



上市时间





上市时间



上市时间



[ MVP ]



实现你的**最小可行产品**来测试你的概念。





上市时间

[ MVP ]

您的MVP的**周长**必须减少\*\*，同时允许您推销您的产品。

投注**早期采用者**并获得**最多反馈**。

您的MVP已部署并可用于**制作**。



上市时间



[ 快速失败 ]



失败快速\*\* \*\*快速。





上市时间

## 「快速失败」

快速体验解决方案（几周），收集来自用户的反馈并从错误中学习。

不要害怕改变一切\*\*。

别忘了，你会失败！



上市时间



[ 吻 ]



\*\* K \*\* eep \*\* l \*\* t \*\* S 实现和  
S \*\* tupid。

为什么复杂的时候可以很简单？





上市时间

[ 吻 ]

避免过度工程化，如果“纸张”模型或 Google 表单足以测试您的概念，请不要继续。

保持简单！在技术上和功能上都是如此。





上市时间



「生产力」

指定较少，**展开更多。**





上市时间

〔生产力〕

将您的规格限制在最基本的要领中，  
\*\*关注“什么”而不是“如何”。

该产品必须是最自我记录的。

文档必须以与代码相同的方式版本化。



上市时间



[ SAAS ]

系统研究\*\* SaaS \*\*解决方案。





上市时间

[ SAAS ]

**\*\* SaaS 解决方案具有可持续性和成本效益\*\*。**

在某些情况下，**\*\* SaaS 可以加速执行\*\* MVP \*\*。**

以总成本（**\*\* TCO : T \*\* total \*\* C \*\*  
\*\* O 的所有权为准），考虑替代方案  
的经济愿景），而不仅仅是许可成本。**



上市时间



[ 业务之心 ]

**核心业务**不应成为新服务和应用构建的障碍。





上市时间

## 「业务之心」

核心业务的演进和交付速度必须与消费它的服务的灵活性\*\*兼容。

核心业务必须公开服务。

核心业务必须采用事件驱动原则，以事件形式报告管理行为。



上市时间



「连续部署」

生产中的部署是非事件。





上市时间

## 「连续部署」

利用连续部署来使\*\* 生产适应企业要求，而不是相反。

环境中的部署，直到生产，必须是自动和频繁。





上市时间



[ PERPETUAL BETA ]

**\*\*永久beta \*\***方法可以让您的用户参与开发过程。





上市时间

[ PERPETUAL BETA ]

随意使用**用户参与开发**的永久beta原则。

术语“永久测试版”是指及时开发的应用程序，\*\*不断发展，而不是不完整的产品。





# 用户体验





# 用户体验





用户觉得的\*\*体验是根本。

人体工程学不可议付。



# 用户体验

## [ 知觉 ]

不要忽视\*\* UX设计师的工作\*\*，这是开发应用程序的基础。

整合用户的反馈，这是必不可少的。



用户体验



[ 性能 ]

内部和外部使用强大的接  
口。





# 用户体验

## [ 性能 ]

接口适用于**效率**。

接口的**性能**可以节省时间，**提高用户**的满意度，**因此**可以节省他们的沮丧

\*\*

。





用户体验



[ 移动第一 ]

采用\*\* Mobile First \*\*策略。





用户体验

[ 移动第一 ]

移动设备是**市场**中最重要的部分。

思维移动正在考虑基本。

响应式设计是常态，它是储蓄的来源  
( \*\* MVP \*\* ) 。



用户体验



[ OMNI-CANAL ]

适应用途，全渠道是常态。





## 用户体验

### [ OMNI-CANAL ]

全渠道方法为用户提供**统一体验**（例如：Netflix）。

不同的**通道是同步的和一致**（不同于批处理过程）。

所有演员（客户，顾问）都可以访问相同的信息。



**\*\* 用户是所有者的数据和课程。**



用户体验

[ SELF-DATA ]

个人，随时\*\* \*\* \* 个人数据。

通过允许用户实时跟踪和控制，建立信心。

子系统必须符合相同的要求。



用户体验



[ CRM / SFA ]



客户关系需要通过灵活，统一和事件驱动的CRM / SFA  
统一和背景化。 \*\*





## 用户体验

[ CRM / SFA ]

选择管理客户关系和销售队伍领导力的\*\* CRM （ SFA : S \*\* ales \*\* F \*\* orce \*\* A \*\* utomation ）。

