

# 选个大咖来 拜师



# Table of Contents

---

1. [Introduction](#)
2. [前言](#)
3. [\[职业规划\]](#)
  - i. [TODO 半年前的目录](#)
  - ii. [如何提问](#)
  - iii. [读书清单](#)
  - iv. [必读书籍](#)
  - v. [程序员的台阶很高](#)
  - vi. [程序员的十种去路](#)
  - vii. [沟通最重要, 技术排第二。](#)
  - viii. [收起你的个性, 要谦虚, 不要任性\(控制好你的脾气\)。](#)
  - ix. [不要内向。内向的人是失败者。](#)
  - x. [不要踢皮球](#)
  - xi. [要有个人的技术博客。](#)
  - xii. [有机会就要带领团队](#)
  - xiii. [我不看好的职业：产品经理？测试？运维？](#)
  - xiv. [每天都要学习, 不要沦于平庸（买菜接孩子, 算计各种羊毛）](#)
  - xv. [创业团队务必有个靠谱的CTO](#)
  - xvi. [可以把任务分派出去, 不要放在自己肩头硬抗, 最后事情没做完, 耽误了。](#)
  - xvii. [敏捷方法论](#)
  - xviii. [目前我很推崇的职业方向：新手 -> 熟手 -> 技术经理 -> CTO\(创业\)](#)
  - xix. [多关注职业前辈的博客：robbin, gigix, iamhukai, 国外的：martin, kent,](#)
  - xx. [使用最好的键盘, 显示器。而不是一流的办公室, 30块钱的破烂键盘鼠标。](#)
  - xxi. [办公室中的健康问题：视力, 脊柱, 脾胃。 按时的离开显示器是为了更好的工作。](#)
  - xxii. [多参加程序员聚会, 与人交流。](#)
4. [\[技术建议\]](#)
  - i. [使用chrome 加快调试速度](#)
  - ii. [敏捷方法论: 测试驱动, \(难于测试的软件, 就难于开发\)](#)
  - iii. [敏捷方法论: 极限编程](#)
  - iv. [世界上只有三种编辑器：Vim, Emacs, 其他。](#)
  - v. [忘掉你的鼠标, 用好快捷键](#)
  - vi. [使用编辑器时, 最重要的一些快捷键](#)
  - vii. [SCM: 只用Git, 忘掉其他。](#)
  - viii. [在技术的天空中留下痕迹](#)
  - ix. [人跟人的能力不一样。](#)
  - x. [永远不要人肉重复。](#)
  - xi. [命令行](#)
  - xii. [放弃mac, 使用linux.](#)
  - xiii. [技术很杂\(好多列表, 不要误以为一种技术通吃\), 入门用的技术\(各种设计模式啥的\), 进阶：ThoughtWorks 技术雷达](#)
  - xiv. [对于ruby 程序员来说, 机器语言, 汇编语言, object c 都是反人类的语言](#)
  - xv. [招聘候选人时, 务必识别项目毒药。](#)
  - xvi. [技术债是要命的, 代码, 坏味道,](#)
  - xvii. [引导客户明确需求](#)
  - xviii. [团队内要及时传播知识。成为进取型团队（每个人都争先恐后的进步）](#)
  - xix. [招人是个长期的任务, 从头培养的人比高薪挖来的人好的多。](#)
  - xx. [导致团队分崩离析的因素](#)
  - xxi. [告诉大家好代码跟烂代码的区别](#)
  - xxii. [软件项目中看起来的美好](#)

5. [软件开发之殇]

- i. 给其他行业的朋友
- ii. 所见即所得设计法
- iii. 小步快跑
- iv. 中国软件公司的特点
- v. 开源项目中的坑：需要先理解，再改
- vi. 如何找到CTO
- vii. 正视技术的作用
- viii. 好技术与坏技术
- ix. 外包的现状
- x. 自己养团队
- xi. 跟家装比较
- xii. 这是一个不透明的行业
- xiii. 全栈工程师-新兴起来的解决方案
- xiv. 死亡案例

软件开发之殇

写给程序员的职业规划 写给要找CTO的创业者 写给有项目要外包的人。

# 前言

---

这本书里面的知识,不会很深奥,不会涉及到算法,数据结构,操作系统.

可以让人快速上手干活儿.

本书的最终目的在于建立起一套快速培训新员工的知识体系. 基本上就是:

- 需求分析.
- 前端开发, 部署
- 后端开发, 服务器的部署和维护.
- 各种有用的方法论
- 职业发展

本书是供给新入行的新人来读的,其中 Titanium 是国内的第一个本中文书, Rails 虽然国内也有中文书, 但是大部分是由 国外文档翻译而成. 希望本书对Rails的描述能有自己的特点.

我对Linux只能算得上熟悉, 书中的内容, 可以让一名初学者调试,操作Linux 入门.

总之, 里面的各种技术, 都只是入门, 进一步的学习,请参考后面的读书清单.

答应了人家要出个书稿，结果现在还没弄好。

想要出三本书，其中之一就是 如何在国内带团队，以及当前软件外包的现状。

程序员如何成长？最终出路？

会用好各种工具。VIM ,GIT, SVN , LINUX,

方法论：全都是自动化。要有单元测试。

键盘是最重要的工具。告别鼠标流。

漫长的编译时间是大坑！

务必想尽一切办法缩短不必要的时间：编译时间，部署时间，测试时间，合并代码时间，以及对业务的理解时间。（详见 如何与客户做需求分析）

英语要好。不要做汉语文档的奴隶。英语好可以让你在平庸程序员中脱颖而出。

平庸程序员的特点：不爱学习，英文差，容易自我满足，水平一般。

要善于表达自己。多写博客，多与非技术的人沟通。不要做闷骚男。

程序员如何处事？与人为善。

最终出路：只有30%的人会做跟技术相关的职位。其他70%的出路各有不同。目前国内技术人员，私企在33岁以下，外企在40岁左右。

如何带领团队？

西方：管理学实践

东方：中国式管理

对外：如何与人沟通需求，

对内：管理项目进度。分配任务。协调各个方面。

对于项目，什么样的人 是催化剂，什么样的人 是项目毒药。

项目必死的几个特点

沟通不够。需求人员跟开发人员无法见面。常见于异地外包项目。难以见面聊需求。继而产生信任危机。

不要找兼职人员。软件开发，需要的是频繁沟通。可能一天碰面20次，

面对面是沟通的王道。面对面 > 电话 > QQ > 邮件。所以，重要的问题，永远面对面解决。（记得不要傻呵呵的写表白信了）

成员不要有坏脾气。跟队友吵起来，这个项目不会成功

解决办法：项目的催化剂：多鼓励，多沟通。有个脾气好的妹子做助理，特别重要。

需求变动频繁。把做好的东西推翻重做，会极度扼杀士气。解决办法：保持好节奏。好的节奏是：小步快跑，跑之前想好。

一句话需求。

便宜到不可思议的项目，都是陷阱。比如说200块做个手机app. 一般都是外包公司把做好的东西，修改个标题，背景啥的，再扔给你。解决办法：要知道项目的成本。雇佣个程序员要多少钱，做个东西大约要多久。

人员变动频繁。人永远是最重要的。代码次之。一个项目，无论怎么变动，都要留下一个资深者。（结对编程是个好办法），或者团队内要有个这个项目的使用专家。

交付周期太长。比如说6个月一交付。一年一交付。基本上这样的项目只有最后30%的阶段才能见到演示。而前面4,5个月都看不到成果。很危险。解决办法：小步快跑，频繁交付。每次只做20% 的功能。忍痛割爱。

如何与人谈项目？

外包行业的市场很混乱，打价格战，受损的是客户。

谈需求三句话：能不能做？多少钱？做多久？

价格要透明。

使用简笔画，先画出 操作界面。可视感比 表格和文字要好得多。

对于需求分析师，要善于引导，使用客户可以听懂的比喻，不要使用术语。

要善于引导客户说出需求。

客户都希望项目“明天就交付”。

作为客户，如何提出自己的需求？

不要用“一句话”来描述需求。比如：我要做个卖啤酒的应用。

不要用“做成跟 X X 一样”。

不负责任的外包公司，会回答你：“可以做，肯定没问题”。但一旦交了钱，就不是那回事了。

项目的估期

时间越短，越容易成功！

第一期只做最重要的功能（没有这个功能，项目就会挂掉）

可以带着缺点上线。

如何选择合适的技术（框架）

没有最好的，只有最合适的。

合适的技术的特点：1. 要有良好的技术支持(open source) 2. 认同的人多。

未来技术的发展趋势

程序员门槛低。想做好门槛高。

移动开发将来不会这么累。看看当今的delphi, c++ 适用的领域。再远一些，看看那些年的汇编。

Object C 迟早会退休，跟当年的 汇编一样。

很可能会出现一种语言或者框架 统一大部分的平台（例如目前的 hyperloop)

不要与其他程序员争论。多与国外的人接触，学习，如 github, stackoverflow

要持续不断的学习。不要以为一招可以吃遍天下。

不要为了优化而优化。lotus 为了节省运行时内存耗费了2年时间，直接导致 office 取代了它的位置。

程序员的工作习惯

1. 必须有个杯子。当你觉得疲劳的时候，站起来喝点水。不要吃零食。伤脾胃。不要抽烟。
2. 不要用沙发椅。要用硬板凳子。1。很好的避免腰酸。2. 当你的屁股觉得疼的时候，就该站起来走一走了。
3. 不要用过低的显示器。小心你的颈椎！
4. 每用1小时左右的电脑，就站起来活动会儿。手头放个秒表很有效果。（可惜自己创业后就很难这样了）

## 关于程序的一点儿思考

---

1. 为什么行末尾要加分号？为什么每个方法结束要放(), {}？
2. 为什么 声明变量，常量，数组的方法不一样？

```
var $apple = 'apple';  
  
define('APPLE', 'an apple');  
  
$apples = array('apple1', 'apple2', 'apple3');
```

像ruby, js 那样一个写法多好:

```
$apple = 'apple'  
  
APPLE= 'apple'  
  
$apples = ['apple1', 'apple2']
```

1. 难道是我 MVC框架用多了？感觉不用MVC 就 没法干活。特别看不了这样的代码：

```
<?php  
  
    echo " ... "  
  
?>
```

## 软件项目中的看起来的美好 ( some useless technologies)

---

显示适配：一套代码，多种设备都可以适配。如果用纯 CSS 写的话，就会特别麻烦。移动屏幕：一套代码 PC 屏幕：一套代码。（优酷的移动下载页面）

I8N：internationalization：一套代码，显示多种不同的语言。

数据库的适配：hibernate，rails Active Record: 都支持数据库的平滑迁移。支持多种数据库。



# 【提问的智慧】手把手教你如何百度

---

提问的智慧一词，来源于著名计算机大师 Eric Raymond所写的同名文章《How to ask questions the smart way》。汉语全文见：<http://www.linuxforum.net/doc/smartq-grand.html> 英文地址请百度。

我们不想掩饰对这样一些人的蔑视--他们不愿思考，或者在发问前不去完成他们应该做的事。这种人只会谋杀时间--他们只愿索取，从不付出，无端消耗我们的时间，而我们本可以把时间用在更有趣的问题或者更值得回答的人身上。我们称这样的人为“失败者”（由于历史原因，我们有时把它拼作“lusers”）。

我们在很大程度上属于志愿者，从繁忙的生活中抽出时间来解惑答疑，而且时常被提问淹没。所以我们无情的滤掉一些话题，特别是抛弃那些看起来象失败者的家伙，以便更高效的利用时间来回答胜利者的问题。

——《提问的智慧》引言。

下面我来教大家如何找到问题的答案。

1. 宗旨：中文百度，英文google

我不是一个百度信徒，也不是迷信google的人。以前作为一个程序员，每天都遇到各种计算机问题，而我尝试两者之后，发现，中文的搜索上，百度最有效。而英文上，google最好。

1. 关键字的使用：要关键，要简洁

比如一个问题：“肾虚的症状是什么？”搜索的时候，不要这句话全搜，而是使用：“肾虚 症状”作为关键字来搜索。关键字之间用空格。

再比如“植物神经紊乱用气功能治好吗？”这样的问题，请搜索“植物神经 气功”。

再再比如“马礼堂六字诀该怎么练？”直接搜索“马礼堂 六字诀”

很简单吧？进一步的使用方法，请百度：)

## 读书清单

---

因为我个人从事的工作(做网站,手机app)不需要什么算法, 所以推荐的 书单也都是基于方法论的书记.

其中, 针对具体编程语言的书, 推荐直接读英文版. 第一本会读起来会特别困难, 但是当你的英文词汇量上去之后, 看任何英文文档就没问题了. 第二本第三本就越来越轻松.

