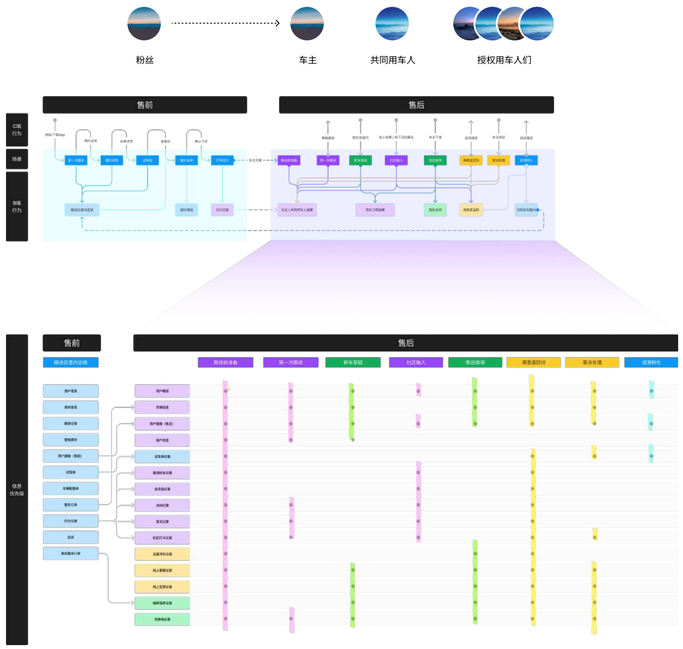
蔚来售后服务管家同时跟进多位车主，信息分散在不同入口，服务动作靠人工记忆和经验判断，效率随用户量增长急速下降。
「平铺结构，在信息增量时，低效查找与响应」的问题

关键
判断

当三个视角问题，都指向： 系统没有帮员工「识别用户」

我知道这不能只靠「加字段」解决

	表象	本质
员工	交接用户不扎实，用户需重复叙述事件	**依赖经验，单靠记忆跟进**
产研	字段平铺，信息拓展性差	**字段平铺结构，无法支撑售后非线性场景**
协作	多人服务同一用户，信息不同步	**没有统一的服务状态语言**



信息降噪与场景化提效：从「找信息」到「数据洞察」的转变

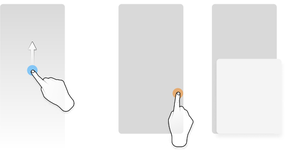
售前是线性的；售后是动态、多线程的。

原有平铺结构失效

补字段无法解决：
多场景并行/动态优先级
所以放弃了。

我选择「重构信息框架」

分类	解决的问题
信息层	什么值得被注意
结构层	什么需要优先判断
交互层	如何降低扫描成本
表现层	如何提升理解速度

分类	原版	新版
页面结构	 长轴下滑	 长轴下滑 + 浮层结构
信息获取	信息堆叠，翻找式，效率低	概览优先，按需下钻
可扩展性	字段堆叠，难以拓展	模块化，场景可独立迭代
状态感知	无，靠人工判断	系统自动推断运营优先级

我的3个设计决策，让服务效率从「依赖记忆」变成「依赖系统」

关键过程
Key Process

STEP 1：前置，高优信息

重构框架，匹配非线性服务场景

原平铺框架无法支撑多场景并行，因此重构为双层结构：首层快速感知用户状态，二层按场景深入明细，让服务者的注意力跟着场景走，而不是跟着字段找。



STEP 2：高亮，重点信息

信息可视化，让系统替代人工判断

信息太多不是问题，看不出重点才是。

- 定义表现优先级：标签（状态）> 数值 > 字段明细
- 系统自动推断运营优先级，并适时介入

服务者一眼知道该做什么，不需要自己判断。



STEP 3：下钻，详细信息

信息规范化，建立跨场景统一语言

原系统格式不一，关键信息需要复制到其他系统才能查询

- 重新定义规范：多用标签、数值、关键词构句，少用描述和赘词。
- 建立看板加筛选器结构：让购车、满意度、活跃度、服务、消费五大场景共用同一套信息语言。



多场景
用户运营工作台
影响力

70 %

场景覆盖增长

过去用户信息多半堆积在「转化前数据」



40 %

数据可视化提升



0 天

新数据无需设计即上线



✓ **提炼服务小管家用户画像**

丰富并迭代NIO B端体验模型

✓ **用户详情页 App/Web双端落地**

提高用户服务效率

✓ **沉淀用户信息组件、车辆信息组件**

场景覆盖购车、满意度、活跃度、服务、消费

✓ **设计特征被其他业务复用**

总分组提升信息传递效率