\*\* CRM 必须开放\*\*给新的机会。

\*\* CRM 产生事件，对应于管理操作以适应平台的事件驱动\*\*逻辑。





用户体验



「大数据」

大数据平台可让您集中并处理用户数据，以最好地服务您的旅程\*\*。





用户体验

[ 大数据 ]

将\*\* Maif Group ，合作伙伴和供应商  
数据集中在途径\*\*逻辑中。

“数据准备”和处理可以合并数据。

大数据团队与特征团队合作以确保数  
据治理。



用户体验



[ DESKTOP ]

该工作站适应并适应**使用**和  
**现代渠道。**





# 用户体验

## [ DESKTOP ]

采用**身份联合会**统一体验。

A 门户允许提供**概述**，它不会取代应用程序。

工作站必须是**移动**，**多通道**和**标准**，以允许在**扩展企业**内开放。



用户体验



「贡献者」

不要忘记，你的**伙伴**正在家中使用现代应用程序。





用户体验

〔贡献者〕

将所有用户视为“客户”\*\*： 互联网用户，管理人员，运营人员，开发人员等.....

不要低估\*\* UX努力\*\*以实施内部使用管理应用程序。



所有可以测量的东西都必须  
是。

没有措施，一切都只是意  
见。



## 用户体验

### [ 全部测量 ]

考虑应用程序\*\*开发期间的指标。 \* **日志必须具有业务以及技术\*维度。**

不要忽视性能指标，它们是根本性的。

功能团队提供操作：它负责使应用程序可用。





**\*\* A / B测试通过允许反馈\*\***  
来决定，从而节省您的时间。



## 用户体验

### [ A / B测试 ]

不要随意决定两种解决方案，不要犹豫设置\*\* A / B测试\*\*。

这种模式包括呈现相同应用的两个不同版本，并基于用户活动的客观度量\*\*选择其中一个。



「退化」



在发生故障时，请考虑降级  
而不是服务中断。



## 用户体验

### 「退化」

在其中一个子系统的失败中，服务的降级版本必须首先被认为是，而不是中断。

使用断路器，在整个系统上隔离故障以避免其影响和传播\*\*。





人类





人类



人类



[ 特色团队 ]



团队围绕**产品或服务**进行组织。





人类

## 「特色团队」

团队是**特征团队**，围绕一个连贯的功能集合进行组织，并由该集合所需的所有**技能**组成。

例如：业务专家+ Web开发人员+  
Java开发人员+架构师+ DBA +运营。

**职责是集体**，特征团队有责任履行这项责任。





人类



[ 2-PIZZA团队 ]

限制Feature Team 的大小  
(从5到12人) 。





人类

[ 2-PIZZA团队 ]

限制特征组的大小： \*\* 5至12人\*\*。

5岁以下，她对外部事件过于敏感，  
缺乏创造力。 12岁以上，会丧失生产  
力。

术语“**2-比萨饼团队**”表示特色团队的  
规模不应超过可用两片比萨饼喂食的  
人数。



人类



[ ARTISAN软件 ]



投注**知道如何做**和谁喜欢做  
\*\*的多才多艺的人。





人类

[ ARTISAN软件 ]

最重要的是文化的发展，可扩展性和适应性。

招聘软件工匠和全栈开发人员，他们通过他们的专业知识和他们的总体愿景带来了真正的附加价值。

不过，例如，移动开发人员通常是专业开发人员。



人类



[ 招聘 ]

\*\* 吸引 最佳。





人类

〔招聘〕

建议适应员工的工作模式：**流动，家庭工作**，\*\* CYOD（C \*\* hoose \*\* Y  
我们 O \*\* wn \*\* D \*\* evice）。

留出时间进行实验并在工作时间中实现。



人类



[ EVE ]

该组织必须是睡眠引擎  
前一天是工作的一部分。





人类

[ EVE ]

该组织必须通过建立诸如**继续教育**或**商业大学**的系统来成为日托引擎。

随意将它们与其他更多**非正式**方式结合使用，例如：**\*\*编码Dojos\*\***，**棕色袋子午餐**，**外部会议**。





人类



[ 共建 ]

打破交易之间的障碍，押注  
**收敛**目标。





人类

〔 共建 〕

为了打破行业之间的障碍，仅仅在一个共同的地方将人们围绕共同的产品进行分组是不够的。

**敏捷方法**消除这些障碍，确保目标的一致。

这些做法是成功的关键的组成部分，该组织是保证人。



人类



[ DEVOPS ]