重要的是, 英文版的内容理解起来比中文翻译过来的词汇容易多了.

## <<从小工到专家>> ( Pragmatic Programmer: From Junior to Master)

---

经典中的经典, 每次翻开这本书, 都可以有新的收获. 曾经为它做过概括, 但是发现不能够. 里面的内容实在太精炼了.

## <<重构>> ( Refactoring )

---

当成API参考书来看吧. 对于 C/JAVA 这样的"传统"语言比较合适. 对于 Ruby 没太大必要, 但是这个属于基本功, 知道设计模式的人写出的代码 远超不知道的.

## 拥抱变化

---

## 测试驱动开发

---

敏捷开发,单元测试, 的必读书籍.

如果你是个java程序员,请看<>

## Code Complete

---

让你知道什么是高质量的代码. 什么是烂代码.

## 黑客与画家

---

## 软件随想录 Joel 谈软件

---

## Ruby 元编程 Metaprogramming Ruby(2)

---

没有读过它,就无法掌握Ruby, 更别提精通.

Ruby 程序员必读书籍. 读了之后会让你对语言的了解更上台阶.

## 设计模式

---

不要看大陆几个人翻译的. 要买就买台湾版.

## 敏捷开发, 原则与模式

---

## 人月神话

---

## 人件 Peopleware

---

人是最重要的.

## CSS the missing manual

---

里面对于CSS的描述特别棒.

## coffeescript

---

薄薄的小册子.看了之后让你对coffee更加有了解.

## Guides.rubyonrails.org

---

建议读英文版. 第一章入门不好入. 后面的文章很有深度.

## Linux : 鸟哥的Linux私房菜

---

Linux 需要在日常的积累中学习, 随便挑本书入门即可. 这本书还好. 台湾的兄台写的.

## Seven Weeks 系列

---

Seven More Languages in Seven Weeks: Seven databases in Seven weeks Seven webframeworks in Seven weeks (还有一个并发的, 这个就不用了,除非你的工作中有并发内容.)

推荐上面两本书, 是为了开阔大家的眼界. 可以不掌握, 但是一定要了解.

## Mac ubuntu kung fu

---

## The Thoughtworks Anthology

---

必读书籍

---

**TODO** 问沛沛。

---

# 程序员的台阶

---

## 英语必须好

---

导致国内的技术人员落后于国际的重要原因，不是不够聪明，而是国内的程序员英文水平不好。

在国内，英语又好，计算机能力又强的程序员，在2001~ 2010年左右，都会被招进 外企。这个情况在BAT等国内互联网公司做大之后，有所好转，但是，我见到的英语好 的程序员，很多走的路线都是：大公司工作 ， 出国。

而国内的程序员为什么会比国外技术落后1~2年呢？ 这个时间就是 认识到这个技术好（大约1年时间）+ 翻译(一年时间）+ 出书( 找出版社，出版，大约4个月）的时间。

现在，由于新兴技术越来越多，我们不能再采取“学中文书”的习惯了。 所以，我对新手的 requirements 是：

- 要么CET6. 过了六级的人，英语肯定没问题的。
- 要么可以进行一段口语对话。口语是英语听说读写四个能力中最薄弱的环节。 如果这个人的口语是60分，那么阅读可以达到80分。

## 思路敏捷，清晰

---

有的人，我给他做培训时，他的思路跟不上我。有的人，我给他做培训时，他往往 能纠正我的错误，提醒我下一步的思路。

我们要的，就是后一种人。

思路的敏捷，直接导致程序员是否有“灵气”。 有灵气的人，几乎都是一点就通。 你给他一个方向，剩下的事他都能自己办完。

思路不行的人，让他做事就会让你特别痛苦，他离不开你，你一离开他就几乎没有 进度。

判断一个人思路是否敏捷，清晰，很重要的一点是看他口齿是否清晰，表述能力是否 足够好。

## 表达沟通能力强

---

表达和沟通能力强是非常重要的因素。一个软件项目能否做好，完全取决于大家的沟通。

比如说，

- 这个需求没有说明白
- 昨天提交的版本为什么没有通过
- 那个bug昨天修改了，为什么今天又出现了
- 用户的需求又改变了

我们在开发过程中，绝大部分出现的问题，我们都要与人商量，跟人沟通。 有沟通恐惧症的人是无法胜任软件开发的。 而“话痨”程序员就特别难得。比如 ThoughtWorks的我认识的朋友，都是非常擅长沟通，口才特别棒的人。

对于不会沟通的人，往往工作就做不好，不受别人的待见。时间一长，这样的人 就容易恶性循环，越不敢跟人沟通。这个问题真的很常见。

## 具备领导气质

---

一个人的能力是极其有限的。一个十年经验的优秀工程师，在做普通难度的编码 方面，也不如2，3个普通人。

而通常，一个项目中70%左右的代码都是“普通难度”的代码。所以，团队的力量就 凸显出来了。你会发现一个5人精英团队做的事儿，比一个独行侠要多的多。

所以，要具备领导气质。因为一旦你的上级发现这个程序员是核心骨干，就会希望 对你委以重任。最直接的就是：让你做小组长。

恭喜你，程序员的晋升之路开始了。把握好这个机会，努力的培养自己的带队能力， 你会发现自己的成就更多了。

## 技术过硬

---

技术人员的世界观中，没有“老资格”一说，能让技术人员服气的，就是实力。一旦你当上了Team Lead的时候，必须具备远超他人的技术实力，比如：

- 对语言的高级特性掌握的清楚
- 能够及时处理其他人遇到的编程难题
- Linux技巧出众，能够轻松化解服务器的压力

只有这样，才能让你的团队成员服气。团队才能在你的带领之下成长。

否则，一旦队伍里其他人发现你的实力还不如他们，你的工作就没法干了。

# 我不看好的职业：架构，测试，运维，产品经理

---

如果你是一名职场新人，我不建议你投上面任何一个职位的简历。如果你是老鸟，可以去做架构师。

## 架构

---

不要做只做架构的架构师。

因为你无法知道第一线工程师面临的问题。

日本的软件公司，专门有个公司叫架构师。收到需求之后，他会把需求一点儿一点儿的做分析，然后设计，从骨架，到伪代码，直到某个按钮的名字。

不要做这样的架构师。能把技术的大方向定下来就可以了，千万不要去做去干预第一线程序员的事情，因为：

- 不参与第一线的工作，就无法准确判断出面临的问题
- 不准确的预判，会导致不合理的架构
- 由第一线的程序员来做写代码最合适。

好的CTO 或者技术经理可以把这些工作做的很好。

## 测试

---

测试的基本功，是人肉测试。好的测试人员，需要熟练运用自动化测试。可惜的是，这些年来，大部分测试人员，都是刚毕业的年轻人，没有接触过自动化的工具。所有的工作就是人肉。在项目上线之前通宵加班。他们的工作内容也很简单，鼠标手指点点点。

这样的工作没有技术含量，是没有价值的。

而且特别容易造成与程序员的摩擦。曾经有个朋友，测试人员一天给他提了200个BUG。沟通无果后，这位程序员朋友离职了。工作没法干了。

我认为，一个好的项目经理 + 懂得测试的程序员，就完全可以承担传统测试人员的工作。而且由产品经理来把握需求的优先级，是特别合适的。

另外，给测试同学的建议：

你可以在大公司里养老。但是你一旦离开就肯定找不到工作。创业公司不会有钱雇佣你。十年前做测试的朋友早都转行了。10个里面留下一个就不错了。

你看看自己会不会 selenium, appium, load runner。如果都不会的话，赶紧现在就转行。

如果你会的话，相信我，其他程序员掌握这些工具的能力比你快还比你强。

## 运维

---

运维的工作包括：

- 管理服务器, 域名。
- 分配账号
- 部署最新代码

我不看好的职业：产品经理？测试？运维？

- 维护wiki, 防火墙, 解决宕机问题
- 需要7x24值班。
- 优化nginx等服务器。

这些工作, 很多都是对 程序员和服务器之间的阻碍。 直接导致程序员的工作效率降低。 导致出错时各种推诿。 没有太高的技术含量。 在BAT这样的大公司会比较有用。但是在其他公司, 日访问量100W以下的, 没有用武之地。

上面这些工作, 熟悉Linux的程序员都会做; 而且做部署, 优化服务器的话, 程序员 做的会更好。因为代码就是程序员写的, 一旦发生问题, 程序员会最快的分析出日志, 会第一时间知道问题出在那里。

加上VPN, 短信报警, 也就不需要7X24的值班了。

更重要的时, 你除了BAT这样的大公司, 无处可去。职业没有出路。

## 产品经理

---

我对这个问题还没有想好。

这个职业不需要高深的知识, 连英语都不需要懂。主要工作是:

- 明确需求
- 提高用户体验
- 跟进项目进度
- 做一些测试

这个职位没有什么发展前景, 但是对公司是很重要的。因为需求是项目的成败所在。 好的产品经理可以很好的掌握需求, 让大家知道哪里应该发力。

所以, 对于产品经理要有好的定位 ( 例如让他来领导程序员), 这样的产品大神来主导 项目, 就可以了。

但是, 在一般的公司, 产品经理, 如果他跟程序员平级, 甚至隶属于不同部门的话, 就会 跟程序员有强大的对抗态度, 而且难于施展拳脚。

从个人的观点出发, 做好产品经理, 记住一点: 用户就是最好的产品经理。

这个职位, 一旦离开公司后没有什么前途。去创业做CEO的话, 是个不错的选择。



# 在技术的天空中留下痕迹

作为一个有追求的程序员，绝对不能啥痕迹都没有。多读英文文档，多参与翻译，多贡献github, 多参加stackoverflow.

## 必须要有技术博客，或者个人站点。

技术博客直接体现了程序员的表述能力和他对问题的思考深度。我们一定要把自己平时遇到的问题，踩过的坑，吸取到的经验，都统统记录下来。

我发现很多表述能力不佳的程序员（比如说话不能说出完整的句子，一个意思需要说出几个分句才能说明白），这样的人绝对没有写博客的习惯。因为所有擅长写博客的人，描述、表达问题的能力都很牛。

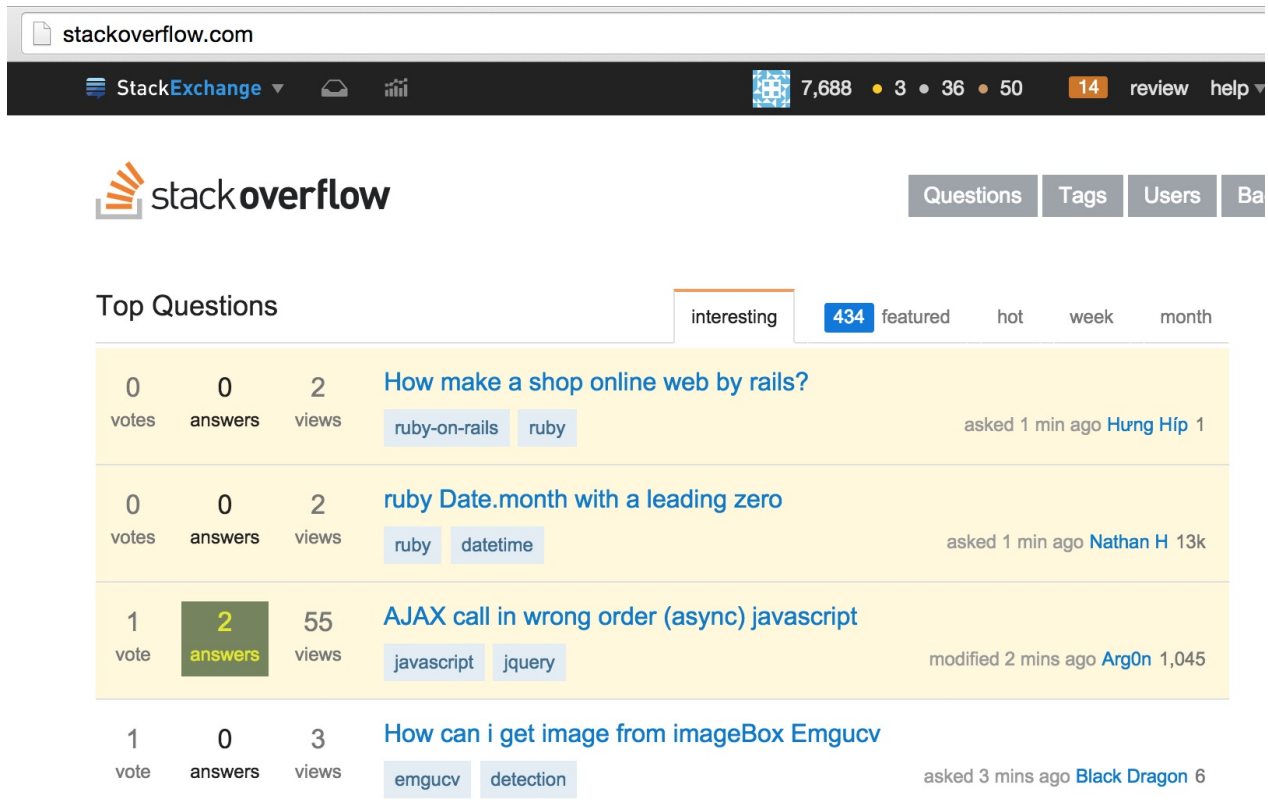
很多人说：自己是新手，怕自己写的东西没有技术含量。我说，怕什么呢？你写博客的目的应该是：

