# “饭否”软件系统的需求构思及描述

## 背景介绍

  随着经济的发展，对生活饮食的要求也随之提高,他们平时的生活节奏非常紧凑，谁也不愿为了填饱肚子, 而浪费更多的学习和工作时间。越来越多的大学生喜欢宅在宿舍，一台电脑，一个电话，解决一天的生活问题，外卖成为众多群体的饮食选择。可以看到外卖市场存在着很大的市场需求，目标顾客主要是学生，白领，市场进入的门槛也比较低。当前外卖市场存在着不少问题，这也将是进入这个市场的有利条件，目前健翔桥周边的外卖快餐店有几百家，但良莠不齐，尤其是一些小型快餐店不尽如人意。叫外卖的顾客，他们在对外卖带来的便利感到满意的同时，也对一些外卖快餐店品种单一、价高质次，送货不及时等问题颇为不满。如果做到专业的，大型的，成规模的，讲究营养搭配，卫生一次性回收，一个电话即及时送餐上门，可单订可团订，形成有品牌的具有良好口碑的大型外卖快餐店，必将有极大的市场回报。

## 软件目的

资料显示，在2016年中国在线订餐市场用户规模达到1.47亿，预计2017年中国在线订餐用户规模将达到2.08亿。2016年上半年中国在线订餐用户职业类型比例从高到低分别是企业普通白领、服务业产业人员、在校学生、企业管理人员、技术人员、自由职业者、公务员/事业单位人员、个体户。因此在2020年的今天，在线订餐市场依旧没有饱和.

## 软件创意

早期外卖平台外卖App和外卖平台外卖频道两个团队的合并，带来的最大痛点是代码复用，而非平台化，而在很长的一段时间内，我们也没有想过从平台化的角度去解决两端代码复用的问题。然而代码复用的一些失败尝试，给后续平台化的架构带来了不少宝贵的经验。当时是怎么解决代码复用问题的呢？我们通过和产品、设计同学的沟通，约定了未来的需求，会从需求内容、交互、样式上，两端尽可能的保持一致。经过多次讨论后，团队发起了两端代码复用的技术方案尝试，我们决定将搜索模块从主工程拆分出来，并实现两端代码复用。然而两端的搜索模块代码底层差异很大，BaseActivity和BaseFragment不统一，UI样式不统一，数据Model不统一，图片、网络、埋点不统一，并且两端发版周期也不一致。针对这些问题的解决方案是： 通过代理屏蔽Activity和Fragment基类不统一的问题； 两端主工程style覆盖搜索库的UI样式； 搜索库使用独立的数据Model，上层去做数据适配； 其他差异通通抛出接口让上层实现； 和PM沟通尽量使产品需求和发版周期一致。业务流程

## 系统功能需求

快餐外卖系统是一套功能强大、操作简便、实用的自动化管理软件，包括客户管理、订餐管理、菜单管理、数据维护、后台管理。

下面概括一下本快餐外卖系统大致的功能需求。

客户登录

在客户进入系统前，首先要求客户进行登录，登录时要验证客户名和密码是否匹配，验证通过后允许客户进入本系统操作，客户的密码需要进行加密算法。客户登录后其信息存入数据库中。

修改注册信息

客户登录后可以修改自己的注册信息，包括修改用户密码，每页显示行数等信息，不允许修改客户名，姓名和部门等信息。

菜单查询

登录后客户可在菜单管理中查询所需快餐，并订餐，也可以直接输入您所需要的饭菜名进行查询并订餐。 4） 订餐管理

若是订餐后，可以填写一张订餐单，详细说明住址和订餐时间。餐到付款。若是有急事取消订餐，可以在半小时内完成，若是您定的餐没有，我们则在订餐单之后3分钟之内告诉您。

## 软件系统的功能描述

5.1 目标用户

在线订餐目标消费群体目前主要集中白领、服务行业人员以及学生等日常就餐依赖于餐馆、食堂的人群。

5.2 竞争对手

目前比较主流的外卖APP有：外卖平台外卖，饿了么，百度外卖，淘点点。以及线下针对小商家设计的小程序，自配送等方式。

## 需求总结

从搜索库拆分的第一次尝试算起，外卖Android客户端在架构上的持续探索和实践已经经历了2年多的时间。起初为了解决两端代码复用的问题，我们尝试过自上而下的强行拆分和复用，但很快就暴露出层次混乱、边界模糊带来的问题，并且认识到如果不能提供两端差异化的解决方案，代码复用是很难持续的。后来我们又尝试过运用设计模式约束边界，先实现解耦再进行复用，但在推广落地过程中认识到复杂的设计很难快速推进下去。

在平台化开始的时候，团队已经形成了设计简单、边界清晰的架构理念。我们将整体结构划分为宿主层、业务层、平台层，并严格约束层次间的依赖关系。在业务模块拆分的过程中，我们借鉴微信的工程结构方案，按照三级工程结构划分业务边界，实现灵活的代码隔离，并降低了后续模块迁出和迁入成本，使得架构动态满足康威定律。

在两端代码复用的问题上，我们认识到要实现可持续的代码复用，必须自下向上的逐步统一两端底层的基础依赖，同时又能容易的支持两端上层业务的差异化处理。使用Flavor管理两端的差异代码，尽量减少向上依赖，在具体实施时应用之前积累的解耦设计的经验，从而满足了架构的可伸缩性。

没有一个方案能获得每个人的赞同。在平台化的实施过程中，团队成员多次对方案选型发生过针锋相对的讨论。这时我们会抛开技术方案，回到问题本身，去重新审视业务的痛点，列出要解决的问题，再回过头来看哪一个方案能够解决问题。虽然我们并不常常这么做，但某些时刻也会强制决策和实施，遇到问题再复盘和调整。

任何一种设计理念都有其适用场景。我们在不断关注业内一些优秀的架构和设计理念，以及公司内部外卖平台App、点评App团队的平台化实践经验，学习和借鉴了许多优秀的设计思想，但也由于盲目滥用踩过不少坑。我们认识到架构的选择正如其他技术问题一样，应该是面向问题的，而不是面向技术本身。架构的演进必须在理论和实践中交替前行，脱离了其中一个谈论架构，都将是个悲剧。