\*\* DevOps \*\*的做法允许墙壁  
在构建和运行之间。





人类

[ DEVOPS ]

采用\*\* DevOps 将 Dev 和 Ops 集中到一个共同目标：为组织\*\*服务。

交易仍然不同！ DevOps 并不意味着同一个人执行 Dev 和 Ops 的任务。开发人员和运营必须合作，以便从\*\*技能中受益并改善同理心\*\*。



人类



[ PAIN ]



功能团队执行艰巨的任务

\*\*  
。

自动化如下。





人类

[ PAIN ]

在传统组织中，团队之间缺乏理解通常与距离和缺乏沟通有关。

特色小组的成员负责所有任务的共同负责和联合\*\*。

疼痛是持续改善的关键因素。



人类



[ CDS ]

这些服务中心很难与**集体承  
诺**协调一致。





人类

[ CDS ]

特征团队围绕着协作和集体参与的原则而建立。

服务中心正在朝着企业IT合理化和合并的方向发展，这与集体承诺的观念背道而驰。





人类



[ 验证 ]

该组织具有验证\*\*的作用，  
而不是教条式的。





人类

[ 验证 ]

确保组织在工具和用途上保留验证角色。特别是影响遗产的工具（例如：源代码的管理）。

提供功能团队意味着支持他们的选择。

不要教条，并确保鼓励实验。

人类



[ 横截 ]



功能团队希望交流并分享他们的经验和技能。





人类

[ 横截 ]

不要在\*\* Feature Teams \*\*之间创建障碍。

设立一个**组织**和功能团队需要的**敏捷性**以相互沟通并分享他们的技能和经验。

\*\* Spotify \*\*（部落，章节和公会）的横向组织是一个雄辩的例子。





# 互操作性





# 互操作性



互操作性



[ API FOR ALL ]



**\*\*所有用途的API\*\*：内部，  
客户和合作伙伴，公共。**





## 互操作性

### [ API FOR ALL ]

使用\*\*公共API\*\*将您的组织开放给新用户和新客户。

在商业合作，客户作为提供商\*\*，API是标准交换格式。

\*\* APIs 也打算用于组织的内部使用

\*\*





互操作性



〔自助服务〕



使用API必须是**简单和快速**。





互操作性

〔自助服务〕

API的使用应尽可能简单。想想开发者的经验。

根据需要验证充分性的最佳解决方案是\*\*快速测试API\*\*：几分钟就够了！

该平台必须提供一个**图形界面**来简单测试API。



互操作性



[ API管理 ]

这些API的使用必须是**控制**和**控制**。



互操作性

[ API管理 ]

实施API管理解决方案来管理配额，  
限制，验证和记录。

收集指标以管理监控，过滤和报告。

互操作性



〔要求〕

为平台内置的外部系统和服  
务设置**要求**。



互操作性

[ 要求 ]

**要求外部系统符合与内部系统相同的要求。**

**外部系统必须发布事件并允许技术监测。**

**在必须集成外部系统数据的情况下，总同步必须可能。**

互操作性



「多租户」

该架构必须被认为是**多租户**。



互操作性

[ 多租户 ]

即使在底座上没有考虑白色标记，也要设置多租户架构。你的初始申请是第一次持有。

从一开始就想到系统的多功能实例。



互操作性



[ SETTING ]

系统必须是**本地配置**。





互操作性

[ SETTING ]

语言，货币，业务规则，安全配置文件必须易于设置。

谨防超级通用性，它通常是无用的\*\*  
成本来源。

设置必须可扩展，并根据需要快速提供。



互操作性



〔功能翻转〕



使用**功能翻转**创建灵活的通用系统。





互操作性

## [ 功能翻转 ]

功能翻转是关于将应用程序设计为一组功能，可以启用或禁用热，生产。

在多租户应用程序中，功能翻转允许您定制支持者。

Le功能翻转简化了A / B测试。





# 游戏规则





# 游戏规则





游戏规则

[ 技术选择 ]

技术选择由\*\* Feature Team  
\*\* 制作和假定。





## 游戏规则

### [ 技术选择 ]

特征小组必须采取**负责的行动**来确定仅影响其的选择以及影响组织的选择。

必须由组织或对等收敛过程**验证 超过特征小组范围的选项**（例如，许可证，不经常使用的编程语言）。







游戏规则

[ 良好的使用 ]