- 记录自己的经验。以后再次出现的时候，一翻博客就全出来了。
- 间接锻炼了自己的表述能力。如果一个人的表达能力不好，他在写出句子的第一时间就会发现这一点，因为自己是第一个读者。让他坚持写一段时间，表述能力就很快提高了。
- 是一个很好的名片。在外面做程序员之间交流的时候，告诉对方，我的个人网站是 myname.me, 是不是很有型？
- 技术博客直接体现了他过去几年的技术痕迹。面试官会格外青睐有技术博客的候选人。

所以，不要仅仅把问题记录在自己的小本子上。你会发现，把信息公布出来，只会给你带来正面的影响。不要敝帚自珍。聪明的人就算看不到你的信息，该知道的也会知道；愚蠢的人就算你直接告诉他，他还是不明白。

## 必须要有stackoverflow的账号。

stackoverflow.com 是世界上最权威最大的程序员问答社区。



The screenshot shows the Stack Overflow website interface. At the top, there's a navigation bar with the StackExchange logo and various statistics (7,688, 3, 36, 50, 14). Below the navigation bar, the main content area displays the 'Top Questions' section. The 'interesting' tab is selected, showing 434 questions. The list of questions includes:

- How make a shop online web by rails?** (0 votes, 0 answers, 2 views) asked 1 min ago by Hung Hip 1. Tags: ruby-on-rails, ruby.
- ruby Date.month with a leading zero** (0 votes, 0 answers, 2 views) asked 1 min ago by Nathan H 13k. Tags: ruby, datetime.
- AJAX call in wrong order (async) javascript** (1 vote, 2 answers, 55 views) modified 2 mins ago by ArgOn 1,045. Tags: javascript, jquery.
- How can i get image from imageBox Emgucv** (1 vote, 0 answers, 3 views) asked 3 mins ago by Black Dragon 6. Tags: emgucv, detection.

如果你在google上搜索 问题，排在前几位的绝对是 stackoverflow的回答帖子。

The screenshot shows a Google search interface. The search bar contains the text "vim indent by file type". Below the search bar, there are tabs for "网页" (Web), "图片" (Images), "视频" (Videos), "新闻" (News), "更多" (More), and "搜索工具" (Search Tools). The "网页" tab is selected. Below the tabs, it says "找到约 351,000 条结果 (用时 0.14 秒)". There are three search results listed:

- Changing Vim indentation behavior by file type - Stack ...**  
[stackoverflow.com/.../changing-vim-indentation-behavior-by-f...](#) ▼ 翻译此页  
2008年10月1日 - Could someone explain to me simply the easiest way to change the ...  
You can add **.vim files** to be executed whenever **vim** switches to a ...
- Setting Vim whitespace preferences by filetype - Stack ...**  
[stackoverflow.com/.../setting-vim-whitespace-preferences-by-fi...](#) ▼ 翻译此页  
2009年10月13日 - possible duplicate of Changing **Vim indentation** behavior by **file type**  
... autocmd **Filetype** html setlocal ts=2 sts=2 sw=2 autocmd **Filetype** ruby ...
- Indenting source code - Vim Tips Wiki - Wikia**  
[vim.wikia.com/wiki/Indenting\\_source\\_code](#) ▼ 翻译此页  
The **indent** features of **Vim** are very helpful for **indenting** source code. ... 'smartindent'  
and 'cindent' might interfere with **file type** based **indentation**, and should ...  
[Shifting blocks visually](#) - [How to stop auto indenting](#) - [Fix indentation](#)

问答记录可以直接看到这个人是否有公益精神。是否热爱程序员这个行业或者他所掌握的语言。另外，stackoverflow作为英文论坛，如果他能参与到里面的问答的话，说明这个人不但英语够好，还有足够的国际视野。这点对于掌握新技术，有特别好的帮助。

另外，我们每次google问题的时候，肯定发现这个问题很多人都在问。有的问题是没有回答的。如果你发现了这个问题的解决方案，不妨把你的答案放上去。予人玫瑰,手有余香，不是吗？

p.s. 我的stackoverflow账号，截止2015年11月29日。积分达到 7688. 很多很多的分数都是由于上面的原因得到的。

## 必须参与开源项目。

参与开源项目，说明了这个人：

- 对于自己的代码足够自信。因为烂代码会被人喷的。
- 有胸怀，具有公益精神。希望能够帮助到别人。这样的人在技术上才会做大做强
- 跟其他世界级的程序员有交流。这个人的技术实力和眼界一定比敝帚自珍的程序员开阔。

我与本书作者 刘明星 同学就是发掘到他在github上的开源项目。然后通过email联系到他。我的感觉是找到了个宝贝。哈哈



刘明星  
liumingxing

 <http://www.mamashai.com>  
 Beijing






 Contributions

 Repositories

 Public activity

 Follow

#### Popular repositories

	<a href="#">titanium_module_alipay_ios</a> 支付宝的titanium module for ios	5 ★
	<a href="#">titanium_module_jpush_android</a> 极光推送的module, 针对Android平台	4 ★
	<a href="#">TiMediaPicker</a> Media Picker Module (Image and Video) for Titanium Mobile	3 ★
	<a href="#">titanium_module_alipay_android</a> 支付宝的titanium module for android	3 ★
	<a href="#">NappJockey</a> Communication between Titanium and webpage running remotely	2 ★

# 引导客户明确需求

---

需求是需要引导出来的. 很多时候,我们会遇到 [一句话需求](#).

作为一线程序员,如果遇到含糊的需求,肯定会把它往某个方向想.(有经验的程序员偏向于想的简单,无经验的程序员偏向于想的复杂),跟用户真实的需求往往是不符合的.

(TODO 图片,对于不同需求下的人的反应)

所以我们要善于挖掘出客户的需求.

有趣的是,在跟用户的沟通中,我们发现,对于第一次找外包的人,往往是你问他需要什么,他的回答是"都要". 例如:

用户: "我要有用户注册功能" 程序员: 用什么注册? 手机? 邮箱? 最好都要 手机注册的话,要发送验证码吗? 我要 需要第三方的支持吗?比如 QQ,微信? 呃...要的. 需要对支持包,微博的支持吗? 对了!也要!

这种问答,直接体现出了用户对于技术问题的不懂,所以完全就是能要就要,能有就有. 所以,我们问完 这些问题后还不算完, 要告诉对方:

手机注册的工作量, 第三方登陆的工作量. 对微信/微博等支持的话,需要用户提供哪些材料(证照等)

让用户体会到这些功能都是需要消耗人力和财力的.

很多时候,用户的需求也往往[是一句话需求](#)

所以,我们必须让他们耐心的坐下来,把[心中的需求落实到纸面上](#),做到[需求可视化](#).

# 让团队散架的因素

---

## 团队毒药

---

跟“项目的催化剂”(见前面TODO)相对，有一种人是项目毒药，他在哪个项目，哪个项目就会失败。

这样人的特点：

- 脾气臭。导致别人不愿与他沟通。
- 能力差。导致项目无法自主完成。
- 不愿主动与其他人沟通。

直接导致别人无法与他沟通。

## 不公平的薪水

---

公司的薪水必须永远保密。连财务都不能知道。

但是很多职场新人特别喜欢打听别人工资。

永远告诉新人，公司的主要纪律有两条：

- 代码必须保密。
- 不能打听工资。

有的同学，入职一年，工资达到了8k。也有的同学，随着经济形势的提高，一进来就是7K。于是前者就会质疑：后者能力不如我，资格也不如我，干嘛跟我拿着差不多的工资？

也有的公司，为了挖人，直接拿到了别人两倍的薪水。这样的情况下，如果被别人知道，会直接导致低薪水的人离职。

所以，高薪挖人要慎重。会直接打乱自己团队的工资体系。对于公司的员工，也必须让他们有保密薪资的意识。

## 不开心的工作环境

---

根据[2014年的调查](#) 工资低，劳动强度大，上班远是跳槽的三大主因。

如果不看工资的因素，那么“不开心的工作环境”是非常重要的方面。

所以，公司必须以人为本，多为员工着想。例如：

- 雾霾天，为员工发放防霾口罩，办公室里增加净化器
- 员工生日时，为大家唱生日歌，发放生日蛋糕。
- 遇到父亲节，母亲节，替员工为父母发送礼物。

平时公司的气氛不要太压抑。

# 代码质量

---

代码质量是最重要

# 看起来的美好

## 屏幕自动适配

自适应的技术，能够提供在不同尺寸的屏幕上显示同样的内容。例如 手机上（600像素宽度以下）是一种布局，PC显示器上（1024宽度像素）是另外一种布局。

在实现技术上讲，就是一套HTML+CSS代码，让多种设备(横屏，竖屏，大屏，小屏)都可以适配。

目前的实际经验是：

- 要达到最好的效果，必须分别来实现。
- 要让小屏幕跟大屏幕的内容互不影响，否则容易按下葫芦浮起瓢。例如：在PC上明明显示非常正常的页面，放到移动端就显示不好。等移动端H5页面修改好之后，反观PC端页面又不行了。
- 太多时候，两个端的显示内容是完全不一样的。
- 适配也是有限度的适配。例如：
  - 移动web(wap)屏幕：一套代码
  - PC 屏幕：一套代码。（优酷的移动下载页面）

## I18N 一套代码，显示多种不同的语言。

i18n(internationalization): 能在不修改代码的前提下针对不同的语言来显示。下面的例子，是分别使用英文（en.yml）和 法文（fr.yml）向人问好：

```
# config/locales/en.yml
welcome: "Welcome, %{name}!"

# config/locales/fr.yml
welcome: "Bienvenue %{name} !"

> I18n.locale = 'en';
> I18n.t(:welcome, :name => 'Charlie')
=> "Welcome, Charlie!"

> I18n.locale = 'fr';
> I18n.t(:welcome, :name => 'Charlie')
=> "Bienvenue Charlie !"
```

- 让代码变得异常臃肿, 难于调试

```
<% if @promotions.any? %>
  Question promoted by <%= @promoter.name.capitalize %>
  to <%= @promotions.size %>
  <%= @promotions.one? ? 'person' : 'people' %>
<% end %>
```

- 很多语言之间是结构完全不同的。你很难把握翻译的粒度（是根据每个字来翻译？还是根据整个句子来翻译？）
- 无法保证其他语言的代码及时更新。

```
# Schema: group by category(log, notice, etc.), use interpolation(ie: %{item}), anything from user perspective(i
cn:
  name: 蛋白石(Opal)
  greeting: 你好, %{name}! #Hello %{name}! # call with t(:greeting, :name => "John")
  # Seed data used in Installation
  seeds:
    setting:
      site_title: 我的Opal网站.# "My Opal Website"
      site_description: 自由, 开源的内容发布网站 "The Free, Open Source, Item Management System. List Anything!"
    category:
      uncategorized:
        name: 未分类 # "Uncategorized"
        description: 这里面的东东太棒了。# "Things that are just too cool to fit into one category."
    page:
      banner_top:
        title: 顶部横幅 # "Banner Top"
        description: 这里的内容会放在侧栏的顶部。适合放广告图片, 或者 javascript. # "Any content added here will show at
        content: ""
      banner_bottom:
        title: 底部横幅 # "Banner Bottom"
        description: 这里的内容会出现在侧栏底部 # "Any content added here will show at the bottom of your site. Usefi
        content: ""
      terms_of_service:
        title: "Terms of Service"
        description: "The Terms of Service for new users."
        content: "<h1>Terms of Service</h1>By joining this site, you agree not to add or submit any damaging or
      new_item:
        title: 新的产品 # "New Item"
        description: "This page appears when a User is creating a new item."
        content: ""
      email_footer:
        title: Email 签名 # "Email Footer"
        description: "This appears at the bottom of any automated email."
        content: "This is an automated email. Please do not reply."
```

, 可以看出, 作者正在根据英文 来翻译中文。而且其中很多部分都没有及时更新, 显示的内容仍然是英文。

参考: [这个著名的ppt](#) 这篇ppt是我见到的把i18n的问题分析的最透彻的文章了。出现了更加可怕的:

- 语态 :



I saw the cats running away  
I see.PST the cat.PL run.PROG away

6 words : 9 morphemes

- 不同语言的单数、复数问题



Kitabu kimoja kitatosha  
One book will be enough



Vitabu viwili vitatosha  
Two books will be enough

- 在语言的表现能力上，把语言分成 analytic/synthetic, no/hella agreement

最要命的是：中国的软件公司不需要做国际化的东西：

- 99%的公司都不卖windows这样的国际化软件。
- 99%的外包公司都不会承接这样的项目。

所以，把精力专心放在做产品上，而不是翻译上。如果遇到一个开源项目，你需要 把它改造成你自己的项目的话，不要保留 i18n !

## 数据库的适配：

hibernate , rails Active Record: 都支持数据库的平滑迁 移。 支持多种数据库。

实际的困难：

- 不同的数据库，肯定会用到不同的语法结构
- 就算使用的全都是标准sql，在迁移时也会出现结构不统一的现象，例如：bool类型的列，在mysql下是boolean 在 sqlite3下面是 int(1)，需要太久去排错。
- 在国内(无论是软件外包公司还是互联网公司)，一个项目开始之后，基本不会更换数据库。

# 几分钟让外行了解软件开发

---

软件开发，是一个重度依赖脑力的行业。

## 客户特点, 上来就三句话:

---

- 能不能做
- 多少钱
- 多长时间

这三个问题都很难得到准确的回答. 因为几乎所有的销售(外包公司) 和 项目负责人(非外包公司)都会说: 能做! 哪怕他之前根本没接触过这类技术.

多少钱也很难获得准确的数字. 因为需要先估算工作量.

多长时间做完也难说, 因为需求不明确.

所以, 先清晰, 简单扼要的表达出你的需求, 对方才能给出靠谱的答案.

## 客户的第二个特点: 一句话需求

---

- 我要做个跟淘宝差不多的商城.
- 我要做个能聊天的app, 跟微信那样
- 我要做个能发帖子的app, 跟微博那样.

如果你的需求还停留在这个阶段, 赶紧细化一下! 具体见: [所见即所得设计法](#)

## 这个行业的项目成功率是不高的.

---

目前还没有一个准确的数据, 估计将来也很难有. 因为很少有人会自曝家丑. 更不愿意承认自己的项目是失败的而惹上麻烦.

## 很难量化工作量

---

例如, 在传统行业 (如家装) 中, 一个贴砖工人的工作量, 我们可以按照平米来计算. 铺砖到地面, 王师傅一天铺100平米, 李师傅一天铺120平米, 我们可以知道李师傅铺的好.

因为这个活儿大家都能做, 所以市场上的价格也比较透明. 一平米40块是标价。

但是软件开发中, 几乎每个功能都是自定义的。例如：

## 软件开发是重度自定义的。

---

曾经有前辈 (Rational rose) 提到过模块理论。在1980年代左右提出, 意思是20年后的 软件开发将会特别简单。大家只要使用 模块化的思想, 把软件中的一个个“螺丝”生产出来, 那么软件中的可重用的模块将会越来越多, 一段时间之后的软件开发工程师都不需要写新 代码了, 直接把现有的组件组装起来就好了。

甚至 Rational rose 的做法, 就是让架构师设计好架构, 一点按钮, 整个实现代码就自动 生成了。

当时这个思想风靡了软件行业。

但是这个想法经过现实的检验，是根本无法可行的。因为在当今的互联网浪潮中，每个用户都对软件的需求不同。

## 难于对软件系统做分解。

你是把你的功能分成：论坛，博客，CMS？这样的大模块？

还是把它们继续细化，比如，把论坛分成：

- 注册
- 登陆
- 发帖子
- 回帖子

两条路都难走。开源项目有很多，随便google一个CMS，会出现上千个开源CMS项目。但是只要你试用一下，就会发现，适合你的太少。要么页面风格不对，要么功能跟你想的 不一样。要是改的话，嘿嘿。你又掉到坑里了。

参考：[开源项目之坑](#)

所以，我认为软件项目最多细化到“螺丝钉”的层面，例如：

1. 开发框架, 例如：rails
2. 最常见的组件，例如：上传功能，第三方登陆，支付(例如支付宝)

99%的内容都无法重用。就算是同样的一个财务软件，在不同的公司用起来都完全不同的。

## 万能的办公自动化系统？难以重用！

---

曾经有朋友咨询过这样的需求，希望做这样的事：

发出一套万能的办公自动化系统（Office Automation，简称OA），然后卖给各个企业。这个想法特别符合国家的政策。

想一想就是不可能的。比如，同样的一套财务软件，在两个公司内用起来就完全不同。有的公司是实报实销，有的公司是先借款。

而很多专门开发财务软件的公司，都会专门为用户组织培训，让他们使用自己定义好的流程。

所以千万不要幻想开发出一套万能的办公自动化系统, 这个事情永远不存在。

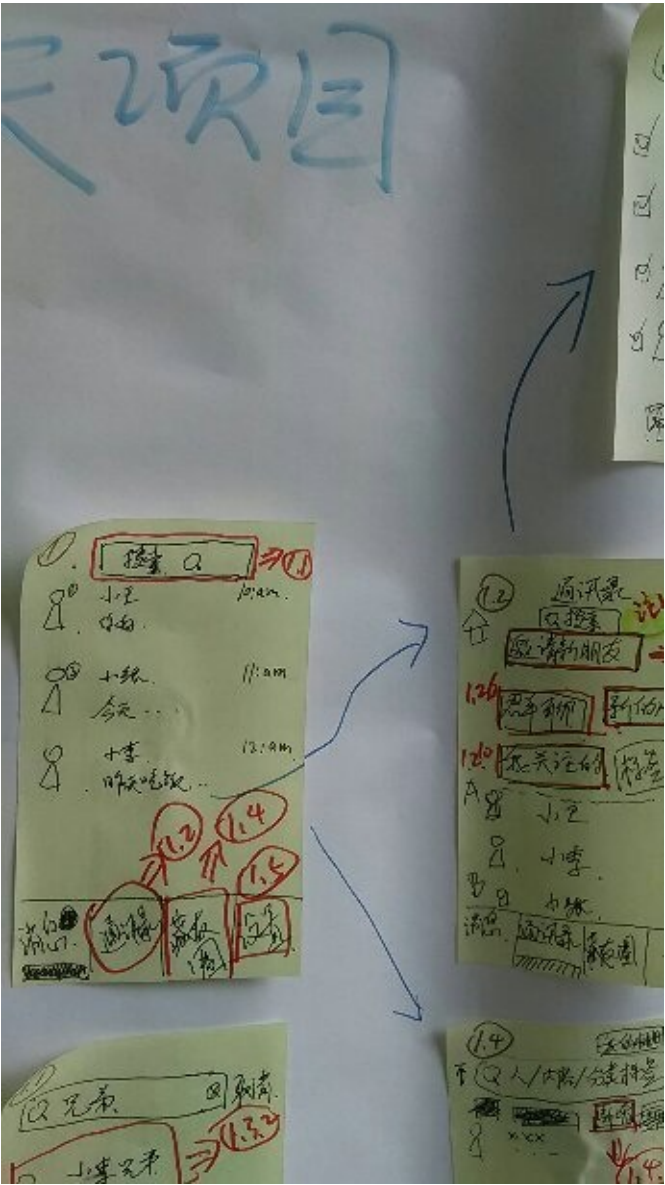
# 所见即所得你的设计

## 概述：

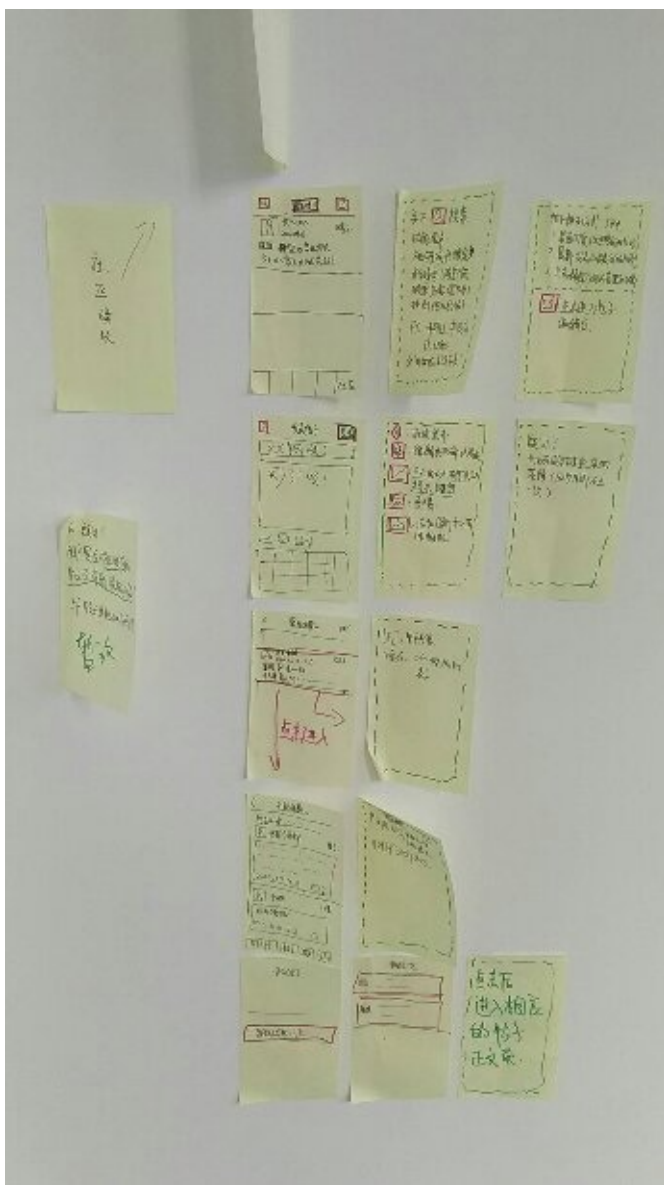
我们团队做产品设计的方式，是先把原型图做出来。

如下面三个图片：

先准备好一张大白纸和黄纸片，然后，快速的手绘项目截图：



有多少个页面，就花多少个，记得标记出每个页面的跳转关系：



经过美工的润色之后，草图就变成这样：



所以,你那边只需要按照这个模式:

1. 整理出所有的功能页面。
2. 标记出页面之间的跳转关系。

即可。我们就可以针对每个页面估算时间,最后得出累计成本。

下面是具体的说明。

这种建模方法,在软件咨询公司 Thoughtworks被称为 Inception (盗梦空间)。我给它起的通俗的名字叫黄纸片建模法。原理是UI驱动设计。

现今有很多电子化的工具可以用,为什么我们要回归原始呢?

我从业十多年,各种各样的建模工具都接触过。UML, RationalRose, VisualView, 在线设计软件等等。不建议使用。

因为电子化办公有很多缺点:

1. 效率低下。

所见即所得设计法



2. 屏幕太小, 无法让人快速理解
3. 越是精心制作的内容, 作者就越不愿意修改. 而我们的目的是拥抱变化. 所以, 有变化我们就改, 拿起笔来就写, 不满意就撕掉. 分分钟的事儿.
4. 我们可以把大白纸悬挂在最醒目的角落. 需要的时候可以随时看. 而电子化的东西则不可以.

这套快速建模的方法, 基本 2~3 个小时就可以把“一句话需求”转化为具体的产品. 然后借助一些工具 (例如 墨刀) 来定型. 直接去讲故事.