正确的工具 好用是节省资金  
的来源。





## 游戏规则

### 〔良好的使用〕

对每个人强加的工具都是**的风险**。滥用的好工具可能会造成非常严重的后果<sup>\*\*</sup>。例如，很少使用的敏捷方法是危险的。

**工具必须质疑。**

<sup>\*\*</sup> Excel 通常是理性的选择，但它不是一个可以做所有事情的工具

<sup>\*\*</sup> (CRM, ERP, Datamart, ...)



Privilege \*\* Build \*\*为核心业务。

考虑购买，其余情况。



## 游戏规则

### [ 构建VS.购买 ]

一个工具带有一个**功能区分功能**的组织越多，它就越有可能被构建。**核心业务必须允许特异性和快速且经常适应**。有些软件包\*\*有时会根据这种需要进行调整。

对于其余的：**SaaS，开源，构建或所有者需要逐案研究\*\*。**





游戏规则

[ 开放源代码 ]

充分利用开源。  
替代选择必须得到支持。





## 游戏规则

### 〔开放源代码〕

专有解决方案对于组织来说是**风险**，必要时必须能够恢复维护。

很少有专有工具没有**开源替代品**。

该组织从开源社区获益，并可以偿还其捐款\*\*。



游戏规则



[ 微服务 ]



开发独立和弱耦合服务。





**弱耦合**必须是标准。

每个微服务都有一个明确定义的界面

★ ★

。

这个**接口**决定**微服务**之间的链接。

**域驱动设计**允许（尤其是有界上下文）预测此问题。





游戏规则

[ DATA ]

每项服务都有自己的\*\*数据  
存储系统。





**A 数据存储仅用于单个微服务。**

从一个微服务到另一个微服务的数据访问仅通过其接口完成。

这种设计意味着整个平台随时间的一致性。它必须在所有级别\*\*被包括，包括用户体验。



每个微服务都必须有一个合理的功能边界，这个“适合头部”\*\*。



## 游戏规则

### [ SCOPE ]

微服务提供合理数量的功能。

毫不犹豫地微服务开始增长时削减服务。

如果需要，一个合理大小的服务使有可能平静地考虑重写。





游戏规则

[ RESPONSIVE ]

反应式宣言为反应式体系结构的设计开辟了道路。





**响应式编程**侧重于数据流和变化传播。它基于“\*\* Observer ”模式，与传统的“ Iterator \*\*”相反。

反应宣言设定了基本轴线：**可用性和速度，韧性到故障，灵活性，弹性和信息定向。**



游戏规则

[ 异步FIRST ]

异步过程有利于解耦和可扩展性，有利于性能。





## 游戏规则

### [ 异步FIRST ]

应用程序之间的交换必须首先是异步。

异步交换自然允许弱耦合，隔离和流量控制（反压\*\*）。

只有当动作需要时才应考虑同步通信

\*\*

○







信息系统必须面向事件。



游戏规则

[ 活动 ]

**\*\* 事件驱动功能过程 自然 异步实现。**

**事件定向**允许有助于实施诸如**\*\* C \*\***  
**ommand \*\* Q \*\* uery \*\* R 责任 \*\* S \*\***  
**egregation (\*\* CQRS \*\*)**和**事件采**  
**购。**



游戏规则



[ 消息经纪人 ]



特权简单，强大且强大的消息代理到“智能管道”。





## 游戏规则

### 「消息经纪人」

**\*\* ESB 显示限制：可扩展维护是严重，无论从技术还是组织\*\*观点。**

**\*\* \*\* 卡夫卡等经纪讯息提供简单，耐用和韧性解决方案。**

**智能端点和简单管道是一种大规模工作的架构：它是\*\* Internet \*\*。**





系统的完全同步应该在设计时尽快考虑。



## 游戏规则

### [ TIMING ]

如果**事件流程**确保两个系统之间的**同步**，则这些系统的**总重新同步**必须在设计时**进行计划**。

自动\*\*\* **同步审计**（例如：按样本）  
允许**测量和检测**任何可能的**同步错误**。



游戏规则

[ 集权 ]