1. 准备一张大白纸. 胶水. 黄色便签纸 (需要长条形)

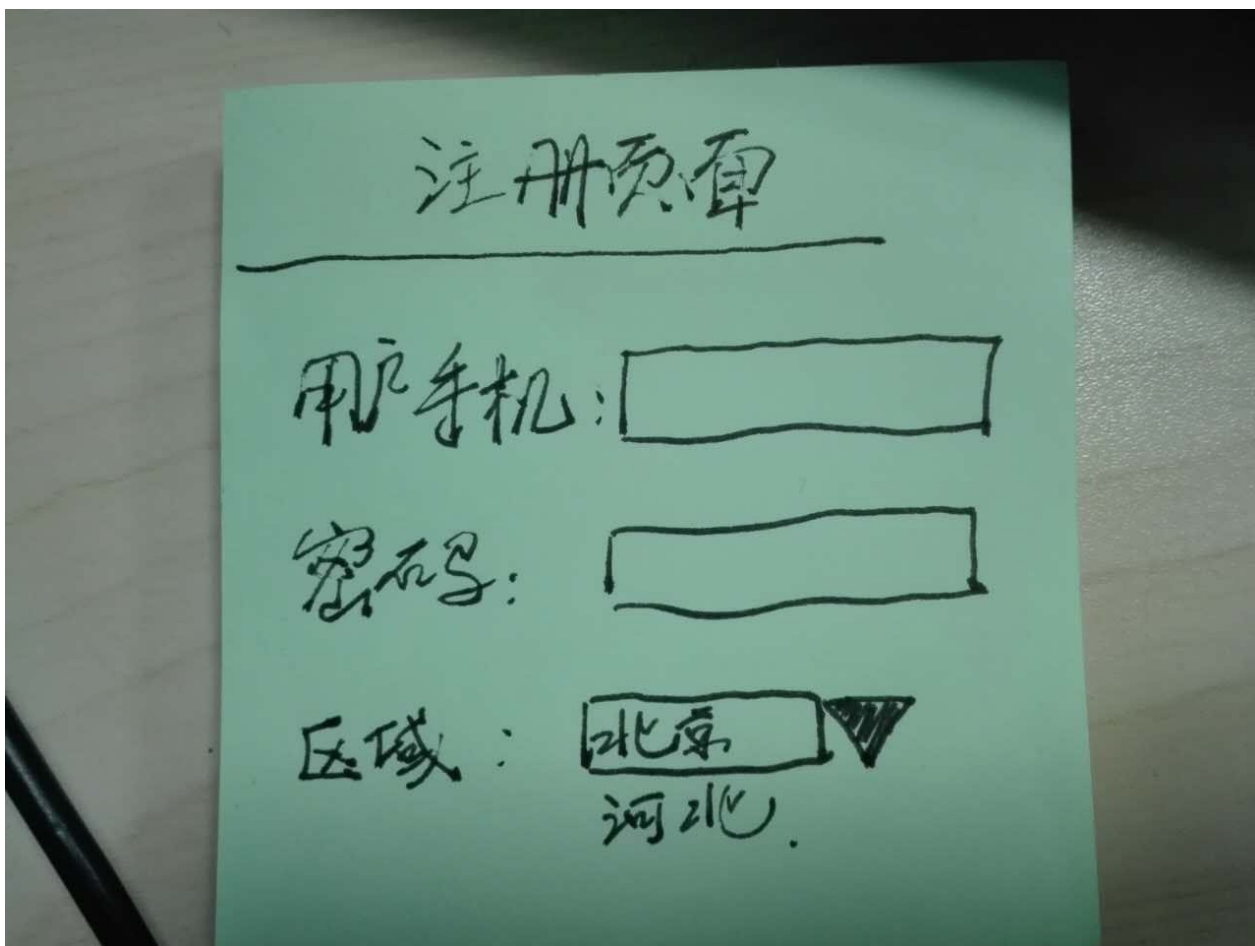
大白纸: 所有的线和字都写在上面. 白纸能重视的记录所有信息. 不用画板的原因是画板一擦, 内容就全没了. 胶水: 为了把便签纸贴牢. 我们在实际操作中发现 便签纸会扭曲变形, 而且沾上去很快就掉下来. 便签纸: 购买黄色的. 视觉效果比其他颜色好很多. 而且记得买 长方形的那种. 横着放就是电脑屏幕, 竖着放就是手机屏幕.

1. 准备笔: 黑色细笔, 红色粗笔, 绿色粗笔. 约定: 黑色细笔 用来描绘 页面的基本结构 红色粗笔: 用来标记页面的跳转 绿色粗笔: 用来表示注释

1. 页面的结构:

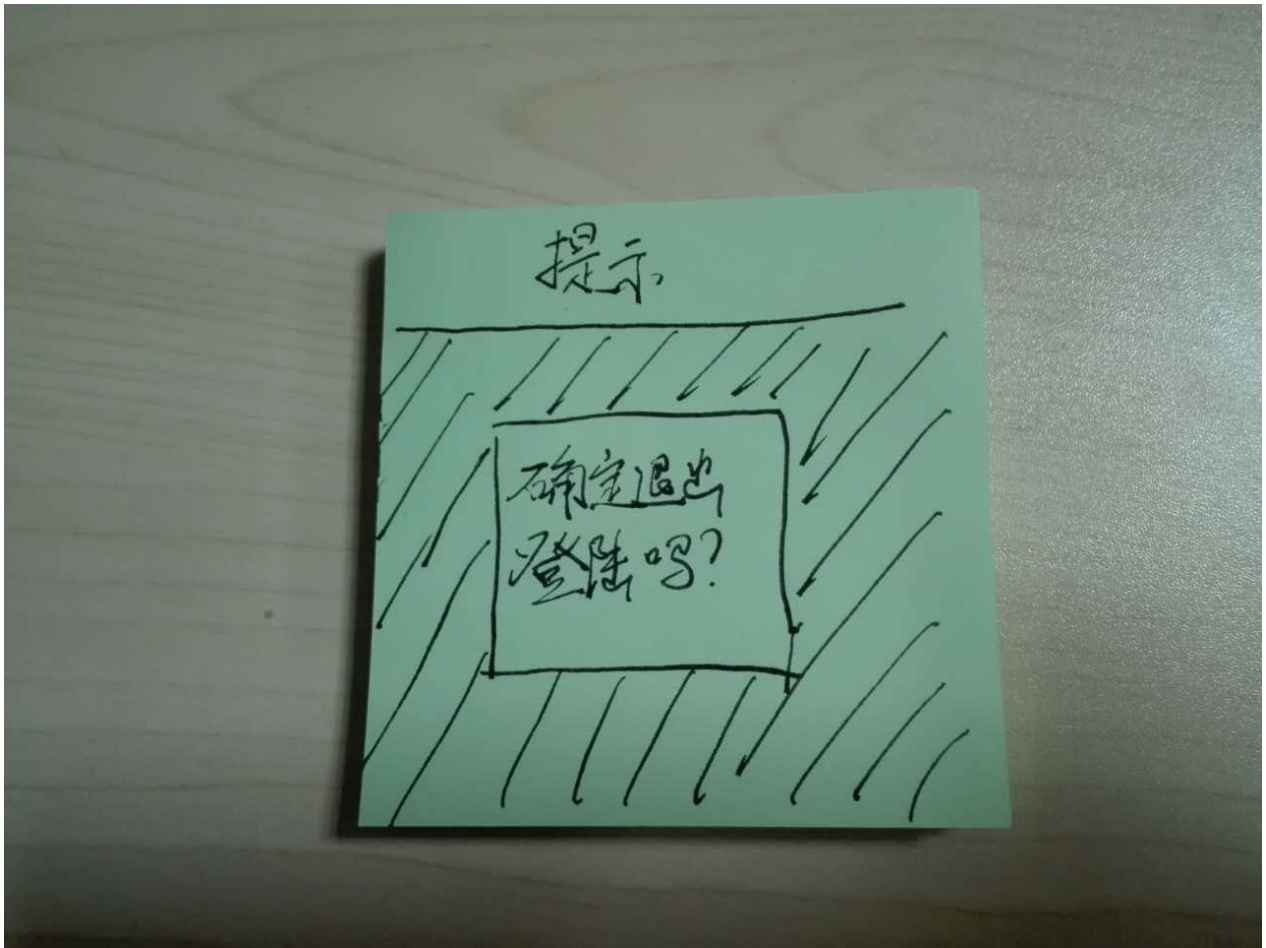
3.1 每个页面都是这样组成的: 页面名称 + 内容. (见图片) 3.2 使用方框, 表示输入文字 3.3 下拉选择框应该是: 一个黑色的, 向下的三角:

下面的例子, 具备了: 标题, 输入框, 下拉选择框.



3.4 消息,警告等,应该有阴影:





还有几个例子:

< 返回

群聚会提醒

群组头像

奥迪车主

报名者(1) 头像

>

+ 报名者(1)

返回

报名成功

报名成功后4S店工作人员会联系您，向您介绍团购相关信息。如果您有任何车的问题也可以通过团购咨询向工作人员问问。

附近的群

有了车怎么玩？看看附近的群吧，加入车友的大家庭。

< 返回

群聚会

报名者(1)

头像

刘冬 25 车标

oookn

回签名

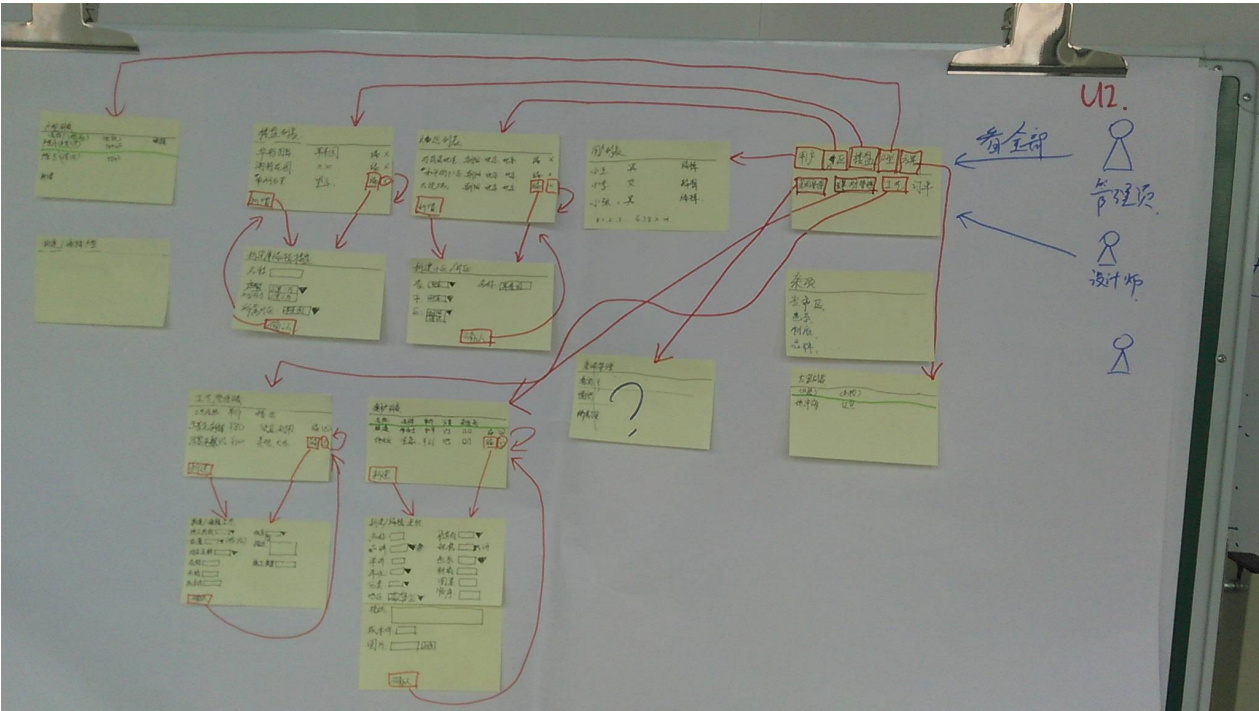
名片

所有可以点击的按钮,都要用红色方框划上,然后标记好它的下一个页面.

1. 大白纸的结构:

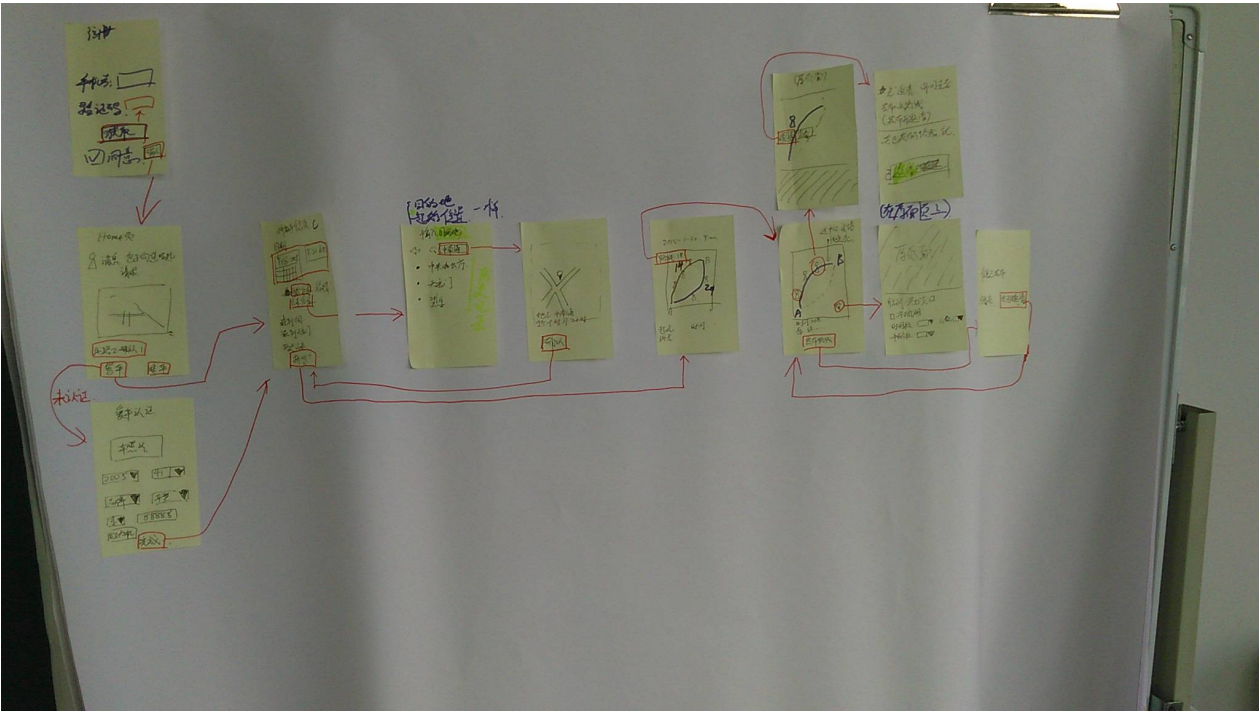
原则上, 从上到下, 从左到右: 左上角是一个用户图标,表示操作的开始.

附件是一些例子.



下面是 2015年4月10日,我们做的一个ipad端app的 页面逻辑图. 可以看到红色的清晰的箭头, 也可以看到 绿色的注释, 左上方的 蓝色 用户图标,是app的起点.

下面是另一个系统的跳转逻辑:(类似于滴滴顺风车)





下面是一个更加完整的例子:

左侧是无线端app, 中间是 PC端管理员后台, 右侧是 服务器的数据库设计.



## 几点注意

- 实际当中,我们发现要让程序员参与进来效果最好.也就是说,谁亲自 动手,谁对整个流程了解的就越深入.
- 一定要用红粗笔来标注页面的跳转. 站在一米外都可以看的很清晰.
- 约定好颜色: 红色表示页面的跳转,绿色表示注释. 不适使用三种以上的颜色.
- 不要连笔字, 字迹要清晰. 因为这是给其他人看的.连笔字会显得效果不好,难于辨认.而且有一种乱乱的感觉,这是我们最应该避免的.
- 表意要明确. 内容要是"具体的例子",而不是概括的说明,例如,在注册页面,应该是:

手机: 13522223333 密码: \*\*

而不是

项目1: xxx 项目2: xxx

- 黄纸片要贴的横平竖直, 例如, 大致都分布在同一横线或者竖线上.
- 只能省略第三方的页面,例如:QQ登陆,淘宝支付, 手机拍照等. 否则的话,再简单的页面也不要省略. 例如: 忘记密码等. 页面越齐全,越能准确的估算工作量.
- 标注好页面的入口(例如从登陆页面开始)

## 一般分成两端

- 手机app端
- 后台管理员(PC端)

如果是B2C的系统,那么就是:

- 普通用户的app

- 商家的app
- 后台管理员的PC端
- 商家的网站.

## 估算工作量

---

分成两种:

- 画出的页面的工作量, 这些在上面已经看到了.
- 对于app, 有看不到工作量, 包括:
  - 消息推送
  - 不同机型和屏幕尺寸的适配

# 小步快跑

---

#

---

签订合同要小而多，频繁沟通才能保证项目失败，兼职的项目都会死掉。

## 本土软件公司的特点

---

哪怕是外企，到了国内也会沦为本土化。具有下面的特点。因为外企虽然有外企的文化，但是归根到底，执行的人都是国内的人。

### 技术含量低

---

技术普遍落后国外。

### 人员素质不高，普遍英语不好

---

英语

要么是外包公司，要么是互联网公司。

---

外包公司大部分都比较烂

---

培训的人的水平太差。归根到底还是英语能力。

---

大公司，里面的软件部门的技术很小，其实跟小作坊差不多。

---

国内的程序员也容易安于现状。

---

# 开源项目之坑

---

开源项目的祖师是 Eric (? TODO详细) 他最先发起了开源项目。 所谓的开源，就是开放源代码(open-source) 的项目。

世界上的开源项目有很多，细分的话分成两类：

1. 工具类, 例如： 各种框架，组件等等。
2. 现成的产品。 例如：各种论坛，博客，CMS，商城。

很多人在创立公司时，会考虑到成本问题，优先采用开源的“现成的产品”， 我听到的 最多的话是：

“我们要做个商城功能很普通，我们就用网上开源的XX商城，随便改一下不就完了么？”

一般说出这个话的人，身边肯定没有经验丰富的老鸟。他很可能是随便找一个经验不太丰富 的程序员，来修改。

于是，花上很短的时间（例如2，3天），这个开源的XX商城就搭建起来了。负责人很满意。 于是提出的问题是：

“很好！我们的这个产品不错！，已经出现了雏形！” “但是，这个背景色要调整，这里的搜索功能不应该是这样的，应该多加几个品类； 这里的用户在查看某个商品的同时，我们也要多给他推荐几个产品” “现有的功能是B2C，只能支持一个商家。这样不行，我们要支持多个商家。” ...

于是，他会发现，修改外观很简单（只涉及到CSS,HTML），但是一旦涉及到修改核心功能， 现有的技术人员就会束手无策了。因为底层架构是不那么好改的：

- 需要先读懂人家开源项目的代码
- 理解人家的代码
- 尝试修改， 出错， 再修改

一般开源项目的代码，都是由行业中的高手写的。新手会看不懂。比如，很多高级 的语言特性，不常见的设计模式，以及各种抽象，新手看到完全就懵了。

于是，这个公司的技术负责人就会发现一个奇怪的现象：项目直接停滞了。什么时候问道 进度，都是没有进展。

到最后的结果往往是：

1. 放弃现有的改动了一半的开源项目，推倒重做。
2. 现有的技术人员离职。

## 开源项目的特点

---

1. 高手写的。 会用到很多高深的技巧。例如，ruby中的元编程，老牌语言（java/c)中的 设计模式。
2. 有些开源项目经历的时间比较久，例如Sugar CRM.(成立于2006年), 里面用的技术 有老的掉渣的组件，也有特别崭新的组件。负责开发的也不是一个人。你会发现一个奇妙 的现象：一个简单的“展示菜单”的功能，有的地方代码写在了数据库，有的代码写在了.js文件中， 有的代码写在了.php文件中。 特别混乱。
3. 要修改开源项目，特别花费时间。 因为你需要先读懂别人的代码，还要了解别人用到 的所有技术和组件。

所以，如果只是为了演示，可以用开源。 如果该项目后期会有很大的想法，那么赶紧自己开发。



# 如何找到靠谱的CTO？

---

这个问题单拎出来，是因为很多人都问过我。

- 有个技术圈子的朋友
- 让这个朋友给你介绍靠谱的人
- 要有技术评估的机制。

## 有个技术圈子的朋友

---

很多朋友都有资本，有资源，就是没有靠谱的技术团队。

因为他不知道如何去寻找。他不懂技术，就没法判断对方的好坏。

所以，首先，要想尽一切办法，认识一个技术高手。找到了他，就可以慢慢的 把他的朋友圈子技术高手介绍给你。

然后，提醒他，给你介绍朋友的时候，要靠谱的。

## 靠谱的CTO，可遇而不可求

---

跟现在的创业市场一样，技术人员也鱼目混珠。

比创业稍微好些的是，技术人员一般比较单纯，能走出来创业的，技术水平要高于打工者的平均水平。所以不必担心找到的人是烂人。

## 绝对不要找兼职的CTO

---

兼职的CTO做不好事：

- 他本身就有全职工作。每天还要做兼职，时间是远远不够的。我之前在摩托罗拉 工作时，每天大约有5小时的空余时间。但是一旦做了兼职，我突然感觉每天除了 吃饭睡觉就是工作。你需要考虑到：
  - 在全职和兼职中的工作切换
  - 你每天的娱乐时间(看看报纸新闻)是必不可少的
  - 沟通是个大问题。我无法随时随地接兼职老板的电话。
  - 由沟通产生的信任问题也是个大问题。
- 与全职工作相比，兼职仅仅是个锦上添花的事情。捞一笔外快。如果出现了项目 要死掉，看不到完成的希望的话，兼职的技术人员会很快想到：大不了我不做了。反正剩下的钱大不了不要就是了。对我也没有太大的影响。但是对这个项目的其他 全职投入的人来说，项目死掉是个极其沉重的打击。

## 宁可下重金，便宜的钱留不住人。

---

在软件人才市场上，一分钱一分货。一个实习生做不了任何事情，反而会拖慢公司的后腿。一个入门级的程序员也只能做搬砖的工作。基本有他没他项目都行。有了2,3个项目经验的程序员，可以继续做一些工作。适合做小弟。但是要多指导 有3,5年工作经验的程序员，这是公司的技术骨干，必须留下来。

现在（2015年12月），北京的软件人才的工资特别高。一个刚毕业的应届生都可以 拿到10K以上的月薪，还是在非BAT的创业公司。对于BAT的抢人就更不用说了。

## 简历中的尴尬

---

对于创业公司，招人也很难。

如果你用的是大众语言: java/php/.net/object c, 那么你可以收到很多应届生的 简历。如果你用的是python/ruby/perl/node等小众语言，就只能自行培养。因为

不过怎么样的语言，如果你不是在一个BAT量级的公司中，就基本只能收到2年以内 经验的程序员。

所以，CTO的另外一个责任，就是要自己能培养团队。

## 培养的途径

---

- 搭建社区，吸引高素质人才 比如，每月一次的社区聚会。是CTO发掘到场新人的好机会。如果你的公司场地 比他目前的公司好，你给的offer也更高，而且他发现你这里的工作内容和环境 都更好，为什么不会来？
- 在校园招聘中，挖掘到有潜力的新人。 华为，百度，阿里，都有校园人才贮备的战略。好的人才：学习能力强，人品又好。

## 把握好你的CTO

---

CTO一走，对公司的损失就大了。

特别是现在创业的，都是互联网创业。没有CTO，什么都做不成。只能做回传统企业。

CTO走掉之后，公司的损失包括但是不限于：

- 技术团队很可能走掉。特别是如果整个团队都是CTO搭建的话，CTO想挖走现有的 团队很容易
  - 现有的团队会特别认同CTO
  - 下个CTO会主动开掉现有团队的技术人员
- 下个CTO会使用新的技术，导致现有的技术人员的技术无法使用。例如，前CTO 用php，新的CTO用java，要不要把所有项目推倒重来？一个公司里，绝对不要同时 存在两种以上的语言。否则对于系统维护就是一个大灾难。因为
  - 多语言开发会导致公司的技术无法交流
  - 多语言会导致系统难于维护和部署
  - 多语言会导致技术成本升高

## 技术团队:少而精

---

少而精的技术团队才是有战斗力的。因为：多人团队需要：

- 人越多，工作量就越容易划分的越不公平。因为很多工作是无法划分的。当有人因为没有新工作空闲起来时，就容易对团队产生负面影响。 让他慢慢的变成项目毒药。而他也会因为自己游离于团队骨干之外而不安。

比如：别人都在干活儿，就我没活儿干，我要不要打游戏？要不要逛淘宝？ 要不要玩论坛？

比如：项目骨干看到有人在玩论坛，他会不会想，为什么我干的最多，但是 工资上却没领先太多？

- 人越多，沟通的成本就越大，效率就越低。 良好的团队最好只有3~5个程序员。 如果团队中有3个精兵，你会发现他们做的事儿比20个平庸的菜鸟都好。

## 正视技术的作用

---

### 短期内被高估，长期内被低估

---

短期内程序员可以为你救火，长期内肯定不行

### 就差一个程序员了

---

<http://www.zhihu.com/question/22989105>

### 2015年程序员的成本

---

# 好的程序员与差程序员的差别

---

## 第一次和第N次的区别。

---

对于程序员来说，存在：

第一次用某个技术时，特别慢，因为他一点儿不懂，需要学习。典型的有：

- 用户注册/登陆
- 上传文件
- 单点登陆

一旦第一次学会了之后，第二，第三次他再遇到时，直接复制粘贴当时自己写的代码即可。

所以，第一次使用某个技术所耗费的时间，如果是10，那么第二次以后的时间，是1或2.