服务的配置是集中，发现由  
目录保证。





## 游戏规则

### [ 集权 ]

微服务的配置对于所有环境都是集中。

中央目录确保微服务的动态发现\*\*。

\*\* 全局可扩展性取决于这个目录\*\*。







游戏规则

[ 沙盘 ]

功能团队提供沙箱环境。





## 游戏规则

### [ 沙盘 ]

Feature Team维护一个\*\* sandbox 环境（当前版本和即将发布的版本）以允许其他团队扩展\*\*。

在一些非名义的情况下，功能可能在开发环境中被禁用。



游戏规则



[ 设计失败 ]



您的系统将崩溃！  
设计它，使其宽容。





## 游戏规则

### 「设计失败」

你的系统会崩溃，这是不可避免的。它必须为此设计（\*\* Design For Failure \*\*）。

在所有级别预测冗余：**硬件**（网络，磁盘等），**应用程序**（多个应用程序实例），**地理区域**，**提供程序**（例如：AWS + OVH）。





游戏规则

[ 工具包 ]

提供工具包，不要强加严格的  
的框架。





## 游戏规则

### 〔工具包〕

**注意技术组件房屋和横向！** 它们是限制性的，昂贵且难以维护。

**加速器，工具箱，技术堆栈可以汇集，免费功能团队，避免教条式的方法。**





公共，私人或混合型，云  
（\*\* IaaS 或 PaaS \*\*）是生  
产标准。



## 游戏规则

### [ 云 ]

**\*\* PaaS 服务优先，简单\*\***，并且快速扩展。

**\*\* IaaS 服务允许您解决需要更大灵活性\*\***的情况，但需要更多的操作工作。

私有云不是传统的虚拟化环境，它依赖商品硬件。







游戏规则

[ 基础设施 ]

功能团队不管理基础设施，  
它由提供和维护。





## 游戏规则

### [ 基础设施 ]

基础设施问题不在**功能团队**内。基础架构必须由**交叉功能**服务提供和维护

★ ★

○





游戏规则

[ 集装箱 ]

容器提供了异构工具所需的  
灵活性。





# 游戏规则

## [ 集装箱 ]

容器提供特性团队所需的灵活性，以便在均匀上下文中启用异构工具\*\*。





游戏规则

[ ENVIRONMENTS ]

使用容器可以解决技术环境的问题。





## 游戏规则

### [ ENVIRONMENTS ]

这些容器（例如： \*\* Docker ）使得可以释放环境的差异。

部署过程对环境必须是不可知的。

一些组件如数据库不应该部署在容器中。他们的部署仍然是自动的。



游戏规则

[ 公制 ]

所有措施必须是集中和可访问。





## 游戏规则

### [ 公制 ]

**指标**对于具有不同粒度级别的所有人都是**可访问**：相关团队特征的详细视图，该组织其他成员的聚合。

**访问指标并不意味着访问单元数据**，必须对其进行控制以保持机密性。

**所有环境都受到影响。**







游戏规则

[ 质量 ]

软件质量是关键因素。





## 游戏规则

### [ 质量 ]

代码评论是系统。作为持续改进的一部分，它们由特征小组的成员或组织的其他成员进行。

那不是你被审计的，而是你的代码：“你不是你的代码！”。

质量可以部分自动化，但没有什么比“新眼睛”。





游戏规则

[ 自动测试 ]

自动化测试是持续部署的不可协商的先决条件。





游戏规则

〔 自动测试 〕

自动测试确保产品的质量随着时间的推移\*\*。

它是持续部署的先决条件，它允许\*\*更改和频繁部署。

生产推出成为轶事事件！





所有级别的测试：单元，集成，功能，弹性，性能。



## 游戏规则

### [ 测试水平 ]

整合和功能测试是最重要的，它们保证\*\* 有效操作\*\*。

单元测试适用于开发。

性能测试随着时间的推移衡量性能。

韧性测试有助于预测失败。





游戏规则

[ COVER ]

封面是测试质量的主要客观指标。





测试的**代码覆盖率**是代码质量的**良好度量**。

这是一个**必要条件**但是**不够**，**不良测试策略**的覆盖率可能很高，而不保证代码的高质量。





安全性是一个过程，不应该  
对问题进行处理。



## 游戏规则

### [ 安全 ]

**\*\* 安全专家可\*\* 如果需要直接集成到功能团队中。**

**安全专家可在组织中获得审计，认识和转发。**