第一次是学习，第二次就是搬砖

所以，大家要把握好心态。尽管很多时候儿你认为自己在搬砖，但是你没办法避免这个问题。做好本职工作是一个人的职业操守。

## 好的程序员都是靠项目磨练出来的。

---

软件项目没有捷径可走。不是会了几种算法之后，就能从一个新手晋升为一个高手。

按照我从业十年的经验来看，难住我的，都不是核心功能。而是一些边缘性的东西。这些边缘性的知识你无法把握住他的主线，只能出现一次解决一次。例如：

- 某个上传组件中的按钮样式
- 某个组件是应该出现在屏幕的上方还是下方
- 某个表格的边缘的线的粗细不一致。

这些问题看起来会特别奇怪，非常不高大上，但是这些BUG就是最高级别的，你必须搞定它们。

所以，不要指望几次培训，就能提高程序员的能力。也不要指望看完一本书，自己就能完全掌握某门技术。

必须靠不断的做项目，来磨练自己。一般说来，做WEB开发，你能独立实现一个博客，一个论坛，就差不多了。对于mobile开发，能做2，3个app也就出徒了。

## 程序员永远会遇到新问题。

---

google才是你最好的老师，兵来将挡水来土掩。

活到老学到老，这句话用在程序员身上没错。

因为你现在掌握的常见技术，会很快被新技术取代。

如果你的学习能力不行，如果你的英语不行，靠吃老本的花，那么你做不了程序员。

## 核心的技术变更的比较慢。

---

比如MVC架构，现在的Rails跟十几年前的java struts框架是一样的。 比如request/response 这种http的基础，是一点儿没变的。 比如持久层，现在的各种主流框架跟 Hibernate 是一样的。

你现在学到的任何东西，只要是核心的技术，肯定是不过时的。也许30年后，大家 还在谈MVC架构呢。

为什么同样的新技术，老鸟上手就比新鸟快？就是因为老鸟的以往的经验会给他很大的 帮助。他一看到某个技术，似曾相识，不过是包了一层新外衣而已。上手自然特别快。

## 80-20 定律

---

一个技术中，20%的内容是核心技术，它会出现80%的地方。

按照我看到的情况，真是这样。20，80代表了人脑中的（相对很少/很多）这样的概念。

所以，大家完全不要被浩如烟海的技术文档所吓倒。也许看起来厚厚的一本技术书，你 学会其中30~50页，就可以上手干活儿了。 其他的几百页内容是用不上的，到时候 随用随学吧！

# 外包的乱象

---

这个市场太乱。根本原因是靠谱的人不多，忽悠的人不少。

## 行业门槛低

---

有很多手工作坊。很多人还在大三大四，找到了几个开源项目，稍作修改，说是自己的 就开始出来接活儿。

这样的人对价格的期待也不高，几千到几万都做。能赚点儿是点儿。 他们也不会对客户的要求从零做起。 做之前，都是什么都可以做。 签了合同之后，就不是了。 开源项目中存在的功能，就可以给你做。 如果没有这个功能，就没办法了。

不要幻想几千块的项目就要求对方给你定制化开发。不可能的。成本摆在那里。 到时候他给你的解决方案，永远都是用现有项目的功能给你套用的。

到最后你的解决就是：妥协 或者 终止合作。

两个结果都对他有利：他拿到了预付款，做不了就跟你失联，反正你也找不到他。

## 绝对不要找外地的

---

见 [沟通至上](#) 沟通是对项目的最好保证。

## 绝对不要找太便宜的软件承接方。

---

互联网创业，绝对不要找太便宜的人，和团队。跟其他行业一样。软件行业也是一分钱一分货。出的钱高，就能找到靠谱的人或者团队。 出的钱低，那么获得的软件质量也不敢恭维。

案例：某位朋友，需要做的项目量级上千万。但是在软件投入方面仅仅是10万。 结果对方无法保证项目的按时交付。

深层分析：在北京，一个3年经验的程序员月薪假设1W， 那么在一个正规的公司，每个月要为他消耗2W。 如果某个项目，需要5人月（就是需要一个人干5个月。 人月：是衡量软件工期的单位。）那么它的成本就是10W。 如果你的项目以低于成本价的价格 外包出去。那么80%是失败的。 因为对方是一个连成本都不会估算的软件承接方。不要指望他能很好的控制工期进度。

## 经验：明确互联网在自己项目中的位置。

---

很多朋友，在自己的项目中，不太明确互联网技术的地位。比如：某个项目，又有线下的实体商店，又有线上的 网店，有移动应用APP。那么，该如何确认自己的项目是否是一个“互联网项目”呢？很简单。 你就假设自己没有移动APP，没有线上的网店，你看看这个项目能不能正常运转。 如果不能的话，你就要小心了。 如果你为自己的项目整体估值1000W。 那么你务必不要把自己的互联网技术上的预算弄得太低。

## 经验：一个靠谱的技术开发团队的运营成本。

---

什么样的团队是靠谱的？

- 诚信，不忽悠客户。
- 有扎实的技术，和成功的案例。

一般来说，这样团队的组成是：开发人员（2，3名）+ 测试人员（1名）+ 项目经理（1名）+ 运维人员（1名）

而一个互联网项目，给这个团队做的话，最少要做6个月，要做成熟的按照1年时间来算，5人 X 12 个月 X 2W。 = 120W

如果考虑到这个项目除了PC端，还要上移动端（只考虑IOS，ANDROID，不考虑winphone等），那么开发人员还要增加2~4名，

这个时候的成本就是：170 ~ 220 W.

结论：如果没有足够的干粮（比如说200W）做好未来一年挨饿的准备，那么团队是养不起的。

## 经验：外包项目与自己培养团队的比较

把项目外包出去是一个不得已的选择。如果资金允许，务必要培养自己的团队。因为在软件项目中，“人”才是最重要的。代码不重要。只要有高素质的人存在，那么你的项目就会一帆风顺。我见过的比较多的情况是：某项目，干了1期。交付了。然后开发人员消息了（可能是他离职，可能是该团队解散，公司关掉等等）然后来了一个人，继续接手。如果这个人是个经验丰富的老手，可以看懂前任程序员的代码，那么他大约需要一段时间来很痛苦的阅读和理解前任的代码。同时做各种修改，各种重构（改善现有代码的结构）。如果这个人是个菜鸟，前任作者的代码他看不懂，那么完了，估计前任作者30分钟的工作，这个新人可能要做一周。（在《人件》以及《软件工程的45个事实和悖论》中有更加精彩的论述）。

所以，除非你的项目不需要谁维护，否则不要外包出去。（我所有见过的成功交付的项目，都会要求做第二期，第三期。。。）

外包的唯一优势就是：我们不需要一直“养”着一个团队。需要人的时候我直接找外包团队来做就可以。如果你手头没有足够的资金，但是又需要在1个月内做出一款线上的产品的话，确实一个可行的选择。但是如同“壮士断腕”，它的副作用相当巨大。可能你在做第二期时候把前期外包的成果“推倒重来”。——原因是外包软件的整体架构不被新团队所理解，而且你也找不到原来作者了。

结论：能不外包就绝对不要外包。一旦做了外包，就做好承担更大失败风险和日后还债的准备。

## 经验：如何保证你的项目进度？

务必要让程序员过来跟你一起工作。最好是面对面，或者坐邻居工位。能够有个开放式的环境，大家围坐在一起更好。程序员有了问题可以直接问项目经理，项目经理想查看进度时，也可以面对面找程序员，大家还可以有每天上午10点的站立会议（注意：要站着，每个人发言不超过1分钟，讲述自己今天要做的事情和昨天遇到的问题）每天都做交付（见敏捷开发的“项目自动化”）。做到一键部署，这样我们的产品经理每天下班前都会看到哪个哪个新特性上了线。

## 经验：产品经理如何提需求？

要“小步快跑”。比如，老板脑子里有100个点子，而这一百个点子在目前的项目资源中是无法得到全部分配的（通俗的说，我们目前只能在一个月内完成5个点子），那么就把这100个点子做个排序：第一期项目先做这5个点子。等项目一期做了交付之后，我们再加上另外10个点子（或者需求）。。。等等等等。这样小步快跑，项目才能成功。

《人月神话》：如果某个项目的时间估算（交付时间）超过了1年，那么它基本上会失败。如果超过2年，那么绝对会失败。

## 绝对不要贪图便宜

很多没毕业的学生都会自己建个工作室。绝对不要因为便宜就找。风险极大。如果你的项目是花5千块请的大四学生做的，那么它肯定会在2个月内垮掉。

## 自有团队

---

自有团队最大的好处是：能够召之即来，马上就用。不像找外包那样不稳定。缺点就是比较贵。

## 开发初期的费用

---

2~3年的web程序员：基本在10k~ 15k 2~3年的app程序员：基本在10k~ 15k 2~3年的产品经理：基本在10k~ 15k 2~3年的UI设计：10K 以上 技术小组长：20K起。

这样算来，web程序员 + ios + android + 产品经理 + UI + 技术负责人 各一位，每个月工资在 6W- 8W。

随着公司的发展，你会发现有更多的人需要你招募，例如：产品经理，测试人员，运维人员，数据统计人员。。。

## 自有团队的好处

---

- 可以召之即来，有需求就用。
- 比外包团队稳定的多。
- 方便交流。比如大家都在同一办公室工作。对整个项目的把控更加稳定。

## 自建团队的关键

---

技术负责人最重要。

可以说，技术负责人是一个种子。整个公司，就这个人最懂技术。技术团队肯定要他来 组件。如果这个种子好，能够慧眼识人，那么整个技术团队的质量就会很好。如果这个技术种子滥竽充数，那么这个技术团队也都是平庸的人。

因为，在招聘的时候，面试官很容易招聘不如他的人进来。80分的人招进来70分的，70分的人招进来60分的。

## 如何招聘？

---

对于应届生：

- 可以是一张白纸
- 英语要么过6级，要么能够做基本流利完整的英语对话。
- 性格必须开朗

在实际的招聘中，我发现多子女家庭的孩子性格普遍更容易相处。

英语好，我们就可以很好的培养他。我们用的技术是国外流传很广，但是在国内使用不多 的技术。(Rails/titanium)，几乎所有的文档都是英文的。

英语好，也可以让这个同学遇到问题时更好的google, 解决问题。

性格开朗，直接关系到整个项目的成败。

对于社招生，则要求务必严格。除了上述的英语能力，性格，还要有专业的经验。如果无从判断的话，有个方法很靠谱：看他过往留下的技术痕迹。

## 由技术痕迹判断一个人最靠谱

---



面试仅仅是一面之缘，多说一两个小时。根本无从观察这个人的实际经验。所以，我们必须通过其他方法来考察他。最好的办法是：

- 看他是否有技术博客，有的话，仔细查看里面的内容。技术博客直接体现了他的表述能力和他对问题的思考深度。我发现很多表述能力不佳的程序员（比如说话不能说出完整的句子，一个意思需要说出几个分句才能说明白），这样的人绝对没有写博客的习惯。因为所有擅长写博客的人，描述、表达问题的能力都很牛。另外，技术博客直接体现了他过去几年的技术痕迹。这些东西完全可以作为面试的补充。
- 看他是否有社区的问答记录，例如stackoverflow(专门的程序员问答社区) 问答记录可以直接看到这个人是否有公益精神。是否热爱程序员这个行业或者他所掌握的语言。另外，stackoverflow作为英文论坛，如果他能参与到里面的问答的话，说明这个人不但英语够好，还有足够的国际视野。这点对于掌握新技术，有特别好的帮助。
- 看他是否参与过开源项目。例如github上的项目。我与本书作者 刘明星 同学就是发掘到他再github上的开源项目。然后通过email联系到 他。我的感觉是找到了个宝贝。哈哈

参与开源项目，说明了这个人具备两个能力：

1. 对于自己的代码足够自信。因为烂代码会被人喷的。
2. 有胸怀，具有公益精神。希望能够帮助到别人。这样的人在技术上才会做大做强
3. 跟其他世界级的程序员有交流。程序员不能敝帚自珍。

上面三条任一具备即可。

## 与家装的比较

---

我用了2个月的时间去专门研究家装。下过工地，读过很多资料，跟业内人士交流，发现家装与软件开发特别像。

## 都是复杂的流程

---

家装：业主找到公司，设计师交流设计，主材进场，工人施工，验收，交付。

软件：用户找到公司，业务分析师分析需求，架构设计，编写代码，测试，交付。

软件中的业务分析师与家装中的设计师几乎是一样的。

这个流程最初都没有这么复杂。家装：工人没有什么文化，泥腿子。软件：都是手工作坊，一个人就从设计到编码都做了。

后来，两种行业都出现了公司：软件外包公司和家装公司，都需要有人专门精通一种业务，所以出现了不同的角色：

家装：谈单师（设计 + 谈单 + 出图），工长、工人，监理。软件：UI设计师，业务分析师，程序员，测试。

都是几乎一样。

## 都主要靠人的技艺。不是靠流程

---

为什么家装和软件都难以做好？

软件外包公司，有东软等。国外的有thoughtworks等。没见到有能大规模批量生产的。家装公司，如东易日盛，实创，业之峰，做了十几年，口碑做的好的也不多。

我觉得，这两者，到最终干活儿的时候，最终决定质量的，都是由人的素质决定的。家装公司是工人、工长，软件公司是程序员。

无论你的流程设计的多么合理，多么严格，但是到最终实施的时候，第一线的施工者能力不足，（代码质量低劣，贴砖时工艺不行，墙面贴的外起裂吧的，）再好的管控，再牛的监理/测试人员，也无法给出合格的结果来。

而对于能力强的第一线人员，不管外部的管理是否严格，他都能做出令客户满意的产品来。所以，对于家装公司来说，靠谱的工人/工长很重要。对软件公司来说，经验丰富的技术带头人特别重要。

很多家装企业对于吸纳新的工长特别慎重。都要严加考核。

## 都不透明

---

软件：我们很难衡量一个软件的工作量。比较常见的方式是人月或者人天。例如：注册功能2人、天，消息推送功能1人天。

但是用户是完全无法判断这个估算的是否是准确的。

这个估算的时间，是内部一份（给公司核算成本用），外部一份（给软件客户用）这两种时间肯定是完全不一样的。

对于谈下来的价钱，完全取决于甲方可以给多少。同样的项目，甲方给到乙方的价格是100W，乙方可以转手就20W外包给第三方。最后这个项目可能被再次以5W的成本转包给在校研究生。质量就根本无法保证了。

对于家装来说，大部分业主也都不懂装修，看到里面各种 主材，辅料，工人的成本核算，马上晕菜。而且 材料商给内线人员（设计师）的价格，跟给到业主的价格完全不一样，回扣可能会达到20%之多。另外，施工过程中也存在着各种的不透明，如：隐蔽工程的要价（水电），多余的辅料和主材 的处理，其他地方。。。另外，回扣也充斥着 家装公司内部成员。例如，设计师的回扣是3，4，5的逻辑。（公司给的单子，扣3%，工长介绍的：4%，自己拿的：5%），设计师跟工长勾结，就可以把单子给以手工作坊的形式从公司手中拿下来。

再如：各种主材的报价，也都完全不一样。工厂给到代理的价格，可能是3.8折，而对于用户来说，价格就有很大的浮动。

而且，现在各种 688, 599这样的套餐，都是价格不透明的产物。我们去买电脑，内存，显卡，机箱，CPU，都是单独报价的。价格非常透明。我认为合理的主材报价应该用多少就报多少。每项加起来核算，用户口服心服。

## TODO：其他章节。关于软件的估算。

---

且对于新手来说，99%的估算不准确。因为他没有经历过的东西，是无法给出估算时间的。在有经验的技术负责人带领下，可以给出时间，但是也不准确。只有经验丰富的老手，对于同样的技术用过了几次，才能给出估算时间。

## 都期待着变革，但是步履缓慢

---

## 都需要用户给到频繁的反馈

---

## 水平都参差不齐

---

## 都无法大规模的被改造。

---

# 这是一个不明确的行业

---

家装行业,充满了不透明.例如:看不到某个单品的真实报价.你去实体店买个沙发,一万.设计师去买,4千.

软件行业,则充满了不明确.一方面是脑力劳动难于衡量(你能说方程一的工作量是方程二的工作量的X倍吗),一方面是用户的需求是不明确的(软件开发,不变的是需求一直在变化),一方面是外行人对某个技术的细节不了解.哪怕同是技术人员,做移动app的也不懂后台服务器的技术,做web的也不懂移动app的技术.所以,处处不明确,到处都是坑.

## 脑力劳动难于衡量.

---

举个例子, (图片)

## 工作量无法明确衡量.

---

只有亲身经历过技术细节,程序员才能对它进行准确估算.否则,对于一个他本身不熟悉的问题,他无法给出准确时间.

另外,同样是做一个用户注册的需求,用手机/邮箱注册,跟用第三方(微信/微博)注册相比,后者的难度就大了不少.不但涉及到程序方面的问题,还要做各种申请,填写各种表格,阅读第三方平台的接口.

## 用户的需求是不明确的.

---

见: 一句话需求

# 全栈工程师

---

不要指望靠流程解决问题。最终是靠人。

## 角色的缘起

---

受丰田的影响，软件行业中的从业人员，被人为的分成若干角色：

架构师（项目管理者） UI设计， 程序员（码农） 测试 运维 产品经理

如果是从产品的角度来看，这些角色需要精简（一个人身兼多职）

## 沟通的成本太高

---

一个人做项目，可以不需要与人交流，这个人在沟通上的成本是0。

两个人做项目，必须两个人互相交流。

根据实际情况来看，一个项目的最佳人选是4个人以下。最好是2，3个精英。因为划分任务 很负责：

- 你必须做到尽量公平：每个人的任务一样多
- 你必须做好划分的合理：每个子任务之间是可以独立拆分开，不互相依赖的。
- 你必须考虑到项目的工期，让擅长的人做擅长的事。或者预先留出学习的时间。

## 不好的流程会催生出坏人

---

最常见的例子：为什么 产品经理与程序员的关系一直被人诟病？（产品经理被打）

为什么测试人员跟开发人员是矛盾？

为什么运维人员讨厌部署？

例如，在互联网公司中，程序员的代码要生效，是需要给运维人员写申请的。过程一般是：

- 程序员填写好一份部署文档
- 程序员把部署文档 写邮件给运维人员
- 运维人员要求程序员打印出来，签字
- 运维人员按照程序员的要求，操作服务器。
- 运维人员告诉程序员：部署完毕，你快测试看有没有问题。

这里的切身体会是，程序员每做一次部署，需要：

- 写一份部署文档
- 签字，证明除了问题都是程序员的责任
- 给运维的人发邮件，打电话
- 做最简单的一个部署，也要2，3个小时。

运维人员每做一次部署，也仿佛掉了一层皮：

- 拿到部署文档
- 按照部署文档的提示，一步步的操作数据库。

- 由于程序不是运维人员写的，所以他完全不知道服务器变慢时，是哪里出的问题，如何调试
- 部署的时候会想，怎么你的python 部署过程跟java部署完全不一样？
- 出问题的时候会想，怎么你的代码出问题要我给你承担责任？

所以，出了问题的时候，大家就会互相指责：

程序人员：服务器不归我管，是运维没有及时增加服务器数量。 运维人员：代码不是我写的。是程序写的不好，测试不完备。 测试人员：代码不是我写的。我在线上前一直在加班。人肉测试达不到100%覆盖率。

## 不要把程序员分成后端和前端

---

对于java/ios/php这样的经典开发语言，是比较啰嗦的。java程序员无法同时掌握多种语言。比如 CSS, javascript.

所以在十年前，招聘帖子就分成 web开发工程师和 html 前端工程师。对于HTML前端工程师, 他的工作任务是把美工设计的静态网站图片，设计成html + CSS + javascript 代码。

这是我知道最早的前后端之说。

后来，随着移动端开发的兴起，又有很多人开始参与android/ios/winphone等的开发。随便一款语言的学习都要读厚厚的一本书，所以做android/ios的人根本无暇学习web端 的知识。

所以，大家也就接受了这样的观点：必须分成前后端。

web中的前后端能相处的相安无事。但是web端 与 app端的开发者就不友好了。因为很多时候 会出现踢皮球的现象：

某个项目开始后，需求定好了，该划分工作了，但是发现有个需求，放在app端也可以， 放在web端也可以。该怎么办？

于是踢皮球的情况就产生了。web端认为这个东西应该做在app端，自己的任务已经够多了。 app端认为应该做在web端，这个事情可以在web端做，为什么就不做呢？

所以，这样的结果往往是：

- 前后端关系不好，各种踢皮球
- 没出问题还好。出了问题互相推诿，几次下来不踢皮球的人发现自己卖力不讨好，也开始踢了。
- 工作效率低下。半天可以做好的东西，往往按照一周来估计。
- 产品经理两面不是人。 前后端都觉得你干嘛要设计这个功能呢？

## 全栈工程师的特点

---

全栈工程师就可以解决这个问题。

他的特点是：

- 又能做web端程序
- 又能做app端程序
- 又能做HTML/CSS/JQUERY程序
- 还能把静态图片切图。
- 还能做自动化的测试。
- 还能参与需求，做原型图。

特点是：

出了问题不会互相指责 技术全面，又懂服务器，又懂代码，开发的质量自然就好。 解决问题速度特别快 为公司省钱。

在悦家 (www.yuewz.com) 和优优宝(www.uubpay.com), 我们完全采用了这样的方式: 一个全栈工程师 集: 开发人员、运维人员、测试人员于一身, 又会写移动端代码, 又会写服务器端代码, 又会部署, 极大的减轻了沟通的成本, 减少了不同角色之间的纠纷, 提高了效率。

我们程序员的工作内容是这样的:

项目开始时, 先跟产品经理, 一起把需求确定下来。参与到定制流程图。

需求定好后, 自己主动申领任务。

美工做UI设计。与此同时, 程序员开始工作, 开两个调试窗口: app端是一个, web页面是一个。他需要什么功能, 先在app 端写一部分程序, 需要接口的时候, 再切换到web端的代码, 写程序, 调试。然后再切换回来。。。因为自己要什么接口自己是最清楚的, 不需要沟通, 立马写好了。

在每天下班前: 程序员提交代码到web服务器, 部署, 重启服务器使之生效。程序员发布最新的app代码到应用商店。或者把它按照到产品经理的手中。

产品经理每天早上, 把测试机上的最新代码过一遍, 看看其中有哪些BUG。记录在 BUG系统中。然后带领程序员开早会: 哪些功能已经实现了, 哪些BUG已经修改完, 还差哪些任务。

这样的团队, 只需要:

产品经理兼测试: 一名 全栈工程师: 2~4名(这里面有一个技术小组长) UI美工: 1名

技术负责人只需要:

- 提供技术支持
- 出现技术争论时, 直接拍板。
- 查看存在哪些问题。

## 实战情况

---

根据实际情况来看, 全栈工程师完全避免了前后端踢皮球的问题, 完全避免了沟通耗时的问题。哪怕是菜鸟, 都可以通过避免上述的问题而间接提高生产率。

在悦家和优优宝, 一个项目, 通常有2个一年经验的全栈工程师就足够了。

## 有可能产生全栈攻城狮的技术背景

---

一般来说, java, php, c 等传统语言背景的人, 无法做成全栈攻城狮. 因为他们所使用的语言过于复杂, 笨重, 很多时候他们是有心无力去学习其他知识的。

全站工程师比较多的是 ruby 等新兴语言, 这类语言的特点是: 轻盈, 表达力强, 简洁。

粗略的讲: 如果一门语言不需要使用什么设计模式, 就能工作的很好, 那么这门语言就是比较轻盈的。

## 死亡案例

---

### 一句话需求

---

很多时候, 甲方没有提出明确的需求, [仅仅有一句话](#), 乙方就会按照自己的想法去做. 结果导致做出来的完全不一样.

深层的意义, 就是甲方还没把自己的需求细化出来. 这样的状态是根本无法开工的.

### 大而长的交付周期

---

交付周期在半年以上的项目, 必死.

### 不合理的价格

---

过低和过高都不行.

### 层层转包

---

某公司, 接手到200W的项目, 转手50W 二次转包给了B公司. B公司10W 转包给了C公司. 到最后C公司用10 W的价格做出来的东西, 原甲方根本无法接受.

### 不能频繁见面

---

见面或者沟通的频次, 应该在每周至少3次. 这样才能保证有了问题随时发问.

### 不靠谱的程序员

---

因为最终干活儿靠的是程序员. 如果程序员不给力, 其他事情做的再好也不行.

### 异地外包

---

沟通的优先级是: (括号内是优点)

面对面(可以看到对方的面孔) > 电话(听到声音) > 即时聊天(及时得到反馈) > Email

只能电话沟通的项目, 无法有视觉的交流, 无法有全方位的信任, 直接导致沟通不畅. 最好的沟通方式应该是: 两个人同时站在大白板边上, 一边比划一边说. 因为很多情况下, 人是无法把内心的东西光靠说就能说明白的.

### 用户不切实际的过高期待

---

比如花了10块钱, 却一定要得到100块的东西.

### 被人用现成的项目去套

---



如上面题目, 在当今的市场上,就算你花10块钱,去 要求对方给你提供100块的软件,对方很可能也会答应的 为什么呢? 因为对方用现成的软件给你套:

以商城为例:

- 基本功能都有 (上架个商品, 制定价格,支付)
- 我修改下标题
- 再修改下背景色.

2,3 天搞定了.

但是, 进一步的话,就没法做了. 比如,我要增加个会员系统, 我要为某个商品设置三种价格: 正常价格, 会员价格,团购价格. 这些都涉及到修改底层架构. 这样的情况乙方是无法交付的.